aws

開発者ガイド

Amazon Lookout for Vision

Amazon Lookout for Vision: 開発者ガイド

Table of Contents

Amazon Lookout for Vision とは？

Amazon Lookout for Vision を使用して、工業製品の視覚的欠陥を正確かつ大規模に見つけることができます。工業製品の部品の欠落、車両や構造物の損傷、生産ラインの異常、シリコンウエハーやプリント基板のコンデンサの欠落など、品質が重要視される物品の微小な欠陥もコンピュータビジョンで識別することができます。

主な利点

Amazon Lookout for Vision には次の利点があります。

- プロセスを迅速かつ効率的に改善 — Amazon Lookout for Vision を使用して、コンピュータビジョンベースの検査を産業プロセスに迅速かつ効率的に大規模に実装できます。ベースラインの良品画像を 30 枚程度用意すれば、Lookout for Vision が欠陥検出用のカスタム ML モデルを自動で構築します。その後、IP カメラからの画像をバッチまたはリアルタイムで処理して、へこみ、スクラッチ、傷などの異常を迅速かつ正確に特定できます。
- 生産品質を高め、迅速に — Amazon Lookout for Vision を使用すると、生産プロセスの欠陥をリアルタイムで削減できます。視覚的な異常をダッシュボードで確認し、報告することで、不良の発生を未然に防ぎ、生産品質の向上とコスト削減を実現します。
- 運用コストの削減 — Amazon Lookout for Visionは、目視検査データの傾向をレポートし、不良率の高い工程を特定したり、最近の不良のばらつきにフラグを立てたりすることができます。この情報を使って、コストのかかる計画外のダウンタイムが発生する前に、プロセスラインのメンテナンスを予定するか、別の機械に生産を振り向けるかを決定することができます。

Amazon Lookout for Vision をエンドユーザーとして初めてお使いですか？

初めて Amazon Lookout for Vision をご利用になる方は、以下の項目を順番にお読みいただくことをお勧めします。

1. Amazon Lookout for Vision のセットアップ – このセクションでは、お客様のアカウント情報を設定します。
2. Amazon Lookout for Vision コンソールの開始 – このセクションでは、最初の Amazon Lookout for Vision モデルの作成について説明します。

Amazon Lookout for Vision のセットアップ

このセクションでは、AWS アカウントにサインアップして Amazon Lookout for Vision をセットアップします。

Amazon Lookout for Vision をサポートする AWS リージョンの詳細については、「Amazon Lookout for Vision エンドポイントとクォータ」を参照してください。

トピック

ステップ 1: AWS アカウントを作成する

このステップでは、 AWS アカウントにサインアップし、管理ユーザーを作成します。

トピック

にサインアップする AWS アカウント

がない場合は AWS アカウント、次の手順を実行して作成します。

にサインアップするには AWS アカウント

1. https://portal.aws.amazon.com/billing/signup を開きます。
2. オンラインの手順に従います。

   サインアップ手順の一環として、通話呼び出しを受け取り、電話キーパッドで検証コードを入力するように求められます。

にサインアップすると AWS アカウント、 AWS アカウントのルートユーザーが作成されます。ルートユーザーには、アカウントのすべての AWS サービス とリソースへのアクセス権があります。セキュリティのベストプラクティスとして、ユーザーに管理アクセスを割り当て、ルートユーザーのみを使用してルートユーザーアクセスが必要なタスクを実行してください。

AWS サインアッププロセスが完了すると、 から確認メールが送信されます。https://aws.amazon.com/ の [マイアカウント] を選んで、いつでもアカウントの現在のアクティビティを表示し、アカウントを管理できます。

管理アクセスを持つユーザーを作成する

にサインアップしたら AWS アカウント、 を保護し AWS アカウントのルートユーザー、 を有効にして AWS IAM Identity Center、日常的なタスクにルートユーザーを使用しないように管理ユーザーを作成します。

のセキュリティ保護 AWS アカウントのルートユーザー

1. ルートユーザーを選択し、 AWS アカウント E メールアドレスを入力して、アカウント所有者AWS Management Consoleとして にサインインします。次のページでパスワードを入力します。

   ルートユーザーを使用してサインインする方法については、AWS サインイン ユーザーガイドの「ルートユーザーとしてサインインする」を参照してください。
2. ルートユーザーの多要素認証 (MFA) を有効にします。

   手順については、「IAM ユーザーガイド」の AWS アカウント 「ルートユーザーの仮想 MFA デバイスを有効にする (コンソール)」を参照してください。

管理アクセスを持つユーザーを作成する

1. IAM アイデンティティセンターを有効にします。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「AWS IAM Identity Centerの有効化」を参照してください。
2. IAM アイデンティティセンターで、ユーザーに管理アクセスを付与します。

を ID ソース IAM アイデンティティセンターディレクトリ として使用する方法のチュートリアルについては、「 ユーザーガイド」の「デフォルト でユーザーアクセス IAM アイデンティティセンターディレクトリを設定するAWS IAM Identity Center 」を参照してください。

管理アクセス権を持つユーザーとしてサインインする

- IAM アイデンティティセンターのユーザーとしてサインインするには、IAM アイデンティティセンターのユーザーの作成時に E メールアドレスに送信されたサインイン URL を使用します。

  IAM Identity Center ユーザーを使用してサインインする方法については、「 AWS サインインユーザーガイド」の AWS 「 アクセスポータルにサインインする」を参照してください。

追加のユーザーにアクセス権を割り当てる

1. IAM アイデンティティセンターで、最小特権のアクセス許可を適用するというベストプラクティスに従ったアクセス許可セットを作成します。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」を参照してください。
2. グループにユーザーを割り当て、そのグループにシングルサインオンアクセス権を割り当てます。

   手順については、「AWS IAM Identity Center ユーザーガイド」の「グループの参加」を参照してください。

ステップ 2: 権限をセットアップする

Amazon Lookout for Vision を使用するには、Lookout for Vision コンソール、 AWS SDK オペレーション、およびモデルトレーニングに使用する Amazon S3 バケットへのアクセス許可が必要です。

> **Note**
>
> AWS SDK オペレーションのみを使用する場合は、 AWS SDK オペレーションを対象とするポリシーを使用できます。詳細については、「SDK 権限をセットアップする」を参照してください。

トピック

AWS 管理ポリシーによるコンソールアクセスの設定

次の AWS マネージドポリシーを使用して、Amazon Lookout for Vision コンソールおよび SDK オペレーションに適切なアクセス許可を適用します。

- AmazonLookoutVisionConsoleFullAccess — Amazon Lookout for Vision コンソールと SDK オペレーションへのフルアクセスを許可します。コンソールバケットを作成するための `AmazonLookoutVisionConsoleFullAccess` アクセス許可が必要です。詳細については、「ステップ 3: コンソールバケットを作成する」を参照してください。
- AmazonLookoutVisionConsoleReadOnlyAccess— Amazon Lookout for Vision コンソールおよび SDK オペレーションへの読み取り専用アクセスを許可します。

権限を割り当てるには、「権限の割り当て」をご覧ください。

AWS 管理ポリシーの詳細については、「AWS 管理ポリシー」を参照してください。

Amazon S3 バケット許可をセッティングする

Amazon Lookout for Vision では、Amazon S3 バケットを使用して以下のファイルを保存します。

- データセットイメージ — モデルのトレーニングに使用される画像。詳細については、「データセットの作成」を参照してください。
- Amazon SageMaker Ground Truth 形式のマニフェストファイル。例えば、 SageMaker GroundTruth ジョブからのマニフェストファイルの出力などです。詳細については、「Amazon SageMaker Ground Truth マニフェストファイルを使用したデータセットの作成」を参照してください。
- モデルトレーニングからの出力。

コンソールを使用する場合、Lookout for Vision はプロジェクトを管理するために Amazon S3 バケット (コンソールバケット) を作成します。`LookoutVisionConsoleReadOnlyAccess` および

LookoutVisionConsoleFullAccess 管理ポリシーには、コンソールバケットの Amazon S3 アクセス許可が含まれます。

コンソールバケットを使用して、データセットイメージと SageMaker Ground Truth 形式のマニフェストファイルを保存できます。または、別の Amazon S3 バケットを使用できます。バケットは AWS アカウントによって所有されている必要があり、Lookout for Vision を使用している AWS リージョンにある必要があります。

別のバケットを使用するには、必要なユーザーまたはグループに次のポリシーを追加します。my-bucket を希望するバケットの名前に置き換えます。IAM ポリシー追加の詳細については、「IAM ポリシーの作成」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionS3BucketAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetBucketLocation",
                "s3:ListBucket"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket"
            ]
        },
        {
            "Sid": "LookoutVisionS3ObjectAccessPermissions",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket/*"
            ]
        }
    ]
}
```

権限を割り当てるには、「権限の割り当て」をご覧ください。

権限の割り当て

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM Identity Center :

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。

- IAM 内で、ID プロバイダーによって管理されているユーザー:

  ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。

- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。IAM ユーザーガイドの「ユーザー (コンソール) へのアクセス許可の追加」の指示に従います。

ステップ 3: コンソールバケットを作成する

Amazon Lookout for Vision コンソールを使用するには、コンソールバケットと呼ばれる Amazon S3 バケットが必要です。コンソールバケットには以下が格納されます:

- コンソールによりデータセットにアップロードする画像。
- コンソールにより開始するモデルトレーニングのトレーニング結果。
- トライアル検出の結果。
- コンソールを使用して、S3 バケット内の画像への自動ラベリングによりデータセットを作成したときに、コンソールが作成する一時的なマニフェストファイル。コンソールはマニフェストファイルを削除しません。

新しい AWS リージョンで Amazon Lookout for Vision コンソールを初めて開くと、Lookout for Vision がユーザーに代わってコンソールバケットを作成します。 AWS SDK オペレーションやデータセットの作成などのコンソールタスクでバケット名を使用する必要がある可能性があるため、コンソールバケット名を書き留めます。

または、コンソールバケットは、Amazon S3 コンソールを使用して作成できます。Amazon S3 バケットポリシーによって Amazon Lookout for Vision コンソールがコンソールバケットを正常に作成できない場合は、この方法を使用してください。たとえば、Amazon S3 バケットの自動作成を禁止するポリシーなどによってです。

> **Note**
>
> Lookout for Vision コンソールではなく AWS SDK のみを使用する場合は、コンソールバケットを作成する必要はありません。選択した名前で別の S3 バケットを使用できます。

コンソールバケット名の形式は、`lookoutvision-<region>-<random value>` です。ランダム値により、バケット名の間に衝突が発生しないようにします。

トピック

- Amazon Lookout for Vision コンソールによるコンソールバケットの作成
- Amazon S3 によるコンソールバケットの作成
- コンソールバケット設定

Amazon Lookout for Vision コンソールによるコンソールバケットの作成

Amazon Lookout for Vision コンソールを使用して AWS リージョンのコンソールバケットを作成するには、次の手順に従います。有効化されている S3 バケット設定については、コンソールバケット設定 を参照してください。

Amazon Lookout for Vision コンソールを使用してコンソールバケットを作成するには

1. 使用しているユーザーまたはグループに `AmazonLookoutVisionConsoleFullAccess` 権限があることを確かめてください。詳細については、「ステップ 2: 権限をセットアップする」を参照してください。
2. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
3. ナビゲーションバーで、[リージョンを選択] を選びます。次に、コンソールバケットを作成する AWS リージョンを選択します。
4. [開始する] を選びます。

5. 現在の AWS リージョンでコンソールを初めて開いた場合は、「初回セットアップ」ダイアログボックスで以下を行います:

   a. 表示されている Amazon S3 バケットの名前をコピーします。この情報は後で必要になります。

   b. Amazon Lookout for Vision にコンソールバケットの作成を代行させる場合は、[S3バケットを作成] を選択します。

   現在の AWS リージョンのコンソールバケットが既に存在する場合、初回設定ダイアログボックスは表示されません。

6. ブラウザウィンドウを閉じます。

Amazon S3 によるコンソールバケットの作成

Amazon S3 を使用してコンソールバケットを作成できます。バケットの作成では、Amazon S3 バージョニングを有効にする必要があります。Amazon S3 ライフサイクル構成を使用して、最新でない (以前の) バージョンのオブジェクトを削除し、不完全なマルチパートアップロードを削除することをお勧めします。オブジェクトの現在のバージョンを削除するライフサイクル構成はお勧めしません。Amazon Lookout for Vision コンソールにより作成したコンソールバケットで有効にされる S3 バケット設定の詳細については、コンソールバケット設定 を参照してください。

1. コンソールバケットを作成する AWS リージョンを決定します。サポート対象リージョンについては、「Amazon Lookout for Vision エンドポイントおよびクォータ」を参照してください。
2. 「バケットの作成」にある S3 コンソールの手引きに従ってバケットを作成します。以下の操作を実行します。

   a. ステップ 3 では、先頭に `lookoutvision-`*`region`*`-`*`your-identifier` が付いたバケット名を指定します。`region` を、前のステップで選択したリージョンコードに変更します。`your-identifier` を、選択した固有の識別子に変更します。例えば、次のようになります: `lookoutvision-us-east-1-my-console-bucket-1`

   b. ステップ 4 では、使用する AWS リージョンを選択します。

3. 「バケットのバージョニングの有効化」にある S3 コンソールの手引きに従って、バケットのバージョニングを有効にします。
4. (オプション) 「バケットのライフサイクル構成の設定」にある S3 コンソールの手引きに従って、バケットのライフサイクル構成を指定します。次を行い、最新でない (以前の) バージョン

のオブジェクトを除去するとともに、不完全なマルチパートアップロードを削除します。ステップ 6、8、9、10 を行う必要はありません。

a. ステップ 5 で、[バケット内のすべてのオブジェクトに適用] を選択します。

b. ステップ 7 では、[最新でないバージョンのオブジェクトを完全に削除] と [期限切れのオブジェクト削除マーカー、または不完全なマルチパートアップロードを削除] を選択します。

c. ステップ 11 では、最新でないバージョンのオブジェクトを削除するまでの待機日数を入力します。

d. ステップ 12 では、不完全なマルチパートアップロードを削除するまでの待機日数を入力します。

コンソールバケット設定

Amazon Lookout for Vision コンソールによりコンソールバケットを作成すると、コンソールバケットで次の設定が有効になります。

- コンソールバケット内のオブジェクトのバージョニング。
- コンソールバケット内のオブジェクトのサーバー側暗号化。
- 最新でないオブジェクト (30 日) と不完全なマルチパートアップロード (3 日間) を削除するためのライフサイクル構成。
- コンソールバケットへのパブリックアクセスをブロックします。

ステップ 4: AWS CLI と AWS SDKsを設定する

次の手順は、 AWS Command Line Interface （ AWS CLI) と AWS SDKsをインストールする方法を示しています。このドキュメントの例では、 AWS CLI、Python、Java AWS SDKsを使用しています。

トピック

- AWS SDK のインストール
- プログラマチックアクセス権を付与する
- SDK 権限をセットアップする
- Amazon Lookout for Vision オペレーションを呼び出す

AWS SDK のインストール

AWS SDKs をダウンロードして設定するには、次の手順に従います。

AWS CLI と AWS SDKsをセットアップするには

- および使用する AWS SDKsをダウンロードしてインストールAWS CLIします。このガイドでは、、Java AWS CLI、および Python の例を示します。 AWS SDKs「Amazon Web Services のツール」を参照してください。

プログラマチックアクセス権を付与する

このガイドの AWS CLI および コード例は、ローカルコンピュータまたは Amazon Elastic Compute Cloud インスタンスなどの他の AWS 環境で実行できます。例を実行するには、例が使用する AWS SDK オペレーションへのアクセスを許可する必要があります。

トピック

- ローカルコンピュータでのコードの実行
- AWS 環境でのコードの実行

ローカルコンピュータでのコードの実行

ローカルコンピュータでコードを実行するには、短期間の認証情報を使用して AWS SDK オペレーションへのアクセス権をユーザーに付与することをお勧めします。ローカルコンピュータで AWS CLI および コード例を実行する方法の詳細については、「」を参照してくださいローカルコンピュータでのプロファイルの使用。

ユーザーが の AWS 外部で を操作する場合は、プログラムによるアクセスが必要です AWS Management Console。プログラムによるアクセスを許可する方法は、 にアクセスするユーザーのタイプによって異なります AWS。

ユーザーにプログラマチックアクセス権を付与するには、以下のいずれかのオプションを選択します。

プログラマチックアクセス権を必要とするユーザー	目的	方法
ワークフォースアイデンティティ (IAM Identity Center で管理されているユーザー)	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 ユーザーガイド」の「 を使用する AWS CLI ためのの設定 AWS IAM Identity CenterAWS Command Line Interface 」を参照してください。 • AWS SDKs 、ツール、AWS APIs「 SDK とツールのリファレンスガイド」の「IAM Identity Center 認証」を参照してください。AWS SDKs
IAM	一時的な認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	「IAM ユーザーガイド」の「 AWS リソースでの一時的な認証情報の使用」の手順に従います。
IAM	(非推奨) 長期認証情報を使用して、、 AWS SDKs AWS CLI、または AWS APIs。	使用するインターフェイス用の手引きに従ってください。 • については AWS CLI、「 AWS Command Line Interface ユーザーガイド」の「IAM ユーザー認証情報を使用した認証」を参照してください。 • AWS SDKs「 SDK とツールのリファレンスガイド」の「長期的な認証情報を使

プログラマチックアクセス権を必要とするユーザー	目的	方法
		用した認証」を参照してください。 AWS SDKs • AWS APIsユーザーガイド」の「IAM ユーザーのアクセスキーの管理」を参照してください。

ローカルコンピュータでのプロファイルの使用

このガイドの AWS CLI および コード例は、 で作成した短期認証情報を使用して実行できますローカルコンピュータでのコードの実行。認証情報や他の設定情報を取得するため、たとえばサンプルでは `lookoutvision-access` という名前のプロファイルを使用しています:

```
session = boto3.Session(profile_name='lookoutvision-access')
lookoutvision_client = session.client("lookoutvision")
```

プロファイルが表すユーザーには、Lookout for Vision SDK オペレーションおよび例で必要なその他の AWS SDK オペレーションを呼び出すアクセス許可が必要です。詳細については、「SDK 権限をセットアップする」を参照してください。権限を割り当てるには、「権限の割り当て」を参照してください。

AWS CLI および コード例で動作するプロファイルを作成するには、次のいずれかを選択します。作成するプロファイルの名前が `lookoutvision-access` であることを確かめてください。

- IAM が管理するユーザー - 「IAM ロール (AWS CLI) の切り替え」の手順に従います。
- ワークフォース ID (によって管理されるユーザー AWS IAM Identity Center) — を使用するように AWS CLI を設定するの手順 AWS IAM Identity Centerに従います。コード例については統合開発環境 (IDE) を使用することが推奨されます。こちらは、IAM Identity Center による認証を許可する AWS Toolkit をサポートしています。Java の例については「Start building with Java」を参照してください。Python の例については「Start building with Python」を参照してください。その他の詳細については「IAM Identity Center credentials」を参照してください。

> **Note**
>
> コードを使用して、短期間の認証情報を取得できます。詳細については「IAM ロール (AWS API) の切り替え」を参照してください。IAM Identity Center の場合は、「Getting IAM role credentials for CLI access」にある手順に従って、ロールの短期間の認証情報を取得します。

AWS 環境でのコードの実行

AWS Lambda 関数で実行されている本番コードなどの AWS 環境で、ユーザー認証情報を使用して AWS SDK 呼び出しに署名しないでください。代わりに、コードに必要なアクセス権限を定義するロールを設定します。次に、コードを実行する環境にそのロールをアタッチします。ロールをアタッチして一時的な認証情報を利用できるようにする方法は、コードを実行する環境によって異なります。

- AWS Lambda 関数 — Lambda 関数の実行ロールを引き受けるときに Lambda が自動的に関数に提供する一時的な認証情報を使用します。認証情報は Lambda の環境変数で使用できます。プロファイルを指定する必要はありません。詳細については、「Lambda 実行ロール」を参照してください。
- Amazon EC2 - Amazon EC2 のインスタンスメタデータエンドポイント認証情報プロバイダーを使用します。このプロバイダーは、Amazon EC2 インスタンスにアタッチされた Amazon EC2 インスタンスプロファイルを使用して、認証情報を自動的に生成して更新します。詳細については、「Amazon EC2 インスタンスで実行されるアプリケーションに IAM ロールを使用してアクセス許可を付与する」を参照してください。
- Amazon Elastic Container Service - コンテナ認証情報プロバイダーを使用します。Amazon ECS は認証情報をメタデータエンドポイントに送信して更新します。指定するタスク IAM ロールは、アプリケーションが使用する認証情報を管理するための戦略を提供します。詳細については、「AWS サービスとやり取りする」を参照してください。
- Greengrass コアデバイス — TLS 相互認証プロトコルを使って AWS IoT Core に接続するには、X.509 証明書を使用します。これらの証明書により、デバイスは AWS 認証情報なしで AWS IoT とやり取りできます。AWS IoT 認証情報プロバイダは、X.509 証明書を使用してデバイスを認証し、特権が制限された一時的セキュリティトークンの形で AWS 認証情報を発行します。詳細については、「AWS サービスとやりとりする」を参照してください。

認証情報プロバイダーの詳細については、「標準認証情報プロバイダー」を参照してください。

SDK 権限をセットアップする

Amazon Lookout for Vision SDK オペレーションを使用するには、Lookout for Vision API や、モデルトレーニングに使用する Amazon S3 バケットへのアクセス権限が必要です。

トピック

- SDK オペレーション権限の付与
- Amazon S3 バケット権限の付与
- 権限の割り当て

SDK オペレーション権限の付与

タスクの実行に必要な権限 (最小特権) のみを付与することをお勧めします。例えば、 を呼び出すにはDetectAnomalies、 を実行するためのアクセス許可が必要ですlookoutvision:DetectAnomalies。オペレーションの権限を確認するには、「API リファレンス」をチェックしてください。

アプリケーションを始めたばかりの場合は、必要な特定の権限がわからない場合があるため、幅広い権限から始めることができます。 AWS 管理ポリシーは、開始に役立つ権限を提供します。

- AmazonLookoutVisionFullAccess — Amazon Lookout for Vision SDK オペレーションへのフルアクセスを許可します。
- AmazonLookoutVisionReadOnlyAccess — 読み取り専用 SDK オペレーションへのアクセスを許可します。

コンソールの管理ポリシーは、SDK オペレーションのアクセス権限も提供します。詳細については、「ステップ 2: 権限をセットアップする」を参照してください。

AWS 管理ポリシーの詳細については、「AWS 管理ポリシー」を参照してください。

アプリケーションに必要な権限がわかったら、ユースケースに応じたカスタマー管理ポリシーを定義することで、権限をさらに減らします。詳細については、「カスタマー管理ポリシー」を参照してください。

> **Note**
>
> 開始の手引きには s3:PutObject 権限が必要です。詳細については、「ステップ1: マニフェストファイルとアップロード画像を作成する」を参照してください。

権限を割り当てるには、「権限の割り当て」を参照してください。

Amazon S3 バケット権限の付与

モデルをトレーニングするには、イメージ、マニフェストファイル、およびトレーニング出力を格納するための適切なアクセス許可を持つ Amazon S3 バケットが必要です。バケットは AWS アカウントによって所有され、Amazon Lookout for Vision を使用している AWS リージョンにある必要があります。

SDK 専用の管理ポリシー (AmazonLookoutVisionFullAccessおよびAmazonLookoutVisionReadOnlyAccess) には Amazon S3 バケットの許可が含まれておらず、既存のコンソールバケットを含む使用するバケットにアクセスするには、以下の許可ポリシーを適用する必要があります。

コンソールの管理ポリシー (AmazonLookoutVisionConsoleFullAccessおよびAmazonLookoutVisionConsoleReadOnlyAccess) はコンソールバケットへのアクセス許可を含めます。SDK オペレーションでコンソールバケットにアクセスしていて、コンソール管理ポリシーのアクセス許可を持っている場合は、次のポリシーを使用する必要はありません。詳細については、「ステップ 2: 権限をセットアップする」を参照してください。

タスク許可の決定

次の情報を使用して、実行するタスクに必要なアクセス許可を決定します。

データセットの作成

を使用してデータセットを作成するにはCreateDataset、次のアクセス許可が必要です。

- s3:GetBucketLocation — Lookout for Vision により、バケットがLookout for Visionを使用しているのと同じ地域にあることを検証することができます。
- s3:GetObject — DatasetSource 入力パラメータで指定されたマニフェストファイルへのアクセスを許可します。。マニフェストファイルの S3 オブジェクトのバージョンを正確に指定し

たい場合は、マニフェストファイル上で s3:GetObjectVersion も必要です。詳細については、「S3 バケットでバージョニングを使用する」を参照してください。

モデルの作成

でモデルを作成するにはCreateModel、次のアクセス許可が必要です。

- s3:GetBucketLocation — Lookout for Vision により、バケットがLookout for Visionを使用しているのと同じ地域にあることを検証することができます。
- s3:GetObject — プロジェクトのトレーニングデータセットとテストデータセットで指定された画像へのアクセスを許可します。
- s3:PutObject — 指定したバケットにトレーニング出力を格納するアクセス許可。OutputConfig パラメータで出力バケットの場所を指定します。オプションで、S3Location 入力フィールドの Prefix フィールドで指定されたオブジェクトキーのみに許可を絞り込むことができます。詳細については、「」を参照してくださいOutputConfig。

イメージ、マニフェストファイル、およびトレーニング出力へアクセスする

Amazon S3 バケットのアクセス許可は、Amazon Lookout for Vision オペレーションのレスポンスを表示するために必要ありません。操作の応答で参照される画像、マニフェストファイル、トレーニング出力にアクセスする場合は、s3:GetObject 許可が必要です。バージョン管理された Amazon S3 オブジェクトにアクセスする場合は、s3:GetObjectVersion アクセス許可が必要です。

Amazon S3 バケットポリシーをセッティングする

以下のポリシーを使用して、データセットの作成 (CreateDataset)、モデルの作成 (CreateModel)、画像、マニフェストファイル、およびトレーニング出力へのアクセスに必要な Amazon S3 バケットの権限を指定できます。*my-bucket* の値を使用したいバケットの名前に変更します。

ポリシーは、ニーズに合わせて調整できます。詳細については、「タスク許可の決定」を参照してください。必要なユーザーにポリシーを追加します。IAM ポリシーの作成の詳細については、「IAM ポリシーを作成する」 を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": "LookoutVisionS3BucketAccess",
            "Effect": "Allow",
            "Action": "s3:GetBucketLocation",
            "Resource": [
                "arn:aws:s3:::my-bucket"
            ],
            "Condition": {
                "Bool": {
                    "aws:ViaAWSService": "true"
                }
            }
        },
        {
            "Sid": "LookoutVisionS3ObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::my-bucket/*"
            ],
            "Condition": {
                "Bool": {
                    "aws:ViaAWSService": "true"
                }
            }
        }
    ]
}
```

権限を割り当てるには、「権限の割り当て」を参照してください。

権限の割り当て

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- のユーザーとグループ AWS IAM Identity Center :

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。
- IAM 内で、ID プロバイダーによって管理されているユーザー:

ID フェデレーションのロールを作成します。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。

- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (お奨めできない方法) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加する。詳細については、「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス権限の追加」を参照してください。

Amazon Lookout for Vision オペレーションを呼び出す

次のコードを実行して、Amazon Lookout for Vision API に呼び出せることを確認します。このコードは、現在の AWS リージョンの AWS アカウント内のプロジェクトを一覧表示します。まだプロジェクトを作成していない場合、レスポンスは空ですが、`ListProjects` オペレーションを呼び出せることを確認します。

一般的に、サンプル関数を呼び出すには、AWS SDK Lookout for Vision クライアントと他の必須パラメータが必要です。AWS SDK Lookout for Vision クライアントはメイン関数で宣言されます。

コードが失敗した場合は、使用するユーザーが適切な権限を持つことをチェックします。また、Amazon Lookout for Vision として使用している AWS リージョンが、一部の AWS リージョンで利用できないことも確認してください。

Amazon Lookout for Vision オペレーションを呼び出すには

1. まだインストールしていない場合は、 と AWS SDKsをインストール AWS CLI して設定します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のサンプルコードを使用して、プロジェクトを表示します。

   CLI

   `list-projects` コマンドを使用して、アカウントのプロジェクトを一覧表示します。

   ```
   aws lookoutvision list-projects \
   --profile lookoutvision-access
   ```

Python

このコードは AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0


from botocore.exceptions import ClientError
import boto3

class GettingStarted:

    @staticmethod
    def list_projects(lookoutvision_client):
        """
        Lists information about the projects that are in in your AWS account
        and in the current AWS Region.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        """
        try:
            response = lookoutvision_client.list_projects()
            for project in response["Projects"]:
                print("Project: " + project["ProjectName"])
                print("ARN: " + project["ProjectArn"])
                print()
            print("Done!")
        except ClientError:
            raise
def main():
    session = boto3.Session(profile_name='lookoutvision-access')
    lookoutvision_client = session.client("lookoutvision")

    GettingStarted.list_projects(lookoutvision_client)

if __name__ == "__main__":
    main()
```

Java V2

このコードは AWS Documentation SDK サンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/

package com.example.lookoutvision;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.services.lookoutvision.LookoutVisionClient;
import software.amazon.awssdk.services.lookoutvision.model.ProjectMetadata;
import
 software.amazon.awssdk.services.lookoutvision.paginators.ListProjectsIterable;
import software.amazon.awssdk.services.lookoutvision.model.ListProjectsRequest;
import
 software.amazon.awssdk.services.lookoutvision.model.LookoutVisionException;

import java.util.ArrayList;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;

public class GettingStarted {

    public static final Logger logger =
 Logger.getLogger(GettingStarted.class.getName());

    /**
     * Lists the Amazon Lookoutfor Vision projects in the current AWS account
 and
     * AWS Region.
     *
     * @param lfvClient   An Amazon Lookout for Vision client.
     * @return List<ProjectMetadata> Metadata for each project.
     */
    public static List<ProjectMetadata> listProjects(LookoutVisionClient
 lfvClient)
            throws LookoutVisionException {
```

```
        logger.log(Level.INFO, "Getting projects:");
        ListProjectsRequest listProjectsRequest = ListProjectsRequest.builder()
                .maxResults(100)
                .build();

        List<ProjectMetadata> projectMetadata = new ArrayList<>();

        ListProjectsIterable projects =
lfvClient.listProjectsPaginator(listProjectsRequest);

        projects.stream().flatMap(r -> r.projects().stream())
                .forEach(project -> {
                    projectMetadata.add(project);
                    logger.log(Level.INFO, project.projectName());
                });

        logger.log(Level.INFO, "Finished getting projects.");

        return projectMetadata;

    }

    public static void main(String[] args) throws Exception {

        try {

            // Get the Lookout for Vision client.
            LookoutVisionClient lfvClient = LookoutVisionClient.builder()

.credentialsProvider(ProfileCredentialsProvider.create("lookoutvision-access"))
                    .build();

            List<ProjectMetadata> projects = Projects.listProjects(lfvClient);

            System.out.printf("Projects%n--------%n");

            for (ProjectMetadata project : projects) {
                System.out.printf("Name: %s%n", project.projectName());
                System.out.printf("ARN: %s%n", project.projectArn());
                System.out.printf("Date: %s%n%n",
project.creationTimestamp().toString());
            }

        } catch (LookoutVisionException lfvError) {
```

```
            logger.log(Level.SEVERE, "Could not list projects: {0}: {1}",
                    new Object[] { lfvError.awsErrorDetails().errorCode(),
                            lfvError.awsErrorDetails().errorMessage() });
            System.out.println(String.format("Could not list projects: %s",
 lfvError.getMessage()));
            System.exit(1);
        }

    }

}
```

ステップ 5: (オプション) 独自の AWS Key Management Service キーの使用

AWS Key Management Service (KMS) を使用して、Amazon S3 バケットに保存する入力画像および動画のキーを管理できます。

デフォルトでは、イメージは AWS が所有および管理するキーで暗号化されます。また、独自の AWS Key Management Service (KMS) キーを使用することもできます。詳細については、「AWS Key Management Service の概念」を参照してください。

独自の KMS キーを使用する場合は、次のポリシーを使用して KMS キーを指定します。*kms_key_arn* を使用する KMS キー (または KMS エイリアス ARN) の ARN に変更します。または、* を指定して、任意の KMS キーを使用します。ユーザーまたはロールへのポリシーの追加に関しては、「IAM ポリシーの作成」を参照してください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionKmsDescribeAccess",
            "Effect": "Allow",
            "Action": "kms:DescribeKey",
            "Resource": "kms_key_arn"
        },
        {
            "Sid": "LookoutVisionKmsCreateGrantAccess",
            "Effect": "Allow",
            "Action": "kms:CreateGrant",
```

```
            "Resource": "kms_key_arn",
            "Condition": {
                "StringLike": {
                    "kms:ViaService": "lookoutvision.*.amazonaws.com"
                },
                "Bool": {
                    "kms:GrantIsForAWSResource": "true"
                }
            }
        }
    ]
}
```

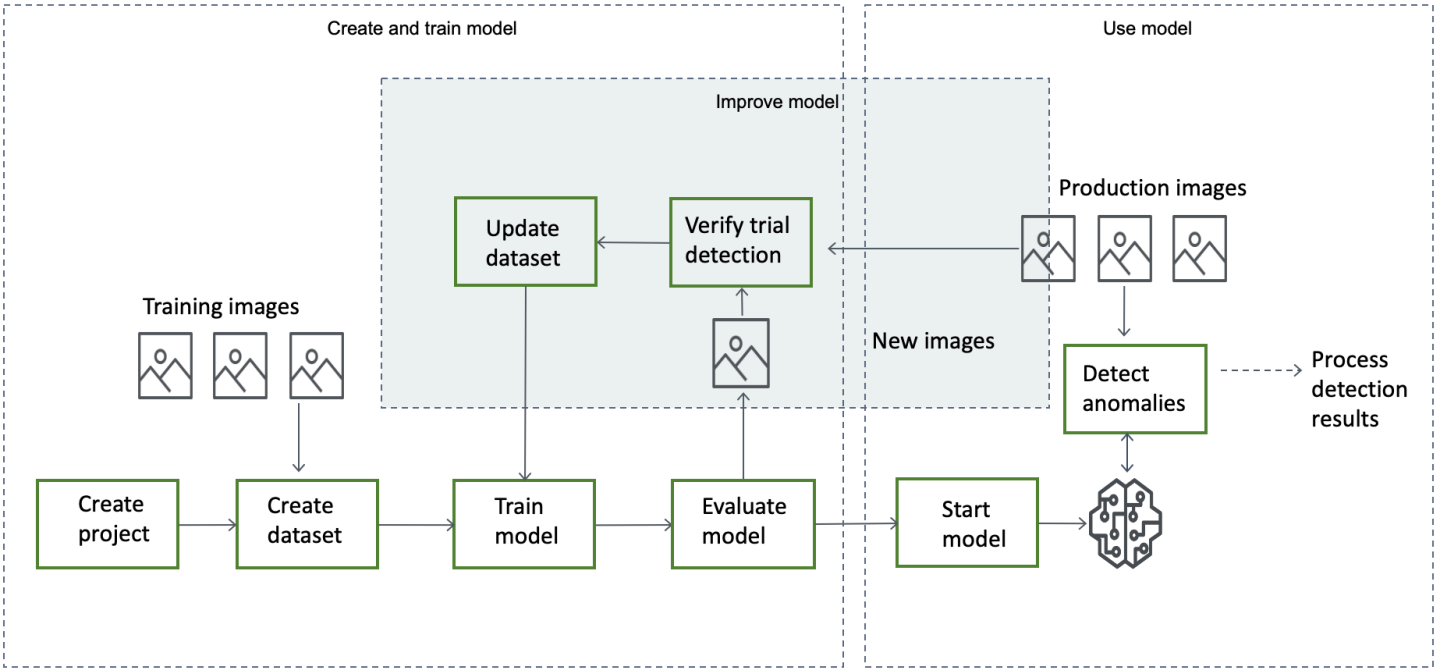
Amazon Lookout for Vision について

Amazon Lookout for Vision を使えば、以下のように、工業製品の視覚的欠陥を正確かつ大規模に見つけることができます:

- 損傷部分の検出 — 製造や組み立ての過程で、製品の表面品質、色、形状の損傷を特定します。
- 欠落部品の特定 — 物体の非適用、存在、配置の情報に基づいて欠落している部品を特定します。たとえば、プリント回路基板で欠けているコンデンサなどです。
- プロセス問題の発見 — シリコンウェーハの同じ箇所に繰り返し傷がつくなど、パターンが繰り返される欠陥を検出します。

Lookout for Vision により、画像内の異常の存在を予測するコンピュータービジョンモデルを作成できます。Amazon Lookout for Vision がモデルのトレーニングとテストに使用する画像を提供してください。Amazon Lookout for Vision は、トレーニング済みモデルの評価と改善に使用できるメトリクスを提供しています。トレーニング済みモデルを AWS クラウドでホストしたり、エッジデバイスにデプロイしたりできます。簡単な API オペレーションにより、モデルの作成する予測が返されます。

モデルを作成、評価、使用するための一般的なワークフローは次のとおりです:



トピック

モデルタイプを選択する

モデルを作成する前に、必要なモデルのタイプを決定する必要があります。画像分類と画像セグメンテーションという 2 タイプのモデルを作成できます。お客様はユースケースに基づいて作成するモデルのタイプを決定します。

画像分類モデル

画像に異常があるかどうかを確認するだけでよく、その位置を知る必要がない場合は、画像分類モデルを作成します。画像分類モデルは、画像に異常があるかどうかを予測します。予測には、予測の精度に対するモデルの信頼度が含まれます。このモデルは、画像上で見つかった異常の位置に関する情報を提供しません。

画像セグメンテーションモデル

傷の場所など、異常箇所を知る必要がある場合は、画像セグメンテーションモデルを作成します。Amazon Lookout for Vision モデルはセマンティックセグメンテーションを使用して、異常のタイプ (傷や部分的欠落など) が存在する画像でのピクセルを識別します。

> **Note**
>
> セマンティックセグメンテーションモデルはさまざまなタイプの異常を検出します。個々の異常に関するインスタンス情報は提供されません。たとえば、画像に 2 つのへこみが含まれている場合、Lookout for Vision は、単一のエンティティ内で両方のへこみに関する情報を返し、へこみの異常タイプを示します。

Amazon Lookout for Vision のセグメンテーションモデルでは、次のことが予測されます:

分類

モデルは、分析された画像の分類 (正常/異常) を返し、そこには予測でのモデルの信頼度が含まれます。分類情報はセグメンテーション情報とは別に計算されるため、それらの関係は想定しないでください。

セグメンテーション

モデルは、画像上で異常が発生している、ピクセルを示す画像マスクを返します。データセット内の異常ラベルに割り当てられた色に従って、さまざまなタイプの異常が色分けされます。異常ラベルは異常のタイプを表します。たとえば、次の画像内の青いマスクは、自動車で見つかった異常として傷の位置を示しています。

モデルはマスク内の各異常ラベルのカラーコードを返します。また、モデルは異常ラベルの画像の被覆率も返します。

Lookout for Vision セグメンテーションモデルにより、さまざまな基準を使用してモデルからの分析結果を分析できます。例:

- 異常箇所 — 異常箇所を知る必要がある場合は、セグメンテーション情報を利用して、異常をカバーするマスクを確認してください。
- 異常のタイプ — セグメンテーション情報を使用して、許容数を超える異常のタイプを画像が含んでいるかどうかを判断します。
- 対象範囲 — セグメンテーション情報を使用して、ある異常タイプが画像の許容範囲を超えているかどうかを判断します。
- 画像分類 — 異常箇所を知る必要がない場合は、分類情報を使用して画像に異常があるかどうかを判断します。

サンプルコードについては、「画像内の異常を検出する」を参照してください。

必要なモデルのタイプを決定したら、モデルを管理するためのプロジェクトとデータセットを作成します。ラベルを使用すると、画像を通常または異常に分類できます。ラベルは、マスクや異常タイプなどのセグメンテーション情報も識別します。データセット内の画像にどのようにラベルを付けるかによって、Lookout for Vision が作成するモデルのタイプが決まります。

画像セグメンテーションモデルのラベリングは、画像分類モデルのラベリングよりも複雑です。セグメンテーションモデルをトレーニングするには、トレーニング画像を正常または異常として分類する必要があります。また、異常画像ごとに異常マスクと異常タイプを定義する必要があります。分類モデルでは、トレーニング画像が正常か異常かを識別するだけで済みます。

モデルを作成する

モデルを作成するステップは、以下のように、プロジェクトの作成、データセットの作成、モデルのトレーニングとなります:

プロジェクトを作成する

作成したデータセットとモデルを管理するためのプロジェクトを作成します。プロジェクトは、単一タイプの機械部品の異常を検出するなど、単一のユースケースに使用する必要があります。

プロジェクトの概要はダッシュボードで確認できます。詳細については、「Amazon Lookout for Vision ダッシュボードを使用する」を参照してください。

詳細情報:「プロジェクトを作成する」を参照してください。

データセットを作成する

Amazon Lookout for Vision でモデルをトレーニングするには、ユースケースに合った通常のオブジェクトと異常なオブジェクトの画像が必要です。これらの画像はデータセットで提供します。

データセットとは、複数の画像とそれらの画像を説明するラベルのセットのことです。画像では、異常が発生する単一タイプのオブジェクトが表されている必要があります。詳細については、「データセットの画像の準備」を参照してください。

Amazon Lookout for Vision では、シングルデータセットを使用するプロジェクトを作成したり、個別のトレーニングデータセットとテストデータセットを持つプロジェクトを作成できます。トレーニング、テスト、およびパフォーマンスチューニングをより細かく制御する必要がある場合を除き、単一のデータセットプロジェクトを使用することをお勧めします。

データセットは、画像をインポートして作成します。画像のインポート方法によっては、画像にラベルが付けられる場合もあります。ラベルを付けない場合は、コンソールを使用して画像にラベルを付けます。

画像のインポート

Lookout for Vision コンソールでデータセットを作成した場合、次のいずれかの方法で画像をインポートできます:

- ローカルコンピュータから画像をインポートします。画像はラベル付けされません。
- S3 バケットから画像をインポートします。Amazon Lookout for Vision では、画像を含むフォルダー名を使用して画像を分類できます。`normal` を正常な画像に使用します。`anomaly` を異常な画像に使用します。セグメンテーションラベルを自動的に割り当てることはできません。
- Amazon SageMaker Ground Truth マニフェストファイルをインポートします。マニフェストファイル内の画像はラベル付けされます。独自のマニフェストファイルを作成してインポートできます。画像が多い場合は、SageMaker Ground Truth ラベリングサービスの使用を検討してください。次に、Amazon SageMaker Ground Truth ジョブから出力マニフェストファイルをインポートします。

画像のラベリング

ラベルは、データセット内の画像について記述します。ラベルは、画像が正常か異常かを指定します (分類)。ラベルは画像上の異常箇所も記述します (セグメンテーション)。

画像にラベルが付いていない場合は、コンソールを使用してラベルを付けることができます。

データセット内の画像に割り当てるラベルによって、Lookout for Vision が作成するモデルのタイプが決まります:

画像分類

画像分類モデルを作成するには、Lookout for Vision コンソールを使用して、データセット内の画像を正常または異常に分類します。

`CreateDataset` オペレーションを使用して、分類情報を含むマニフェストファイルからデータセットを作成することもできます。

画像セグメンテーション

画像セグメンテーションモデルを作成するには、Lookout for Vision コンソールを使用して、データセット内の画像を正常または異常として分類します。また、画像上の異常領域 (存在する場合) にはピクセルマスクを指定し、個々の異常マスクには異常ラベルを指定します。

また、この CreateDataset オペレーションを使用して、セグメンテーションと分類の情報を含むマニフェストファイルからデータセットを作成することもできます。

プロジェクトに個別のトレーニングデータセットとテストデータセットがある場合、Lookout for Vision はトレーニングデータセットを使用してモデルタイプの学習と決定行います。テストデータセットの画像には同じ方法でラベルを付ける必要があります。

詳細情報: 「データセットの作成」を参照してください。

モデルをトレーニングする

トレーニングは、モデルを作成し、画像内の異常の存在を予測するようにトレーニングします。モデルをトレーニングするたびに、モデルの新しいバージョンが作成されます。

トレーニングのスタート時に、Amazon Lookout for Vision は、モデルのトレーニングに最適なアルゴリズムを選択します。モデルはトレーニングされ、テストされます。Amazon Lookout for Vision コンソールの開始 で、単一データセットプロジェクトのトレーニングを行う場合、データセットは内部で分割され、トレーニングデータセットとテストデータセットが作成されます。個別のトレーニングデータセットとテストデータセットを持つプロジェクトを作成することもできます。この構成では、Amazon Lookout for Vision はトレーニングデータセットを使用してモデルをトレーニングし、テストデータセットでモデルをテストします。

> **Important**
>
> 課金はモデルのトレーニングに成功するまでの時間に対して行われます。

詳細情報:モデルのトレーニング。

モデルの評価

テスト中に作成されたパフォーマンスメトリックを使用して、モデルのパフォーマンスを評価します。

パフォーマンスメトリクスにより、トレーニングしたモデルのパフォーマンスをより適切に理解し、実運用で使用する準備ができたかどうかを判断できます。

詳細: 「モデルの改善」を参照してください。

パフォーマンスメトリクスの結果、改善が必要な場合は、新しい画像で検出タスクを試行することにより、トレーニングデータを追加することができます。タスクが完了したら、結果を確認し、検証済みの画像をトレーニングデータセットに追加できます。または、新しいトレーニング画像をデータセットに直接追加することもできます。次に、モデルを再トレーニングし、パフォーマンスメトリクスを再チェックします。

詳細: 「トライアル検出タスクでモデルを検証する」を参照してください。

モデルを使用する

AWS クラウドでモデルを使用する前に、StartMode オペレーションでモデルを開始します。コンソールからモデルの `StartModel` CLI コマンドを取得できます。

詳細情報:モデルの開始。

トレーニング済みの Amazon Lookout for Vision モデルは、入力された画像に正常なコンテンツまたは異常なコンテンツが含まれているかどうかを予測します。モデルがセグメンテーションモデルの場合、予測には異常が見つかったピクセルをマークする異常マスクが含まれます。

モデルで予測を行うには、DetectAnomalies オペレーションを実行し、ローカルコンピュータから入力画像を渡します。`DetectAnomalies` を呼び出す CLI コマンドは、コンソールから取得することができます。

詳細情報:画像内の異常を検出する。

> **Important**
>
> モデルの稼働時間に応じて課金されます。

モデルを使用しなくなった場合は、StopModel オペレーションを使用してモデルを停止します。CLI コマンドは、コンソールから取得できます。

詳細情報:モデルを停止する。

モデルをエッジデバイスで使用する

Lookout for Vision モデルは AWS IoT Greengrass Version 2 コアデバイスで使用できます。

詳細: 「エッジデバイスでの Amazon Lookout for Vision モデルの使用」を参照してください。

ダッシュボードを使用する

ダッシュボードを使用して、すべてのプロジェクトの概要と個々のプロジェクトの概要情報を取得できます。

詳細: 「ダッシュボードを使用する」を参照してください。

Amazon Lookout for Vision コンソールの開始

開始の手引きを始める前に、「Amazon Lookout for Vision について」を読むことをお勧めします。

開始の手引きでは、サンプルの画像セグメンテーションモデルの使用と作成の方法を説明します。サンプルの画像分類モデルを作成する場合は、画像分類データセット を参照してください。

サンプルモデルをすぐに試してみたい場合は、サンプルのトレーニング画像とマスク画像を提供します。また、画像セグメンテーションマニフェストファイルを作成する Python スクリプトも提供します。マニフェストファイルを使用してプロジェクトのデータセットを作成すれば、データセット内の画像にラベルを付ける必要はありません。独自の画像を使用してモデルを作成する場合は、データセット内の画像にラベルを付ける必要があります。詳細については、「データセットの作成」を参照してください。

私たちが提供する画像は、通常のクッキーと異常クッキーのものです。異常クッキーは、クッキーの形状全体にひびが入っています。画像を使ってトレーニングしたモデルは、次の例に示すように、分類 (正常または異常) を予測し、異常な Cookie の欠陥の領域 (マスク) を検出します。

トピック

ステップ1: マニフェストファイルとアップロード画像を作成する

この手続きでは、Amazon Lookout for Vision ドキュメンテーションリポジトリをコンピュータに複製します。次に、Python (バージョン 3.7 以降) スクリプトを使用してマニフェストファイルを作成し、指定した Amazon S3 の場所にトレーニング画像とマスク画像をアップロードします。マニフェストファイルを使用してモデルを作成します。後で、ローカルリポジトリのテスト画像を使用してモデルを試します。

マニフェストファイルを作成して画像をアップロードするには

1. 「Amazon Lookout for Vision をセットアップする」の手順に従って Amazon Lookout for Vision をセットアップします。必ず Python 用 AWS SDK をインストールしてください。
2. Lookout for Vision を使用したい AWS リージョンで、S3 バケットを作成します。
3. Amazon S3 バケットで、`getting-started` という名前のフォルダを作成します。
4. Amazon S3 URI and Amazon リソースネーム (ARN) をメモします。これらを使用して権限をセットアップし、スクリプトを実行します。
5. スクリプトを呼び出すユーザーに、`s3:PutObject` オペレーションを呼び出す権限があることを確かめてください。以下のポリシーを使用できます。権限を割り当てるには 権限の割り当て を参照してください。

```
{
 "Version": "2012-10-17",
 "Statement": [{
  "Sid": "Statement1",
  "Effect": "Allow",
  "Action": [
   "s3:PutObject"
  ],
  "Resource": [
   "arn:aws:s3::: ARN for S3 folder in step 4/*"
  ]
 }]
}
```

6. `lookoutvision-access` という名前のローカルプロファイルがあり、そのプロファイルユーザーが前のステップの権限を持っていることを確かめてください。詳細については、「ローカルコンピュータでのプロファイルの使用」を参照してください。
7. zip ファイル getting-started.zip をダウンロードします。zip ファイルには、開始用データセットとセットアップスクリプトが含まれています。

8. ファイル getting-started.zip を解凍します。
9. コマンドプロンプトで、以下を行います:

   a. getting-started フォルダに移動します。

   b. 次のコマンドを実行してマニフェストファイルを作成し、ステップ 4 でメモした Amazon S3 パスにトレーニング画像と画像マスクをアップロードします。

   ```
   python getting_started.py S3-URI-from-step-4
   ```

   c. スクリプトが完了したら、スクリプトで Create dataset using manifest file: の後に表示される train.manifest ファイルへのパスをメモします。パスは s3://*path to getting started folder*/manifests/train.manifest に似たものとなるはずです。

ステップ 2: ロールを作成する

この手続きでは、前に Amazon S3 バケットにアップロードした画像とマニフェストファイルを使ってプロジェクトとデータセットを作成します。次に、モデルを作成し、モデルトレーニングの評価結果を表示します。

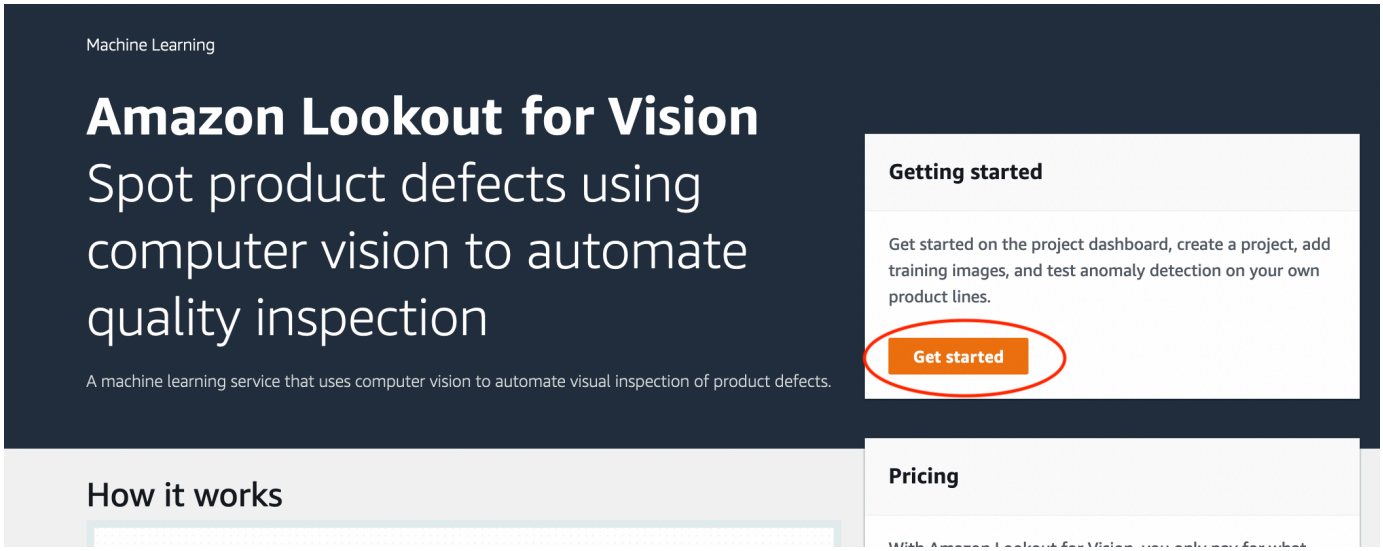
データセットは開始用マニフェストファイルから作成するため、データセットの画像にラベルを付ける必要はありません。独自の画像を使用してデータセットを作成する場合、画像にラベルを付ける必要があります。詳細については、「画像のラベリング」を参照してください。

> **Important**
>
> 課金はモデルのトレーニングが完了するごとに行われます。

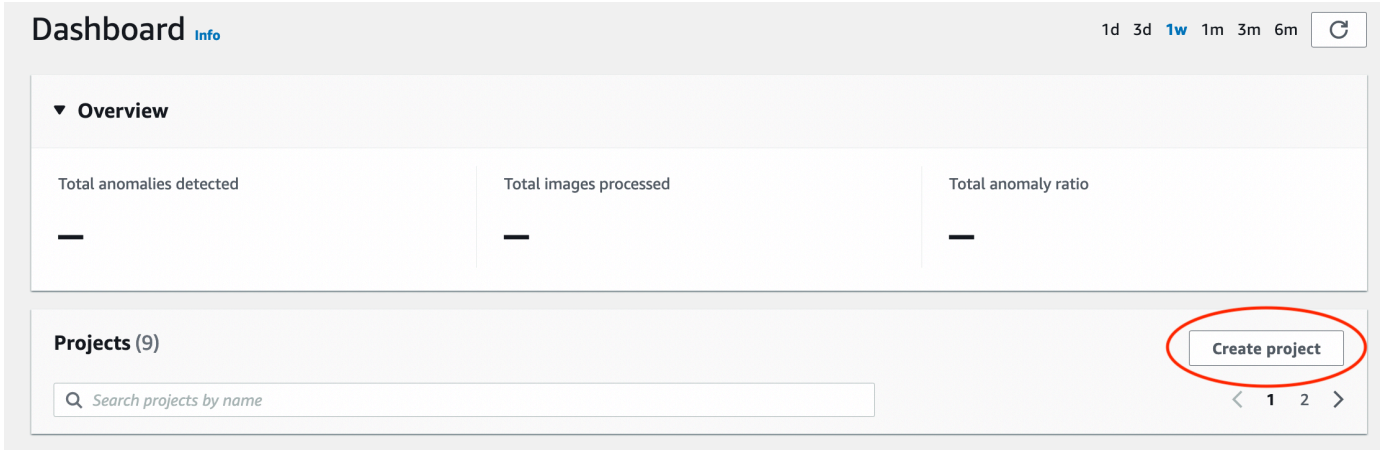
モデルを作成するには

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. ステップ1: マニフェストファイルとアップロード画像を作成する で Amazon S3 バケットを作成したリージョンと同じ AWS リージョンに存在することを確かめます。リージョンを変更するには、ナビゲーションバーで、現在表示されているリージョン名を選択します。切り替え先となるリージョンを選択します。

3. [開始する] を選択します。



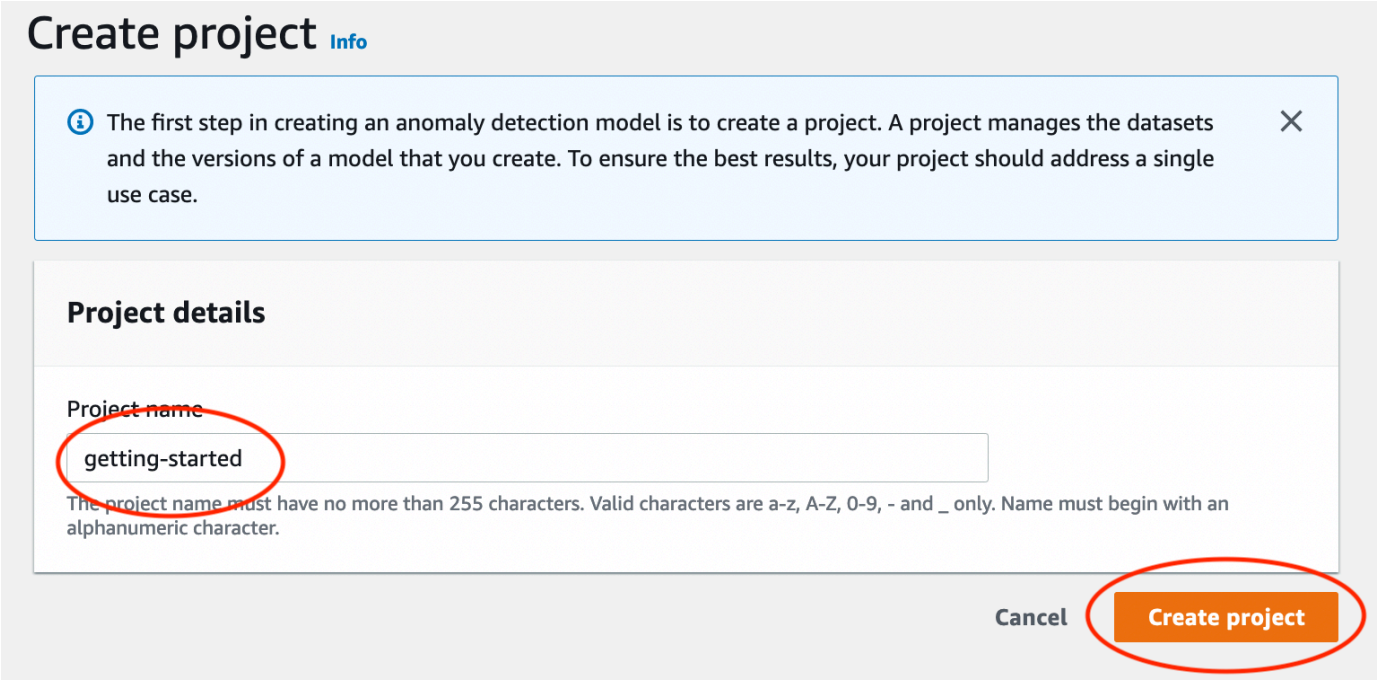
4. [プロジェクト] セクションで、[プロジェクトを作成] を選択します。



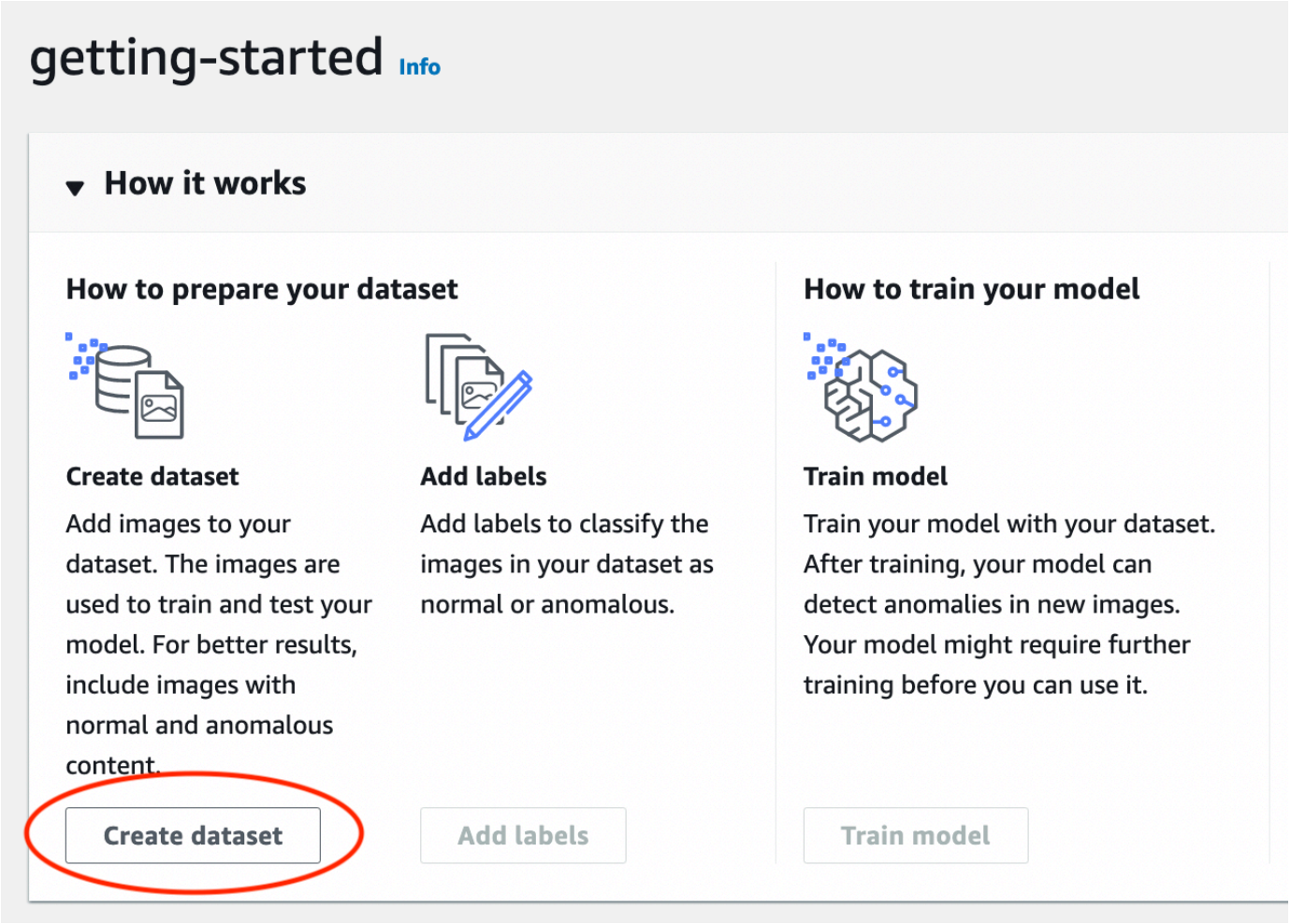
5. 「プロジェクトを作成」ページで、以下を行います:

   a. [プロジェクト名] に `getting-started` を入力します。

   b. [プロジェクトを作成] を選択します。



6. [動作方法] セクションで、[データセットを作成] を選択します。



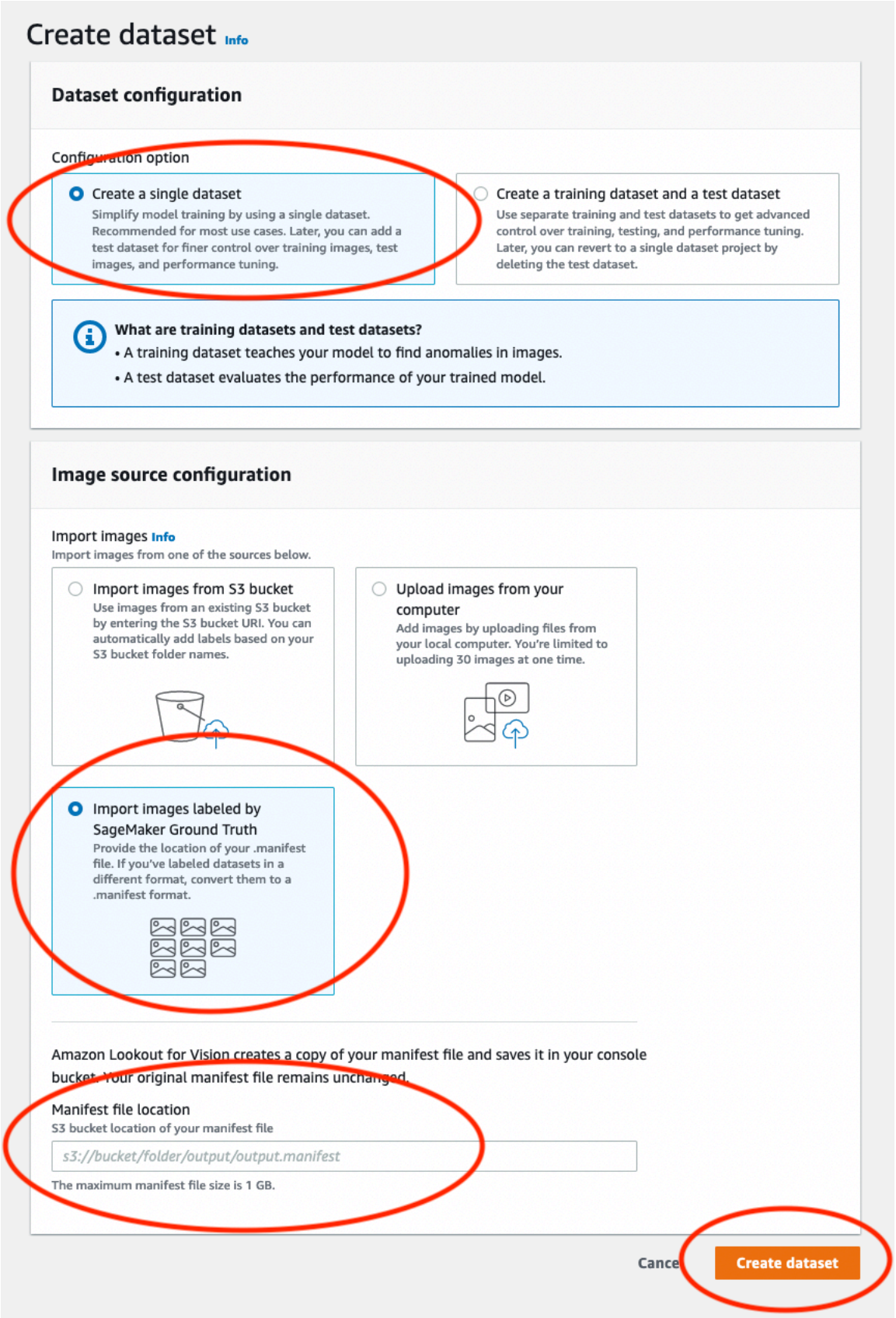
7. 「データセットを作成する」ページで、次の操作を行います。

   a. [シングルデータセットを作成] を選択します。

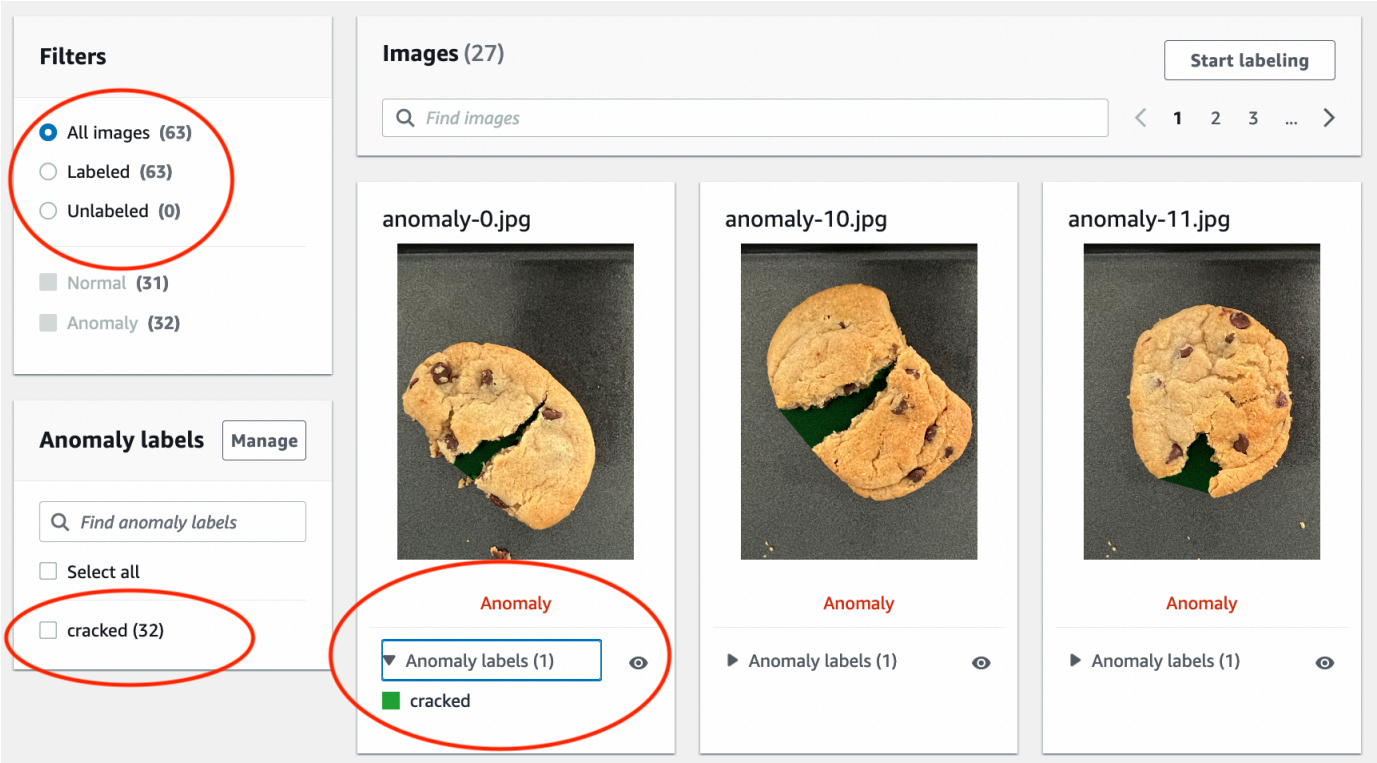
   b. [画像ソース構成] セクションで、[SageMaker Ground Truth によってラベル付けされた画像をインポート] を選択します。

   c. [.manifest ファイルの場所] には、ステップ1: マニフェストファイルとアップロード画像を作成する のステップ 6.c. でメモしたマニフェストファイルの Amazon S3 の場所を入力します。Amazon S3 の場所は `s3://path to getting started folder/manifests/train.manifest` のようになっているはずです

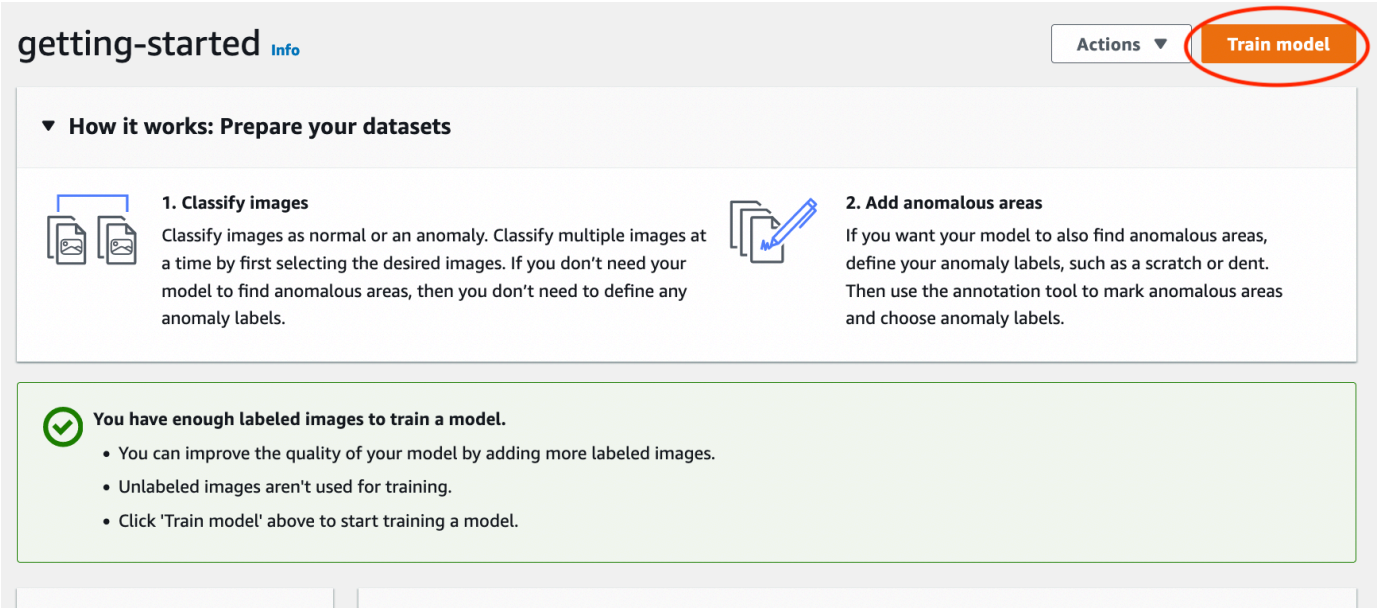
   d. [データセットを作成] を選択します。


Create dataset Info
Dataset configuration
Configuration option
Create a single dataset
Simplify model training by using a single dataset. Recommended for most use cases. Later, you can add a test dataset for finer control over training images, test images, and performance tuning.
Create a training dataset and a test dataset
Use separate training and test datasets to get advanced control over training, testing, and performance tuning. Later, you can revert to a single dataset project by deleting the test dataset.
What are training datasets and test datasets?
• A training dataset teaches your model to find anomalies in images.
• A test dataset evaluates the performance of your trained model.
Image source configuration
Import images Info
Import images from one of the sources below.
Import images from S3 bucket
Use images from an existing S3 bucket by entering the S3 bucket URI. You can automatically add labels based on your S3 bucket folder names.
Upload images from your computer
Add images by uploading files from your local computer. You're limited to uploading 30 images at one time.
Import images labeled by SageMaker Ground Truth
Provide the location of your .manifest file. If you've labeled datasets in a different format, convert them to a .manifest format.
Amazon Lookout for Vision creates a copy of your manifest file and saves it in your console bucket. Your original manifest file remains unchanged.
Manifest file location
S3 bucket location of your manifest file
s3://bucket/folder/output/output.manifest
The maximum manifest file size is 1 GB.
Cancel
Create dataset

8. プロジェクト詳細ページの [画像] セクションに、データセットの画像が表示されます。各データセット画像の分類と画像セグメンテーション情報 (マスクラベルと異常ラベル) を表示できます。また、画像を検索したり、ラベリングステータス (ラベル付き/ラベルなし) で画像をフィルターしたり、割り当てられた異常ラベルで画像をフィルターしたりすることもできます。

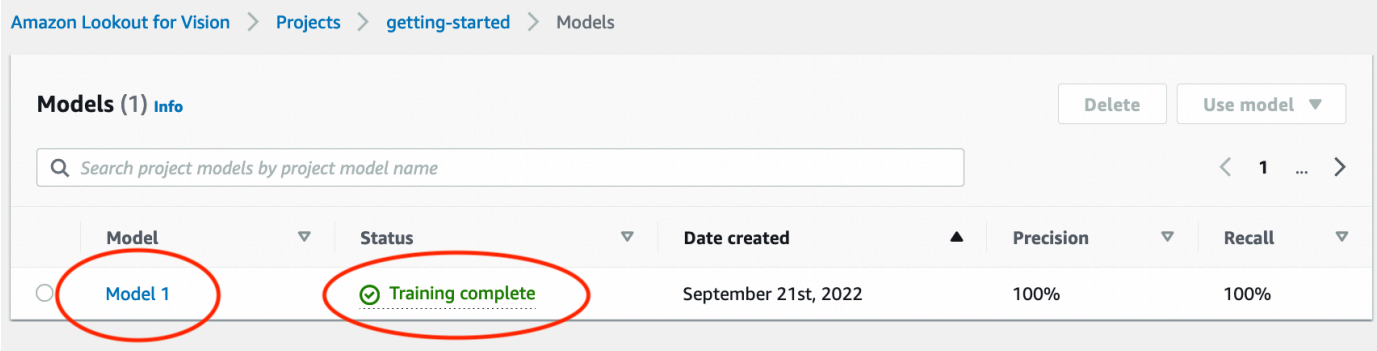


9. プロジェクト詳細ページで、[モデルをトレーニング] を選択します。



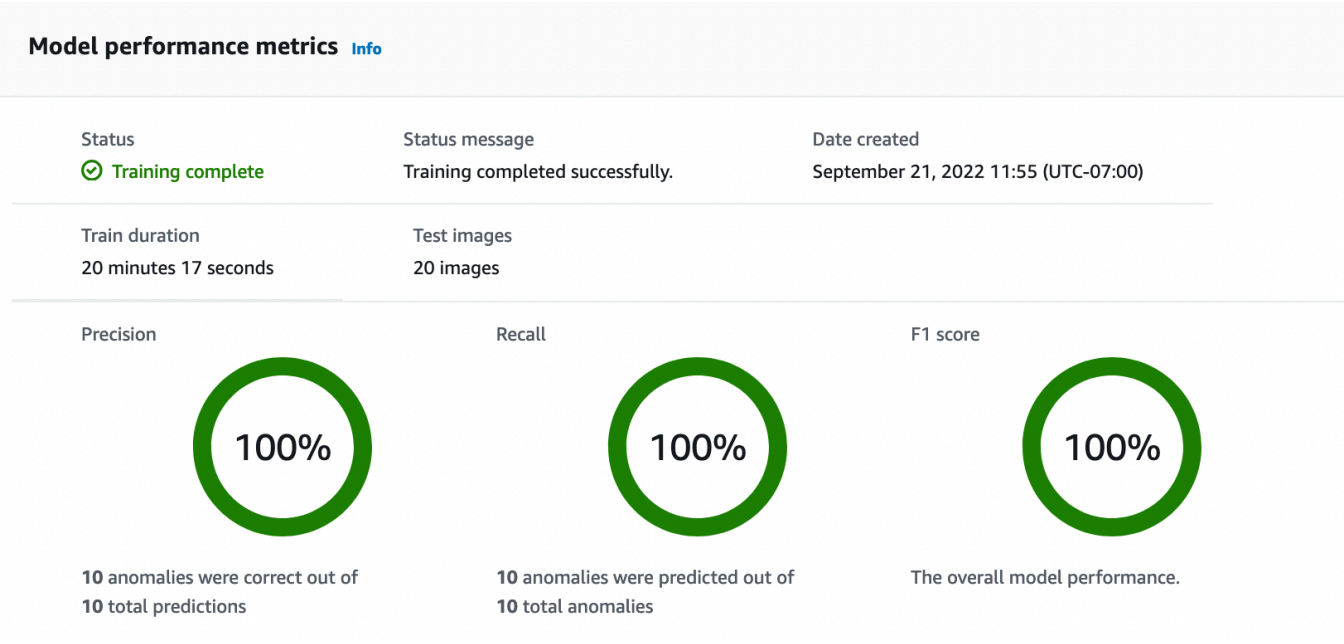
10. 「モデルをトレーニングする」詳細ページで、[モデルをトレーニング] を選択します。

11. 「モデルをトレーニングしますか?」ダイアログボックスで、[モデルをトレーニング] を選択します。
12. プロジェクトの「モデル」ページでは、トレーニングが開始されたことを確認できます。モデルの現在のステータスは、「ステータス」列に表示されます。モデルのトレーニングには 30 分ほどかかります。ステータスが「トレーニング完了」に変わると、トレーニングは正常に終了します。
13. トレーニングが終了したら、「モデル」ページで [モデル 1] を選択します。

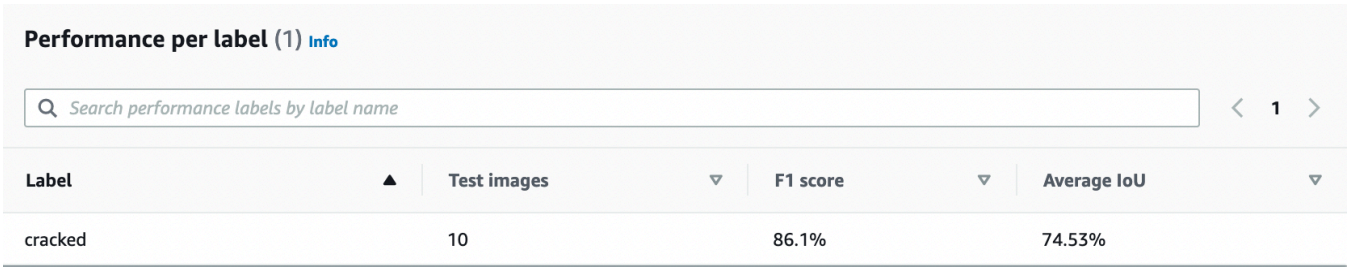


14. モデルの詳細ページの「パフォーマンスメトリクス」タブに評価結果が表示されます。次のメトリクスを参照できます:

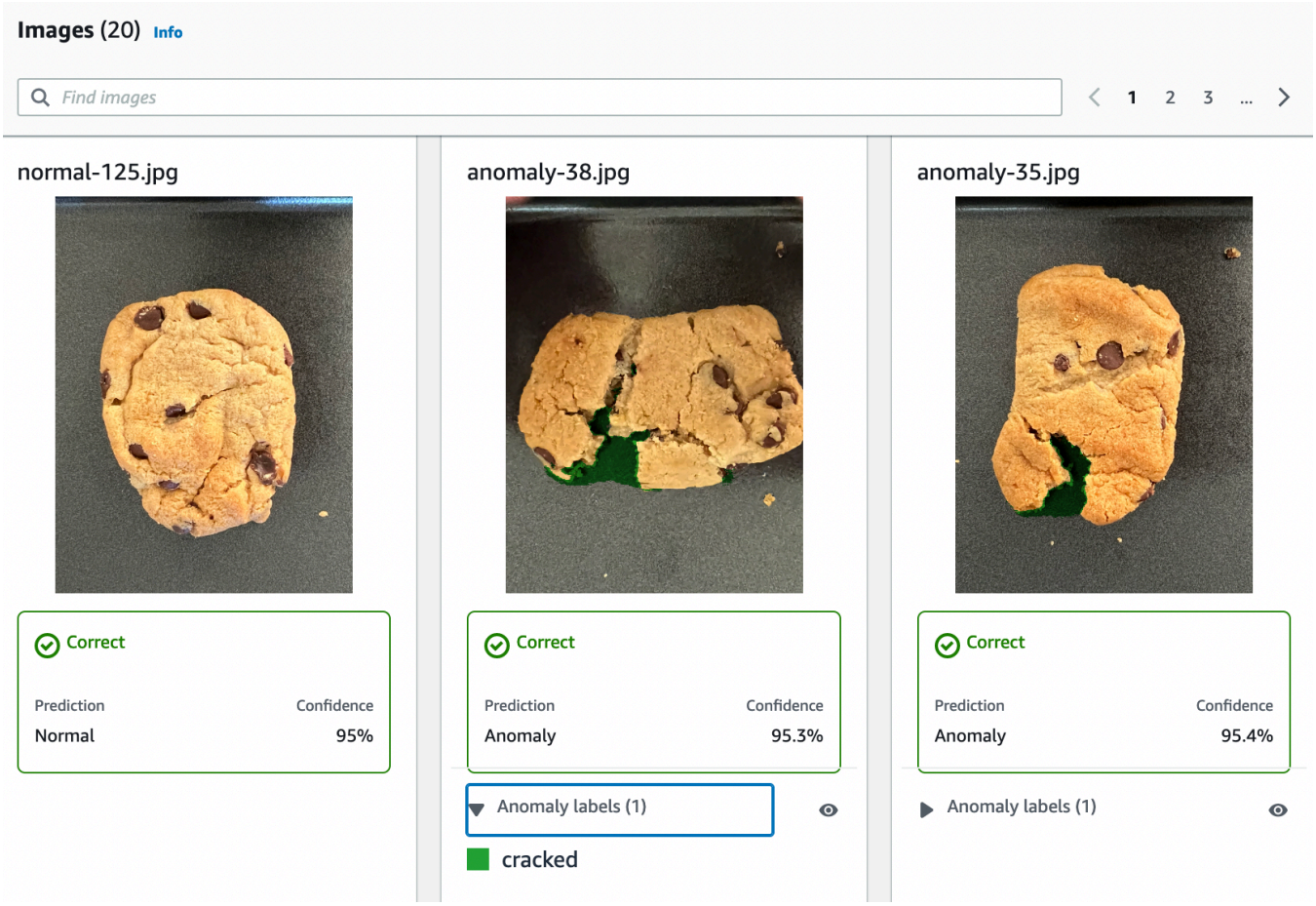
    - モデルによって行われた分類予測の全体的モデルパフォーマンス指標 (精度、再現率、F1 スコア)。



    - テスト画像に含まれる異常ラベルのパフォーマンス指標 (平均 IoU、F1 スコア)



- テスト画像に対する予測 (分類、セグメンテーションマスク、異常ラベル)



モデルトレーニングは非決定的であるため、評価結果がこのページに表示されている結果と異なる場合があります。詳細については、「Amazon Lookout for Vision モデルの改善」を参照してください。

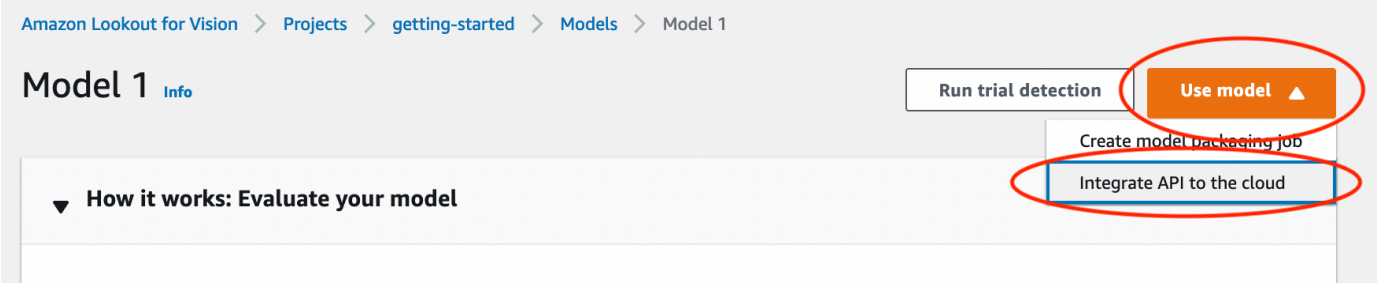
ステップ 3: モデルを開始する

このステップでは、モデルのホスティングを開始して、画像を分析できる状態にします。詳細については、「トレーニング済みの Amazon Lookout for Vision モデルの実行」を参照してください。

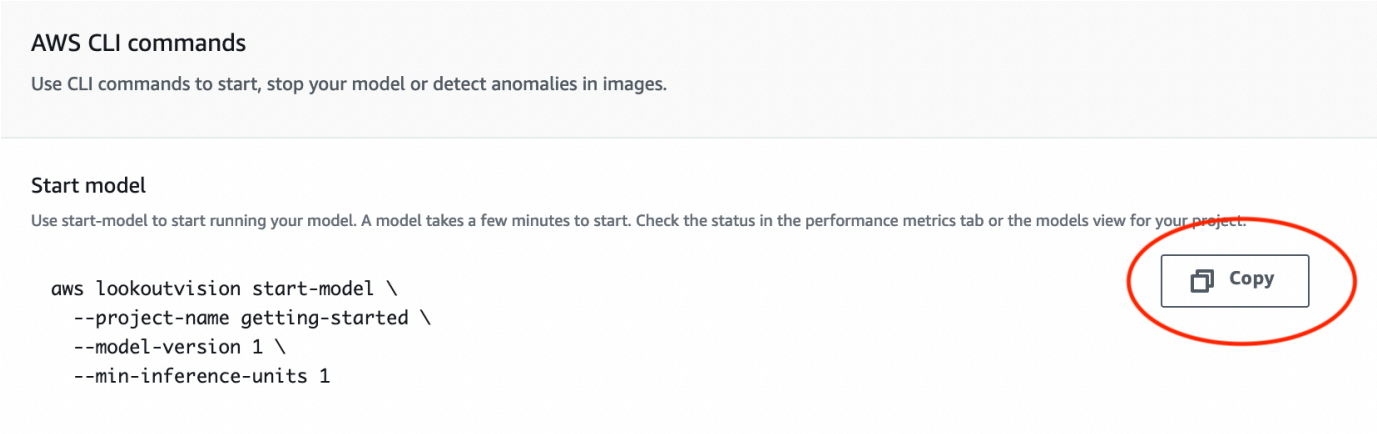
> **Note**
>
> モデルの稼働時間に応じて課金されます。ステップ 5: モデルを停止する でモデルを停止します。

モデルを開始します。

1. モデルの詳細ページで [モデルを使用] を選択し、[API をクラウドに統合] を選択します。



2. [AWS CLI コマンド] セクションで、`start-model` AWS CLI コマンドをコピーします。



3. AWS CLI が Amazon Lookout for Vision コンソールを使用しているのと同じ AWS リージョンで実行するように構成されていることを確かめてください。AWS CLI が使用する AWS リージョンを変更するには、AWS SDK のインストール を参照してください。
4. コマンドプロンプトで、`start-model` コマンドの入力によりモデルを開始します。`lookoutvision` プロファイルを使用して認証情報を取得する場合は、`--profile lookoutvision-access` パラメーターを追加します。例:

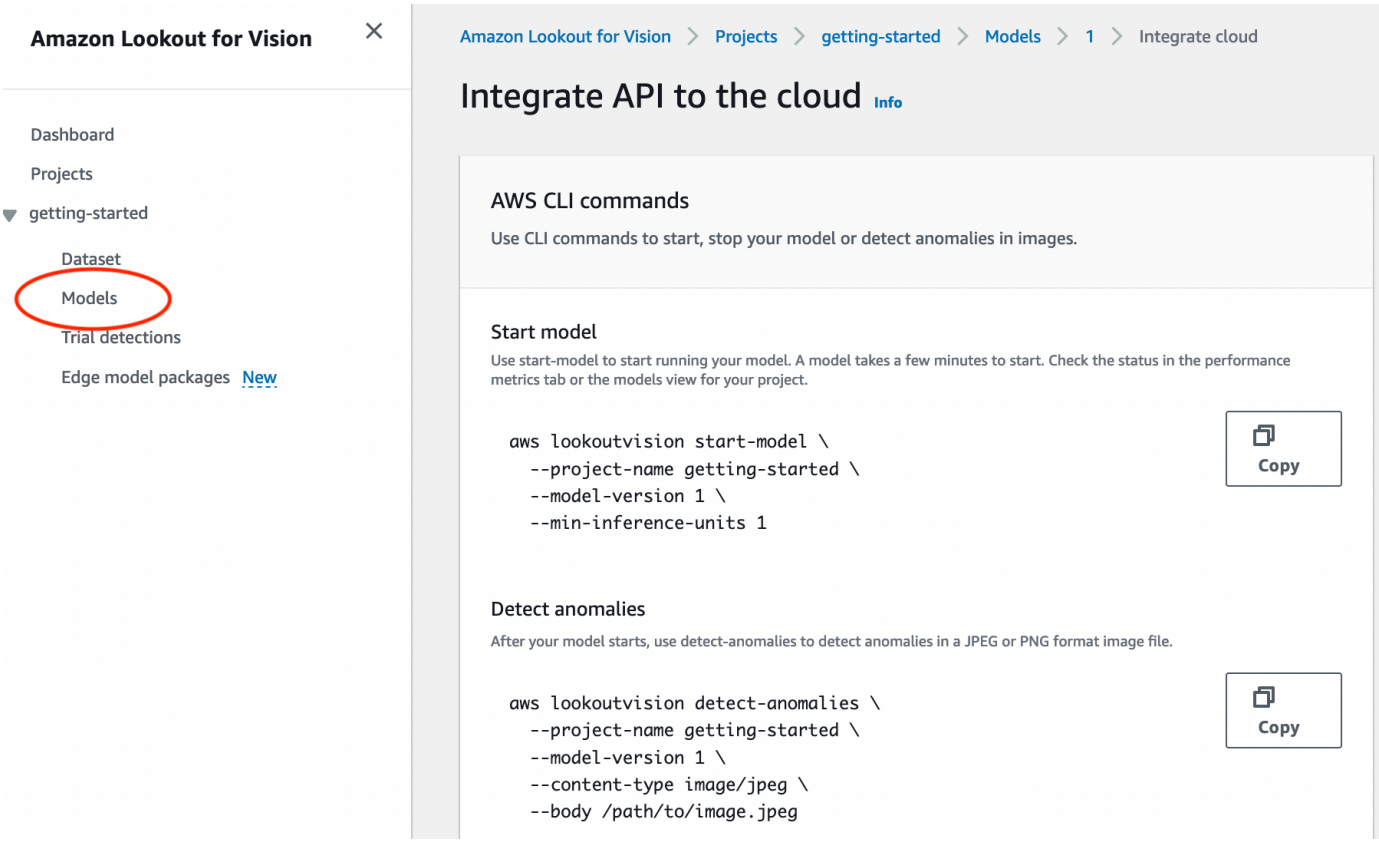
```
aws lookoutvision start-model \
  --project-name getting-started \
  --model-version 1 \
  --min-inference-units 1 \
```

```
  --profile lookoutvision-access
```

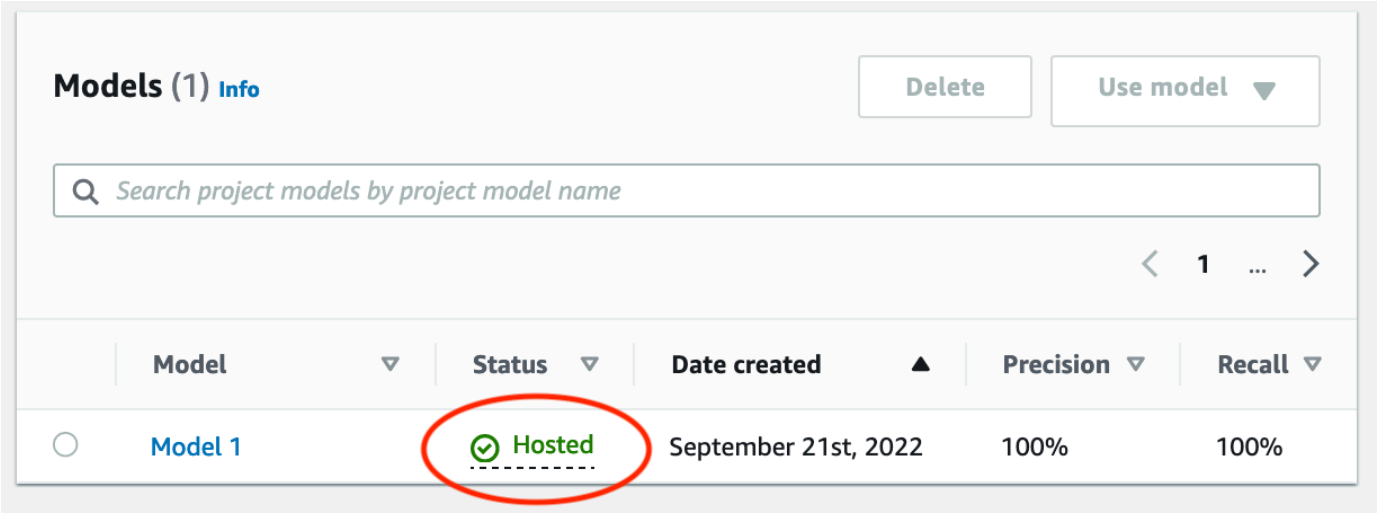
クラスターが作成されると、次の出力が表示されます:

```
{
    "Status": "STARTING_HOSTING"
}
```

5. コンソールに戻り、ナビゲーションペインの [モデル] を選択します。



6. [ステータス] 列でモデル (Model 1) のステータスが「ホスト済み」と表示されるまで待ってください。以前にプロジェクトでモデルをトレーニングしたことがある場合は、最新のモデルバージョンが完了するまでお待ちください。

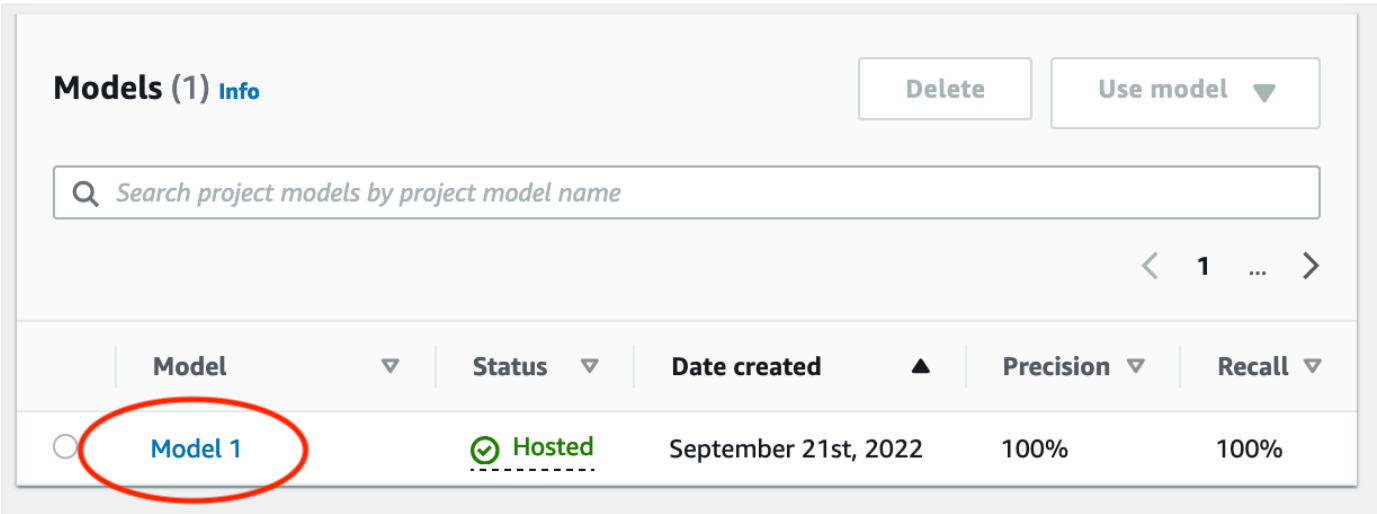


ステップ 4: 画像を分析する

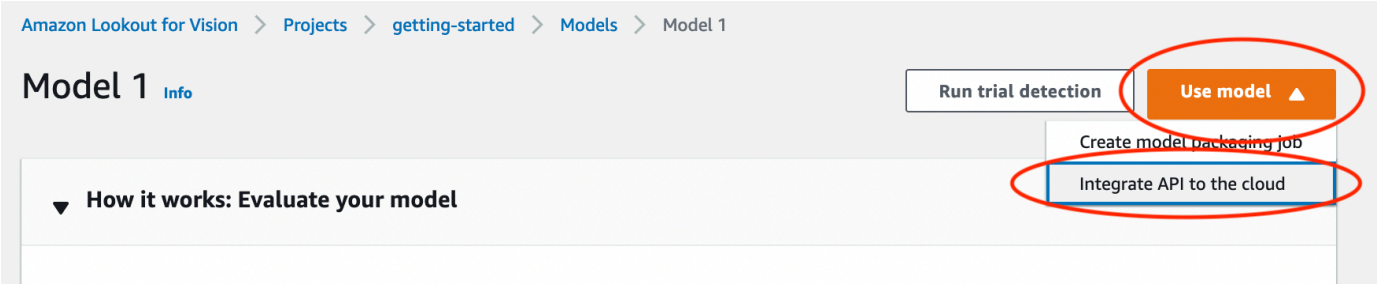
このステップでは、4 つのモデルで画像を分析します。お使いの PC 上の Lookout for Vision ドキュメンテーションリポジトリ内の開始用 `test-images` フォルダーで使えるサンプル画像を提供しています。詳細については、「画像内の異常を検出する」を参照してください。

クエリを分析するには

1. 「モデル」ページで、[モデル 1] を選択します。



2. モデルの詳細ページで [モデルを使用] を選択してから [API をクラウドに統合] を選択します。



3. [AWS CLI コマンド] セクションで、detect-anomalies AWS CLI コマンドをコピーします。

4. コマンドプロンプトで、前のステップの detect-anomalies コマンドを入力して異常画像を分析します。--body パラメータには、PC の開始用 test-images フォルダーにある異常画像を指定します。lookoutvision プロファイルを使用して認証情報を取得する場合は、--profile lookoutvision-access パラメータを追加します。例:

```
aws lookoutvision detect-anomalies \
  --project-name getting-started \
  --model-version 1 \
  --content-type image/jpeg \
  --body /path/to/test-images/test-anomaly-1.jpg \
  --profile lookoutvision-access
```

出力は次の例に類似したものになります:

```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": true,
        "Confidence": 0.983975887298584,
        "Anomalies": [
            {
                "Name": "background",
```

```
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.9818974137306213,
                    "Color": "#FFFFFF"
                }
            },
            {
                "Name": "cracked",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.018102575093507767,
                    "Color": "#23A436"
                }
            }
        ],
        "AnomalyMask": "iVBORw0KGgoAAAANSUhEUgAAAkAAAAMACA......"
    }
}
```

5. 出力について、以下の点に注意してください:

   - IsAnomalous は予測分類のブール値です。画像が異常な場合 true で、そうでない場合 false となります。
   - Confidence は、予測での Amazon Lookout for Vision の信頼度を表す浮動小数点値であり、0 が最低、1 が最高となります。
   - Anomalies は、画像内で見つかった異常のリストです。 Name は異常ラベルです。PixelAnomaly は異常の領域の合計パーセンテージ (TotalPercentageArea) と異常ラベルの色 (Color) を含んでいます。リストには、画像で見つかった異常以外の領域に及ぶ「バックグラウンド」異常も含まれています。
   - AnomalyMask は、分析された画像上の異常の位置を示すマスク画像です。

   次の例のように、レスポンス内の情報を使って、分析された画像と異常マスクを融合させたものを表示できます。サンプルコードについては、「分類とセグメンテーションの情報の表示」を参照してください。


Classification:
Prediction: Anomalous
Confidence: 99.9%
Segmentation:
Anomaly: cracked. Area: 6.2%

6. コマンドプロンプトで、開始用 test-images フォルダーにある正常の画像を分析します。lookoutvision プロファイルを使用して認証情報を取得する場合は、--profile lookoutvision-access パラメータを追加します。例:

```
aws lookoutvision detect-anomalies \
  --project-name getting-started \
  --model-version 1 \
  --content-type image/jpeg \
  --body /path/to/test-images/test-normal-1.jpg \
  --profile lookoutvision-access
```

出力は次の例に類似したものになります:

```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": false,
        "Confidence": 0.9916400909423828,
        "Anomalies": [
            {
                "Name": "background",
                "PixelAnomaly": {
                    "TotalPercentageArea": 1.0,
                    "Color": "#FFFFFF"
                }
            }
        ],
        "AnomalyMask": "iVBORw0KGgoAAAANSUhEUgAAAkAAAA....."
    }
}
```

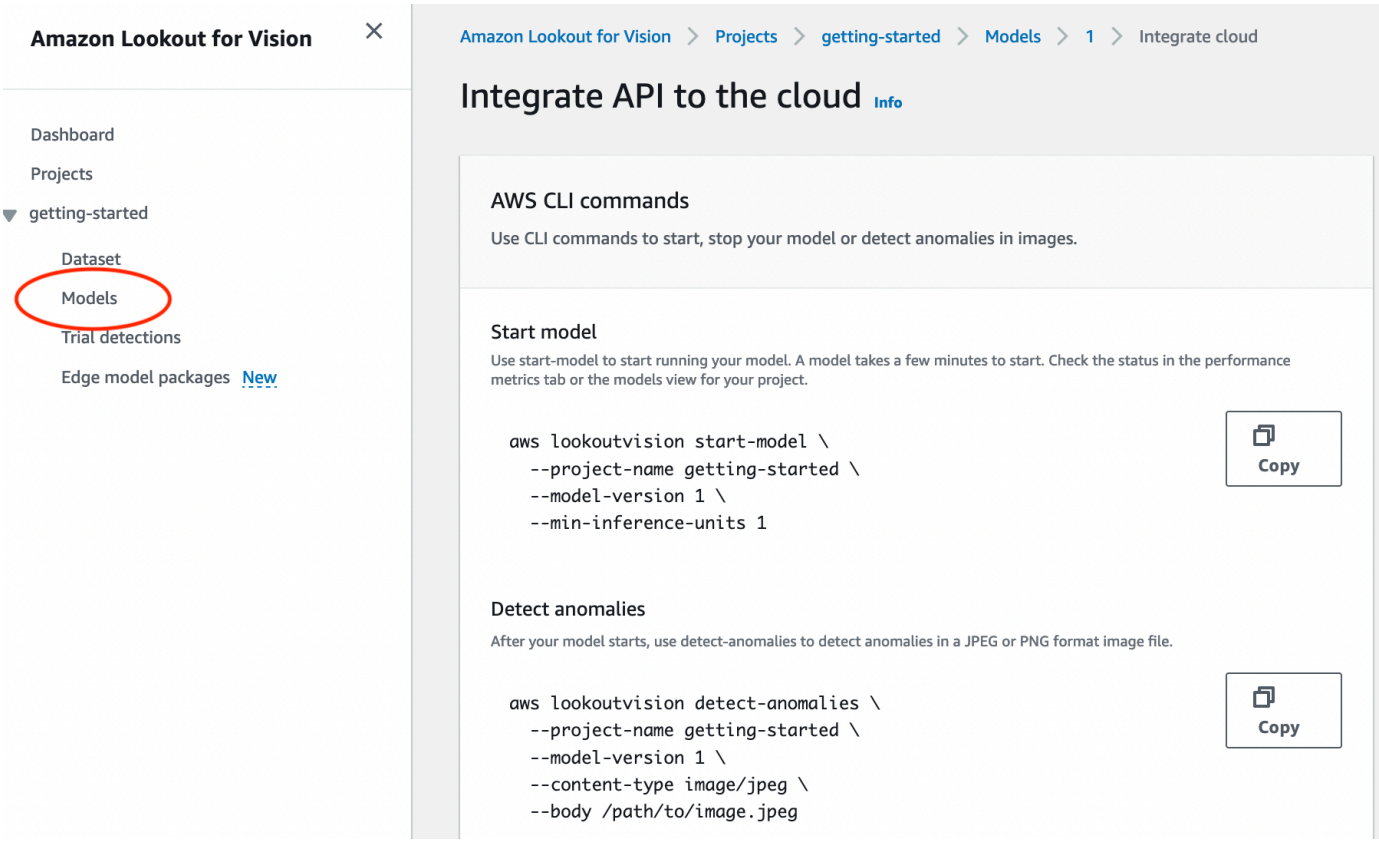
7. 出力では、IsAnomalous に対する false 値によって画像を異常なしと分類することに注意してください。Confidence を分類の信頼度を判断するのに役立ててください。また、Anomalies 配列には background 異常ラベルしか付いていません。

ステップ 5: モデルを停止する

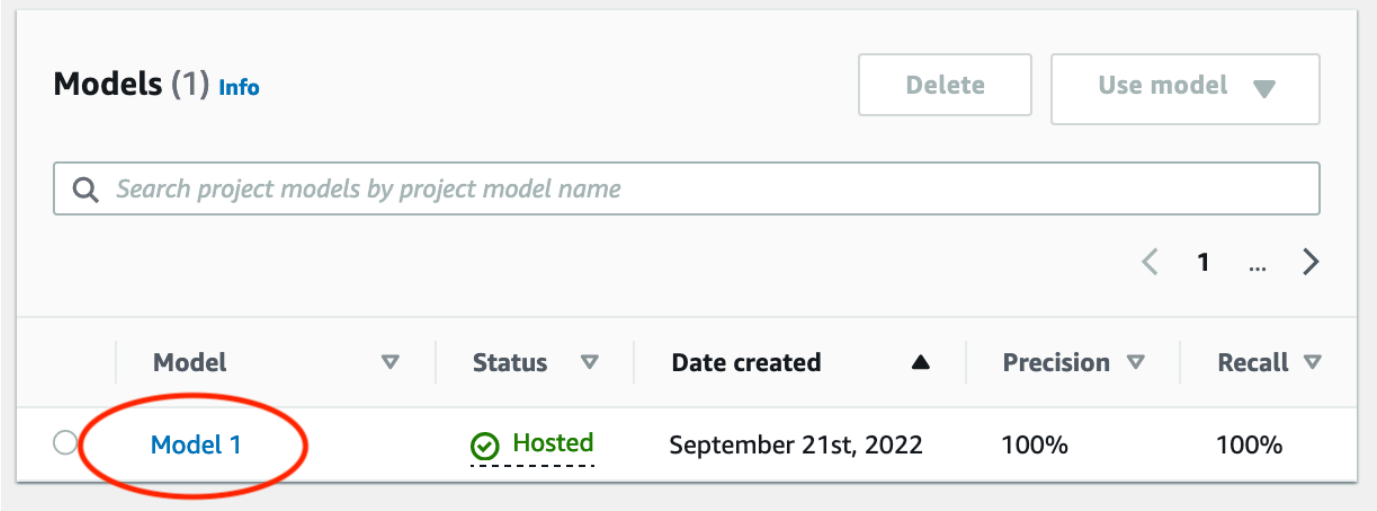
このステップでは、モデルのホスティングを作成します。モデルの実行時間に応じて課金されます。モデルを使用していない場合は、停止できます。モデルは次に必要な時に再開できます。詳細については、「Amazon Lookout for Vision モデルの開始」を参照してください。

モデルを停止するには

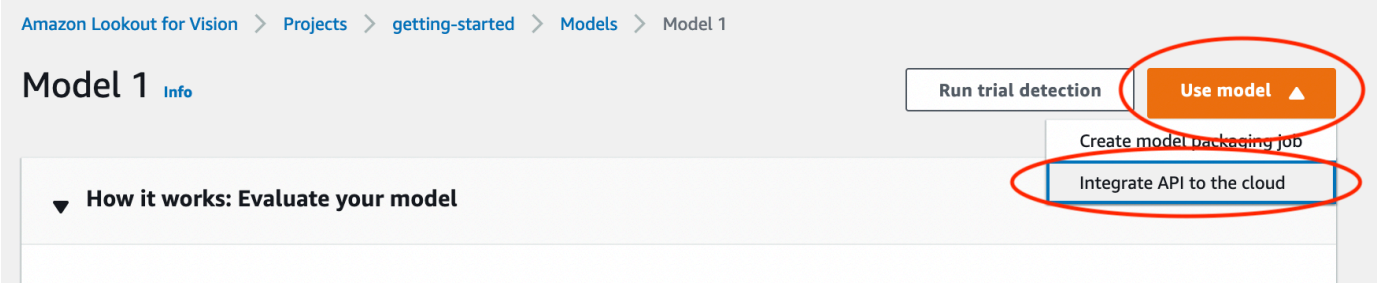
1. ナビゲーションペインで、[モデル] を選択します。



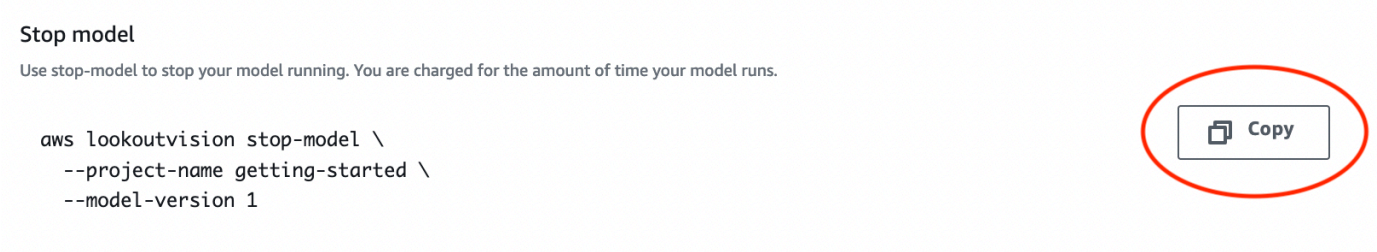
2. 「モデル」ページで、[モデル 1] を選択します。



3. モデルの詳細ページで [モデルを使用] を選択してから [API をクラウドに統合] を選択します。



4. [AWS CLI コマンド] セクションで、stop-model AWS CLI コマンドをコピーします。



5. コマンドプロンプトで、前のステップから stop-model AWS CLI コマンドを入力してモデルを停止します。lookoutvision プロファイルを使用して認証情報を取得する場合は、--profile lookoutvision-access パラメーターを追加します。例:

```
aws lookoutvision stop-model \
  --project-name getting-started \
  --model-version 1 \
  --profile lookoutvision-access
```

呼び出しが完了すると、次の出力が表示されます:

```
{
    "Status": "STOPPING_HOSTING"
}
```

6. コンソールで戻り、左ナビゲーションページで [モデル] を選択します。
7. 「ステータス」列のモデルのステータスが「トレーニング完了」の場合、このモデルは停止しています。

次のステップ

独自の画像によりモデルを作成する準備ができたら、プロジェクトを作成します の手引きに従って開始します。手引きには、Amazon Lookout for Vision コンソールと AWS SDK によりモデルを作成するステップが含まれています。

他のサンプルデータセットを試してみたい場合は、サンプルのコードおよびデータセット を参照してください。

Amazon Lookout for Vision モデルの作成

Amazon Lookout for Vision モデルは、モデルのトレーニングに使用される画像のパターンを見つけることによって、新しい画像の異常の存在を予測する機械学習モデルです。このセクションでは、モデルの作成とトレーニングを行う方法を説明します。モデルをトレーニングしたら、そのパフォーマンスを評価する必要があります。詳細については、「Amazon Lookout for Vision モデルの改善」を参照してください。

最初のモデルを作成する前に、「Amazon Lookout for Vision について」と「Amazon Lookout for Vision コンソールの開始」を読むことをお勧めします。を使用している場合はSDK、 AWS 「」を参照してくださいAmazon Lookout for Vision オペレーションを呼び出す。

トピック

- プロジェクトを作成します
- データセットの作成
- 画像のラベリング
- モデルのトレーニング
- モデルトレーニングのトラブルシューティング

プロジェクトを作成します

プロジェクトは、Amazon Lookout for Vision モデルを作成および管理するために必要なリソースのグループです。プロジェクトでは、次のものが管理されます。

- データセット — モデルのトレーニングに使用される画像と画像ラベル。詳細については、「データセットの作成」を参照してください。
- モデル — 異常を検出するためにトレーニングするソフトウェア。モデルのバージョンは複数持つことができます。詳細については、「モデルのトレーニング」を参照してください。

単一のタイプの機械部品の異常を検出するなど、単一のユースケースにプロジェクトを使用することをお勧めします。

> **Note**
>
> を使用して AWS CloudFormation 、Amazon Lookout for Vision プロジェクトのプロビジョニングと設定を行うことができます。詳細については、「AWS CloudFormation による Amazon Lookout for Vision プロジェクトの作成」を参照してください。

プロジェクトを表示するには、「プロジェクトの表示」を参照してください。または、[Amazon Lookout for Vision ダッシュボードを使用する] を開きます。モデルを削除するには「モデルの削除」を参照してください。

トピック

- プロジェクトの作成 (コンソール)
- プロジェクトの作成 (SDK)

プロジェクトの作成 (コンソール)

次の手続きは、コンソールを使ってプロジェクトを作成する方法です。

プロジェクトを作成するには (コンソール)

1. で Amazon Lookout for Vision コンソールを開きます https://console.aws.amazon.com/lookoutvision/。
2. 左側のナビゲーションペインで、[プロジェクト] を選択します。
3. [プロジェクトを作成] を選択します。
4. [プロジェクト名] にプロジェクトの名前を入力します。
5. [プロジェクトを作成] を選択します。プロジェクトの詳細ページが表示されます。
6. データセットを作成するには、「データセットの作成」の手順に従います。

プロジェクトの作成 (SDK)

CreateProject オペレーションを使用して、Amazon Lookout for Vision プロジェクトを作成します。からのレスポンス`CreateProject`には、プロジェクト名とプロジェクトの Amazon リソースネーム (ARN) が含まれます。その後、 CreateDatasetを呼び出して、トレーニングデータセットとテストデータセットをプロジェクトに追加します。詳細については、「マニフェストファイルを使用したデータセットの作成 (SDK)」を参照してください。

プロジェクトで作成したプロジェクトを表示するには、ListProjects を呼び出します。詳細については、「プロジェクトの表示」を参照してください。

プロジェクトを作成するには (SDK)

1. まだインストールしていない場合は、 と をインストール AWS CLI して設定します AWS SDKs。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 以下のサンプルコードを使用してモデルを作成します。

   CLI

   project-name の値をプロジェクトで使用する名前に変更します。

   ```
   aws lookoutvision create-project --project-name project name \
     --profile lookoutvision-access
   ```

   Python

   このコードは、 AWS ドキュメントSDKサンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

   ```
       @staticmethod
       def create_project(lookoutvision_client, project_name):
           """
           Creates a new Lookout for Vision project.

           :param lookoutvision_client: A Boto3 Lookout for Vision client.
           :param  project_name: The name for the new project.
           :return project_arn: The ARN of the new project.
           """
           try:
               logger.info("Creating project: %s", project_name)
               response =
    lookoutvision_client.create_project(ProjectName=project_name)
               project_arn = response["ProjectMetadata"]["ProjectArn"]
               logger.info("project ARN: %s", project_arn)
           except ClientError:
               logger.exception("Couldn't create project %s.", project_name)
               raise
           else:
               return project_arn
   ```

Java V2

このコードは、 AWS ドキュメントSDKサンプル GitHub リポジトリから取得されます。詳しい例はこちらで参照できます。

```
/**
 * Creates an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that you want to create.
 * @return ProjectMetadata Metadata information about the created project.
 */
public static ProjectMetadata createProject(LookoutVisionClient lfvClient,
 String projectName)
                throws LookoutVisionException {

        logger.log(Level.INFO, "Creating project: {0}", projectName);
        CreateProjectRequest createProjectRequest =
 CreateProjectRequest.builder().projectName(projectName)
                        .build();

        CreateProjectResponse response =
 lfvClient.createProject(createProjectRequest);

        logger.log(Level.INFO, "Project created. ARN: {0}",
 response.projectMetadata().projectArn());

        return response.projectMetadata();

}
```

3. データセットを作成するには「Amazon SageMaker Ground Truth マニフェストファイルを使用したデータセットの作成」の手順に従います。

データセットの作成

データセットには、モデルのトレーニングとテストに使用する画像と割り当てられたラベルが含まれています。プロジェクトのデータセットは、Amazon Lookout for Vision コンソールまたは

CreateDatasetオペレーションを使用して作成します。データセット画像には、作成するモデルのタイプ (画像分類または画像セグメンテーション) に従ってラベルを付ける必要があります。

トピック

データセットの画像の準備

データセットを作成するには、画像のコレクションが必要です。イメージは PNGまたは JPEG形式のファイルである必要があります。必要な画像の数とタイプは、プロジェクトに単一のデータセットがあるのか、トレーニングデータセットとテストデータセットが別々にあるのかによって異なります。

単一データセットプロジェクト

画像分類モデルを作成するには、トレーニング開始で次が必要です:

- 正常なオブジェクトの画像 20 枚以上。
- 異常なオブジェクトの画像 10 枚以上。

画像セグメンテーションモデルを作成するには、トレーニング開始で次が必要です:

- 各異常タイプの画像 20 枚以上。
- 各異常画像 (異常タイプを含む画像) には、1 タイプの異常のみが含まれている必要があります。
- 正常なオブジェクトの画像 20 枚以上。

トレーニングデータセットとテストデータセットを分けるプロジェクト

画像分類モデルを作成するには、次が必要です:

- トレーニングデータセット内の正常なオブジェクトの画像 10 枚以上。
- テストデータセット内の正常なオブジェクトの画像 10 枚以上。

- テストデータセット内の異常なオブジェクトの画像 10 枚以上。

画像セグメンテーションモデルを作成するには、次が必要です:

- 各データセットには、各異常タイプの画像 10 枚が必要です。
- 各異常画像 (異常タイプが存在する画像) には、1 タイプの異常のみが含まれている必要があります。
- 各データセットには正常なオブジェクトの画像 10 枚以上が含まれている必要があります。

より高品質なモデルを作成するには、最小画像数より多くの画像を使用してください。セグメンテーションモデルを作成する場合は、複数の異常タイプの画像を含めることをお勧めしますが、これらは Lookout for Vision がトレーニングを開始するために最低限必要とするデータとしては数えられません。

画像は単一のタイプのオブジェクトのものである必要があります。また、カメラの位置、ライティング、オブジェクトのポーズなど、画像の撮影条件を統一しておく必要があります。

トレーニングとテストのデータセット内の画像はすべて同じ寸法でなければなりません。後で、トレーニング済みモデルで分析する画像は、トレーニングとテストのデータセット画像と同じサイズでなければなりません。詳細については、「画像内の異常を検出する」を参照してください。

トレーニング画像とテスト画像はすべて固有の画像でなければならず、できれば固有のオブジェクトの画像であることが望まれます。正常な画像は、分析対象のオブジェクトにありえる正常な範囲のバリエーションを含んでいる必要があります。異常な画像では、さまざまな異常のサンプルをとらえている必要があります。

Amazon Lookout for Vision では、使用できるサンプル画像を提供しています。詳細については、「画像分類データセット」を参照してください。

画像の制限については、「クォータ」を参照してください。

データセットの作成

プロジェクト用のデータセットを作成するときは、プロジェクトの初期データセット構成を選択します。また、Lookout for Vision が画像をインポートする場所も選択できます。

プロジェクトのデータセット構成の選択

プロジェクトで最初のデータセットを作成するときは、次のデータセット構成のうち 1 つが必要です:

- 単一データセット — 単一データセットプロジェクトは、単一のデータセットを使用して、モデルのトレーニングとテストを行います。単一のデータセットを使用することで、Amazon Lookout for Visionがトレーニング画像とテスト画像を選択し、トレーニングを簡素化することができます。す。トレーニング中、Amazon Lookout for Vision では、データセットをトレーニングデータセットとテストデータセットに分割します。分割されたデータセットにはアクセスできません。ほとんどのシナリオでは、単一データセットプロジェクトを使用することをお勧めします。
- トレーニングデータセットとテストデータセットを分離する — トレーニング、テスト、パフォーマンスチューニングをより細かく制御したい場合は、トレーニングデータセットとテストデータセットを別々に持つようにプロジェクトを構成することができます。テストに使用する画像をコントロールしたい場合、または既に使用したいベンチマーク画像セットがある場合は、別のテストデータセットを使用します。

テストデータセットは、既存の単一データセットプロジェクトに追加できます。これにより、単一データセットがトレーニングデータセットになります。トレーニングデータセットとテストデータセットが別々に存在するプロジェクトからテストデータセットを削除すると、そのプロジェクトは単一データセットのプロジェクトになります。詳細については、「データセットの削除」を参照してください。

画像のインポート

データセットを作成するときは、画像のインポート元を選びます。画像のインポート方法によっては、画像にすでにラベルが付けられている場合があります。データセットの作成後に画像にラベルが付けられていない場合は、画像のラベリング を参照してください。

データセットを作成するには、次のいずれかの方法で画像をインポートします:

- ローカルコンピュータから画像をインポートします。画像にはラベルが付いていません。追加やラベル付けには、Lookout for Vision コンソールを使用します。
- S3 バケットから画像をインポートします。Amazon Lookout for Vision では、画像にラベル付けするフォルダー名で画像を分類できます。正常な画像には `normal` を使用します。異常な画像には `anomaly` を使用します。セグメンテーションラベルを自動的に割り当てることはできません。
- ラベル付きイメージを含む Amazon SageMaker Ground Truth マニフェストファイル をインポートします。独自のマニフェストファイルを作成してインポートできます。イメージが多い場合

は、 SageMaker Ground Truth ラベル付けサービスの使用を検討してください。次に、Amazon SageMaker Ground Truth ジョブから出力マニフェストファイルをインポートします。必要に応じて、Lookout for Vision コンソールを使用してラベルを追加または変更できます。

を使用している場合は AWS SDK、Amazon SageMaker Ground Truth マニフェストファイルを使用してデータセットを作成します。詳細については、「Amazon SageMaker Ground Truth マニフェストファイルを使用したデータセットの作成」を参照してください。

データセットを作成して画像にラベル付けしたら、モデルをトレーニングできます。画像にラベルが付いていない場合は、作成するモデルのタイプに応じてラベルを追加してください。詳細については、「画像のラベリング」を参照してください。

既存のデータセットにはさらに画像を追加できます。詳細については、「データセットへの画像の追加」を参照してください。

ローカルコンピュータに保存されている画像を使用してデータセットを作成する

コンピュータから直接ロードされた画像を使用して、データセットを作成できます。同時にアップロードできる画像は、30 枚までです。この手続きでは、単一データセットプロジェクト、または個別のトレーニングデータセットとテストデータセットを含むプロジェクトを作成できます。

> **Note**
>
> プロジェクトを作成します を完了したばかりの時は、コンソールにプロジェクトダッシュボードが表示され、手順 1～3 を行う必要はありません。

ローカルコンピュータ上の画像を使用してデータセットを作成するには (コンソール)

1. で Amazon Lookout for Vision コンソールを開きます https://console.aws.amazon.com/lookoutvision/。
2. 左側のナビゲーションペインで、[プロジェクト] を選択します。
3. 「プロジェクト」ページで、データセットを追加したいプロジェクトを選択します。
4. プロジェクトの詳細ページで、データセットを作成] を選択します。
5. [シングルデータセット] タブまたは [トレーニングデータセットとテストデータセットを分離] タブをクリックし、手順に従います。

Single dataset

a. [データセット構成] セクションで、[シングルデータセットを作成] を選択します。

b. [画像ソース構成] セクションで、[コンピュータから画像をアップロード] を選択します。

c. [データセットを作成] を選択します。

d. データセットページで、[画像を追加] を選択します。

e. コンピュータファイルからデータセットにアップロードする画像を選択します。画像をドラッグするか、ローカルコンピュータからアップロードする画像を選択できます。

f. [画像をアップロード] を選択します。

Separate training and test datasets

a. [データセット構成] セクションで、[トレーニングデータセットとテストデータセットを作成] を選択します。

b. [トレーニングデータセット詳細] セクションで、[コンピュータから画像をアップロード] を選択します。

c. [テストデータセット詳細] セクションで、[コンピュータから画像をアップロード] を選択します。

> **Note**
>
> トレーニングデータセットとテストデータセットは、異なる画像ソースを持つことができます。

d. [データセットを作成] を選択します。データセットページが表示され、それぞれのデータセットの[トレーニング] タブと[テスト] タブが表示されます。

e. [アクション] を選択し、[トレーニングデータセットに画像を追加] をクリックします。

f. データセットにアップロードする画像を選択します。画像をドラッグするか、ローカルコンピュータからアップロードする画像を選択できます。

g. [画像をアップロード] を選択します。

h. ステップ 5e ~ 5g を繰り返します。ステップ 5e で、[アクション] を選択します。[テストデータセットに画像を追加] をクリックします。

6. 「画像のラベリング」の手順に従って、画像にラベルを付けます。

7. 「モデルのトレーニング」の手順に従って、モデルをトレーニングします。

Amazon S3 バケットに保存されている画像を使用してデータセットを作成します。

Amazon S3 バケットに保存されている画像を使用して、データセットを作成できます。このオプションを使用すると、Amazon S3 バケットのフォルダー構造を使用して、画像を自動的に分類できます。画像は、コンソールバケットまたはアカウントの別の Amazon S3 バケットに保存できます。

自動ラベル付け用のフォルダの設定

データセットの作成中に、画像を含むフォルダの名前に基づいて画像にラベル名を割り当てることを選択できます。フォルダは、データセットの作成時に Amazon S3 S3 URIフォルダパスの子である必要があります。

以下は、スタート用サンプル画像の `train` フォルダです。Amazon S3 フォルダの場所を`S3-bucket/circuitboard/train/` として指定した場合、`normal` フォルダ内の画像には、`Normal` ラベルが付与されます。`anomaly` フォルダ内の画像には、`Anomaly` ラベルが割り当てられています。より深い子フォルダの名前は、画像のラベル付けには使用されません。

```
S3-bucket
  ### circuitboard
    ### train
        ### anomaly
            ### train-anomaly_1.jpg
            ### train-anomaly_2.jpg
            ### .
            ### .
        ### normal
            ### train-normal_1.jpg
            ### train-normal_2.jpg
            ### .
            ### .
```

Amazon S3 バケットの画像を使用してデータセットを作成する

以下の手続きで、Amazon S3 バケットに保存されている分類サンプル画像を使用して、データセットを作成します。独自の画像を使用するには、自動ラベル付け用のフォルダの設定 で説明されているフォルダ構造を作成します。

この手続きでは、単一データセットプロジェクト、または個別のトレーニングデータセットとテストデータセットを使用するプロジェクトを作成する方法も示します。

画像に自動的にラベルを付けるように選択しない場合は、データセットの作成後に画像にラベルを付ける必要があります。詳細については、「画像の分類 (コンソール)」を参照してください。

> **Note**
>
> プロジェクトを作成します を完了したばかりの時は、コンソールにプロジェクトダッシュボードが表示され、手順 1～4 を行う必要はありません。

Amazon S3 バケットに保存されている画像を使用してデータセットを作成するには

1. Amazon S3 バケットにスタート用画像をアップロードしていない場合は、それをアップロードします。詳細については、「画像分類データセット」を参照してください。
2. で Amazon Lookout for Vision コンソールを開きます https://console.aws.amazon.com/lookoutvision/。
3. 左側のナビゲーションペインで、[プロジェクト] を選択します。
4. 「プロジェクト」ページで、データセットを追加したいプロジェクトを選択します。プロジェクトの詳細ページが表示されます。
5. [データセットを作成] を選択します。「データセットを作成」ページが表示されます。

   > **Tip**
   >
   > スタートガイドの手順に従っている場合は、[トレーニングデータセットとテストデータセットを作成] を選択します。

6. [シングルデータセット] タブまたは [トレーニングデータセットとテストデータセットを分離] タブを選択し、手順に従います。

   Single dataset

   a. [データセット構成] セクションで、[シングルデータセットを作成] を選択します。

   b. [画像ソース構成] に手順7～9の情報を入力します。

Separate training and test datasets

a. [データセット構成] セクションで、[トレーニングデータセットとテストデータセットを作成] を選択します。

b. トレーニングデータセットについて、手順 7～9 の情報を [トレーニングデータセット詳細] ( ) セクションに入力します。

c. テストデータセットについて、手順 7～9 の情報を [テストデータセット詳細] ( ) セクションに入力します。

> **Note**
>
> トレーニングデータセットとテストデータセットは、異なる画像ソースを持つことができます。

7. [Amazon S3 バケットから画像をインポート] を選択します。
8. S3 URIで、Amazon S3 バケットの場所とフォルダパスを入力します。「bucket」を Amazon S3 バケットの名前に変更します。

   a. 単一データセットプロジェクトまたはトレーニングデータセットを作成する場合は、次のように入力します。:

   ```
   s3://bucket/circuitboard/train/
   ```

   b. テストデータセットを作成する場合は、次のように入力します。

   ```
   s3://bucket/circuitboard/test/
   ```

9. [フォルダに基づいて画像にラベルを自動的にアタッチ] を選択します。
10. [データセットを作成] を選択します。ラベル付きの画像を含むデータセットページが開きます。
11. 「モデルのトレーニング」の手順に従って、モデルをトレーニングします。

Amazon SageMaker Ground Truth マニフェストファイルを使用したデータセットの作成

マニフェストファイルには、モデルのトレーニングとテストに使用できる画像と画像ラベルに関する情報が含まれています。マニフェストファイルを Amazon S3 バケットに保存し、それを使

用してデータセットを作成できます。独自のマニフェストファイルを作成することも、Amazon SageMaker Ground Truth ジョブからの出力などの既存のマニフェストファイルを使用することもできます。

トピック

- Amazon SageMaker Ground Truth ジョブの使用
- マニフェストファイルの作成

Amazon SageMaker Ground Truth ジョブの使用

画像のラベリングには、かなりの時間がかかる場合があります。たとえば、異常の周囲にマスクを正確に描画するには数十秒かかることがあります。画像が数百枚ある場合は、ラベリングに数時間かかるかもしれません。イメージに自分でラベルを付ける代わりに、Amazon SageMaker Ground Truth の使用を検討してください。

Amazon SageMaker Ground Truth では、選択したベンダー会社の Amazon Mechanical Turk または社内のプライベートワークフォースのいずれかのワーカーを使用して、ラベル付き画像セットを作成できます。詳細については、「Amazon SageMaker Ground Truth を使用してデータにラベルを付ける」を参照してください。

Amazon Mechanical Turk の使用には費用がかかります。また、Amazon Ground Truth のラベリングジョブが完了するには数日かかる場合があります。費用が問題になる場合や、モデルをすばやくトレーニングする必要がある場合は、Amazon Lookout for Vision コンソールを使用して画像にラベル付けすることをお勧めします。

Amazon SageMaker Ground Truth ラベル付けジョブを使用して、画像分類モデルと画像セグメンテーションモデルに適した画像にラベルを付けることができます。ジョブが完了したら、出力マニフェストファイルを使用して Amazon Lookout for Vision データセットを作成します。

画像分類

画像分類モデルの画像にラベルを付けるには、画像分類 (単一ラベル) タスク用のラベリングジョブを作成します。

画像セグメンテーション

画像セグメンテーションモデルの画像にラベルを付けるには、画像分類 (単一ラベル) タスク用のラベリングジョブを作成します。次に、そのジョブを連結して、画像セマンティックセグメンテーションタスク用のラベリングジョブを作成します。

ラベリングジョブを使用して、画像セグメンテーションモデルの部分的なマニフェストファイルを作成することもできます。たとえば、画像分類 (単一ラベル) タスクを使用して画像を分類できます。ジョブ出力を含む Lookout for Vision データセットを作成したら、Amazon Lookout for Vision コンソールを使用して、データセット画像にセグメンテーションマスクと異常ラベルを追加します。

Amazon SageMaker Ground Truth によるイメージのラベル付け

次の手順は、Amazon SageMaker Ground Truth イメージラベル付けタスクを使用してイメージにラベルを付ける方法を示しています。この手続きでは、画像分類マニフェストファイルを作成し、オプションで画像ラベリングタスクを連結して画像セグメンテーションマニフェストファイルを作成します。プロジェクトで別のテストデータセットを扱いたい場合は、この手続きを繰り返してテストデータセットのマニフェストファイルを作成します。

Amazon SageMaker Ground Truth でイメージにラベルを付けるには (コンソール)

1. 「ラベリングジョブを作成する (コンソール)」の手引きに従って、画像分類 (単一ラベル) タスクの Ground Truth ジョブを作成します。

   a. ステップ 10 で、[タスクカテゴリ] ドロップダウンメニューから [画像] を選択し、[画像分類 (単一ラベル)] をタスクタイプとして指定します。
   b. ステップ 16 で、[画像分類 (単一ラベル) ラベリングツール] セクションで 2 つのラベル「正常」と「異常」を追加します。
2. 担当者の方が画像の分類を完了するまでお待ちください。
3. 画像セグメンテーションモデル用のデータセットを作成する場合は、次の操作を行います。それ以外の場合はステップ 4 に進みます。

   a. Amazon SageMaker Ground Truth コンソールで、ラベル付けジョブページを開きます。
   b. 以前に作成した EC2 インスタンスを選択します。これにより、[アクション] メニューが有効になります。
   c. [アクション] メニューから [連鎖] を選択します。ジョブ詳細ページが開きます。
   d. [タスクタイプ] で、[セマンティックセグメンテーション] を選びます。
   e. Choose Next.
   f. [セマンティックセグメンテーションラベリングツール] セクションで、モデルに見つけてほしい各タイプの異常における異常ラベルを追加します。
   g. Choose Create.
   h. 担当者の方が画像にラベルを付けるまでお待ちください。
4. Ground Truth コンソールを開き、「ラベリングジョブ」ページを開きます。

5. 画像分類モデルを作成する場合は、ステップ 1 で作成したタグを選択します。画像セグメンテーションモデルを作成する場合は、ステップ 3 で作成したジョブを選択します。
6. 「ラベリングジョブサマリー」の [出力データセットの場所] で S3 の場所を開きます。マニフェストファイルの場所を書き留めておきます (s3://*output-dataset-location*/manifests/output/output.manifest のはずです)。
7. テストデータセット用マニフェストファイルを作成する場合は、この手続きを繰り返します。それ以外の場合は、データセットの作成 の手引きに従って、マニフェストファイルによりデータセットを作成してください。

データセットの作成

以下の手続きに従って、 Amazon SageMaker Ground Truth によるイメージのラベル付け のステップ 6 でメモしたマニフェストファイルにより Lookout for Vision プロジェクトにデータセットを作成します。マニフェストファイルは単一のデータセットプロジェクトのトレーニングデータセットを作成します。プロジェクトに別のテストデータセットを持たせたい場合は、別の Amazon SageMaker Ground Truth ジョブを実行して、テストデータセットのマニフェストファイルを作成できます。または、マニフェストファイルを自分で作成することもできます。画像は Amazon S3 バケットかローカルコンピューターからテストデータセットにインポートすることもできます。(画像へのラベリングはモデルをトレーニングする前に必要となる場合があります)。

この手続きでは、プロジェクトにデータセットがないことを前提としています。

Lookout for Vision によりデータセットを作成するには (コンソール)

1. で Amazon Lookout for Vision コンソールを開きます https://console.aws.amazon.com/lookoutvision/。
2. [開始する] を選択します。
3. 左側のナビゲーションペインで、[プロジェクト] を選択します。
4. マニフェストファイルで使用するために追加するプロジェクトを選択します。
5. [操作方法] セクションで、[データセットを作成] を選択します。
6. [単一データセット] タブまたは [トレーニングデータセットとテストデータセットを分離] タブをクリックし、手順に従います。

   Single dataset

   1. [単一データセットを作成] をクリックします。

2. 「画像ソース設定」セクションで、「Ground Truth で SageMakerラベル付けされた画像をインポートする」を選択します。
3. [.manifest ファイルの場所] には、 Amazon SageMaker Ground Truth によるイメージのラベル付け のステップ 6 で記したマニフェストファイルの場所を入力します。

Separate training and test datasets

1. [トレーニングデータセットとテストデータセットを作成] を選択します。
2. 「トレーニングデータセットの詳細」セクションで、「Ground Truth で SageMakerラベル付けされたイメージをインポートする」を選択します。
3. [.manifest ファイルの場所] には、 Amazon SageMaker Ground Truth によるイメージのラベル付け のステップ 6 でメモしたマニフェストファイルの場所を入力します。
4. 「データセットの詳細をテストする」セクションで、 SageMaker 「Ground Truth でラベル付けされた画像のインポート」を選択します。
5. [.manifest ファイルの場所] には、 Amazon SageMaker Ground Truth によるイメージのラベル付け のステップ 6 でメモしたマニフェストファイルの場所を入力します。テストデータセットには別のマニフェストファイルが必要なことを忘れないでください。

7. Choose Submit.
8. 「モデルのトレーニング」の手順に従って、モデルをトレーニングします。

マニフェストファイルの作成

SageMaker Ground Truth 形式のマニフェストファイルをインポートしてデータセットを作成できます。イメージが SageMaker Ground Truth マニフェストファイルではない形式でラベル付けされている場合は、次の情報を使用して SageMaker Ground Truth 形式のマニフェストファイルを作成します。

マニフェストファイルはJSON行形式で、各行はイメージのラベル情報を表す完全なJSONオブジェクトです。画像分類と画像セグメンテーションにはさまざまな形式があります。マニフェストファイルは UTF-8 エンコードを使用してエンコードする必要があります。

> **Note**
>
> このセクションのJSON行例は、読みやすくするためにフォーマットされています。

マニフェストファイルで参照される画像は、同じ Amazon S3 バケットに配置する必要があります。マニフェストファイルは別のバケットにあることも可能です。JSON 線の `source-ref`フィールドにイメージの場所を指定します。

コードを使用してマニフェストファイルを作成できます。Amazon Lookout for Vision Lab Python Notebook は、回路基板サンプル画像の画像分類マニフェストファイルを作成する方法を示しています。または、コード例リポジトリのデータセットサンプル AWS コードを使用することもできます。マニフェストファイルは、カンマ区切り値 (CSV) ファイルを使用して簡単に作成できます。詳細については、「ファイルからの分類マニフェストCSVファイルの作成」を参照してください。

トピック

- 画像分類のJSON行の定義
- 画像セグメンテーションのJSON線の定義
- ファイルからの分類マニフェストCSVファイルの作成
- マニフェストファイルを使用したデータセットの作成 (コンソール)
- マニフェストファイルを使用したデータセットの作成 (SDK)

画像分類のJSON行の定義

Amazon Lookout for Vision マニフェストファイルで使用するイメージごとにJSON行を定義します。分類モデルを作成する場合、JSON行には正常または異常のいずれかのイメージ分類が含まれている必要があります。JSON 行は SageMaker Ground Truth 分類ジョブ出力形式です。マニフェストファイルはJSON、インポートするイメージごとに 1 行以上で構成されます。

分類された画像のマニフェストファイルを作成するには

1. 空のテキストファイルを作成します。
2. インポートするイメージごとにJSON行を追加します。各JSON行は次のようになります。

```
{"source-ref":"s3://lookoutvision-console-us-east-1-nnnnnnnnnn/gt-job/manifest/
IMG_1133.png","anomaly-label":1,"anomaly-label-metadata":{"confidence":0.95,"job-
name":"labeling-job/testclconsolebucket","class-name":"normal","human-
annotated":"yes","creation-date":"2020-04-15T20:17:23.433061","type":"groundtruth/
image-classification"}}
```

3. ファイルを保存します。

> **Note**
>
> .manifest 拡張機能を使用できますが、必須ではありません。

4. 作成したマニフェストファイルを使用してデータセットを作成します。詳細については、「マニフェストファイルの作成」を参照してください。

分類JSON行

このセクションでは、イメージを正常または異常として分類するJSON行を作成する方法について説明します。

異常JSON線

次のJSON行は、異常としてラベル付けされた画像を示しています。class-name の値はanomalyである点に注意してください。

```
{
    "source-ref": "s3: //bucket/image/anomaly/abnormal-1.jpg",
    "anomaly-label-metadata": {
        "confidence": 1,
        "job-name": "labeling-job/auto-label",
        "class-name": "anomaly",
        "human-annotated": "yes",
        "creation-date": "2020-11-10T03:37:09.600",
        "type": "groundtruth/image-classification"
    },
    "anomaly-label": 1
}
```

通常のJSON行

次のJSON行は、通常とラベル付けされた画像を示しています。class-name の値はnormalである点に注意してください。

```
{
    "source-ref": "s3: //bucket/image/normal/2020-10-20_12-14-55_613.jpeg",
```

```
    "anomaly-label-metadata": {
        "confidence": 1,
        "job-name": "labeling-job/auto-label",
        "class-name": "normal",
        "human-annotated": "yes",
        "creation-date": "2020-11-10T03:37:09.603",
        "type": "groundtruth/image-classification"
    },
    "anomaly-label": 0
}
```

JSON 行のキーと値

次の情報は、Amazon Lookout for Vision JSON行のキーと値について説明します。

source-ref

(必須) 画像の Amazon S3 の場所。形式は "s3://*BUCKET*/*OBJECT_PATH*" です。インポートされたデータセット内の画像は、同じ Amazon S3 バケットに格納する必要があります。

anomaly-label

(必須) ラベル属性。anomaly-label キー、または選択した別のキー名を使用してください。キーバリュー (前の例では 0) は Amazon Lookout for Vision で必須ですが、使用されていません。Amazon Lookout for Vision が作成する出力マニフェストでは、異常画像の場合は値が 1 に、正常画像の場合は値が 0 に変換されます。class-name の値は、画像が正常か異常かを判断します。

末尾に -metadata が付いたフィールド名により特定される該当メタデータが必要です。例えば、"anomaly-label-metadata" と指定します。

anomaly-label-metadata

(必須) ラベル属性に関するメタデータ。フィールド名は、-metadata を付加したラベル属性と同じでなければなりません。

confidence

(Optional) Currently not used by Amazon Lookout for Vision. 値を指定する場合は、1 の値を使用します。

job-name

(オプション) 画像を処理するジョブに対して選択した名前。

class-name

(必須) 画像に正常なコンテンツが含まれている場合は、normal を指定します。そうでない場合は、anomaly を指定します。class-name がそれ以外の値であれば、画像はラベルなしの画像としてデータセットに追加されます。画像にラベルを付けるには、「データセットへの画像の追加」を参照してください。

human-annotated

(必須) アノテーションが人によって完成されている場合、"yes" を指定します。それ以外の場合は、"no"を指定します。

creation-date

(オプション) ラベルが作成された協定世界時 (UTC) の日時。

type

(必須) 画像に適用する処理のタイプ。画像レベルの異常ラベルの場合、値は "groundtruth/image-classification" です。

画像セグメンテーションのJSON線の定義

Amazon Lookout for Vision マニフェストファイルで使用するイメージごとにJSON行を定義します。セグメンテーションモデルを作成する場合、JSON線には画像のセグメンテーションと分類情報を含める必要があります。マニフェストファイルはJSON、インポートするイメージごとに 1 行以上で構成されます。

セグメント化された画像のマニフェストファイルを作成するには

1. 空のテキストファイルを作成します。
2. インポートするイメージごとにJSON行を追加します。各JSON行は次のようになります。

```
{"source-ref":"s3://path-to-image","anomaly-label":1,"anomaly-label-metadata":
{"class-name":"anomaly","creation-date":"2021-10-12T14:16:45.668","human-
annotated":"yes","job-name":"labeling-job/classification-job","type":"groundtruth/
image-classification","confidence":1},"anomaly-mask-ref":"s3://path-to-
image","anomaly-mask-ref-metadata":{"internal-color-map":{"0":{"class-
name":"BACKGROUND","hex-color":"#ffffff","confidence":0.0},"1":{"class-
name":"scratch","hex-color":"#2ca02c","confidence":0.0},"2":{"class-
name":"dent","hex-color":"#1f77b4","confidence":0.0}},"type":"groundtruth/
```

```
semantic-segmentation","human-annotated":"yes","creation-
date":"2021-11-23T20:31:57.758889","job-name":"labeling-job/segmentation-job"}}
```

3. ファイルを保存します。

> **Note**
>
> .manifest 拡張機能を使用できますが、必須ではありません。

4. 作成したマニフェストファイルを使用してデータセットを作成します。詳細については、「マニフェストファイルの作成」を参照してください。

セグメンテーションJSONライン

このセクションでは、画像のセグメンテーションと分類情報を含むJSON行を作成する方法について説明します。

次のJSON行は、セグメンテーションおよび分類情報を含む画像を示しています。 anomaly-label-metadataには分類情報が含まれています。 anomaly-mask-refと にはセグメンテーション情報anomaly-mask-ref-metadataが含まれています。

```
{
    "source-ref": "s3://path-to-image",
    "anomaly-label": 1,
    "anomaly-label-metadata": {
        "class-name": "anomaly",
        "creation-date": "2021-10-12T14:16:45.668",
        "human-annotated": "yes",
        "job-name": "labeling-job/classification-job",
        "type": "groundtruth/image-classification",
        "confidence": 1
    },
    "anomaly-mask-ref": "s3://path-to-image",
    "anomaly-mask-ref-metadata": {
        "internal-color-map": {
            "0": {
                "class-name": "BACKGROUND",
                "hex-color": "#ffffff",
                "confidence": 0.0
            },
            "1": {
```

```
                "class-name": "scratch",
                "hex-color": "#2ca02c",
                "confidence": 0.0
            },
            "2": {
                "class-name": "dent",
                "hex-color": "#1f77b4",
                "confidence": 0.0
            }
        },
        "type": "groundtruth/semantic-segmentation",
        "human-annotated": "yes",
        "creation-date": "2021-11-23T20:31:57.758889",
        "job-name": "labeling-job/segmentation-job"
    }
}
```

JSON 行のキーと値

次の情報は、Amazon Lookout for Vision JSON行のキーと値について説明します。

source-ref

(必須) 画像の Amazon S3 の場所。形式は "s3://*BUCKET*/*OBJECT_PATH*" です。インポートされたデータセット内の画像は、同じ Amazon S3 バケットに格納する必要があります。

anomaly-label

(必須) ラベル属性。anomaly-label キー、または選択した別のキー名を使用してください。キーバリュー (前の例では 1) は Amazon Lookout for Vision で必須ですが、使用されていません。Amazon Lookout for Vision が作成する出力マニフェストでは、異常画像の場合は値が 1 に、正常画像の場合は値が 0 に変換されます。class-name の値は、画像が正常か異常かを判断します。

末尾に -metadata が付いたフィールド名により特定される該当メタデータが必要です。例えば、"anomaly-label-metadata" と指定します。

anomaly-label-metadata

(必須) ラベル属性に関するメタデータ。分類情報を含みます。フィールド名は、末尾に -metadata が付いたラベル属性と同じでなければなりません。

confidence

(Optional) Currently not used by Amazon Lookout for Vision. 値を指定する場合は、1 の値を使用します。

job-name

(オプション) 画像を処理するジョブに対して選択した名前。

class-name

(必須) 画像に正常なコンテンツが含まれている場合は、`normal` を指定します。そうでない場合は、`anomaly` を指定します。`class-name` がそれ以外の値であれば、画像はラベルなしの画像としてデータセットに追加されます。画像にラベルを付けるには、「データセットへの画像の追加」を参照してください。

human-annotated

(必須) アノテーションが人によって完成されている場合、`"yes"` を指定します。それ以外の場合は、`"no"`を指定します。

creation-date

(オプション) ラベルが作成された協定世界時 (UTC) の日時。

type

(必須) 画像に適用する処理のタイプ。値 `"groundtruth/image-classification"` を使用します。

anomaly-mask-ref

(必須) Amazon S3 におけるマスク画像の場所。キー名用の `anomaly-mask-ref` を使用するか、任意のキー名を使用してください。キーは `-ref` で終わる必要があります。マスク画像は、異常タイプ `internal-color-map` ごとに色付きのマスクを含む必要があります。形式は `"s3://BUCKET/OBJECT_PATH"` です。インポートされたデータセット内の画像は、同じ Amazon S3 バケットに格納する必要があります。マスクイメージは、ポータブルネットワークグラフィックス (PNG) 形式のイメージである必要があります。

anomaly-mask-ref-metadata

(必須) 画像のセグメンテーションメタデータ。キー名用の `anomaly-mask-ref-metadata` を使用するか、任意のキー名を使用してください。キー名は `-ref-metadata` で終わる必要があります。

internal-color-map

(必須) 個々の異常タイプにマップする色のマップ。色はマスク画像 (anomaly-mask-ref) 内の色と一致する必要があります。

key

(必須) マップへのキー。エントリには、イメージの異常以外の領域BACKGROUNDを表すクラス名が含まれている0必要があります。

class-name

(必須) 傷やへこみなどの異常タイプの名前。

hex-color

(必須) 異常タイプに対するウェブカラー (#2ca02c など)。色は anomaly-mask-ref の色と一致する必要があります。異常タイプ BACKGROUND の値は常に #ffffff です。

confidence

(必須) 現在、Amazon Lookout for Vision で使用されてませんが、浮動小数点値が必要です。

human-annotated

(必須) アノテーションが人によって完成されている場合、"yes" を指定します。それ以外の場合は、"no"を指定します。

creation-date

(オプション) セグメンテーション情報が作成された協定世界時 (UTC) の日時。

type

(必須) 画像に適用する処理のタイプ。値 "groundtruth/semantic-segmentation" を使用します。

ファイルからの分類マニフェストCSVファイルの作成

この Python スクリプトの例では、カンマ区切り値 (CSV) ファイルを使用してイメージにラベルを付けることで、分類マニフェストファイルの作成を簡素化します。CSV ファイルを作成します。

マニフェストファイルはモデルのトレーニングに使用される複数の画像について記述したものです。マニフェストファイルは 1 JSON行以上で構成されます。各JSON行は 1 つのイメージを記述します。詳細については、「画像分類のJSON行の定義」を参照してください。

CSV ファイルは、テキストファイル内の複数の行にわたる表形式データを表します。行内のフィールドはカンマで区切られます。詳細については、「カンマ区切り値」を参照してください。このスクリプトでは、CSVファイル内の各行には、イメージの S3 の場所とイメージの異常分類 (normal または anomaly) が含まれます。各行はマニフェストファイルのJSON行にマッピングされます。

例えば、次の CSV ファイルでは、サンプルイメージ 内のイメージの一部について説明しています。

```
s3://s3bucket/circuitboard/train/anomaly/train-anomaly_1.jpg,anomaly
s3://s3bucket/circuitboard/train/anomaly/train-anomaly_10.jpg,anomaly
s3://s3bucket/circuitboard/train/anomaly/train-anomaly_11.jpg,anomaly
s3://s3bucket/circuitboard/train/normal/train-normal_1.jpg,normal
s3://s3bucket/circuitboard/train/normal/train-normal_10.jpg,normal
s3://s3bucket/circuitboard/train/normal/train-normal_11.jpg,normal
```

このスクリプトは、行ごとにJSON行を生成します。例えば、最初の行のJSON行 (s3://s3bucket/circuitboard/train/anomaly/train-anomaly_1.jpg,anomaly) を次に示します。

```
{"source-ref": "s3://s3bucket/csv_test/train_anomaly_1.jpg","anomaly-label":
 1,"anomaly-label-metadata": {"confidence": 1,"job-name": "labeling-job/anomaly-
classification","class-name": "anomaly","human-annotated": "yes","creation-date":
 "2022-02-04T22:47:07","type": "groundtruth/image-classification"}}
```

CSV ファイルにイメージの Amazon S3 パスが含まれていない場合は、--s3-pathコマンドライン引数を使用してイメージへの Amazon S3 パスを指定します。

マニフェストファイルを作成する前に、スクリプトはCSVファイル内の重複イメージと、 normalまたは ではないイメージ分類をチェックしますanomaly。重複画像または画像分類エラーが見つかった場合、スクリプトは次の処理を行います:

- 重複排除されたCSVファイル内のすべてのイメージの最初の有効なイメージエントリを記録します。
- 画像の重複発生をエラーファイルに記録します。
- エラーファイルの、normalではない、またはanomalyな画像分類を記録します。
- マニフェストファイルは作成しません。

エラーファイルには、入力CSVファイルに重複したイメージまたは分類エラーが見つかった行番号が含まれます。エラーCSVファイルを使用して入力CSVファイルを更新し、スクリプトを再度実行します。または、エラーCSVファイルを使用して重複排除されたCSVファイルを更新します。この

ファイルには、一意のイメージエントリとイメージ分類エラーのないイメージのみが含まれます。更新された重複排除CSVファイルを使用してスクリプトを再実行します。

入力CSVファイルに重複またはエラーが見つからない場合、スクリプトは重複排除されたイメージCSVファイルとエラーファイルを空として削除します。

この手順では、 CSV ファイルを作成し、Python スクリプトを実行してマニフェストファイルを作成します。スクリプトは Python 3.7 によりテストされました。

CSV ファイルからマニフェストファイルを作成するには

1. 各行 (イメージごとに 1 行) に次のフィールドを含むCSVファイルを作成します。ファイルにはヘッダー行を追加しないでくださいCSV。

フィールド 1	フィールド 2
画像名または Amazon S3 画像パス。例えば、`s3://s3bucket/circuitboard/train/anomaly/train-anomaly_10.jpg` と指定します。Amazon S3 パスによる画像と画像とパスがない画像を混在させることはできません。	画像の異常分類 (`normal`または`anomaly`)。

   例えば、`s3://s3bucket/circuitboard/train/anomaly/image_10.jpg,anomaly`、`image_11.jpg,normal` などです。
2. CSV ファイルを保存します。
3. 以下の Python スクリプトを実行します。以下の情報を提供します:

   - `csv_file` – ステップ 1 で作成したCSVファイル。
   - (オプション) `--s3-path` *`s3://path_to_folder/`* — 画像ファイル名に追加する Amazon S3 パス (フィールド 1)。フィールド 1 の画像がまだ S3 パスを含んでいない場合は `--s3-path` を使用します。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier:  Apache-2.0
"""
Purpose
```

```
Shows how to create an Amazon Lookout for Vision manifest file from a CSV file.
The CSV file format is image location,anomaly classification (normal or anomaly)
For example:
s3://s3bucket/circuitboard/train/anomaly/train_11.jpg,anomaly
s3://s3bucket/circuitboard/train/normal/train_1.jpg,normal

If necessary, use the bucket argument to specify the Amazon S3 bucket folder for
 the images.
"""

from datetime import datetime, timezone
import argparse
import logging
import csv
import os
import json

logger = logging.getLogger(__name__)


def check_errors(csv_file):
    """
    Checks for duplicate images and incorrect classifications in a CSV file.
    If duplicate images or invalid anomaly assignments are found, an errors CSV
 file
    and deduplicated CSV file are created. Only the first
    occurrence of a duplicate is recorded. Other duplicates are recorded in the
 errors file.
    :param csv_file: The source CSV file
    :return: True if errors or duplicates are found, otherwise false.
    """

    logger.info("Checking %s.", csv_file)

    errors_found = False
    errors_file = f"{os.path.splitext(csv_file)[0]}_errors.csv"
    deduplicated_file = f"{os.path.splitext(csv_file)[0]}_deduplicated.csv"

    with open(csv_file, 'r', encoding="UTF-8") as input_file,\
            open(deduplicated_file, 'w', encoding="UTF-8") as dedup,\
            open(errors_file, 'w', encoding="UTF-8") as errors:

        reader = csv.reader(input_file, delimiter=',')
        dedup_writer = csv.writer(dedup)
```

```
        error_writer = csv.writer(errors)
        line = 1
        entries = set()
        for row in reader:

            # Skip empty lines.
            if not ''.join(row).strip():
                continue

            # Record any incorrect classifications.
            if not row[1].lower() == "normal" and not row[1].lower() == "anomaly":
                error_writer.writerow(
                    [line, row[0], row[1], "INVALID_CLASSIFICATION"])
                errors_found = True

            # Write first image entry to dedup file and record duplicates.
            key = row[0]
            if key not in entries:
                dedup_writer.writerow(row)
                entries.add(key)
            else:
                error_writer.writerow([line, row[0], row[1], "DUPLICATE"])
                errors_found = True
            line += 1

    if errors_found:
        logger.info("Errors found check %s.", errors_file)
    else:
        os.remove(errors_file)
        os.remove(deduplicated_file)

    return errors_found


def create_manifest_file(csv_file, manifest_file, s3_path):
    """
    Read a CSV file and create an Amazon Lookout for Vision classification manifest
 file.
    :param csv_file: The source CSV file.
    :param manifest_file: The name of the manifest file to create.
    :param s3_path: The Amazon S3 path to the folder that contains the images.
    """
    logger.info("Processing CSV file %s.", csv_file)
```

```
    image_count = 0
    anomalous_count = 0

    with open(csv_file, newline='', encoding="UTF-8") as csvfile,\
        open(manifest_file, "w", encoding="UTF-8") as output_file:

        image_classifications = csv.reader(
            csvfile, delimiter=',', quotechar='|')

        # Process each row (image) in the CSV file.
        for row in image_classifications:
            # Skip empty lines.
            if not ''.join(row).strip():
                continue

            source_ref = str(s3_path) + row[0]
            classification = 0

            if row[1].lower() == 'anomaly':
                classification = 1
                anomalous_count += 1

           # Create the JSON line.
            json_line = {}
            json_line['source-ref'] = source_ref
            json_line['anomaly-label'] = str(classification)

            metadata = {}
            metadata['confidence'] = 1
            metadata['job-name'] = "labeling-job/anomaly-classification"
            metadata['class-name'] = row[1]
            metadata['human-annotated'] = "yes"
            metadata['creation-date'] = datetime.now(timezone.utc).strftime('%Y-%m-
%dT%H:%M:%S.%f')
            metadata['type'] = "groundtruth/image-classification"

            json_line['anomaly-label-metadata'] = metadata

            output_file.write(json.dumps(json_line))
            output_file.write('\n')
            image_count += 1

    logger.info("Finished creating manifest file %s.\n"
                "Images: %s\nAnomalous: %s",
```

```
                manifest_file,
                image_count,
                anomalous_count)
    return image_count, anomalous_count


def add_arguments(parser):
    """
    Add command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "csv_file", help="The CSV file that you want to process."
    )

    parser.add_argument(
        "--s3_path", help="The Amazon S3 bucket and folder path for the images."
        " If not supplied, column 1 is assumed to include the Amazon S3 path.",
 required=False
    )


def main():

    logging.basicConfig(level=logging.INFO,
                        format="%(levelname)s: %(message)s")

    try:

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()
        s3_path = args.s3_path
        if s3_path is None:
            s3_path = ""

        csv_file = args.csv_file
        csv_file_no_extension = os.path.splitext(csv_file)[0]
        manifest_file = csv_file_no_extension + '.manifest'

        # Create manifest file if there are no duplicate images.
        if check_errors(csv_file):
```

```
            print(f"Issues found. Use {csv_file_no_extension}_errors.csv "\
                "to view duplicates and errors.")
            print(f"{csv_file}_deduplicated.csv contains the first"\
                "occurrence of a duplicate.\n"
                  "Update as necessary with the correct information.")
            print(f"Re-run the script with
 {csv_file_no_extension}_deduplicated.csv")
        else:
            print('No duplicates found. Creating manifest file.')

            image_count, anomalous_count = create_manifest_file(csv_file,
 manifest_file, s3_path)

            print(f"Finished creating manifest file: {manifest_file} \n")

            normal_count = image_count-anomalous_count
            print(f"Images processed: {image_count}")
            print(f"Normal: {normal_count}")
            print(f"Anomalous: {anomalous_count}")

    except FileNotFoundError as err:
        logger.exception("File not found.:%s", err)
        print(f"File not found: {err}. Check your input CSV file.")

if __name__ == "__main__":
    main()
```

4. 画像が重複したり、分類エラーが発生したりする場合:

   a. エラーファイルを使用して、重複排除されたCSVファイルまたは入力CSVファイルを更新します。

   b. 更新された重複排除CSVファイルまたは更新された入力CSVファイルを使用してスクリプトを再度実行します。

5. テストデータセットを使用する場合は、ステップ 1 ~ 4 を繰り返して、テストデータセットのマニフェストファイルを作成します。

6. 必要に応じて、コンピュータから、CSVファイルの列 1 で指定した (またはコマンドラインで指定した) Amazon S3 バケットパスにイメージをコピーします。 `--s3-path`画像をコピーするには、コマンドプロンプトで次のコマンドを入力します。

```
aws s3 cp --recursive your-local-folder/ s3://your-target-S3-location/
```

7. マニフェストファイルを使用したデータセットの作成 (コンソール) の手引きに従ってデータセットを作成します。を使用している場合は、 AWS SDK「」を参照してくださいマニフェストファイルを使用したデータセットの作成 (SDK) 。

マニフェストファイルを使用したデータセットの作成 (コンソール)

次の手順では、Amazon S3 バケットに保存されている SageMaker 形式のマニフェストファイルをインポートして、トレーニングデータセットまたはテストデータセットを作成する方法を示します。

データセットを作成したら、データセットに画像を追加したり、画像にラベルを追加したりできます。詳細については、「データセットへの画像の追加」を参照してください。

SageMaker Ground Truth 形式のマニフェストファイルを使用してデータセットを作成するには (コンソール)

1. 既存の Amazon Lookout for Vision 互換の SageMaker Ground Truth 形式のマニフェストファイルを作成または使用します。詳細については、「マニフェストファイルの作成」を参照してください。
2. にサインイン AWS Management Console し、 で Amazon S3 コンソールを開きますhttps://console.aws.amazon.com/s3/。
3. Amazon S3 バケットで、フォルダを作成してマニフェストファイルを格納します。
4. 先ほど作成したフォルダに、マニフェストファイルをアップロードしてください。
5. Amazon S3 バケットで、画像を保存するフォルダを作成します。
6. 作成したフォルダに画像をアップロードします。

> **Important**
>
> 各JSON行の`source-ref`フィールド値は、 フォルダ内のイメージにマッピングする必要があります。

7. で Amazon Lookout for Vision コンソールを開きます https://console.aws.amazon.com/lookoutvision/。
8. [開始する] を選択します。
9. 左側のナビゲーションペインで、[プロジェクト] を選択します。
10. マニフェストファイルで使用するために追加するプロジェクトを選択します。
11. [操作方法] セクションで、[データセットを作成] を選択します。

12. [単一データセット] タブまたは [トレーニングデータセットとテストデータセットを分離] タブをクリックし、手順に従います。

    Single dataset

    1. [単一データセットを作成] をクリックします。
    2. 「イメージソース設定」セクションで、「Ground Truth で SageMakerラベル付けされたイメージのインポート」を選択します。
    3. [.manifest ファイルの場所] には、マニフェストファイルの場所を入力します。

    Separate training and test datasets

    1. [トレーニングデータセットとテストデータセットを作成] を選択します。
    2. 「トレーニングデータセットの詳細」セクションで、「Ground Truth で SageMakerラベル付けされた画像のインポート」を選択します。
    3. [.manifest ファイルの場所] には、トレーニングマニフェストファイルの場所を入力します。
    4. 「データセットの詳細をテストする」セクションで、 SageMaker 「Ground Truth でラベル付けされた画像のインポート」を選択します。
    5. [.manifest ファイルの場所] には、テストマニフェストファイルの場所を入力します。

    > **Note**
    >
    > トレーニングデータセットとテストデータセットは、異なる画像ソースを持つことができます。

13. [送信] を選択します。
14. 「モデルのトレーニング」の手順に従って、モデルをトレーニングします。

Amazon Lookout for Vision は Amazon S3 バケット `datasets` フォルダにデータセットを作成します。オリジナル `.manifest` ファイルは変更されないままです。

マニフェストファイルを使用したデータセットの作成 (SDK)

CreateDataset オペレーションを使用して、Amazon Lookout for Vision プロジェクトに関連付けられたデータセットを作成します。

単一データセットをトレーニングとテストに使用したい場合は、DatasetType 値を train に設定して単一データセットを作成します。トレーニング中、データセットは内部的に分割され、トレーニングデータセットとテストデータセットが作成されます。分割されたトレーニングデータセットとテストデータセットにはアクセスできません。別のテストデータセットが必要な場合は、DatasetType 値を test に設定して CreateDataset を2回呼び出します。トレーニング中、トレーニングデータセットとテストデータセットは、モデルのトレーニングとテストに使用されます。

オプションで、 DatasetSourceパラメータを使用して、データセットの入力に使用される SageMaker Ground Truth 形式のマニフェストファイルの場所を指定できます。この場合、CreateDataset への呼び出しは非同期です。現在のステータスを確認するには、DescribeDataset を呼び出します。詳細については、「データセットの表示」を参照してください。インポート中に検証エラーが発生した場合、 の値は CREATE_FAILED Statusに設定され、ステータスメッセージ (StatusMessage) が設定されます。

> **Tip**
>
> 開始用サンプルデータセットを使用してデータセットを作成する場合は、ステップ1: マニフェストファイルとアップロード画像を作成する でスクリプトが作成するマニフェストファイル (getting-started/dataset-files/manifests/train.manifest) を使用してください。
> 回路基板サンプル画像によりデータセットを作成する場合は、2 つのオプションがあります:
>
> 1. コードを使用してマニフェストファイルを作成します。Amazon Lookout for Vision Lab Python Notebook は、回路基板サンプル画像のマニフェストファイルを作成する方法を示しています。または、コード例リポジトリのデータセットサンプル AWS コードを使用します。
> 2. すでに Amazon Lookout for Vision コンソールを使って回路基板サンプル画像でデータセットを作成した場合は、Amazon Lookout for Vision で作成されたマニフェストファイルを再利用してください。トレーニングおよびテストマニフェストファイルの場所は s3://*bucket*/datasets/*project name*/*train or test*/manifests/output/output.manifest です。

DatasetSource 指定しない場合では、空のデータセットが作成されます。この場合、CreateDataset への呼び出しは同期します。後で、 を呼び出すことで、データセットに画像にラベルを付けることができますUpdateDatasetEntries。サンプルコードについては、「画像の追加 (SDK)」を参照してください。

データセットを置き換える場合は、まず既存のデータセットを で削除してからDeleteDataset、 を呼び出して同じデータセットタイプの新しいデータセットを作成しますCreateDataset。詳細については、「データセットの削除」を参照してください。

データセットを作成すると、モデルを作成できます。詳細については、「モデルのトレーニング (SDK) 」を参照してください。

を呼び出すことで、データセット内のラベル付きイメージ (JSON 行) を表示できますListDatasetEntries。ラベル付き画像を追加するには、UpdateDatasetEntries を呼び出します。

テストデータセットとトレーニングデータセットに関する情報を表示するには、「データセットの表示」を参照してください。

データセットを作成するには (SDK)

1. まだインストールしていない場合は、 と をインストール AWS CLI して設定します AWS SDKs。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のサンプルコードを使用して、データセットを作成します。

   CLI

   以下の値を変更します:

   - project-name をデータセットと関連付けるプロジェクトの名前に。
   - dataset-type を作成したいデータセットのタイプ (train または test) に。
   - dataset-source をマニフェストファイルの Amazon S3 の保存場所に設定します。
   - Bucket をマニフェストファイルを含む Amazon S3 バケットの名前に。
   - Key を Amazon S3 バケット内のマニフェストファイルのパスとファイル名に。

```
aws lookoutvision create-dataset --project-name project\
  --dataset-type train or test\
  --dataset-source '{ "GroundTruthManifest": { "S3Object": { "Bucket": "bucket",
 "Key": "manifest file" } } }' \
  --profile lookoutvision-access
```

Python

このコードは、 AWS ドキュメントSDKサンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
    @staticmethod
    def create_dataset(lookoutvision_client, project_name, manifest_file,
 dataset_type):
        """
        Creates a new Lookout for Vision dataset

        :param lookoutvision_client: A Lookout for Vision Boto3 client.
        :param project_name: The name of the project in which you want to
                             create a dataset.
        :param bucket: The bucket that contains the manifest file.
        :param manifest_file: The path and name of the manifest file.
        :param dataset_type: The type of the dataset (train or test).
        """
        try:
            bucket, key = manifest_file.replace("s3://", "").split("/", 1)
            logger.info("Creating %s dataset type...", dataset_type)
            dataset = {
                "GroundTruthManifest": {"S3Object": {"Bucket": bucket, "Key":
 key}}
            }
            response = lookoutvision_client.create_dataset(
                ProjectName=project_name,
                DatasetType=dataset_type,
                DatasetSource=dataset,
            )
            logger.info("Dataset Status: %s", response["DatasetMetadata"]
["Status"])
            logger.info(
                "Dataset Status Message: %s",
                response["DatasetMetadata"]["StatusMessage"],
            )
            logger.info("Dataset Type: %s", response["DatasetMetadata"]
["DatasetType"])

            # Wait until either created or failed.
            finished = False
            status = ""
            dataset_description = {}
```

```
            while finished is False:
                dataset_description = lookoutvision_client.describe_dataset(
                    ProjectName=project_name, DatasetType=dataset_type
                )
                status = dataset_description["DatasetDescription"]["Status"]

                if status == "CREATE_IN_PROGRESS":
                    logger.info("Dataset creation in progress...")
                    time.sleep(2)
                elif status == "CREATE_COMPLETE":
                    logger.info("Dataset created.")
                    finished = True
                else:
                    logger.info(
                        "Dataset creation failed: %s",
                        dataset_description["DatasetDescription"]
["StatusMessage"],
                    )
                    finished = True

            if status != "CREATE_COMPLETE":
                message = dataset_description["DatasetDescription"]
["StatusMessage"]
                logger.exception("Couldn't create dataset: %s", message)
                raise Exception(f"Couldn't create dataset: {message}")

        except ClientError:
            logger.exception("Service error: Couldn't create dataset.")
            raise
```

Java V2

このコードは、 AWS ドキュメントSDKサンプル GitHub リポジトリから取得されます。詳しい例はこちらで参照できます。

```
/**
 * Creates an Amazon Lookout for Vision dataset from a manifest file.
 * Returns after Lookout for Vision creates the dataset.
 *
 * @param lfvClient    An Amazon Lookout for Vision client.
 * @param projectName  The name of the project in which you want to create a
 *                     dataset.
```

```
 * @param datasetType  The type of dataset that you want to create (train or
 *                     test).
 * @param bucket       The S3 bucket that contains the manifest file.
 * @param manifestFile The name and location of the manifest file within the S3
 *                     bucket.
 * @return DatasetDescription The description of the created dataset.
 */
public static DatasetDescription createDataset(LookoutVisionClient lfvClient,
                String projectName,
                String datasetType,
                String bucket,
                String manifestFile)
                throws LookoutVisionException, InterruptedException {

        logger.log(Level.INFO, "Creating {0} dataset for project {1}",
                        new Object[] { projectName, datasetType });

        // Build the request. If no bucket supplied, setup for empty dataset
 creation.
        CreateDatasetRequest createDatasetRequest = null;

        if (bucket != null && manifestFile != null) {

                InputS3Object s3Object = InputS3Object.builder()
                                .bucket(bucket)
                                .key(manifestFile)
                                .build();

                DatasetGroundTruthManifest groundTruthManifest =
 DatasetGroundTruthManifest.builder()
                                .s3Object(s3Object)
                                .build();

                DatasetSource datasetSource = DatasetSource.builder()
                                .groundTruthManifest(groundTruthManifest)
                                .build();

                createDatasetRequest = CreateDatasetRequest.builder()
                                .projectName(projectName)
                                .datasetType(datasetType)
                                .datasetSource(datasetSource)
                                .build();
        } else {
                createDatasetRequest = CreateDatasetRequest.builder()
```

```
                                .projectName(projectName)
                                .datasetType(datasetType)
                                .build();
        }

        lfvClient.createDataset(createDatasetRequest);

        DatasetDescription datasetDescription = null;

        boolean finished = false;

        // Wait until dataset is created, or failure occurs.
        while (!finished) {

                datasetDescription = describeDataset(lfvClient, projectName,
datasetType);

                switch (datasetDescription.status()) {
                        case CREATE_COMPLETE:
                                logger.log(Level.INFO, "{0}dataset created for
project {1}",
                                                new Object[] { datasetType,
projectName });
                                finished = true;
                                break;
                        case CREATE_IN_PROGRESS:
                                logger.log(Level.INFO, "{0} dataset creating for
project {1}",
                                                new Object[] { datasetType,
projectName });

                                TimeUnit.SECONDS.sleep(5);

                                break;

                        case CREATE_FAILED:
                                logger.log(Level.SEVERE,
                                                "{0} dataset creation failed for
project {1}. Error {2}",
                                                new Object[] { datasetType,
projectName,

datasetDescription.statusAsString() });
                                finished = true;
```

```
                                break;
                        default:
                                logger.log(Level.SEVERE, "{0} error when
creating {1} dataset for project {2}",
                                                new Object[] { datasetType,
projectName,

datasetDescription.statusAsString() });
                                finished = true;
                                break;

                }
        }

        logger.log(Level.INFO, "Dataset info. Status: {0}\n Message: {1} }",
                        new Object[] { datasetDescription.statusAsString(),
                                        datasetDescription.statusMessage() });

        return datasetDescription;

}
```

3. モデルのトレーニング (SDK) の手順に従ってモデルをトレーニングします。

画像のラベリング

Amazon Lookout for Vision コンソールを使用して、データセット内の画像に割り当てられるラベルを追加したり変更したりできます。を使用している場合SDK、ラベルは に提供するマニフェストファイルの一部ですCreateDataset。を呼び出すことで、イメージのラベルを更新できますUpdateDatasetEntries。サンプルコードについては、「画像の追加 (SDK)」を参照してください。

モデルタイプの選択

画像に割り当てるラベルによって、Lookout for Vision が作成するモデルのタイプが決まります。プロジェクトに別のテストデータセットがある場合は、同じ方法で画像にラベルを付けます。

画像分類モデル

画像分類モデルを作成するには、各画像を正常または異常として分類します。各画像に対して、画像の分類 (コンソール) を行います。

画像セグメンテーションモデル

画像セグメンテーションモデルを作成するには、各画像を正常または異常として分類します。それぞれの異常画像において、画像上の各異常領域にピクセルマスクを指定するとともに、ピクセルマスク内の異常のタイプを示す異常ラベルも指定します。たとえば、次の画像の青いマスクは、異常となる自動車の傷の位置を示しています。画像には、複数タイプの異常ラベルを指定できます。各画像に対して、画像のセグメント化 (コンソール) を行ってください。

画像の分類 (コンソール)

Lookout for Vision コンソールを使用して、データセット内の画像を正常または異常として分類できます。分類されていない画像は、モデルのトレーニングに使用されません。

画像セグメンテーションモデルを作成する場合は、この手続きを飛ばして、画像を分類するステップを含む 画像のセグメント化 (コンソール) を行ってください。

> **Note**
>
> データセットの作成 を完了した場合、コンソールにはモデルダッシュボードが表示されているはずであり、ステップ 1〜4 を行う必要はありません。

画像を分類するには (コンソール)

1. で Amazon Lookout for Vision コンソールを開きます https://console.aws.amazon.com/lookoutvision/。
2. 左ナビゲーションペインで、[プロジェクト] を選択します。
3. 「プロジェクト」 ページで、削除するプロジェクトを選択します。
4. プロジェクトの左ナビゲーションペインで、[データセット] を選択します。

5. 個別のトレーニングデータセットとテストデータセットがある場合は、使用するデータセットのタブを選択します。
6. [ラベリングを開始] を選択します。
7. Choose Select all images on this page.
8. 画像が正常の場合は、[正常として分類] を選択します。そうでない場合は [異常として分類] を選択します。各画像の下にラベルが表示されます。
9. 画像のラベルを変更する必要がある場合は、次の操作を行います。

   a. 画像の下にある [異常] または [正常] を選択します。

   b. 画像の正しいラベルを特定できない場合は、ギャラリーで画像を選択して画像を拡大してください。

> **Note**
>
> [フィルタ] セクションで目的のラベル、またはラベルの状態を選択することで、画像ラベルをフィルターすることができます。

10. データセットの全画像が正しくラベル付けされるまで、必要に応じて各ページで手順 7 ~ 9 を繰り返します。
11. [変更を保存] をクリックします。
12. 画像のラベリングが完了したら、モデルをトレーニングできます。

画像のセグメント化 (コンソール)

画像セグメンテーションモデルを作成する場合は、画像を「正常」または「異常」に分類する必要があります。また、異常のある画像にはセグメンテーション情報を追加する必要があります。セグメンテーション情報を指定するには、まず、へこみや傷など、モデルに検出させたい異常のタイプごとに異常ラベルを指定します。次に、データセット内の異常画像の各異常について、異常マスクと異常ラベルを指定します。

> **Note**
>
> 画像分類モデルを作成する場合は、画像をセグメント化する必要や異常ラベルを指定する必要はありません。

トピック

異常ラベルの指定

データセット画像に含まれる異常のタイプごとに異常ラベルを定義します。

異常ラベルを指定する

1. で Amazon Lookout for Vision コンソールを開きます https://console.aws.amazon.com/lookoutvision/。
2. 左ナビゲーションペインで、[プロジェクト] を選択します。
3. 「プロジェクト」 ページで、削除するプロジェクトを選択します。
4. プロジェクトの左ナビゲーションペインで、[データセット] を選択します。
5. 「異常ラベル」で「異常ラベルを追加」を選択します。以前に異常ラベルを追加したことがある場合は、[管理] を選択します。
6. ダイアログボックスで、次の操作を実行します。
    a. 追加する異常ラベルを入力し、[異常ラベルを追加] を選択します。
    b. モデルに検出させたい異常ラベルをすべて入力するまで、前のステップを繰り返します。
    c. (オプション) ラベル名を変更する編集アイコンを選びます。
    d. (オプション) 削除アイコンを選んで、新しい異常ラベルを削除します。データセットが現在使用している異常タイプは削除できません。
7. [確定] を選択して、新しい異常ラベルをデータセットに追加します。

異常ラベルを指定したら、画像へのラベリング を行って画像にラベルを付けます。

画像へのラベリング

画像セグメンテーション用に画像にラベルを付けるには、画像を正常または異常として分類します。次に、注釈ツールを使用して、画像に存在する各タイプの異常の領域をしっかりと覆うマスクを描画して、画像をセグメント化します。

画像にラベルを付けるには

1. 個別のトレーニングデータセットとテストデータセットがある場合は、使用するデータセットのタブを選択します。
2. まだ行っていない場合は、異常ラベルの指定 を行ってデータセットにおいて異常タイプを指定します。
3. [ラベリングを開始] を選択します。
4. Choose Select all images on this page.
5. 画像が正常な場合は [正常として分類] を選び、それ以外の場合は [異常として分類] を選びます。
6. 単一の画像でラベルを変更するには、画像で [正常] か [異常] を選びます。

> **Note**
>
> You can filter image labels by choosing the desired label, or label state, in the Filters section. 分類オプションセクションの信頼性スコアで分類できます。

7. 異常のある画像ごとに、画像を選択して注釈ツールを開きます。注釈ツールによる画像のセグメント化 を行ってセグメンテーション情報を追加します。
8. Choose Save changes.
9. 画像のラベリングが完了したら、モデルをトレーニングできます。

注釈ツールによる画像のセグメント化

注釈ツールを使用して、異常な領域をマスクでマークして画像をセグメント化します。

注釈ツールで画像をセグメント化するには

1. データセットギャラリーで画像を選択して、注釈ツールを開きます。必要な場合は、[ラベリングを開始] を選んでラベリングモードに入ります。
2. [異常ラベル] セクションではマークしたい異常ラベルを選びます。必要な場合は、[異常ラベルを追加] を選んで新しい異常ラベルを追加します。
3. ページ下部にある描画ツールを選択し、異常領域をしっかりと覆います。次のコード例は、異常領域をしっかりと覆うマスクの例を示しています。

次のコード例は、異常領域をしっかりと覆っていないマスクの例を示しています。

4. セグメント化する画像が多数ある場合は、[次へ] を選んでステップ 2 と 3 を繰り返します。
5. [送信して閉じる] を選んで画像のセグメント化を完了します。

モデルのトレーニング

データセットを作成して画像にラベル付けしたら、モデルをトレーニングできます。トレーニングプロセスの一環として、テストデータセットが使用されます。単一データセットプロジェクトがある場合、データセット内の画像は、トレーニングプロセスの一部として、テストデータセットとトレーニングデータセットに自動的に分割されます。プロジェクトにトレーニングデータセットとテストデータセットがある場合は、データセットを個別にトレーニングおよびテストするために使用されます。

トレーニングが完了したら、モデルのパフォーマンスを評価し、必要な改善を行うことができます。詳細については、「Amazon Lookout for Vision モデルの改善」を参照してください。

モデルをトレーニングするために、Amazon Lookout for Vision はソーストレーニングとテスト画像のコピーを作成します。デフォルトでは、コピーされたイメージは、 がAWS所有および管理するキーで暗号化されます。独自の Key Management Service (KMS) AWS キーを使用することもできます。詳細については、AWS「 Key Management Service の概念」を参照してください。ソース画像には影響がありません。

モデルには、タグという形でメタデータを付与することができます。詳細については、「モデルのタグ付け」を参照してください。

モデルをトレーニングするたびに、モデルの新しいバージョンが作成されます。モデルのバージョンが不要になった場合には、それを削除することができます。詳細については、「モデルの削除」を参照してください。

モデルを正常にトレーニングするのにかかる時間に対して課金されます。詳細については、「トレーニング時間」を参照してください。

プロジェクト内の既存のモデルを表示するには、「モデルの表示」に進んでください。

> **Note**
>
> 「データセットの作成」または「データセットへの画像の追加」を完了したばかりの場合、コンソールにはモデルダッシュボードが表示されるはずです。手順 1～4 を行う必要はありません。

トピック

- モデルのトレーニング (コンソール)
- モデルのトレーニング (SDK)

モデルのトレーニング (コンソール)

次の手続きでは、 コンソールを使用したモデルのトレーニング方法を示します。

モデルをトレーニングするには (コンソール)

1. で Amazon Lookout for Vision コンソールを開きます https://console.aws.amazon.com/lookoutvision/。
2. In the left navigation pane, choose Projects.
3. 「プロジェクト」 ページで、トレーニングしたいモデルを含むプロジェクトを選択します。
4. On the project details page, choose Train model. The Train model button is available if you have enough labeled images to train the model. ボタンがない場合は、十分なラベル付き画像が揃うまで画像を追加してください。
5. (オプション) 独自のAWSKMS暗号化キーを使用する場合は、次の操作を行います。
   a. In Image data encryption choose Customize encryption settings (advanced).
   b. encryption.aws_kms_key で、キーの Amazon リソースネーム (ARN) を入力するか、既存のAWSKMSキーを選択します。新しいキーを作成するには、AWSIMSキーの作成 を選択します。
6. (オプション) モデルにタグを追加する場合は、次の操作を行います。
   a. In the Tags section, choose Add new tag.
   b. 次のように入力します。
      i. The name of the key in Key.
      ii. The value of the key in Value.
   c. タグをさらに追加するには、手順 6a と 6b を繰り返します。
   d. (Optional) If you want to remove a tag, choose Remove next to the tag that you want to remove. 以前に保存したタグを削除する場合、変更を保存するとそのタグが削除されます。
7. Choose Train model.
8. In the Do you want to train your model? dialog box, choose Train model.
9. In the Models view, you can see that training has started and you can check the current status by viewing the `Status` column for the model version. モデルのトレーニングは、完了までに時間がかかります。

10. トレーニングが終了したら、そのパフォーマンスを評価できます。詳細については、「Amazon Lookout for Vision モデルの改善」を参照してください。

モデルのトレーニング (SDK)

CreateModel オペレーションを使用して、モデルのトレーニング、テスト、評価を開始します。Amazon Lookout for Vision は、プロジェクトに関連付けられたトレーニングおよびテストデータセットを使用してモデルをトレーニングします。詳細については、「プロジェクトの作成 (SDK)」を参照してください。

CreateModel を呼び出すたびに、モデルの新しいバージョンが作成されます。CreateModel からの応答には、モデルのバージョンが含まれます。

モデルトレーニングが成功するごとに課金されます。ClientToken 入力パラメータを使用することで、ユーザーによる不要なモデルトレーニングの繰り返しによる課金を防止することができます。ClientToken は冪等の入力パラメータで、CreateModel が特定のパラメータセットに対して一度だけ完了するようにします。– 同じ ClientToken 値で CreateModel を繰り返し呼び出すと、トレーニングが繰り返されないようにします。に値を指定しない場合ClientToken、使用している AWSSDKによって値が挿入されます。これにより、ネットワークエラー後の再試行が複数のトレーニングジョブを開始することを防ぎますが、ご自身のユースケースに合わせて独自の値を用意する必要があります。詳細については、「」を参照してくださいCreateModel。

トレーニングが完了するまで少し時間がかかります。現在の状態を確認するには、DescribeModel を呼び出し、プロジェクト名 (CreateProject への呼び出しで指定) とモデルのバージョンを渡します。status フィールドは、モデルトレーニングの現在のステータスを示します。サンプルコードについては、「モデルの表示 (SDK)」を参照してください。

トレーニングが成功すれば、モデルを評価できます。詳細については、「Amazon Lookout for Vision モデルの改善」を参照してください。

プロジェクトで作成したモデルを表示するには、ListModels を呼び出します。サンプルコードについては、「モデルの表示」を参照してください。

モデルをトレーニングするには (SDK)

1. まだインストールしていない場合は、 と をインストール AWS CLI して設定します AWS SDKs。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のサンプルコードを使用して、モデルをトレーニングします。

CLI

以下の値を変更します:

- project-name を、作成したいモデルを含むプロジェクト名に。
- output-config を、トレーニング結果を保存したい場所に。以下の値を置き換えます:
  - output bucket を、Amazon Lookout for Vision がトレーニング結果を保存する Amazon S3 バケットの名前に。
  - output folder を、トレーニング結果を保存したいフォルダの名前に。
  - Key を、タグキーの名前に。
  - Value を、tag_key と関連する値に。

```
aws lookoutvision create-model --project-name "project name"\
  --output-config '{ "S3Location": { "Bucket": "output bucket", "Prefix":
 "output folder" } }'\
  --tags '[{"Key":"Key","Value":"Value"}]' \
  --profile lookoutvision-access
```

Python

このコードは、 AWS ドキュメントSDKサンプル GitHub リポジトリから取得されます。詳しい事例はこちらです。

```
    @staticmethod
    def create_model(
        lookoutvision_client,
        project_name,
        training_results,
        tag_key=None,
        tag_key_value=None,
    ):
        """
        Creates a version of a Lookout for Vision model.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project in which you want to create
 a
                             model.
```

```
        :param training_results: The Amazon S3 location where training results
 are stored.
        :param tag_key: The key for a tag to add to the model.
        :param tag_key_value - A value associated with the tag_key.
        return: The model status and version.
        """
        try:
            logger.info("Training model...")
            output_bucket, output_folder = training_results.replace("s3://",
 "").split(
                "/", 1
            )
            output_config = {
                "S3Location": {"Bucket": output_bucket, "Prefix": output_folder}
            }
            tags = []
            if tag_key is not None:
                tags = [{"Key": tag_key, "Value": tag_key_value}]

            response = lookoutvision_client.create_model(
                ProjectName=project_name, OutputConfig=output_config, Tags=tags
            )

            logger.info("ARN: %s", response["ModelMetadata"]["ModelArn"])
            logger.info("Version: %s", response["ModelMetadata"]
["ModelVersion"])
            logger.info("Started training...")

            print("Training started. Training might take several hours to
 complete.")

            # Wait until training completes.
            finished = False
            status = "UNKNOWN"
            while finished is False:
                model_description = lookoutvision_client.describe_model(
                    ProjectName=project_name,
                    ModelVersion=response["ModelMetadata"]["ModelVersion"],
                )
                status = model_description["ModelDescription"]["Status"]

                if status == "TRAINING":
                    logger.info("Model training in progress...")
                    time.sleep(600)
```

```
                    continue

                if status == "TRAINED":
                    logger.info("Model was successfully trained.")
                else:
                    logger.info(
                        "Model training failed: %s ",
                        model_description["ModelDescription"]["StatusMessage"],
                    )
                finished = True
        except ClientError:
            logger.exception("Couldn't train model.")
            raise
        else:
            return status, response["ModelMetadata"]["ModelVersion"]
```

Java V2

このコードは、 AWS ドキュメントSDKサンプル GitHub リポジトリから取得されます。詳しい例はこちらで参照できます。

```
/**
 * Creates an Amazon Lookout for Vision model. The function returns after model
 * training completes. Model training can take multiple hours to complete.
 * You are charged for the amount of time it takes to successfully train a
 model.
 * Returns after Lookout for Vision creates the dataset.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project in which you want to create a
 *                    model.
 * @param description A description for the model.
 * @param bucket      The S3 bucket in which Lookout for Vision stores the
 *                    training results.
 * @param folder      The location of the training results within the S3
 *                    bucket.
 * @return ModelDescription The description of the created model.
 */
public static ModelDescription createModel(LookoutVisionClient lfvClient, String
 projectName,
                String description, String bucket, String folder)
                throws LookoutVisionException, InterruptedException {
```

```
        logger.log(Level.INFO, "Creating model for project: {0}.", new Object[]
{ projectName });

        // Setup input parameters.
        S3Location s3Location = S3Location.builder()
                        .bucket(bucket)
                        .prefix(folder)
                        .build();

        OutputConfig config = OutputConfig.builder()
                        .s3Location(s3Location)
                        .build();

        CreateModelRequest createModelRequest = CreateModelRequest.builder()
                        .projectName(projectName)
                        .description(description)
                        .outputConfig(config)
                        .build();

        // Create and train the model.
        CreateModelResponse response =
lfvClient.createModel(createModelRequest);

        String modelVersion = response.modelMetadata().modelVersion();
        boolean finished = false;
        DescribeModelResponse descriptionResponse = null;

        // Wait until training finishes or fails.

        do {
                DescribeModelRequest describeModelRequest =
DescribeModelRequest.builder()
                                .projectName(projectName)
                                .modelVersion(modelVersion)
                                .build();

                descriptionResponse =
lfvClient.describeModel(describeModelRequest);

                switch (descriptionResponse.modelDescription().status()) {
                        case TRAINED:
                                logger.log(Level.INFO, "Model training completed
for project {0} version {1}.",
```

```
                                                    new Object[] { projectName,
modelVersion });
                                finished = true;
                                break;

                        case TRAINING:
                                logger.log(Level.INFO,
                                                "Model training in progress for
project {0} version {1}.",
                                                new Object[] { projectName,
modelVersion });
                                TimeUnit.SECONDS.sleep(60);

                                break;

                        case TRAINING_FAILED:
                                logger.log(Level.SEVERE,
                                                "Model training failed for for
project {0} version {1}.",
                                                new Object[] { projectName,
modelVersion });
                                finished = true;
                                break;

                        default:
                                logger.log(Level.SEVERE,
                                                "Unexpected error when training
model project {0} version {1}: {2}.",
                                                new Object[] { projectName,
modelVersion,

descriptionResponse.modelDescription()

.status() });
                                finished = true;
                                break;

                }
        } while (!finished);

        return descriptionResponse.modelDescription();

}
```

3. トレーニングが終了したら、そのパフォーマンスを評価できます。詳細については、「Amazon Lookout for Vision モデルの改善」を参照してください。

モデルトレーニングのトラブルシューティング

マニフェストファイルやトレーニング画像に問題があると、モデルトレーニングが失敗する場合があります。モデルを再トレーニングする前に、発生する可能性がある以下の問題をチェックしてください。

異常ラベルの色がマスク画像の異常の色と一致しない

画像セグメンテーションモデルをトレーニングする場合、マニフェストファイル内の異常ラベルの色はマスク画像の色と一致しなければなりません。マニフェストファイル内のイメージのJSON行には、どの色が異常ラベルに対応するかを Amazon Lookout for Vision に伝えるメタデータ (`internal-color-map`) があります。例えば、次のJSON行の`scratch`異常ラベルの色は です`#2ca02c`。

```
{
    "source-ref": "s3://path-to-image",
    "anomaly-label": 1,
    "anomaly-label-metadata": {
        "class-name": "anomaly",
        "creation-date": "2021-10-12T14:16:45.668",
        "human-annotated": "yes",
        "job-name": "labeling-job/classification-job",
        "type": "groundtruth/image-classification",
        "confidence": 1
    },
    "anomaly-mask-ref": "s3://path-to-image",
    "anomaly-mask-ref-metadata": {
        "internal-color-map": {
            "0": {
                "class-name": "BACKGROUND",
                "hex-color": "#ffffff",
                "confidence": 0.0
            },
            "1": {
                "class-name": "scratch",
                "hex-color": "#2ca02c",
                "confidence": 0.0
```

```
            },
            "2": {
                "class-name": "dent",
                "hex-color": "#1f77b4",
                "confidence": 0.0
            }
        },
        "type": "groundtruth/semantic-segmentation",
        "human-annotated": "yes",
        "creation-date": "2021-11-23T20:31:57.758889",
        "job-name": "labeling-job/segmentation-job"
    }
}
```

マスク画像の色が hex-color 内の値と一致しない場合、トレーニングは失敗し、マニフェストファイルを更新する必要があります。

マニフェストファイル内のカラー値を更新するには

1. テキストエディタを使って、データセットを作成したマニフェストファイルを開きます。
2. JSON 行 (イメージ) ごとに、internal-color-mapフィールド内の色 (hex-color) がマスクイメージ内の異常ラベルの色と一致することを確認します。

   マスク画像の位置は *anomaly-mask*-ref フィールドから取得できます。画像をコンピューターにダウンロードし、次のコードを使用して画像内の色を取得します。

   ```
   from PIL import Image
   img = Image.open('path to local copy of mask file')
   colors = img.convert('RGB').getcolors() #this converts the mode to RGB
   for color in colors:
       print('#%02x%02x%02x' % color[1])
   ```

3. 色割り当てが正しくないイメージごとに、イメージのJSON行の hex-colorフィールドを更新します。
4. 更新されたマニフェストファイルを保存します。
5. 既存のデータセットをプロジェクトから削除します。
6. 更新されたマニフェストファイルにより、プロジェクトに新しいデータセットを作成します。

7. モデルをトレーニングします。

または、ステップ 5 と 6 では、 UpdateDatasetEntriesオペレーションを呼び出し、更新するイメージの更新JSON行を指定することで、データセット内の個々のイメージを更新できます。サンプルコードについては、「画像の追加 (SDK)」を参照してください。

マスク画像が PNG 形式ではない

画像セグメンテーションモデルをトレーニングする場合、マスク画像は PNG 形式である必要があります。マニフェストファイルからデータセットを作成する場合は、 で参照するマスクイメージ*anomaly-mask*-refが PNG形式であることを確認します。マスクイメージが PNG 形式でない場合は、 PNG形式に変換する必要があります。画像ファイルの拡張子を .png に変更するだけでは不十分です。

Amazon Lookout for Vision コンソールまたは SageMaker Ground Truth ジョブで作成したマスクイメージは、 PNG形式で作成されます。これらの画像の形式を変更する必要はありません。

マニフェストファイル内の非PNGフォーマットマスクイメージを修正するには

1. テキストエディタを使って、データセットを作成したマニフェストファイルを開きます。
2. JSON 行 (イメージ) ごとに、 がPNGフォーマットイメージ*anomaly-mask*-refを参照していることを確認してください。詳細については、「マニフェストファイルの作成」を参照してください。
3. 更新されたマニフェストファイルを保存します。
4. 既存のデータセットをプロジェクトから削除します。
5. 更新されたマニフェストファイルにより、プロジェクトに新しいデータセットを作成します。
6. モデルをトレーニングします。

セグメンテーションラベルまたは分類ラベルが不正確または欠落している

ラベルが欠けていたり、不正確だったりすると、トレーニングが失敗したり、モデルのパフォーマンスが低下したりする可能性があります。データセット内のすべての画像にラベルを付けることをお勧めします。すべての画像にラベルを付けず、モデルトレーニングが失敗したり、モデルのパフォーマンスが悪い場合は、さらに画像を追加してください。

以下をチェックしてください:

- セグメンテーションモデルを作成する場合、マスクはデータセット画像での異常をしっかりと力バーする必要があります。データセット内のマスクをチェックするには、プロジェクトのデータセットギャラリーで画像を表示します。必要に応じて、画像マスクを再描画します。詳細については、「画像のセグメント化 (コンソール)」を参照してください。

- データセット画像に含まれる異常画像が分類されていることを確かめてください。画像セグメンテーションモデルを作成する場合は、異常画像に異常ラベルと画像マスクが付いていることを確かめてください。

  作成するモデルのタイプ (セグメンテーションまたは分類) を覚えておくことは重要です。分類モデルでは、異常画像の画像マスクは必要ありません。分類モデル用のデータセット画像にはマスクを追加しないでください。

  見つからないラベルを更新するには

  1. プロジェクトのデータセットギャラリーを開きます。
  2. ラベルなしの画像をフィルターして、ラベルなしの画像を確認します。
  3. 次のいずれかを行います:
     - 画像分類モデルを作成する場合は、ラベルが付いていない各画像を分類します。
     - 画像セグメンテーションモデルを作成する場合は、ラベルなしの各画像を分類してセグメント化します。
  4. 画像セグメンテーションモデルを作成する場合は、分類された異常画像でマスクが欠落しているものすべてにマスクを追加します。
  5. モデルをトレーニングします。

不適切なラベルや欠落しているラベルを修正しない場合は、ラベル付けされた画像をさらに追加するか、影響を受ける画像をデータセットから削除することをお勧めします。コンソールから、または UpdateDatasetEntriesオペレーションを使用して、さらに追加できます。詳細については、「データセットへの画像の追加」を参照してください。

画像を削除する場合は、データセットから画像を削除できないため、影響を受けた画像を含まないデータセットを再作成する必要があります。詳細については、「データセットからの画像の削除」を参照してください。

Amazon Lookout for Vision モデルの改善

トレーニング中、Lookout for Vision はテストデータセットを使用してモデルをテストし、その結果を使用してパフォーマンスメトリクスを作成します。パフォーマンスメトリクスを使用して、モデルのパフォーマンスを評価できます。必要に応じて、データセットを改善し、モデルを再トレーニングする手順を実行できます。

モデルのパフォーマンスに満足している場合は、使用を開始できます。詳細については、「トレーニング済みの Amazon Lookout for Vision モデルの実行」を参照してください。

トピック

ステップ 1: モデルのパフォーマンスを評価する

パフォーマンスメトリクスは、コンソールからも DescribeModel オペレーションからもアクセスできます。Amazon Lookout for Vision は、テストデータセットのパフォーマンスメトリクスのサマリーと、すべての個別画像に対する予測結果を提供します。モデルがセグメンテーションモデルの場合、コンソールには各異常ラベルのサマリーメトリクスも表示されます。

コンソールでパフォーマンスメトリクスとテスト画像予測を表示するには、「パフォーマンスメトリクスの表示 (コンソール)」を参照してください。パフォーマンスメトリクスへのアクセスや、`DescribeModel` オペレーションによるテスト画像の予測については、「パフォーマンスメトリクスの表示 (SDK)」を参照してください。

画像分類指標

Amazon Lookout for Vision では、モデルがテスト中に行う分類について、次のようなサマリーメトリクスを提供します:

- 適合率
- リコール

- F1 スコア

画像セグメンテーションモデルメトリクス

モデルが画像セグメンテーションモデルの場合、Amazon Lookout for Vision は各異常ラベルのサマリー画像分類メトリクスとサマリーパフォーマンスメトリクスを提供します:

- F1 スコア
- アベレージ・インターセクション・オーバー・ユニオン (IoU)

適合率

適合率のメトリクスは、ある画像に異常があるとモデルが予測したとき、その予測はどれくらいの頻度で正しいか、という問いに答えるものです。

適合率は、偽陽性のコストが高い場合に便利なメトリクスです。例えば、組み立てられた機械から欠陥ではない機械部品を取り外すコストなどです。

Amazon Lookout for Vision は、テストデータセット全体の適合率メトリクス値のサマリーを提供します。

適合率は、すべての予測された異常 (真陽性および偽陽性) に対する正しく予測された異常 (真陽性) の割合です。適合率の計算式は次のとおりです。

適合率値 = 真陽性/(真陽性 + 偽陽性)

適合率の範囲は 0～1 です。Amazon Lookout for Vision コンソールでは、適合率がパーセンテージ値 (0～100) として表示されます。

適合率の値が高いほど、予測された異常のうち正しいものが多いことを示します。例えば、モデルで 100 個の画像が異常であると予測するとします。85 個の予測が正しく (真陽性)、15 が誤り (偽陽性) の場合、適合率は次のように計算されます。

85 真陽性/(85 真陽性 + 15 偽陽性) = 0.85 適合率値

ただし、モデルが 100 個の異常予測のうち 40 個の画像しか正しく予測しなかった場合、結果の適合率値は、0.40 (40/(40 + 60) = 0.40) と低い値になります。この場合、モデルは正しい予測よりも間違った予測を多く行っているということになります。これを修正するには、モデルを改善することを検討してください。詳細については、「ステップ 2: モデルを改善する」を参照してください。

詳細については、「適合率と再現率」を参照してください。

リコール

再現率メトリクスは、テストデータセットに含まれる異常画像の総数のうち、何枚が正しく異常画像と予測されたか、という問いに答えるものです。

再現率メトリクスは、偽陰性のコストが高い場合に便利です。例えば、欠陥部品を取り外さないコストが高い場合です。Amazon Lookout for Vision では、テストデータセット全体の再現率メトリクス値のサマリーが提供されます。

再現率は、正しく検出された異常テスト画像の割合です。これは、テストデータセットの画像に実際に異常な画像が存在する場合、モデルが正しく予測できる頻度を示す指標です。再現率の計算式は次のとおりです。

再現率値 = 真陽性/(真陽性 + 偽陰性)

再現率の範囲は 0～1 です。Amazon Lookout for Vision コンソールでは、再現率がパーセント値 (0～100) として表示されます。

再現率値が大きいほど、より多くの異常な画像が正しく識別されることを示します。例えば、テストデータセットに 100 個の異常画像が含まれているとします。モデルが 100 個の異常画像のうち 90 個を正しく検出した場合、再現率は次のようになります。

90 真陽性/(90 真陽性 + 10 偽陰性) = 0.90 再現率値

再現率値 0.90 は、モデルがテストデータセット内のほとんどの異常画像を正しく予測していることを示します。モデルが異常画像を 20 個しか正しく予測しない場合、再現率は 0.20 (20/(20 + 80) = 0.20) と低い値になります。

この場合、モデルの改善を検討する必要があります。詳細については、「ステップ 2: モデルを改善する」を参照してください。

詳細については、「適合率と再現率」を参照してください。

F1 スコア

Amazon Lookout for Vision では、テストデータセットの平均モデルパフォーマンススコアが提供されます。具体的には、異常分類のモデルのパフォーマンスは F1 スコアというメトリクスで測定されます。これは適合率スコアと再現率スコアの調和平均となります。

F1 スコアは、適合率と再現率の両方を考慮した総合的な指標です。モデルのパフォーマンススコアは 0～1 の値をとります。値が大きいほど、再現率と適合率の両方でモデルのパフォーマンスが高い事を意味します。例えば、適合率が 0.9、再現率が 1.0 のモデルの場合、F1 スコアは 0.947 になります。

モデルのパフォーマンスが良好でない場合、例えば、適合率が 0.30 と低いと、再現率が 1.0 と高くても F1 スコアは0.46になります。同様に、適合率が高くとも (0.95)、再現率が低い場合 (0.20)、F1 スコアは0.33 になります。どちらの場合も、F1 スコアは低く、これはモデルに問題があることを示しています。

詳細については、「F1 スコア」を参照してください。

アベレージ・インターセクション・オーバー・ユニオン (IoU)

テスト画像の異常マスクと、モデルがテスト画像について予測する異常マスクとの間の重なりの平均パーセンテージ。Amazon Lookout for Vision は、各異常ラベルの平均 IoU を返します。画像セグメンテーションモデルによってのみ返されます。

低いパーセンテージ値は、モデルがラベルの予測マスクとテスト内のマスクを正確に一致させていないことを示しています。

次の画像は IoU が低くなっています。オレンジ色のマスクはモデルからの予測であり、テスト画像のマスクを表す青いマスクをしっかりと覆っていません。

次の画像は IoU が高くなっています。青いマスク (テスト画像) はオレンジ色のマスク (予測マスク) でしっかりと覆われています。

テスト結果

テスト中、モデルはテストデータセット内の各テスト画像の分類を予測します。各予測の結果は、以下のように対応するテスト画像のラベル (正常または異常) と比較されます。

- 画像が異常であると正しく予測された場合、真陽性とみなされます。
- 画像が異常であると誤って予測された場合、偽陽性とみなされます。
- 画像が正常であると正しく予測された場合、真陰性とみなされます。
- 画像が正常であると誤って予測された場合、偽陰性とみなされます。

モデルがセグメンテーションモデルの場合、モデルはテスト画像上の異常箇所に対するマスクと異常ラベルも予測します。

Amazon Lookout for Vision では、比較の結果を使用してパフォーマンスメトリクスが生成されます。

ステップ 2: モデルを改善する

パフォーマンスメトリクスは、モデルを改善できることを示している場合があります。例えば、モデルがテストデータセット内のすべての異常を検出しない場合、モデルの再現率は低い (再現率メトリクスの値が低い) ことになります。モデルを改善する必要がある場合は、以下の点を考慮します。

- トレーニングデータセット画像とテストデータセット画像に適切なラベルが付けられていることを確認します。
- 照明や物体の外観など、画像の撮影条件のばらつきを抑え、同じ種類の物体でモデルをトレーニングさせます。
- 画像には必要なコンテンツのみが表示されていることを確認してください。例えば、プロジェクトで機械部品の異常が検出された場合は、画像内に他のものがないことを確認してください。

- トレーニングデータセットとテストデータセットにラベル付き画像を追加します。テストデータセットが優れた再現率と適合率を持つにもかかわらず、モデルをデプロイしたときのパフォーマンスが低い場合、テストデータセットの表示が十分でない可能性があり、それを拡張する必要があります。
- テストデータセットの再現率と適合率が低い場合、トレーニングとテストのデータセットにおける異常と画像のキャプチャ条件がどれくらい一致しているかに配慮してください。トレーニング画像が予想される異常や条件を表していないのに、テスト画像内の画像がそうである場合は、画像を予想される異常や条件を含むトレーニングデータセットに画像を追加します。テストデータセットの画像は予想どおりの状態ではないものの、トレーニング画像は予想どおりの状態である場合は、テストデータセットを更新してください。

  詳細については、「画像をさらに追加する」を参照してください。ラベル付けされた画像をトレーニングデータセットに追加する別の方法として、検出タスクを試行し、結果を検証する方法があります。その後、検証済み画像をトレーニングデータセットに追加できます。詳細については、「トライアル検出タスクでモデルを検証する」を参照してください。
- トレーニングおよびテストデータセットに、色々な正常画像と異常画像が十分あることを確認します。画像は、モデルが遭遇する正常画像と異常画像のタイプを表現している必要があります。例えば、回路基板を解析する場合、抵抗やトランジスタなどの部品の位置やハンダ付けのばらつきを正常な画像として表現する必要があります。異常画像は、部品の取り違えや紛失など、システムが遭遇する可能性のあるさまざまな種類の異常を表現している必要があります。
- モデルで、検出された異常タイプの平均 IoU が低い場合は、セグメンテーションモデルからのマスク出力をチェックします。スクラッチなどの一部のユースケースでは、モデルが出力するスクラッチは、テスト画像のグラウンドトゥルーススクラッチに非常に近いものの、ピクセルオーバーラップが低いことがあります。たとえば、1 ピクセル離れた 2 本の平行線のようにです。このような場合、平均 IoU は予測の成功度を測定する指標としては信頼性が低くなります。
- 画像サイズが小さい場合や画像の解像度が低い場合は、より高い解像度で画像をキャプチャすることを検討してください。画像のサイズは、64 x 64 ピクセルから最大 4096 ピクセル x 4096 ピクセルまでさまざまです。
- 異常サイズが小さい場合は、画像を別々のタイルに分割し、そのタイル画像をトレーニングとテストに使用してください。これにより、モデルは画像内で欠陥をより大きなサイズで見ることができます。

トレーニングデータセットとテストデータセットを改善したら、モデルを再トレーニングして再評価します。詳細については、「モデルのトレーニング」を参照してください。

メトリクスによってモデルの性能が許容範囲であることが示された場合、テストデータセットに試行検出タスクの結果を追加することによって、その性能を検証することができます。再トレーニング後、パフォーマンスメトリクスは前回のトレーニングでのパフォーマンスメトリクスを確認する必要があります。詳細については、「トライアル検出タスクでモデルを検証する」を参照してください。

パフォーマンスメトリクスの表示

コンソールから、DescribeModel オペレーションを呼び出すことで、パフォーマンスメトリクスを取得することができます。

トピック

- パフォーマンスメトリクスの表示 (コンソール)
- パフォーマンスメトリクスの表示 (SDK)

パフォーマンスメトリクスの表示 (コンソール)

トレーニングが完了すると、コンソールにパフォーマンスメトリックが表示されます。

Amazon Lookout for Vision コンソールには、次のパフォーマンスメトリクスが表示されます:

- 適合率
- リコール
- F1 スコア

モデルがセグメンテーションモデルの場合、コンソールには各異常ラベルについて以下のパフォーマンス指標も表示されます:

- 異常ラベルが見つかったテスト画像の数。
- F1 スコア
- アベレージ・インターセクション・オーバー・ユニオン (IoU)

テスト結果の概要セクションには、テストデータセット内の画像の正しい予測と誤った予測の合計が表示されます。また、テストデータセットの個々の画像について、予測されたラベルの割り当てと実際のラベルの割り当てを見ることができます。

以下の手順では、プロジェクトのモデルリストビューからパフォーマンスメトリクスを取得する方法を示します。

パフォーマンスメトリクスを表示するには (コンソール)

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [開始する] を選択します。
3. 左側のナビゲーションペインで、[プロジェクト] を選択します。
4. プロジェクトビューで、トレーニングしたいモデルを含むプロジェクトを選択します。
5. 左側のナビゲーションペインで、プロジェクト名の下にある [モデル] を選択します。
6. モデルリストビューで、表示するモデルのバージョンを選択します。
7. モデルの詳細ページで、[パフォーマンスメトリクス] タブでパフォーマンスメトリクスを表示します。
8. 次の点に注意してください。

    a. [モデル性能指標] セクションには、モデルがテスト画像に対して行った分類予測の全体的なモデル指標 (精度、再現率、F1 スコア) が含まれます。

    b. モデルが画像セグメンテーションモデルの場合、[ラベルごとのパフォーマンス] セクションには、異常ラベルが見つかったテスト画像の数が含まれます。また、各異常ラベルの指標 (F1 スコア、平均 IoU) も表示されます。

    c. [テスト結果概要] セクションには、Lookout for Vision がモデルの評価に使用する各テスト画像の結果が表示されます。この情報には以下が含まれます。

        - すべてのテスト画像の正しい (真陽性) 分類予測と正しくない (偽陰性) 分類予測 (正常または異常) の総数。
        - 各テスト画像の分類予測。画像の下に「正しい」と表示されている場合、予測された分類は画像の実際の分類と一致します。そうでなければ、モデルは画像を正しく分類しませんでした。
        - 画像セグメンテーションモデルでは、モデルが画像に割り当てた異常ラベルと、異常ラベルの色と一致する画像上のマスクが表示されます。

パフォーマンスメトリクスの表示 (SDK)

DescribeModel オペレーションを使用すると、モデルに対するサマリーパフォーマンスメトリクス (分類)、評価マニフェスト、モデルに対する評価結果を取得できます。

パフォーマンスメトリクスのサマリーを取得

テスト中にモデルによって行われた分類予測のサマリーパフォーマンスメトリクス (適合率、リコール、および F1 スコア) は、DescribeModel への呼び出しによって返される Performance フィールドで返されます。

```
        "Performance": {
            "F1Score": 0.8,
            "Recall": 0.8,
            "Precision": 0.9
        },
```

Performance フィールドには、セグメンテーションモデルによって返された異常ラベルのパフォーマンスメトリクスは含まれません。それらは EvaluationResult フィールドから取得できます。詳細については、「評価結果の検証」を参照してください。

パフォーマンスメトリクスのサマリーの詳細については、「ステップ 1: モデルのパフォーマンスを評価する」を参照してください。サンプルコードについては、「モデルの表示 (SDK)」を参照してください。

評価マニフェストの使用

評価マニフェストは、モデルのテストに使用される個々の画像のテスト予測メトリクスを提供します。テストデータセット内の各画像について、JSON line には、元のテスト (ground truth) 情報と画像のモデルの予測が含まれます。Amazon Lookout for Vision では、評価マニフェストを Amazon S3 バケットに格納します。DescribeModel オペレーションからのレスポンスにある EvaluationManifest フィールドから位置情報を取得することができます。

```
"EvaluationManifest": {
            "Bucket": "lookoutvision-us-east-1-nnnnnnnnnn",
            "Key": "my-sdk-project-model-output/EvaluationManifest-my-sdk-
project-1.json"
        }
```

ファイル名の形式は、EvaluationManifest-*project name*.json です。サンプルコードについては、「モデルの表示」を参照してください。

以下のJSON line のサンプルでは、class-name が画像の内容のグラウンドトゥルースです。この例では、画像に異常が含まれています。confidence フィールドは、Amazon Lookout for Vision の予測に対する信頼度を示しています。

```
{
    "source-ref"*: "s3://customerbucket/path/to/image.jpg",
    "source-ref-metadata": {
        "creation-date": "2020-05-22T21:33:37.201882"
    },

    // Test dataset ground truth
    "anomaly-label": 1,
    "anomaly-label-metadata": {
        "class-name": "anomaly",
        "type": "groundtruth/image-classification",
        "human-annotated": "yes",
        "creation-date": "2020-05-22T21:33:37.201882",
        "job-name": "labeling-job/anomaly-detection"
    },
    // Anomaly label detected by Lookout for Vision
    "anomaly-label-detected": 1,
    "anomaly-label-detected-metadata": {
        "class-name": "anomaly",
        "confidence": 0.9,
        "type": "groundtruth/image-classification",
        "human-annotated": "no",
        "creation-date": "2020-05-22T21:33:37.201882",
        "job-name": "training-job/anomaly-detection",
        "model-arn": "lookoutvision-some-model-arn",
        "project-name": "lookoutvision-some-project-name",
        "model-version": "lookoutvision-some-model-version"
    }
}
```

評価結果の検証

評価結果は、テスト画像の全セットについて、以下のような総計パフォーマンスメトリクス (分類) を示します:

- 適合率
- リコール
- ROC カーブ (コンソールには表示されません)
- 平均適合率 (コンソールには表示されません)
- F1 スコア

評価結果には、モデルのトレーニングとテストに使用される画像の数も含まれます。

モデルがセグメンテーションモデルの場合、評価結果にはテストデータセットで見つかった各異常ラベルに関する次のメトリクスも含まれます:

- 適合率
- リコール
- F1 スコア
- アベレージ・インターセクション・オーバー・ユニオン (IoU)

Amazon Lookout for Vision では、評価結果を Amazon S3 バケットに格納します。DescribeModel オペレーションからの応答で EvaluationResult の値を確認することで、位置を取得することができます。

```
"EvaluationResult": {
            "Bucket": "lookoutvision-us-east-1-nnnnnnnnnn",
            "Key": "my-sdk-project-model-output/EvaluationResult-my-sdk-project-1.json"
        }
```

ファイル名の形式は、EvaluationResult-*project name*.json です。例については、「モデルの表示」を参照してください。

次のスキーマは、評価結果を示しています。

```
    {
        "Version": 1,
        "EvaluationDetails":
        {
            "ModelArn": "string", // The Amazon Resource Name (ARN) of the model
version.
```

```
            "EvaluationEndTimestamp": "string",   // The UTC date and time that
evaluation finished.
            "NumberOfTrainingImages": int,        // The number of images that were
successfully used for training.
            "NumberOfTestingImages": int          // The number of images that were
successfully used for testing.
        },
        "AggregatedEvaluationResults":
        {
            "Metrics":
            {                       //Classification metrics.
                "ROCAUC": float,            // ROC area under the curve.
                "AveragePrecision": float,  // The average precision of the model.
                "Precision": float,         // The overall precision of the model.
                "Recall": float,            // The overall recall of the model.
                "F1Score": float,           // The overal F1 score for the model.

                "PixelAnomalyClassMetrics":     //Segmentation metrics.
                [
                    {
                        "Precision": float,       // The precision for the anomaly
label.
                        "Recall": float,          // The recall for the anomaly label.
                        "F1Score": float,         // The F1 score for the anomaly
label.
                        "AIOU" : float,           // The average Intersection Over
Union for the anomaly label.
                        "ClassName": "string"   // The anomaly label.
                    }
                ]
            }
        }
    }
```

トライアル検出タスクでモデルを検証する

モデルの品質を検証または改善したい場合は、トライアル検出タスクを実行できます。トライアル検出タスクは、提供された新しい画像の異常を検出します。

検出結果を確認し、検証した画像をデータセットに追加できます。個別のトレーニングデータセットとテストデータセットがある場合、検証済みの画像がトレーニングデータセットに追加されます。

ローカルコンピュータからの画像、または Amazon S3 バケットにある画像を検証できます。検証済み画像をデータセットに追加する場合、S3 バケットに配置された画像は、データセット内の画像と同じ S3 バケット内に存在する必要があります。

> **Note**
>
> トライアル検出タスクを実行するには、S3 バケットでバージョニングが有効になっていることを確認します。詳細については、「バージョニングの使用」を参照してください。コンソールバケットは、バージョニングが有効になっている状態で作成されます。

デフォルトでは、画像は AWS が所有および管理するキーで暗号化されます。また、独自の AWS Key Management Service (KMS) キーを使用することもできます。詳細については、「AWS Key Management Service の概念」を参照してください。

トピック

- トライアル検出タスクの実行
- トライアル検出結果の確認
- 注釈ツールによるセグメンテーションラベルの修正

トライアル検出タスクの実行

トライアル検出タスクを実行するには、次の手順を実行します。

トライアル検出を実行するには (コンソール)

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [開始する] を選択します。
3. 左側のナビゲーションペインで、[プロジェクト] を選択します。
4. プロジェクトビューで、トレーニングしたいモデルを含むプロジェクトを選択します。
5. 左側のナビゲーションペインで、プロジェクト名の下にある [トライアル検出] を選択します。
6. 「トライアル検出」ビューで、[トライアル検出を実行] を選択します。
7. 「トライアル検出を実行」ページで、[タスク名] にトライアル検出タスクの名前を入力します。
8. [モデルを選択] で、使用するそのモデルのバージョンを選択します。

9. 次のように、画像のソースに従って画像をインポートします。

   - Amazon S3 バケットからソース画像をインポートする場合は、「S3 URI」を入力します。

     > **Tip**
     >
     > スタートガイドのサンプル画像を使用している場合は、extra_images フォルダーを使用します。Amazon S3 URI は `s3://your bucket/circuitboard/extra_images` です。

   - コンピュータから画像をアップロードする場合は、[異常を検出] を選択した後に画像を追加してください。

10. (オプション) 独自の AWS KMS 暗号化キーを使用する場合は、次の手順を実行します。

    a. [画像データ暗号化] では、[暗号化設定をカスタマイズ (詳細)] を選択します。

    b. encryption.aws_kms_key で、キーの Amazon リソースネーム (ARN) を入力するか、既存の AWS KMS キーを選択します。新しいキーを作成するには、[AWS IMS キーを作成] を選択します。

11. [異常を検出 を選択してから、[トライアル検出を実行] を選択して、トライアル検出タスクを開始します。

12. トライアル検出 ビューで現在のステータスを確認します。トライアル検出が完了するまでに時間がかかる場合があります。

トライアル検出結果の確認

トライアル検出の結果を検証すると、モデルの改善に役立ちます。

パフォーマンスメトリクスが悪い場合は、トライアル検出を実行してモデルを改善し、検証済みの画像をデータセット (別のデータセットがある場合は、トレーニングデータセット) に追加します。

モデルのパフォーマンスメトリクスは良好で、トライアル検出の結果が悪い場合は、検証済み画像をデータセット (トレーニングデータセット) に追加してモデルを改善できます。別のテストデータセットがある場合は、テストデータセットに画像を追加することを検討してください。

検証済み画像をデータセットに追加したら、モデルを再トレーニングして再評価します。詳細については、「モデルのトレーニング」を参照してください。

トライアル検出の結果を検証するには

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. 左側のナビゲーションペインで、[プロジェクト] を選択します。
3. 「プロジェクト」ページで、削除するプロジェクトを選択します。プロジェクトのダッシュボードが表示されます。
4. 左側のナビゲーションペインで、[トライアル検出] を選択します。
5. 検証したいトライアル検出を選択します。
6. 「トライアル検出」ページで、[マシン予測を確認] を選択します。
7. [このページのすべての画像を選択] をクリックします。
8. 予測が正しい場合は、[正しいことを確認] を選択します。それ以外の場合は、[正しくないことを確認] を選択します。予測と予測の信頼スコアは、各画像の下に表示されます。
9. 画像のラベルを変更する必要がある場合は、以下を実行します。

    a. 画像の下の [正しい] または [正しくない] を選びます。

    b. 画像の正しいラベルを特定できない場合は、ギャラリーで画像を選択して画像を拡大してください。

    > **Note**
    >
    > [フィルター] セクションで目的のラベル、またはラベルの状態を選択することで、画像ラベルをフィルターすることができます。[並べ替えオプション] セクションで、信頼度スコアによる並べ替えができます。

10. 使用しているモデルがセグメンテーションモデルで、画像のマスクまたは異常ラベルが間違っている場合は、画像の下の [異常領域] を選択し、注釈ツールを開きます。注釈ツールによるセグメンテーションラベルの修正 を実行してセグメンテーション情報を更新します。
11. すべての画像が検証されるまで、必要に応じて各ページで手順 7 ~ 10 を繰り返します。
12. [検証済み画像をデータセットに追加] を選択します。個別のデータセットがある場合、画像はトレーニングデータセットに追加されます。
13. モデルをトレーニングする 詳細については、「the section called “モデルのトレーニング”」を参照してください。

注釈ツールによるセグメンテーションラベルの修正

注釈ツールを使用して、異常な領域をマスクでマークして画像をセグメント化します。

注釈ツールを使用して画像のセグメンテーションラベルを修正するには

1. データセットギャラリーの画像で [異常領域] を選択して、注釈ツールを開きます。
2. マスクの異常ラベルが正しくない場合は、マスクを選択し、[異常ラベル] で正しい異常ラベルを選択します。必要に応じて、[異常ラベルを追加] を選択して新しい異常ラベルを追加します。
3. マスクが正しくない場合は、ページの下部にある描画ツールを選択し、異常領域をしっかりと覆うマスクを描画して異常ラベルを作成します。次の図は、異常をしっかりとカバーするマスクの例です。

以下は、異常をしっかりとカバーしていないマスクの例です。

4. 修正する画像が他にもある場合は、[次へ] を選択し、ステップ 2 と 3 を繰り返します。
5. [送信して終了] を選択し、画像の更新を完了します。

トレーニング済みの Amazon Lookout for Vision モデルの実行

モデルにより画像内の異常を検出するためには、まず[StartModel] オペレーションでモデルを開始する必要があります。Amazon Lookout for Vision コンソールは、モデルの開始と停止に使用できる AWS CLI コマンドを提供しています。このセクションは、使用できるサンプルコードを含みます。

モデルの開始後は、`DetectAnomalies` オペレーションを使用して画像の異常を検出することができます。詳細については、「画像内の異常を検出する」を参照してください。

トピック

- 推論単位
- アベイラビリティーゾーン
- Amazon Lookout for Vision モデルの開始
- Amazon Lookout for Vision モデルの停止

推論単位

モデルを開始すると、Amazon Lookout for Vision は、推論単位と呼ばれる最低 1 つのコンピューティングリソースをプロビジョニングします。`StartModel` API への `MinInferenceUnits` 入力パラメータで、使用する推論単位数を指定します。モデルのデフォルトの割り当ては 1 推論単位です。

> **Important**
>
> 課金は、モデルが実行される時間数と、モデルがその実行中に使用する推論単位の数に基づいて行われます。たとえば、モデルを 2 推論単位で開始し、モデルを 8 時間使用すると、16 推論時間 (8 時間の実行時間× 2 推論単位) に対して課金されます。詳細については、「Amazon Lookout for Vision の料金」を参照してください。StopModel を呼び出してモデルを明示的に停止しないと、モデルで画像をアクティブに分析していない場合でも課金されます。

単一の推論単位がサポートする 1 秒あたりのトランザクション (TPS) は、次の影響を受けます:

- Lookout for Vision がモデルのトレーニングに使用するアルゴリズム。モデルをトレーニングすると、複数のモデルがトレーニングされます。Lookout for Vision では、データセットのサイズと正常画像と異常画像の構成に基づいて、最適なパフォーマンスを備えたモデルが選択されます。
- 高解像度の画像では、解析に時間がかかります。
- 小さいサイズの画像 (MB単位) は、大きな画像よりも速く分析されます。

推論単位によるスループットの管理

アプリケーションの要求に応じて、モデルのスループットを増やしたり減らしたりすることができます。スループットを増やすには、追加の推論単位を使用します。推論単位を追加するたびに、処理速度が 1 推論単位増加します。必要な推論単位数の計算方法については、「Amazon Rekognition Custom Labels と Amazon Lookout for Vision モデルの推論単を計算する」を参照してください。モデルのサポート対象スループットを変更する場合は、次の 2 つのオプションがあります:

手動で推論単位を追加または削除する

モデルを停止し、必要な推論単位数で再開します。この方法の欠点は、再開中はモデルがリクエストを受け取ることができず、需要の急増に対処できないことです。モデルのスループットが安定していて、ユースケースが 10 ~ 20 分のダウンタイムを許容できる場合は、このアプローチを使用してください。例としては、週単位のスケジュールを使用してモデルへの呼び出しをバッチ処理する場合などが挙げられます。

推論単位数を自動スケーリングする

モデルが需要急増に対応する必要がある場合、Amazon Lookout for Vision は、モデルが使用する推論単位数を自動的にスケーリングできます。需要が高まると、Amazon Lookout for Vision はモデルに推論単位数を追加し、需要が減少するとそれを削除します。

Lookout for Vision がモデルの推論単位数を自動的にスケーリングできるようにするには、モデルを開始し、使用できる推論単位の最大数を `MaxInferenceUnits` パラメーターにより設定します。推論単位の最大数を設定すると、使用可能な推論単位の数を制限することで、モデルの実行コストを管理できます。最大単位数を指定しない場合、Lookout for Vision はモデルを自動的にスケーリングせず、最初に設定した推論単位の数を使用するだけです。推論単位の最大数については、「Service Quotas」を参照してください。

`MinInferenceUnits` パラメータを使用して推論単位の最小数も指定できます。これにより、モデルの最小スループットを指定できます。1 つの推論単位は 1 時間の処理時間を表します。

> **Note**
>
> Lookout for Vision コンソールでは推論単位の最大数を設定できません。代わりに、StartModel オペレーションに MaxInferenceUnits 入力パラメーターを指定してください。

Lookout for Vision には、モデルの現在の自動スケーリングステータスを判断するために使用できる以下の Amazon CloudWatch Logs メトリクスが用意されています。

メトリクス	説明
DesiredInferenceUnits	Lookout for Vision のスケールアップまたはスケールダウンの対象となる推論単位数。
InServiceInferenceUnits	モデルが使用する推論単位数。

DesiredInferenceUnits = InServiceInferenceUnits の場合、Lookout for Vision は現在、推論単位数をスケーリングしていません。

DesiredInferenceUnits > InServiceInferenceUnits の場合、Lookout for Vision は DesiredInferenceUnits の値までスケールアップしています。

DesiredInferenceUnits < InServiceInferenceUnits の場合、Lookout for Vision は DesiredInferenceUnits の値までスケールダウンしています。

Lookout for Vision によって返されるメトリクスに関する詳細については、「Amazon CloudWatch による Lookout for Vision のモニタリング」を参照してください。

モデルに対してリクエストした推論単位の最大数を確認するには、DescribeModel を呼び出してレスポンス内の MaxInferenceUnits フィールドをチェックします。

アベイラビリティーゾーン

Amazon Lookout for Vision は、AWS リージョン内の複数のアベイラビリティーゾーンに推論単位を分散させ、可用性を高めます。詳細については、「アベイラビリティーゾーン」を参照してください。アベイラビリティーゾーンの停止や推論単位の障害から量産モデルを保護するために、少なくとも 2 つの推論単位を使用して量産モデルを開始することを強くお勧めします。

アベイラビリティーゾーンの停止が発生すると、アベイラビリティーゾーン内のすべての推論単位が使用できなくなり、モデル容量が削減されます。DetectAnomalies への呼び出しは、残りの推論単位に再配分されます。このような呼び出しは、残りの推論単位の 1 秒あたりのトランザクション(TPS)がサポート範囲を超えていなければ成功します。AWS がアベイラビリティーゾーンを修復した後、推論単位が再開され、完全な容量が復元されます。

1 つの推論単位に障害が発生すると、Lookout for Vision は同じアベイラビリティーゾーンで新しい推論単位を自動的に開始します。新しい推論単位が始まるまで、モデル容量は減少します。

Amazon Lookout for Vision モデルの開始

Amazon Lookout for Vision モデルを使用して異常を検出するには、まずモデルを開始する必要があります。StartModel API を呼び出してモデルを開始し、以下を渡します:

- ProjectName — 開始するモデルを含むプロジェクトの名前。
- ModelVersion — 開始するモデルのバージョン。
- MinInferenceUnits — 推論単位の最小数。詳細については、「推論単位」を参照してください。
- (オプション) MaxInferenceUnits — Amazon Lookout for Vision がモデルを自動的にスケーリングするために使用できる推論単位の最大数。詳細については、「推論単位数を自動スケーリングする」を参照してください。

Amazon Lookout for Vision コンソールには、モデルの開始と停止に使用できるサンプルコードが用意されています。

> **Note**
>
> モデルの稼働時間に応じて課金されます。実行中のモデルを停止するには、「Amazon Lookout for Vision モデルの停止 」を参照してください。
> AWS SDKを使うと、Lookout for Vision を利用可能なすべての AWS リージョンにおける実行モデルを見ることができます。コード例については、find_running_models.py を参照してください。

トピック

- モデルの開始 (コンソール)
- Amazon Lookout for Vision モデル (SDK) を開始する

モデルの開始 (コンソール)

Amazon Lookout for Vision コンソールは、モデルの開始と停止に使用できる AWS CLI コマンドを提供しています。モデルの開始後、画像内の異常の検出を開始できます。詳細については、「画像内の異常を検出する」を参照してください。

モデル(コンソール)を開始するには

1. まだ行っていない場合は、AWS CLI とAWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
3. [開始する] を選択します。
4. 左側のナビゲーションペインで、[プロジェクト] を選択します。
5. [プロジェクト] リソースページで、開始するモデルを含むプロジェクトを選択します。
6. [モデル] セクションで、開始するモデルを選択します。
7. モデルの詳細ページで [モデルを使用] を選択し、[API をクラウドに統合] を選択します。

   > **Tip**
   >
   > モデルをエッジデバイスにデプロイする場合は、[モデルパッケージングジョブを作成] を選択します。詳細については、「Amazon Lookout for Vision モデルのパッケージング」を参照してください。

8. [AWS CLI コマンド] で、AWS呼び出す CLI コマンド`start-model`をコピーします。
9. コマンドプロンプトで、前のステップでコピーした `start-model` コマンドを入力します。`lookoutvision` プロファイルを使用して認証情報を取得する場合は、`--profile lookoutvision-access` パラメータを追加します。
10. コンソールで、左のナビゲーションページから[モデル] を選択します。
11. モデルの現在のステータスは、[ステータス] 欄をチェックします。ステータスが[ホスト済み] の場合、このモデルを使用して画像内の異常を検出できます。詳細については、「画像内の異常を検出する」を参照してください。

Amazon Lookout for Vision モデル (SDK) を開始する

StartModel オペレーションを呼び出してモデルを開始します。

モデルの開始までに時間がかかる場合があります。DescribeModel を呼び出して現在のステータスを確認できます 詳細については、「モデルの表示」を参照してください。

モデル (SDK) を起動するには

1. まだ行っていない場合は、AWS CLI とAWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のサンプルコードを使用して、モデルを開始します。

   CLI

   以下の値を変更します。

   - project-name に開始したいモデルを含むプロジェクト名を入力します。
   - model-version に開始したいモデルのバージョンを入力します。
   - --min-inference-units で使用する推論単位の数を指定します。
   - (オプション) --max-inference-units を、自動的にモデルをスケーリングするために Amazon Lookout for Vision が使用できる推論単位の最大数に。

```
aws lookoutvision start-model --project-name "project name"\
  --model-version model version\
  --min-inference-units minimum number of units\
  --max-inference-units max number of units \
  --profile lookoutvision-access
```

   Python

   このコードは、AWS ドキュメンテーション SDK サンプル GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
    @staticmethod
    def start_model(
            lookoutvision_client, project_name, model_version,
min_inference_units, max_inference_units = None):
        """
        Starts the hosting of a Lookout for Vision model.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
```

```
        :param project_name:  The name of the project that contains the version
of the
                              model that you want to start hosting.
        :param model_version: The version of the model that you want to start
hosting.
        :param min_inference_units: The number of inference units to use for
hosting.
        :param max_inference_units: (Optional) The maximum number of inference
units that
        Lookout for Vision can use to automatically scale the model.
        """
        try:
            logger.info(
                "Starting model version %s for project %s", model_version,
project_name)

            if max_inference_units is None:
                lookoutvision_client.start_model(
                    ProjectName = project_name,
                    ModelVersion = model_version,
                    MinInferenceUnits = min_inference_units)

            else:
                lookoutvision_client.start_model(
                    ProjectName = project_name,
                    ModelVersion = model_version,
                    MinInferenceUnits = min_inference_units,
                    MaxInferenceUnits = max_inference_units)

            print("Starting hosting...")

            status = ""
            finished = False

            # Wait until hosted or failed.
            while finished is False:
                model_description = lookoutvision_client.describe_model(
                    ProjectName=project_name, ModelVersion=model_version)
                status = model_description["ModelDescription"]["Status"]

                if status == "STARTING_HOSTING":
                    logger.info("Host starting in progress...")
                    time.sleep(10)
                    continue
```

```
                if status == "HOSTED":
                    logger.info("Model is hosted and ready for use.")
                    finished = True
                    continue

                logger.info("Model hosting failed and the model can't be used.")
                finished = True

            if status != "HOSTED":
                logger.error("Error hosting model: %s", status)
                raise Exception(f"Error hosting model: {status}")
        except ClientError:
            logger.exception("Couldn't host model.")
            raise
```

Java V2

このコードは、AWS ドキュメンテーション SDK サンプル GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
/**
 * Starts hosting an Amazon Lookout for Vision model. Returns when the model has
 * started or if hosting fails. You are charged for the amount of time that a
 * model is hosted. To stop hosting a model, use the StopModel operation.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that contains the model that you
 *                    want to host.
 * @modelVersion The version of the model that you want to host.
 * @minInferenceUnits The number of inference units to use for hosting.
 * @maxInferenceUnits The maximum number of inference units that Lookout for
 *                    Vision can use for automatically scaling the model. If the
 *                    value is null, automatic scaling doesn't happen.
 * @return ModelDescription The description of the model, which includes the
 *         model hosting status.
 */
public static ModelDescription startModel(LookoutVisionClient lfvClient, String
 projectName, String modelVersion,
        Integer minInferenceUnits, Integer maxInferenceUnits) throws
 LookoutVisionException, InterruptedException {

    logger.log(Level.INFO, "Starting Model version {0} for project {1}.",
```

```
            new Object[] { modelVersion, projectName });

   StartModelRequest startModelRequest = null;

   if (maxInferenceUnits == null) {

       startModelRequest =
StartModelRequest.builder().projectName(projectName).modelVersion(modelVersion)
               .minInferenceUnits(minInferenceUnits).build();
   } else {
       startModelRequest =
StartModelRequest.builder().projectName(projectName).modelVersion(modelVersion)

.minInferenceUnits(minInferenceUnits).maxInferenceUnits(maxInferenceUnits).build();
   }

   // Start hosting the model.
   lfvClient.startModel(startModelRequest);

   DescribeModelRequest describeModelRequest =
DescribeModelRequest.builder().projectName(projectName)
           .modelVersion(modelVersion).build();

   ModelDescription modelDescription = null;

   boolean finished = false;
   // Wait until model is hosted or failure occurs.
   do {

       modelDescription =
lfvClient.describeModel(describeModelRequest).modelDescription();

       switch (modelDescription.status()) {

       case HOSTED:
           logger.log(Level.INFO, "Model version {0} for project {1} is
running.",
                   new Object[] { modelVersion, projectName });
           finished = true;
           break;

       case STARTING_HOSTING:
           logger.log(Level.INFO, "Model version {0} for project {1} is
starting.",
```

```
                    new Object[] { modelVersion, projectName });

            TimeUnit.SECONDS.sleep(60);

            break;
        case HOSTING_FAILED:
            logger.log(Level.SEVERE, "Hosting failed for model version {0} for
 project {1}.",
                    new Object[] { modelVersion, projectName });
            finished = true;
            break;

        default:
            logger.log(Level.SEVERE, "Unexpected error when hosting model
 version {0} for project {1}: {2}.",
                    new Object[] { projectName, modelVersion,
 modelDescription.status() });
            finished = true;
            break;

        }

    } while (!finished);

    logger.log(Level.INFO, "Finished starting model version {0} for project {1}
 status: {2}",
            new Object[] { modelVersion, projectName,
 modelDescription.statusMessage() });

    return modelDescription;

}
```

3. コードの出力が `Model is hosted and ready for use` の場合、このモデルを使用して画像内の異常を検出できます。詳細については、「画像内の異常を検出する」を参照してください。

Amazon Lookout for Vision モデルの停止

実行中のモデルを停止するには、`StopModel` オペレーションを呼び出し、以下を渡します:

- プロジェクト — 停止するモデルを含むプロジェクトの名前。

- ModelVersion — 停止するモデルのバージョン。

Amazon Lookout for Vision コンソールには、モデルの停止に使用できるサンプルコードが用意されています。

> **Note**
>
> モデルの稼働時間に応じて課金されます。

トピック

- モデルの停止 (コンソール)
- Amazon Lookout for Vision モデル (SDK) を停止する

モデルの停止 (コンソール)

次の手順のステップを実行し、コンソールを使用してモデルを停止します。

モデルを停止するには (コンソール)

1. まだ行っていない場合は、AWS CLI とAWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
3. [開始する] を選択します。
4. 左側のナビゲーションペインで、[プロジェクト] を選択します。
5. [プロジェクト] リソースページで、停止したいモデルを含むプロジェクトを選択します。
6. [モデル] セクションで、停止するモデルを選択します。
7. モデルの詳細ページで [モデルを使用] を選択し、[API をクラウドに統合] を選択します。
8. [AWS CLI コマンド] で、`stop-model` を呼び出す AWS CLI コマンドをコピーします。
9. コマンドプロンプトで、前のステップでコピーした `stop-model` コマンドを入力します。`lookoutvision` プロファイルを使用して認証情報を取得する場合は、`--profile lookoutvision-access` パラメータを追加します。
10. コンソールで、左のナビゲーションページから[モデル] を選択します。

11. モデルの現在のステータスは、[ステータス] 欄をチェックします。[ステータス] 欄の値が [トレーニング完了] の場合、このモデルは停止しています。

Amazon Lookout for Vision モデル (SDK) を停止する

StopModel オペレーションを呼び出してモデルを停止します。

モデルの停止までに時間がかかる場合があります。現在のステータスを確認するには、DescribeModel を使用します。

モデルを停止するには (SDK)

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のサンプルコードを使用して、実行中のモデルを停止します。

   CLI

   以下の値を変更します。

   - project-name に停止するモデルが含まれているプロジェクトの名前を入力します。
   - model-version に停止するモデルのバージョンを入力します。

```
aws lookoutvision stop-model --project-name "project name"\
    --model-version model version \
  --profile lookoutvision-access
```

   Python

   このコードは、AWS ドキュメンテーション SDK サンプル GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
    @staticmethod
    def stop_model(lookoutvision_client, project_name, model_version):
        """
        Stops a running Lookout for Vision Model.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
```

```
        :param project_name: The name of the project that contains the version
of
                             the model that you want to stop hosting.
        :param model_version:  The version of the model that you want to stop
hosting.
        """
        try:
            logger.info("Stopping model version %s for %s", model_version,
project_name)
            response = lookoutvision_client.stop_model(
                ProjectName=project_name, ModelVersion=model_version
            )
            logger.info("Stopping hosting...")

            status = response["Status"]
            finished = False

            # Wait until stopped or failed.
            while finished is False:
                model_description = lookoutvision_client.describe_model(
                    ProjectName=project_name, ModelVersion=model_version
                )
                status = model_description["ModelDescription"]["Status"]

                if status == "STOPPING_HOSTING":
                    logger.info("Host stopping in progress...")
                    time.sleep(10)
                    continue

                if status == "TRAINED":
                    logger.info("Model is no longer hosted.")
                    finished = True
                    continue

                logger.info("Failed to stop model: %s ", status)
                finished = True

            if status != "TRAINED":
                logger.error("Error stopping model: %s", status)
                raise Exception(f"Error stopping model: {status}")
        except ClientError:
            logger.exception("Couldn't stop hosting model.")
            raise
```

Java V2

このコードは、AWS ドキュメンテーション SDK サンプル GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
/**
 * Stops the hosting an Amazon Lookout for Vision model. Returns when model has
 * stopped or if hosting fails.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that contains the model that you
 *                    want to stop hosting.
 * @modelVersion The version of the model that you want to stop hosting.
 * @return ModelDescription The description of the model, which includes the
 *         model hosting status.
 */

public static ModelDescription stopModel(LookoutVisionClient lfvClient, String
 projectName,
                String modelVersion) throws LookoutVisionException,
 InterruptedException {

        logger.log(Level.INFO, "Stopping Model version {0} for project {1}.",
                        new Object[] { modelVersion, projectName });

        StopModelRequest stopModelRequest = StopModelRequest.builder()
                        .projectName(projectName)
                        .modelVersion(modelVersion)
                        .build();

        // Stop hosting the model.

        lfvClient.stopModel(stopModelRequest);

        DescribeModelRequest describeModelRequest =
 DescribeModelRequest.builder()
                        .projectName(projectName)
                        .modelVersion(modelVersion)
                        .build();

        ModelDescription modelDescription = null;
```

```
        boolean finished = false;
        // Wait until model is stopped or failure occurs.
        do {

                modelDescription =
lfvClient.describeModel(describeModelRequest).modelDescription();

                switch (modelDescription.status()) {

                        case TRAINED:
                                logger.log(Level.INFO, "Model version {0} for
project {1} has stopped.",
                                                new Object[] { modelVersion,
projectName });
                                finished = true;
                                break;

                        case STOPPING_HOSTING:
                                logger.log(Level.INFO, "Model version {0} for
project {1} is stopping.",
                                                new Object[] { modelVersion,
projectName });

                                TimeUnit.SECONDS.sleep(60);

                                break;

                        default:
                                logger.log(Level.SEVERE,
                                                "Unexpected error when stopping
model version {0} for project {1}: {2}.",
                                                new Object[] { projectName,
modelVersion,

modelDescription.status() });
                                finished = true;
                                break;

                }

        } while (!finished);
```

```
        logger.log(Level.INFO, "Finished stopping model version {0} for project
 {1} status: {2}",
                        new Object[] { modelVersion, projectName,
 modelDescription.statusMessage() });

        return modelDescription;

}
```

画像内の異常を検出する

トレーニング済みの Amazon Lookout for Vision モデルを使用して画像内の異常を検出するには、DetectAnomalies オペレーションを実行します。DetectAnomalies の結果には、1 つ以上の異常を含むような画像を分類するブーリアン予測と、その予測に対する信頼値が含まれます。モデルが画像セグメンテーションモデルの場合、結果にはさまざまなタイプの異常の位置を示す色付きのマスクも含まれます。

DetectAnomalies に提供する画像の幅と高さは、モデルのトレーニングに使用した画像と同じである必要があります。

DetectAnomalies は PNG または JPG 形式の画像を受け付けます。画像のエンコードと圧縮形式は、モデルのトレーニングに使用したものと同じにすることをお勧めします。たとえば、PNG 形式の画像を使用してモデルをトレーニングする場合は、PNG 形式の画像で DetectAnomalies を呼び出します。

DetectAnomalies を呼び出す前に、まずモデルを StartModel オペレーションで開始してください。詳細については「Amazon Lookout for Vision モデルの開始」を参照してください。モデルが実行される時間 (分単位) と、モデルが使用する異常検出ユニットの数に対して課金されます。モデルを使用していない場合は、StopModel オペレーションでモデルを停止します。詳細については「Amazon Lookout for Vision モデルの停止」を参照してください。

トピック

- DetectAnomalies の呼び出し
- DetectAnomalies からのレスポンスの把握
- 画像が異常であるかどうかの判断
- 分類とセグメンテーションの情報の表示
- AWS Lambda 関数による異常の検出

DetectAnomalies の呼び出し

DetectAnomalies を実行するには、以下を指定します。

- [プロジェクト] — 使用するモデルを含むプロジェクトの名前。
- [モデルバージョン] — 使用するモデルのバージョン。

- [コンテンツタイプ] — 分析する画像のタイプ。有効な値は、image/png (PNG 形式の画像) とimage/jpeg (JPG 形式の画像) です。
- [本文] — 画像を表す暗号化されていないバイナリバイト。

  画像は、モデルのトレーニングに使用された画像と同じ寸法である必要があります。

次の例は、DetectAnomalies を——————び出す方法を示しています。Python と Java の例の関数レスポンスを使用して、画像が異常であるかどうかの判断 内の関数を呼び出すことができます。

AWS CLI

この AWS CLI コマンドでは、DetectAnomalies CLI オペレーションの JSON 出力を表示します。次の入力パラメータの値を変更します。:

- project name に使用するプロジェクトの名前を入力します。
- model version に使用するモデルのバージョンを入力します。
- content type に使用したい画像のタイプを入力します。有効な値は、image/png (PNG 形式の画像) とimage/jpeg (JPG 形式の画像) です。
- file name に使用する画像のパスとファイル名を指定します。必ずファイルタイプが content-type の値と一致しているようにしてください。

```
aws lookoutvision detect-anomalies  --project-name project name\
    --model-version model version\
    --content-type content type\
    --body file name \
    --profile lookoutvision-access
```

Python

完全なコード例は、GitHub でご覧ください。

```
    def detect_anomalies(lookoutvision_client, project_name, model_version, photo):
        """
        Calls DetectAnomalies using the supplied project, model version, and image.
        :param lookoutvision_client: A Lookout for Vision Boto3 client.
        :param project: The project that contains the model that you want to use.
        :param model_version: The version of the model that you want to use.
```

```
        :param photo: The photo that you want to analyze.
        :return: The DetectAnomalyResult object that contains the analysis results.
        """

        image_type = imghdr.what(photo)
        if image_type == "jpeg":
            content_type = "image/jpeg"
        elif image_type == "png":
            content_type = "image/png"
        else:
            logger.info("Invalid image type for %s", photo)
            raise ValueError(
                f"Invalid file format. Supply a jpeg or png format file: {photo}")


        # Get images bytes for call to detect_anomalies
        with open(photo, "rb") as image:
            response = lookoutvision_client.detect_anomalies(
                ProjectName=project_name,
                ContentType=content_type,
                Body=image.read(),
                ModelVersion=model_version)

        return response['DetectAnomalyResult']
```

Java V2

```
  public static DetectAnomalyResult detectAnomalies(LookoutVisionClient lfvClient,
String projectName,
            String modelVersion,
            String photo) throws IOException, LookoutVisionException {
        /**
         * Creates an Amazon Lookout for Vision dataset from a manifest file.
         * Returns after Lookout for Vision creates the dataset.
         *
         * @param lfvClient    An Amazon Lookout for Vision client.
         * @param projectName  The name of the project in which you want to create a
         *                     dataset.
         * @param modelVersion The version of the model that you want to use.
         *
         * @param photo        The photo that you want to analyze.
         *
         * @return DetectAnomalyResult The analysis result from DetectAnomalies.
```

```
         */

        logger.log(Level.INFO, "Processing local file: {0}", photo);

        // Get image bytes.

        InputStream sourceStream = new FileInputStream(new File(photo));
        SdkBytes imageSDKBytes = SdkBytes.fromInputStream(sourceStream);
        byte[] imageBytes = imageSDKBytes.asByteArray();

        // Get the image type. Can be image/jpeg or image/png.
        String contentType = getImageType(imageBytes);

        // Detect anomalies in the supplied image.
        DetectAnomaliesRequest request =
DetectAnomaliesRequest.builder().projectName(projectName)
                .modelVersion(modelVersion).contentType(contentType).build();

        DetectAnomaliesResponse response = lfvClient.detectAnomalies(request,
                RequestBody.fromBytes(imageBytes));

        /*
         * Tip: You can also use the following to analyze a local file.
         * Path path = Paths.get(photo);
         * DetectAnomaliesResponse response = lfvClient.detectAnomalies(request,
path);
         */
        DetectAnomalyResult result = response.detectAnomalyResult();

        String prediction = "Prediction: Normal";

        if (Boolean.TRUE.equals(result.isAnomalous())) {
            prediction = "Prediction: Anomalous";
        }

        // Convert confidence to percentage.
        NumberFormat defaultFormat = NumberFormat.getPercentInstance();
        defaultFormat.setMinimumFractionDigits(1);
        String confidence = String.format("Confidence: %s",
defaultFormat.format(result.confidence()));

        // Log classification result.
        String photoPath = "File: " + photo;
        String[] imageLines = { photoPath, prediction, confidence };
```

```
        logger.log(Level.INFO, "Image: {0}\nAnomalous: {1}\nConfidence {2}",
imageLines);

        return result;

    }

    // Gets the image mime type. Supported formats are image/jpeg and image/png.
    private static String getImageType(byte[] image) throws IOException {

        InputStream is = new BufferedInputStream(new ByteArrayInputStream(image));
        String mimeType = URLConnection.guessContentTypeFromStream(is);

        logger.log(Level.INFO, "Image type: {0}", mimeType);

        if (mimeType.equals("image/jpeg") || mimeType.equals("image/png")) {
            return mimeType;
        }
        // Not a supported file type.
        logger.log(Level.SEVERE, "Unsupported image type: {0}", mimeType);
        throw new IOException(String.format("Wrong image type. %s format isn't
supported.", mimeType));
    }
```

DetectAnomalies からのレスポンスの把握

DetectAnomalies からのレスポンスは、トレーニングするモデルのタイプ (分類モデルまたはセグメンテーションモデル) によって異なります。いずれの場合も、レスポンスは DetectAnomalyResult オブジェクトです。

分類モデル

モデルが 画像分類モデル の場合、DetectAnomalies からのレスポンスには以下が含まれます:

- IsAnomalous — 画像に 1 つ以上の異常があることを示すブーリアン指標。
- Confidence — 異常予測 (IsAnomalous) の精度における Amazon Lookout for Vision の信頼度。Confidence は 0 ~ 1 の浮動小数点値です。値が高いほど、信頼度が高いことを示します。
- Source — DetectAnomalies に渡された画像に関する情報。

```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": true,
        "Confidence": 0.9996867775917053
    }
}
```

画像に異常があるかどうかを判断するには、IsAnomalous フィールドをチェックし、Confidence の値が二ーズを満たすほど大きいことを確認します。

DetectAnomalies で返される信頼値が低すぎると感じる場合は、モデルの再トレーニングを検討してください。サンプルコードについては、「分類」を参照してください。

セグメンテーションモデル

モデルが 画像セグメンテーションモデル の場合、レスポンスには分類情報と、画像マスクや異常タイプなどのセグメンテーション情報が含まれます。分類情報はセグメンテーション情報とは別に計算されるため、両者の関係は想定しないでください。レスポンスにセグメンテーション情報が表示されない場合は、最新バージョンの AWS SDK がインストールされていることをチェックしてください (AWS CLI を使用している場合 AWS Command Line Interface)。サンプルコードについては、セグメンテーション と 分類とセグメンテーションの情報の表示 を参照してください。

- IsAnomalous (分類) — 画像を正常または異常のいずれかに分類するブーリアン指標。
- Confidence (分類) — 異常予測の精度における Amazon Lookout for Vision の信頼度 (IsAnomalous)。Confidence は 0 ~ 1 の浮動小数点値です。値が高いほど、信頼度が高いことを示します。
- Source — DetectAnomalies に渡された画像に関する情報。
- AnomalyMask (セグメンテーション) — 分析された画像で見つかった異常をカバーするピクセルマスク。画像には異常が複数存在する場合があります。マスクマップの色は異常のタイプを示します。マスクカラーは、トレーニングデータセット内の異常タイプに割り当てられた色に対応しています。マスクカラーから異常タイプを見つけるには、Anomalies リストで返された各異常の PixelAnomaly フィールドで Color をチェックします。サンプルコードについては、「分類とセグメンテーションの情報の表示」を参照してください。

- Anomalies (セグメンテーション) — 画像で見つかった異常のリスト。各異常には、異常タイプ (Name) とピクセル情報 (PixelAnomaly) が含まれます。TotalPercentageArea は異常がカバーする画像の面積のパーセンテージです。Color は異常のマスクカラーです。

  リストの最初の要素は常に画像の背景 (BACKGROUND) を表す異常タイプなので、異常と見なすべきではありません。Amazon Lookout for Vision では、背景の異常タイプが自動的にレスポンスに追加されます。データセットで背景異常タイプを宣言する必要はありません。


```
{
    "DetectAnomalyResult": {
        "Source": {
            "Type": "direct"
        },
        "IsAnomalous": true,
        "Confidence": 0.9996814727783203,
        "Anomalies": [
            {
                "Name": "background",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.998999834060669,
                    "Color": "#FFFFFF"
                }
            },
            {
                "Name": "scratch",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.0004034999874420464,
                    "Color": "#7ED321"
                }
            },
            {
                "Name": "dent",
                "PixelAnomaly": {
                    "TotalPercentageArea": 0.0005966666503809392,
                    "Color": "#4DD8FF"
                }
            }
        ],
        "AnomalyMask": "iVBORw0....."
    }
}
```

画像が異常であるかどうかの判断

画像が異常であるかどうかは、さまざまな方法で判断できます。選択する方法は、ユースケースとモデルのタイプによって異なります。考えられる解決策は次のとおりです。

トピック

分類

IsAnomalous は画像を異常として分類し、画像が実際に異常であるかどうかを判断するために Confidence フィールドを使用します。値が高いほど、信頼度が高いことを示します。たとえば、信頼度が 80% を超えた場合にのみ製品に欠陥があるとすることを判断基準とできます。分類モデルまたは画像セグメンテーションモデルによって分析した画像を分類できます。

Python

完全なコード例は、GitHub でご覧ください。

```
    def reject_on_classification(image, prediction, confidence_limit):
        """
        Returns True if the anomaly confidence is greater than or equal to
        the supplied confidence limit.
        :param image: The name of the image file that was analyzed.
        :param prediction: The DetectAnomalyResult object returned from
DetectAnomalies
        :param confidence_limit: The minimum acceptable confidence. Float value
between 0 and 1.
        :return: True if the error condition indicates an anomaly, otherwise False.
        """

        reject = False

        logger.info("Checking classification for %s", image)

        if prediction['IsAnomalous'] and prediction['Confidence'] >=
confidence_limit:
            reject = True
            reject_info=(f"Rejected: Anomaly confidence
({prediction['Confidence']:.2%}) is greater"
```

```
                    f" than limit ({confidence_limit:.2%})")
            logger.info("%s", reject_info)

        if not reject:
            logger.info("No anomalies found.")
        return reject
```

Java V2

```
   public static boolean rejectOnClassification(String image, DetectAnomalyResult
prediction, float minConfidence) {
        /**
         * Rejects an image based on its anomaly classification and prediction
         * confidence
         *
         * @param image         The file name of the analyzed image.
         * @param prediction    The prediction for an image analyzed with
         *                      DetectAnomalies.
         * @param minConfidence The minimum acceptable confidence for the prediction
         *                      (0-1).
         *
         * @return boolean True if the image is anomalous, otherwise False.
         */

        Boolean reject = false;

        logger.log(Level.INFO, "Checking classification for {0}", image);

        String[] logParameters = { prediction.confidence().toString(),
String.valueOf(minConfidence) };

        if (Boolean.TRUE.equals(prediction.isAnomalous()) && prediction.confidence()
>= minConfidence) {
            logger.log(Level.INFO, "Rejected: Anomaly confidence {0} is greater than
confidence limit {1}",
                    logParameters);
            reject = true;
        }
        if (Boolean.FALSE.equals(reject))
            logger.log(Level.INFO, ": No anomalies found.");

        return reject;
```

```
    }
```

セグメンテーション

モデルが画像セグメンテーションモデルの場合は、セグメンテーション情報を使用して、画像に異常があるかどうかを判断できます。画像セグメンテーションモデルを使用して画像を分類することもできます。画像マスクを取得して表示するコードの例については、分類とセグメンテーションの情報の表示 を参照してください。

異常領域

画像上の異常のカバー率 (TotalPercentageArea) を使用してください。たとえば、異常領域が画像の 1% を超える場合、商品に欠陥があるとすることを判断基準とできます。

Python

完全なコード例は、GitHub でご覧ください。

```
   def reject_on_coverage(image, prediction, confidence_limit, anomaly_label,
coverage_limit):
       """
       Checks if the coverage area of an anomaly is greater than the coverage limit
and if
       the prediction confidence is greater than the confidence limit.
       :param image: The name of the image file that was analyzed.
       :param prediction: The DetectAnomalyResult object returned from
DetectAnomalies
       :param confidence_limit: The minimum acceptable confidence (float 0-1).
       :anomaly_label: The anomaly label for the type of anomaly that you want to
check.
       :coverage_limit: The maximum acceptable percentage coverage of an anomaly
(float 0-1).
       :return: True if the error condition indicates an anomaly, otherwise False.
       """

       reject = False

       logger.info("Checking coverage for %s", image)

       if prediction['IsAnomalous'] and prediction['Confidence'] >=
confidence_limit:
```

```
            for anomaly in prediction['Anomalies']:
                if (anomaly['Name'] == anomaly_label and
                        anomaly['PixelAnomaly']['TotalPercentageArea'] >
 (coverage_limit)):
                    reject = True
                    reject_info=(f"Rejected: Anomaly confidence
 ({prediction['Confidence']:.2%}) "
                        f"is greater than limit ({confidence_limit:.2%}) and
 {anomaly['Name']} "
                        f"coverage ({anomaly['PixelAnomaly']
['TotalPercentageArea']:.2%}) "
                        f"is greater than limit ({coverage_limit:.2%})")

                    logger.info("%s", reject_info)

        if not reject:
            logger.info("No anomalies found.")

        return reject
```

Java V2

```
   public static Boolean rejectOnCoverage(String image, DetectAnomalyResult
 prediction, float minConfidence,
            String anomalyType, float maxCoverage) {
        /**
         * Rejects an image based on a maximum allowable coverage area for an
 anomaly
         * type.
         *
         * @param image         The file name of the analyzed image.
         * @param prediction    The prediction for an image analyzed with
         *                      DetectAnomalies.
         * @param minConfidence The minimum acceptable confidence for the prediction
         *                      (0-1).
         * @param anomalyTypes  The anomaly type to check.
         * @param maxCoverage   The maximum allowable coverage area of the anomaly
 type.
         *                      (0-1).
         *
         * @return boolean True if the coverage area of the anomaly type exceeds the
         *         maximum allowed, otherwise False.
```

```
         */

        Boolean reject = false;

        logger.log(Level.INFO, "Checking coverage for {0}", image);

        if (Boolean.TRUE.equals(prediction.isAnomalous()) && prediction.confidence()
>= minConfidence) {
            for (Anomaly anomaly : prediction.anomalies()) {

                if (Objects.equals(anomaly.name(), anomalyType)
                        && anomaly.pixelAnomaly().totalPercentageArea() >=
maxCoverage) {

                    String[] logParameters = { prediction.confidence().toString(),
                            String.valueOf(minConfidence),

String.valueOf(anomaly.pixelAnomaly().totalPercentageArea()),
                            String.valueOf(maxCoverage) };
                    logger.log(Level.INFO,
                            "Rejected: Anomaly confidence {0} is greater than
confidence limit {1} and " +
                                    "{2} anomaly type coverage is higher than
coverage limit {3}\n",
                            logParameters);
                    reject = true;

                }
            }
        }

        if (Boolean.FALSE.equals(reject))
            logger.log(Level.INFO, ": No anomalies found.");

        return reject;
    }
```

異常タイプの数

画像にあるさまざまな異常タイプ (Name) の数を使用します。たとえば、異常が 3 タイプ以上存在する場合、製品に欠陥があるとすることを判断基準とできます。

Python

完全なコード例は、GitHub でご覧ください。

```
   def reject_on_anomaly_types(image, prediction, confidence_limit,
anomaly_types_limit):
       """
       Checks if the number of anomaly types is greater than than the anomaly types
       limit and if the prediction confidence is greater than the confidence limit.
       :param image: The name of the image file that was analyzed.
       :param prediction: The DetectAnomalyResult object returned from
DetectAnomalies
       :param confidence: The minimum acceptable confidence. Float value between 0
and 1.
       :param anomaly_types_limit: The maximum number of allowable anomaly types
(Integer).
       :return: True if the error condition indicates an anomaly, otherwise False.
       """

       logger.info("Checking number of anomaly types for %s",image)

       reject = False

       if prediction['IsAnomalous'] and prediction['Confidence'] >=
confidence_limit:

           anomaly_types = {anomaly['Name'] for anomaly in prediction['Anomalies']\
               if anomaly['Name'] != 'background'}

           if len (anomaly_types) > anomaly_types_limit:
                   reject = True
                   reject_info = (f"Rejected: Anomaly confidence
({prediction['Confidence']:.2%}) "
                   f"is greater than limit ({confidence_limit:.2%}) and "
                   f"the number of anomaly types ({len(anomaly_types)-1}) is "
                   f"greater than the limit ({anomaly_types_limit})")

                   logger.info("%s", reject_info)

       if not reject:
           logger.info("No anomalies found.")
       return reject
```

Java V2

```
   public static Boolean rejectOnAnomalyTypeCount(String image, DetectAnomalyResult
prediction,
           float minConfidence, Integer maxAnomalyTypes) {

       /**
        * Rejects an image based on a maximum allowable number of anomaly types.
        *
        * @param image           The file name of the analyzed image.
        * @param prediction      The prediction for an image analyzed with
        *                        DetectAnomalies.
        * @param minConfidence   The minimum acceptable confidence for the
predictio
        *                        (0-1).
        * @param maxAnomalyTypes The maximum allowable number of anomaly types.
        *
        * @return boolean True if the image contains more than the maximum allowed
        *         anomaly types, otherwise False.
        */

       Boolean reject = false;

       logger.log(Level.INFO, "Checking coverage for {0}", image);

       Set<String> defectTypes = new HashSet<>();

       if (Boolean.TRUE.equals(prediction.isAnomalous()) && prediction.confidence()
>= minConfidence) {
           for (Anomaly anomaly : prediction.anomalies()) {
               defectTypes.add(anomaly.name());
           }
           // Reduce defect types by one to account for 'background' anomaly type.
           if ((defectTypes.size() - 1) > maxAnomalyTypes) {
               String[] logParameters = { prediction.confidence().toString(),
                       String.valueOf(minConfidence),
                       String.valueOf(defectTypes.size()),
                       String.valueOf(maxAnomalyTypes) };
               logger.log(Level.INFO, "Rejected: Anomaly confidence {0} is >=
minimum confidence {1} and " +
                       "the number of anomaly types {2} > the allowable number of
anomaly types {3}\n", logParameters);
               reject = true;
           }
```

```
        }

        if (Boolean.FALSE.equals(reject))
            logger.log(Level.INFO, ": No anomalies found.");

        return reject;
    }
```

分類とセグメンテーションの情報の表示

この例では、分析した画像を表示し、分析結果を重ね合わせます。レスポンスに異常マスクが含まれる場合、マスクは関連する異常タイプの色で表示されます。

画像分類と画像セグメンテーションの情報を表示するには

1. まだの場合は、 以下を行ってください:
   a. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
   b. モデルをトレーニングします。
   c. モデルを開始する。
2. DetectAnomalies を呼び出す IAM ユーザーが、使用するモデルのバージョンへのアクセス権を持っていることを確かめます。詳細については、「SDK 権限をセットアップする」を参照してください。
3. 以下のコードを使用します。

   Python

   次のサンプルコードは、提供された画像の異常を検出します。この例では、次のコマンドラインオプションを使用します。

   - project — 使用するプロジェクトの名前。
   - version — プロジェクト内で使用するモデルのバージョンを指定します。
   - image — ローカル画像ファイル (JPEG または PNG 形式) のパスとファイル。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
# SPDX-License-Identifier: Apache-2.0
"""
Shows how to detect and show anomalies in an image using a trained Amazon
 Lookout
for Vision model. The script displays the analysed image and overlays mask and
 analysis
output.
"""

import argparse
import logging
import io
import boto3
from PIL import Image, ImageDraw, ImageFont

from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


class ShowAnomalies:
    """
    Class to detect and show anomalies in an image analyzed by detect_anomalies.
    """



    @staticmethod
    def draw_line(draw, text, fnt,  y_coordinate, color):
        """
        Draws a line of text on the supplied drawing surface.
        :param draw: The surface on which to draw the text.
        :param text: The text to draw in the drawing surface.
        :param fnt: The font for the text.
        :param y_coordinate: The y position for the text.
        :param color: The color for the text.
        :returns The y coordinate for the next line of text.
        """
        text_width, text_height = draw.textsize(text, fnt)
        draw.rectangle([(10, y_coordinate), (text_width + 10,
                                             y_coordinate + text_height)],
 fill="black")
        draw.text((10, y_coordinate), text, fill=color, font=fnt)

        y_coordinate += text_height
```

```
        return y_coordinate

    @staticmethod
    def draw_analysis_text(image, analysis):
        """
        Draws classification and segmentation info onto supplied image
        overlay analysis results on an image analyzed by detect_anomalies.
        :param analysis: The response from a call to detect_anomalies.
        :returns Nothing
        """

        ## Calculate a reasonable font size based on image width.
        font_size = int(image.size[0]/32)

        fnt = ImageFont.truetype('/Library/Fonts/Tahoma.ttf', font_size)

        draw = ImageDraw.Draw(image)

        y_coordinate = 0

        # Draw classification information.
        prediction = "Anomalous" if analysis["DetectAnomalyResult"]
["IsAnomalous"] \
            else "Normal"

        confidence = analysis["DetectAnomalyResult"]["Confidence"]
        found_anomalies = analysis["DetectAnomalyResult"]['Anomalies']
        segmentation_info = False

        logger.info("Prediction: %s", format(prediction))
        logger.info("Confidence: %s", format(confidence))

        y_coordinate = 0
        y_coordinate = ShowAnomalies.draw_line(
            draw, "Classification", fnt, y_coordinate, "white")
        y_coordinate = ShowAnomalies.draw_line(
            draw, f"  Prediction: {prediction}", fnt, y_coordinate, "white")
        y_coordinate = ShowAnomalies.draw_line(
            draw, f"  Confidence: {confidence:.2%}", fnt, y_coordinate, "white")

        # Draw segmentation information, if present.
        if (len(found_anomalies)) > 1:
            logger.info("Anomalies:")
```

```
            y_coordinate = ShowAnomalies.draw_line(
                draw, "Segmentation:", fnt, y_coordinate, "white")
            for i in range(1, len(found_anomalies)):

                # Only display info if more than 0% coverage found.
                percent_coverage = found_anomalies[i]['PixelAnomaly']
['TotalPercentageArea']
                if percent_coverage > 0:
                    segmentation_info = True
                    logger.info("  %s", found_anomalies[i]['Name'])
                    logger.info("    Color: %s",
                                found_anomalies[i]['PixelAnomaly']['Color'])
                    logger.info("    Area: %s", percent_coverage)
                    y_coordinate = ShowAnomalies.draw_line(
                        draw,
                        f"  Anomaly: {found_anomalies[i]['Name']}. Area:
 {percent_coverage:.2%}",
                        fnt,
                        y_coordinate,
                        found_anomalies[i]['PixelAnomaly']['Color'])

            if not segmentation_info:
                y_coordinate = ShowAnomalies.draw_line(
                    draw, "No segmentation information found.", fnt,
 y_coordinate, "white")




    @staticmethod
    def show_anomaly_prediction(lookoutvision_client, project_name,
 model_version, photo):
        """
        Detects anomalies in an image (jpg/png) by using your Amazon Lookout for
 Vision
        model. Displays the image and overlays prediction information text.
        :param lookoutvision_client: An Amazon Lookout for Vision Boto3 client.
        :param project_name: The name of the project that contains the model
 that
        you want to use.
        :param model_version: The version of the model that you want to use.
        :param photo: The path and name of the image in which you want to detect
        anomalies.
```

```
        """
        try:

            logger.info("Detecting anomalies in %s", photo)

            image = Image.open(photo)
            image_type = Image.MIME[image.format]

            # Check that image type is valid.
            if image_type not in ("image/jpeg", "image/png"):
                logger.info("Invalid image type for %s", photo)
                raise ValueError(
                    f"Invalid file format. Supply a jpeg or png format file:
{photo}"
                )

            # Get images bytes for call to detect_anomalies.
            image_bytes = io.BytesIO()
            image.save(image_bytes, format=image.format)
            image_bytes = image_bytes.getvalue()

            # Analyze the image.
            response = lookoutvision_client.detect_anomalies(
                ProjectName=project_name,
                ContentType=image_type,
                Body=image_bytes,
                ModelVersion=model_version
            )

            # Overlay mask onto analyzed image.
            image_mask_bytes = response["DetectAnomalyResult"]["AnomalyMask"]
            image_mask = Image.open(io.BytesIO(image_mask_bytes))

            final_img = Image.blend(image, image_mask, 0.5) \
                if response["DetectAnomalyResult"]["IsAnomalous"] else image

            # Overlay analysis output on image.
            ShowAnomalies.draw_analysis_text(final_img, response)

            final_img.show()

        except ClientError as err:
            logger.info(format(err))
            raise
```

```
def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "project", help="The project containing the model that you want to use."
    )
    parser.add_argument(
        "version", help="The version of the model that you want to use."
    )
    parser.add_argument(
        "image",
        help="The file that you want to analyze. "
        "Supply a local file path.",
    )


def main():
    """
    Entrypoint for anomaly detection example.
    """

    try:
        logging.basicConfig(level=logging.INFO,
                            format="%(levelname)s: %(message)s")

        session = boto3.Session(
            profile_name='lookoutvision-access')

        lookoutvision_client = session.client("lookoutvision")

        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)

        add_arguments(parser)

        args = parser.parse_args()

        # Analyze the image and show results.
        ShowAnomalies.show_anomaly_prediction(
            lookoutvision_client, args.project, args.version, args.image
```

```
        )

    except ClientError as err:
        print("A service error occured: " +
              format(err.response["Error"]["Message"]))
    except FileNotFoundError as err:
        print("The supplied file couldn't be found: " + err.filename)
    except ValueError as err:
        print("A value error occured. " + format(err))
    else:
        print("Successfully completed analysis.")


if __name__ == "__main__":
    main()
```

Java 2

次のサンプルコードは、提供された画像の異常を検出します。この例では、次のコマンドラインオプションを使用します。

- project — 使用するプロジェクトの名前。
- version — プロジェクト内で使用するモデルのバージョンを指定します。
- image — ローカル画像ファイル (JPEG または PNG 形式) のパスとファイル。

```
/*
   Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
   SPDX-License-Identifier: Apache-2.0
*/


package com.example.lookoutvision;

import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.core.SdkBytes;
import software.amazon.awssdk.core.sync.RequestBody;
import software.amazon.awssdk.services.lookoutvision.LookoutVisionClient;
import software.amazon.awssdk.services.lookoutvision.model.Anomaly;
import
 software.amazon.awssdk.services.lookoutvision.model.DetectAnomaliesRequest;
```

```
import
 software.amazon.awssdk.services.lookoutvision.model.DetectAnomaliesResponse;
import software.amazon.awssdk.services.lookoutvision.model.DetectAnomalyResult;
import
 software.amazon.awssdk.services.lookoutvision.model.LookoutVisionException;

import java.io.BufferedInputStream;
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.net.URLConnection;

import java.text.NumberFormat;
import java.awt.*;
import java.awt.font.LineMetrics;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import javax.swing.*;

import java.util.logging.Level;
import java.util.logging.Logger;

// Finds anomalies on a supplied image.
public class ShowAnomalies extends JPanel {
/**
 * Finds and displays anomalies on a supplied image.
 */

    private static final long serialVersionUID = 1L;
    private transient BufferedImage image;
    private transient BufferedImage maskImage;
    private transient Dimension dimension;
    public static final Logger logger =
 Logger.getLogger(ShowAnomalies.class.getName());

    // Constructor. Finds anomalies in a local image file.
    public ShowAnomalies(LookoutVisionClient lfvClient, String projectName,
 String modelVersion,
            String photo) throws IOException, LookoutVisionException {

        logger.log(Level.INFO, "Processing local file: {0}", photo);
```

```
        maskImage = null;

        // Get image bytes and buffered image.
        InputStream sourceStream = new FileInputStream(new File(photo));
        SdkBytes imageSDKBytes = SdkBytes.fromInputStream(sourceStream);
        byte[] imageBytes = imageSDKBytes.asByteArray();
        ByteArrayInputStream inputStream = new
ByteArrayInputStream(imageSDKBytes.asByteArray());
        image = ImageIO.read(inputStream);

        // Get the image type. Can be image/jpeg or image/png.
        String contentType = getImageType(imageBytes);

        // Set the size of the window that shows the image.
        setWindowDimensions();

        // Detect anomalies in the supplied image.
        DetectAnomaliesRequest request =
DetectAnomaliesRequest.builder().projectName(projectName)
                .modelVersion(modelVersion).contentType(contentType).build();

        DetectAnomaliesResponse response = lfvClient.detectAnomalies(request,
                RequestBody.fromBytes(imageBytes));

        /*
         * Tip: You can also use the following to analyze a local file.
         * Path path = Paths.get(photo);
         * DetectAnomaliesResponse response = lfvClient.detectAnomalies(request,
path);
         */
        DetectAnomalyResult result = response.detectAnomalyResult();


        if (result.anomalyMask() != null){
            SdkBytes maskSDKBytes = result.anomalyMask();

            ByteArrayInputStream maskInputStream = new
ByteArrayInputStream(maskSDKBytes.asByteArray());
            maskImage = ImageIO.read(maskInputStream);
        }

        drawImageInfo(result);
```

```
    }

    // Sets window dimensions to 1/2 screen size, unless image is smaller.
    public void setWindowDimensions() {
        dimension = java.awt.Toolkit.getDefaultToolkit().getScreenSize();

        dimension.width = (int) dimension.getWidth() / 2;
        dimension.height = (int) dimension.getHeight() / 2;

        if (image.getWidth() < dimension.width || image.getHeight() <
dimension.height) {
            dimension.width = image.getWidth();
            dimension.height = image.getHeight();
        }
        setPreferredSize(dimension);

    }

    private int drawLine(Graphics2D g2d, String line, FontMetrics metrics, int
yPos, Color color) {
    /**
    * Draws a line of text at the spsecified y position and color.
    * confidence
    *
    * @param g2D The Graphics2D object for the image.
    * @param line The line of text to draw.
    * @param metrics The font information to use.
    * @param yPos The y position for the line of text.
    *
    * @return  The yPos for the next line of text.
    */

        int indent = 10;

        // Get text height, width, and descent.
        int textWidth = metrics.stringWidth(line);
        LineMetrics lm = metrics.getLineMetrics(line, g2d);
        int textHeight = (int) lm.getHeight();
        int descent = (int) lm.getDescent();

        int y2Pos = (yPos + textHeight) - descent;

        // Draw black rectangle.
        g2d.setColor(Color.BLACK);
```

```
        g2d.fillRect(indent, yPos, textWidth, textHeight);

        // Draw text.
        g2d.setColor(color);
        g2d.drawString(line, indent, y2Pos);

        yPos += textHeight;

        return yPos;

    }

    public void drawImageInfo(DetectAnomalyResult result) {
    /**
     * Draws the results from DetectAnomalies onto the output image.
     *
     * @param result The response from a call to
     *               DetectAnomalies.
     *
     */

        // Set up drawing.
        Graphics2D g2d = image.createGraphics();

        if (result.anomalyMask() != null){
            Composite composite = g2d.getComposite();
            g2d.setComposite(AlphaComposite.SrcOver.derive(0.5f));
            int x = (image.getWidth() - maskImage.getWidth()) / 2;
            int y = (image.getHeight() - maskImage.getHeight()) / 2;
            g2d.drawImage(maskImage, x, y, null);
            // Set composite for overlaying text.
            g2d.setComposite(composite);
        }

        //Calculate font size based on arbitary 32 pixel image width.
        int fontSize = (image.getWidth() / 32);


        g2d.setFont(new Font("Tahoma", Font.PLAIN, fontSize));
        Font font = g2d.getFont();
        FontMetrics metrics = g2d.getFontMetrics(font);

        // Get classification information.
```

```
        String prediction = "Prediction: Normal";

        if (Boolean.TRUE.equals(result.isAnomalous())) {
            prediction = "Prediction: Anomalous";
        }

        // Convert prediction to percentage.
        NumberFormat defaultFormat = NumberFormat.getPercentInstance();
        defaultFormat.setMinimumFractionDigits(1);
        String confidence = String.format("Confidence: %s",
defaultFormat.format(result.confidence()));

        // Draw classification information.
        int yPos = 0;

        yPos = drawLine(g2d, "Classification:", metrics, yPos, Color.WHITE);
        yPos = drawLine(g2d, prediction, metrics, yPos, Color.WHITE);
        yPos = drawLine(g2d, confidence, metrics, yPos, Color.WHITE);

        // Draw segmentation info.
        yPos = drawLine(g2d, "Segmentation:", metrics, yPos, Color.WHITE);

        // Ignore background label, so size must be > 1
        if (result.anomalies().size() > 1) {
            for (Anomaly anomaly : result.anomalies()) {
                if (anomaly.name().equals("background"))
                    continue;
                String label = String.format("Anomaly: %s. Area: %s",
anomaly.name(),

defaultFormat.format(anomaly.pixelAnomaly().totalPercentageArea()));
                Color anomalyColor =
Color.decode((anomaly.pixelAnomaly().color()));
                yPos = drawLine(g2d, label, metrics, yPos, anomalyColor);

            }

        } else {
            drawLine(g2d, "None found.", metrics, yPos, Color.WHITE);

        }

        g2d.dispose();
```

```
    }

    @Override
    public void paintComponent(Graphics g)
    /**
     * Draws the image and analysis results.
     *
     * @param g The Graphics context object for drawing.
     *          DetectAnomalies.
     *
     */
    {

        Graphics2D g2d = (Graphics2D) g; // Create a Java2D version of g.

        // Draw the image.
        g2d.drawImage(image, 0, 0, dimension.width, dimension.height, this);

    }

    // Gets the image mime type. Supported formats are image/jpeg and image/png.

    private String getImageType(byte[] image) throws IOException
    /**
     * Gets the file type of a supplied image. Raises an exception if the image
     * isn't compatible with with Amazon Lookout for Vision.
     *
     * @param image The image that you want to check.
     *
     * @return String The type of the image.
     */

    {
        InputStream is = new BufferedInputStream(new
ByteArrayInputStream(image));
        String mimeType = URLConnection.guessContentTypeFromStream(is);

        logger.log(Level.INFO, "Image type: {0}", mimeType);

        if (mimeType.equals("image/jpeg") || mimeType.equals("image/png")) {
            return mimeType;
        }
        // Not a supported file type.
        logger.log(Level.SEVERE, "Unsupported image type: {0}", mimeType);
```

```
        throw new IOException(String.format("Wrong image type. %s format isn't
supported.", mimeType));
    }

    public static void main(String[] args) throws Exception {

        String photo = null;
        String projectName = null;
        String modelVersion = null;

        final String USAGE = "\n" +
                "Usage:\n" +
                "    DetectAnomalies <project> <version> <image> \n\n" +
                "Where:\n" +
                "    project - The Lookout for Vision project.\n\n" +
                "    version - The version of the model within the project.\n\n"
+
                "    image - The path and filename of a local image. \n\n";

        try {

            if (args.length != 3) {
                System.out.println(USAGE);
                System.exit(1);
            }

            projectName = args[0];
            modelVersion = args[1];
            photo = args[2];
            ShowAnomalies panel = null;

            // Get the Lookout for Vision client.
            LookoutVisionClient lfvClient = LookoutVisionClient.builder()

.credentialsProvider(ProfileCredentialsProvider.create("lookoutvision-access"))
                .build();

            // Create frame and panel.
            JFrame frame = new JFrame(photo);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

            panel = new ShowAnomalies(lfvClient, projectName, modelVersion,
photo);
```

```
            frame.setContentPane(panel);
            frame.pack();
            frame.setVisible(true);


        } catch (LookoutVisionException lfvError) {
            logger.log(Level.SEVERE, "Lookout for Vision client error: {0}:
 {1}",
                    new Object[] { lfvError.awsErrorDetails().errorCode(),
                            lfvError.awsErrorDetails().errorMessage() });
            System.out.println(String.format("lookout for vision client error:
 %s", lfvError.getMessage()));
            System.exit(1);

        } catch (FileNotFoundException fileError) {
            logger.log(Level.SEVERE, "Could not find file: {0}",
 fileError.getMessage());
            System.out.println(String.format("Could not find file: %s",
 fileError.getMessage()));
            System.exit(1);

        } catch (IOException ioError) {
            logger.log(Level.SEVERE, "IO error {0}", ioError.getMessage());
            System.out.println(String.format("IO error: %s",
 ioError.getMessage()));
            System.exit(1);
        }

    }
}
```

4. モデルを引き続き使用する予定がない場合は、モデルを停止してください。

AWS Lambda 関数による異常の検出

AWS Lambda は、サーバーをプロビジョニングまたは管理せずにコードを実行できるようにするコンピューティングサービスです。たとえば、アプリケーションコードをホストするサーバーを作成しなくても、モバイルアプリケーションから送信された画像を分析できます。以下の手引きは、DetectAnomalies を呼び出すLambda 関数を Python で作成する方法を示しています。この関数は提供された画像を分析し、その画像での異常の存在に対する分類を返します。手引きに

は、Amazon S3 バケット内の画像またはローカルコンピュータから提供された画像を使用して Lambda 関数を呼び出す方法を示す Python コードの例が含まれています。

トピック

ステップ 1: AWS Lambda 関数を作成する (コンソール)

このステップでは、空の AWS 関数と、お使いの関数に `DetectAnomalies` オペレーションを呼び出させる IAM 実行ロールを作成します。分析用の画像を保存する Amazon S3 バケットへのアクセス権も付与します。また、以下の環境変数も指定します:

- Lambda 関数に使用させたい Amazon Lookout for Vision プロジェクトとモデルバージョン。
- モデルに使用させたい信頼限度。

後で、ソースコードとレイヤー (オプション) を Lambda 関数に追加します。

AWS Lambda 関数を作成するには (コンソール)

1. AWS Management Console にサインインして AWS Lambda コンソール (https://console.aws.amazon.com/lambda/) を開きます。
2. [関数を作成] を選択します。詳細については、「コンソールで Lambda 関数を作成する」を参照してください。
3. 次のオプションを選択します。
   - [最初から作成] を選択します。
   - [関数名] の値を入力します。
   - [ランタイム] では、[Python 3.10] を選択します。
4. [関数を作成] を選択し AWS Lambda 関数を作成します。
5. 「関数」ページで、[構成] タブを選択します。
6. [環境変数] ペインで、[編集] を選択します。

7. 次の環境変数を追加します。変数ごとに [環境変数を追加] を選択し、変数のキーと値を入力します。

キー	値
PROJECT_NAME	使用したいモデルが含まれている Lookout for Vision プロジェクト。
MODEL_VERSION	使用したいモデルのバージョン。
信頼度	予測が異常であるというモデルの信頼度の最小値 (0 ~ 100)。信頼度が低い場合、分類は正常とみなされます。

8. [保存] を選択して環境変数を保存します。
9. [権限] ペインの [ロール名] で、実行ロールを選択して IAM コンソールで開きます。
10. [権限] タブで、[権限を追加]、[インラインポリシーを作成] をクリックします。
11. [JSON] を選択し、既存の JSON を次のポリシーに置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": "lookoutvision:DetectAnomalies",
            "Resource": "*",
            "Effect": "Allow",
            "Sid": "DetectAnomaliesAccess"
        }
    ]
}
```

12. [次へ] をクリックします。
13. 「ポリシー詳細」に、「DetectAnomalies-access」などのポリシーの名前を入力します。
14. [ポリシーを作成] を選択します。
15. 分析用の画像を Amazon S3 バケットに保存する場合は、ステップ 10 ~ 14 を繰り返します。
    a. ステップ 11 では、次のポリシーを使用します。*####/######* を Amazon S3 バケット、分析する画像へのフォルダパスに置き換えます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "S3Access",
            "Effect": "Allow",
            "Action": "s3:GetObject",
            "Resource": "arn:aws:s3:::bucket/folder path/*"
        }
    ]
}
```

b. ステップ 13 では、S3bucket-access などの別のポリシー名を選択します。

ステップ 2: (オプション) レイヤーを作成する (コンソール)

この例を実行するには、このステップを行う必要はありません。DetectAnomalies オペレーションは、AWS SDK for Python (Boto3) の一部としてデフォルトの Lambda Python 環境に含まれています。Lambda 関数の他の部分で、デフォルトの Lambda Python AWS 環境にはない最新のサービス更新が必要な場合は、このステップを実行して、最新の Boto3 SDK リリースを関数のレイヤーとして追加してください。

まず、Boto3 SDK を含む.zip ファイルアーカイブを作成します。次に、レイヤーを作成し、.zip ファイルアーカイブをそのレイヤーに追加します。詳細については、「Lambda 関数でのレイヤーの使用」を参照してください。

レイヤーを作成して追加するには (コンソール)

1. コマンドプロンプトを開き、次のコマンドを入力します。

```
pip install boto3 --target python/.
zip boto3-layer.zip -r python/
```

2. zip ファイル (boto3-layer.zip) の名前をメモします。それは、この手順のステップ 6 で必要になります。
3. AWS Lambda コンソールを https://console.aws.amazon.com/lambda/ で開きます。
4. ナビゲーションペインで [レイヤー] を選択します。
5. [レイヤーを作成] を選択します。

6. [名前] と [説明] の値を入力します。
7. [.zip ファイルをアップロード]、[アップロード] の順に選択します。
8. ダイアログボックスで、この手順のステップ 1 で作成した.zip ファイルアーカイブ (boto3-layer.zip) を選択します。
9. 互換性のあるランタイムについては、Python 3.9 を選択してください。
10. [作成] を選択してレイヤーを作成します。
11. ナビゲーションペインのメニューアイコンを選択します。
12. ナビゲーションペインで、[関数] を選択します。
13. リソースリストで、ステップ 1: AWS Lambda 関数を作成する (コンソール) で作成した関数を選択します。
14. [コード] タブを選択します。
15. [レイヤー] セクションで、[レイヤーを追加] を選択します。
16. [カスタムレイヤー] を選択します。
17. [カスタムレイヤー] で、ステップ 6 で入力したレイヤー名を選択します。
18. [バージョン] で、レイヤーのバージョン「1」を選択します。
19. [追加] を選択します。

ステップ 3: Python コードを追加する (コンソール)

このステップでは、Lambda コンソールコードエディタを使用して Lambda 関数に Python コードを追加します。コードは提供された画像を `DetectAnomalies` により分析して分類を返します (画像に異常がある場合は true、画像が正常であれば false)。提供される画像は、Amazon S3 バケットに配置したり、byte64 でエンコードされた画像バイトとすることもできます。

Python コードを追加するには (コンソール)

1. Lambda コンソールを使用していない場合は、次の操作を行います:
   a. AWS Lambda コンソールを https://console.aws.amazon.com/lambda/ で開きます。
   b. ステップ 1: AWS Lambda 関数を作成する (コンソール) で作成した Lambda 関数を開きます。
2. [コード] タブを選択します。
3. [コードソース] で、lambda_function.py のコードを次のコードに置き換えます。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose
An AWS lambda function that analyzes images with an Amazon Lookout for Vision
 model.
"""
import base64
import imghdr
from os import environ
from io import BytesIO
import logging
import boto3

from botocore.exceptions import ClientError


logger = logging.getLogger(__name__)


# Get the model and confidence.
project_name = environ['PROJECT_NAME']
model_version = environ['MODEL_VERSION']
min_confidence = int(environ.get('CONFIDENCE', 50))

lookoutvision_client = boto3.client('lookoutvision')


def lambda_handler(event, context):
    """
    Lambda handler function
    param: event: The event object for the Lambda function.
    param: context: The context object for the lambda function.
    return: The labels found in the image passed in the event
    object.
    """

    try:

        file_name = ""

        # Determine image source.
```

```
        if 'image' in event:
            # Decode the encoded image
            image_bytes = event['image'].encode('utf-8')
            img_b64decoded = base64.b64decode(image_bytes)
            image_type = get_image_type(img_b64decoded)
            image = BytesIO(img_b64decoded)
            file_name = event['filename']

        elif 'S3Object' in event:
            bucket = boto3.resource('s3').Bucket(event['S3Object']['Bucket'])
            image_object = bucket.Object(event['S3Object']['Name'])
            image = image_object.get().get('Body').read()
            image_type = get_image_type(image)
            file_name = f"s3://{event['S3Object']['Bucket']}/{event['S3Object']
['Name']}"

        else:
            raise ValueError(
                'Invalid image source. Only base 64 encoded image bytes or images
 in S3 buckets are supported.')

        # Analyze the image.
        response = lookoutvision_client.detect_anomalies(
            ProjectName=project_name,
            ContentType=image_type,
            Body=image,
            ModelVersion=model_version)

        reject = reject_on_classification(
            response['DetectAnomalyResult'],
 confidence_limit=float(environ['CONFIDENCE'])/100)

        status = "anomalous" if reject else "normal"

        lambda_response = {
            "statusCode": 200,
            "body": {
                "Reject": reject,
                "RejectMessage": f"Image {file_name} is {status}."
            }
        }

    except ClientError as err:
        error_message = f"Couldn't analyze {file_name}. " + \
```

```
            err.response['Error']['Message']

        lambda_response = {
            'statusCode': 400,
            'body': {
                "Error": err.response['Error']['Code'],
                "ErrorMessage": error_message,
                "Image": file_name
            }
        }
        logger.error("Error function %s: %s",
                     context.invoked_function_arn, error_message)

    except ValueError as val_error:

        lambda_response = {
            'statusCode': 400,
            'body': {
                "Error": "ValueError",
                "ErrorMessage": format(val_error),
                "Image": event['filename']
            }
        }
        logger.error("Error function %s: %s",
                     context.invoked_function_arn, format(val_error))

    return lambda_response

def get_image_type(image):
    """
    Gets the format of the image. Raises an error
    if the type is not PNG or JPEG.
    :param image: The image that you want to check.
    :return The type of the image.

    """
    image_type = imghdr.what(None, image)

    if image_type == "jpeg":
        content_type = "image/jpeg"
    elif image_type == "png":
        content_type = "image/png"
    else:
        logger.info("Invalid image type")
```

```
        raise ValueError(
            "Invalid file format. Supply a jpeg or png format file.")
    return content_type


def reject_on_classification(prediction, confidence_limit):
    """
    Returns True if the anomaly confidence is greater than or equal to
    the supplied confidence limit.
    :param image: The name of the image file that was analyzed.
    :param prediction: The DetectAnomalyResult object returned from DetectAnomalies
    :param confidence_limit: The minimum acceptable confidence. Float value between
 0 and 1.
    :return: True if the error condition indicates an anomaly, otherwise False.
    """

    reject = False

    if prediction['IsAnomalous'] and prediction['Confidence'] >= confidence_limit:
        reject = True
        reject_info = (f"Rejected: Anomaly confidence
 ({prediction['Confidence']:.2%}) is greater"
                       f" than limit ({confidence_limit:.2%})")
        logger.info("%s", reject_info)

    if not reject:
        logger.info("No anomalies found.")
    return reject
```

4. Lambda 関数をデプロイするには、[デプロイ] を選択します。

ステップ 4: Lambda 関数を試す

このステップでは、コンピューター上の Python コードを使用して、ローカル画像または Amazon S3 バケット内の画像を Lambda 関数に渡します。ローカルコンピュータから渡される画像は 6291456 バイト未満でなければなりません。画像が大きい場合は、画像を Amazon S3 バケットにアップロードし、画像への Amazon S3 パスを使用してスクリプトを呼び出します。Amazon S3 バケットへの画像ファイルのアップロードに関しては、「オブジェクトのアップロード」を参照してください。

必ず Lambda 関数を作成したのと同じ AWS リージョンでコードを実行するようにします。Lambda 関数の AWS リージョンは、Lambda コンソールの関数詳細ページのナビゲーションバーで確認できます。

AWS Lambda 関数がタイムアウトエラーを返す場合は、Lambda 関数のタイムアウト期間を延長します。詳細については、「関数タイムアウトの構成 (コンソール)」を参照してください。

コードから Lambda 関数を呼び出す場合の詳細については、「AWS Lambda 関数の呼び出し」を参照してください。

Lambda 関数を試すには

1. まだの場合は、 以下を実行してください:

   a. クライアントコードを使用するユーザーに `lambda:InvokeFunction` 権限があることを確かめてください。以下の権限を使用できます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LambdaPermission",
            "Effect": "Allow",
            "Action": "lambda:InvokeFunction",
            "Resource": "ARN for lambda function"
        }
    ]
}
```

   Lambda 関数の ARN は、Lambda コンソールの関数概要から取得できます。

   アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

   - AWS IAM Identity Center のユーザーとグループ:

     アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「シークレットの作成と管理」の手順に従ってください。

   - ID プロバイダーを通じて IAM で管理されているユーザー:

ID フェデレーションのロールを作成する。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。

- IAM ユーザー:
  - ユーザーが設定できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (非推奨) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。「IAM ユーザーガイド」の「ユーザー (コンソール) へのアクセス許可の追加」の指示に従います。

b. Python 用 AWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。

c. ステップ 1: AWS Lambda 関数を作成する (コンソール) のステップ 7 で指定したモデルを開始します。

2. 次のコードを client.py という名前のファイルに保存します。

```
# Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
# SPDX-License-Identifier: Apache-2.0

"""
Purpose: Shows how to call the anomaly detection
AWS Lambda function.
"""
from botocore.exceptions import ClientError

import argparse
import logging
import base64
import json
import boto3
from os import environ

logger = logging.getLogger(__name__)

def analyze_image(function_name, image):
    """
    Analyzes an image with an AWS Lambda function.
    :param image: The image that you want to analyze.
    :return The status and classification result for
    the image analysis.
```

```
    """

    lambda_client = boto3.client('lambda')

    lambda_payload = {}

    if image.startswith('s3://'):
        logger.info("Analyzing image from S3 bucket: %s", image)
        bucket, key = image.replace("s3://", "").split("/", 1)
        s3_object = {
            'Bucket': bucket,
            'Name': key
        }
        lambda_payload = {"S3Object": s3_object}

    # Call the lambda function with the image.
    else:
        with open(image, 'rb') as image_file:
            logger.info("Analyzing local image image: %s ", image)
            image_bytes = image_file.read()
            data = base64.b64encode(image_bytes).decode("utf8")
            lambda_payload = {"image": data, "filename": image}

    response = lambda_client.invoke(FunctionName=function_name,
                                    Payload=json.dumps(lambda_payload))
    return json.loads(response['Payload'].read().decode())


def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument(
        "function", help="The name of the AWS Lambda function "
        "that you want to use to analyze the image.")
    parser.add_argument(
        "image", help="The local image that you want to analyze.")


def main():
    """
    Entrypoint for script.
```

```
    """
    try:
        logging.basicConfig(level=logging.INFO,
                            format="%(levelname)s: %(message)s")

        # Get command line arguments.
        parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
        add_arguments(parser)
        args = parser.parse_args()

        # Analyze image and display results.

        result = analyze_image(args.function, args.image)

        status = result['statusCode']

        if status == 200:
            classification = result['body']
            print(f"classification: {classification['Reject']}")
            print(f"Message: {classification['RejectMessage']}")
        else:
            print(f"Error: {result['statusCode']}")
            print(f"Message: {result['body']}")

    except ClientError as error:
        logging.error(error)
        print(error)


if __name__ == "__main__":
    main()
```

3. コードを実行します。コマンドライン引数には、Lambda 関数名と分析するローカル画像へのパスを指定します。例:

```
python client.py function_name /bucket/path/image.jpg
```

完了すると、出力は画像で見つかった異常の分類になります。分類が返されない場合は、ステップ 1: AWS Lambda 関数を作成する (コンソール) のステップ 7 で設定した信頼値を下げることを検討してください。

4. Lambda 関数を使い終わっても、そのモデルが他のアプリケーションで使用されていない場合は、モデルを停止します。次回 Lambda 関数を使用するときには、モデルを開始することを忘れないでください。

エッジデバイスでの Amazon Lookout for Vision モデルの使用

Amazon Lookout for Vision モデルは、AWS IoT Greengrass Version 2 によって管理されるエッジデバイスで使用できます。AWS IoT Greengrass はオープンソースのモノのインターネット (IoT) エッジランタイムおよびクラウドサービスです。これを使用して、デバイス上で IoT アプリケーションの構築、デプロイ、および管理ができます。詳細については、「AWS IoT Greengrass」を参照してください。

クラウドでトレーニングしたのと同じ Amazon Lookout for Vision モデルを、AWS IoT Greengrass V2 互換性のあるエッジデバイスにデプロイします。その後、デプロイしたモデルを使用して、データをクラウドに継続的にストリーミングしなくても、工場などのオンプレミスで異常検出を実行できます。そうすれば、リアルタイムの画像分析により帯域幅コストを最小限に抑え、異常をローカルで検出できます。

> **Tip**
>
> Lookout for Vision モデルを AWS IoT Greengrass でデプロイする前に、「AWS IoT Greengrass Version 2 開発者ガイド」を読むことをお勧めします。詳細については、「AWS IoT Greengrass について」を参照してください。

AWS IoT Greengrass V2 コアデバイスで Lookout for Vision モデルを使用するには、モデルとサポートソフトウェアをコアデバイスにコンポーネントとしてデプロイします。コンポーネントは、Greengrass コアデバイス上で動作する Lookout for Vision モデルなどのソフトウェアモジュールです。コンポーネントには 2 つの形式があります。カスタムコンポーネントは自分で作成し、自分だけがアクセスできるコンポーネントです。プライベートコンポーネントとも呼ばれます。AWS 付属コンポーネントは、AWS が提供するビルド済みコンポーネントです。パブリックコンポーネントとも呼ばれます。詳細については「https://docs.aws.amazon.com/greengrass/v2/developerguide/public-components.html」を参照してください。

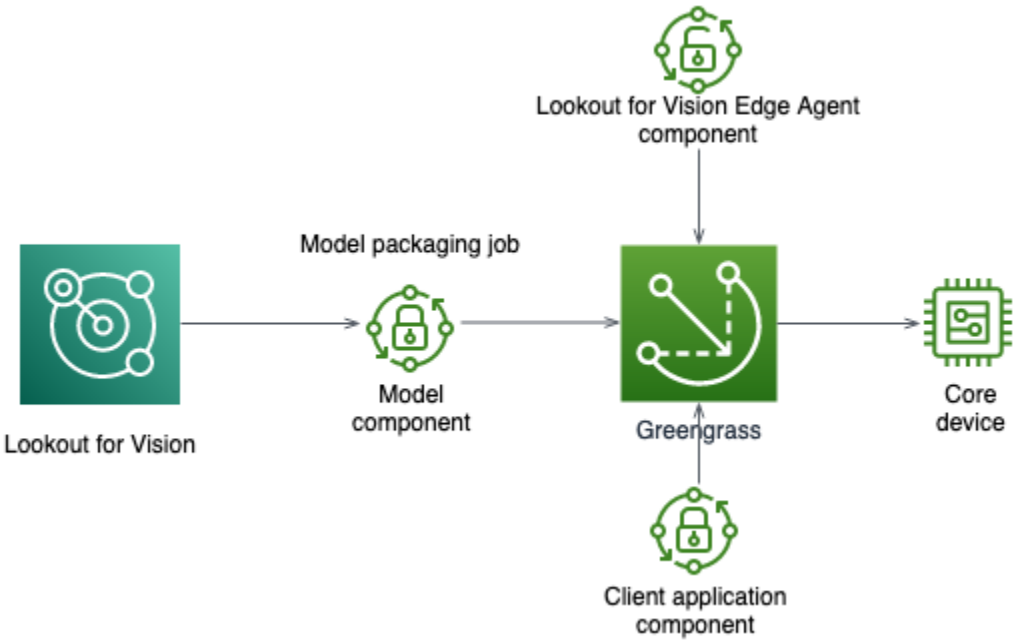
Lookout for Vision モデルのコアデバイスにデプロイするコンポーネントとサポートソフトウェアは次のとおりです:

- モデルコンポーネント。Lookout for Vision モデルを含むカスタムコンポーネント。モデルコンポーネントを作成するには、Lookout for Vision を使用してモデルパッケージングジョブを作成します。モデルパッケージングジョブはモデルのコンポーネントを作成し、そのコンポーネントを

AWS IoT Greengrass V2 内でカスタムコンポーネントとして使用できるようにします。詳細については「Amazon Lookout for Vision モデルのパッケージング」を参照してください。

- クライアントアプリケーションコンポーネント。ビジネス要件に合ったコードを実装する、自分で作成するカスタムコンポーネント。たとえば、組み立て後に撮影した画像から異常な回路基板を見つけたりするものです。詳細については「クライアントアプリケーションコンポーネントの記述」を参照してください。
- Amazon Lookout for Vision Edge Agent コンポーネント。モデルの使用と管理のための API を提供する AWS 付属コンポーネント。たとえば、クライアントアプリケーションコンポーネントのコードは `DetectAnomalies` API を使用して画像の異常を検出できます。Lookout for Vision Edge Agent コンポーネントは、モデルコンポーネントに依存しています。モデルコンポーネントをデプロイすると、コアデバイスに自動的にインストールされます。詳細については「Amazon Lookout for Vision Edge Agent API リファレンス」を参照してください。

モデルコンポーネントとクライアントアプリケーションコンポーネントを作成したら、AWS IoT Greengrass V2 を使用してコンポーネントと依存関係をコアデバイスにデプロイできます。詳細については「デバイスへのコンポーネントのデプロイ」を参照してください。



> **Important**
>
> コアデバイスでモデルが `DetectAnomalies` により行う予測は、クラウドでホストされている同じモデルを使用した予測とは異なる場合があります。製作環境で使用する前に、コアデバイスでモデルをテストすることをお勧めします。

デバイスホストモデルとクラウドホストモデル間の予測の不一致を減らすには、トレーニングデータセット内の正常画像と異常画像の数を増やすことをお勧めします。トレーニングデータセットのサイズを増やすために既存の画像を再利用することはお勧めしません。

AWS IoT Greengrass Version 2 コアデバイスへのモデルとクライアントアプリケーションコンポーネントのデプロイ

Amazon Lookout for Vision モデルとクライアントアプリケーションコンポーネントを AWS IoT Greengrass Version 2 コアデバイスでデプロイする手続きは次のとおりです:

1. AWS IoT Greengrass Version 2 によりコアデバイスをセットアップします。
2. Lookout for Vision を使用してモデルパッケージングジョブを作成します。このジョブはモデルコンポーネントを作成します。
3. クライアントアプリケーションコンポーネントを記述します。このコンポーネントはビジネスロジックを実装します。
4. AWS IoT Greengrass V2 を使用して、コアデバイスにモデルコンポーネントとクライアントアプリケーションコンポーネントをデプロイします。

コンポーネントと依存関係をコアデバイスにデプロイしたら、そのモデルをコアデバイスで使用できます。

> **Note**
>
> Lookout for Vision モデルとクライアントアプリケーションコンポーネントを作成してデプロイするには、同じ AWS リージョンと AWS アカウントを使用する必要があります。

AWS IoT Greengrass Version 2 コアデバイス要件

Amazon Lookout for Vision モデルを AWS IoT Greengrass Version 2 コアデバイスで使用する場合、そのモデルにはコアデバイスに関するさまざまな要件があります。

トピック

- テスト済みのデバイス、チップアーキテクチャー、およびオペレーティングシステム
- コアデバイスのメモリとストレージ

- 必要なソフトウェア

テスト済みのデバイス、チップアーキテクチャー、およびオペレーティングシステム

Amazon Lookout for Vision は以下のハードウェアで動作させることを想定しています:

- CPU アーキテクチャ
  - X86_64 (x86 命令セットの 64 ビットバージョン)
  - Aarch64 (ARMv8 64 ビット CPU)
- (GPU 高速化推論のみ) 十分なメモリ容量 (実行中のモデルにおいて 6.0 GB 以上) を備えた NVIDIA GPU アクセラレーター。

Amazon Lookout for Vision チームは、以下のデバイス、チップアーキテクチャ、およびオペレーティングシステムで Lookout for Vision モデルをテストしました。

デバイス

デバイス	オペレーティングシステム	アーキテクチャ	アクセラレーター	コンパイラオプション
jetson_xavier (NVIDIA® Jetson AGX Xavier)	Linux	Aarch64	NVIDIA	{"gpu-code": "sm_72", "trt-ver": "7.1.3", "cuda-ver": "10.2"} {"gpu-code": "sm_72", "trt-ver": "8.2.1", "cuda-ver": "10.2"}

デバイス	オペレーティングシステム	アーキテクチャ	アクセラレーター	コンパイラオプション
g4dn.xlarge (NVIDIA T4 Tensor Core GPU を搭載した EC2 インスタンス (G4))	Linux	X86_64/X86-64	NVIDIA	{"gpu-code": "sm_75", "trt-ver": "7.1.3", "cuda-ver": "10.2"}
g5.xlarge (NVIDIA A10G Tensor コア GPU を搭載した EC2 インスタンス (G5))	Linux	X86_64/X86-64	NVIDIA	{"gpu-code": "sm_80", "trt-ver": "8.2.0", "cuda-ver": "11.2"}
c5.2xlarge (Amazon EC2 C5 インスタンス)	Linux	X86_64/X86-64	CPU	{"mcpu": "core-avx2"}

コアデバイスのメモリとストレージ

単一モデルと Amazon Lookout for Vision Edge Agent を実行する場合、コアデバイスにはメモリとストレージにおいて次の要件があります。クライアントアプリケーションコンポーネントには、より多くのメモリとストレージが必要になる場合があります。

- ストレージ — 1.5 GB 以上。
- メモリ — 実行中のモデルにおいて 6.0 GB 以上。

必要なソフトウェア

コアデバイスには以下のソフトウェアが必要です。

Jetson デバイス

コアデバイスが Jetson デバイスの場合は、以下のソフトウェアがコアデバイスにインストールされている必要があります。

ソフトウェア	サポートバージョン
Jetpack SDK	4.4 ~ 4.6.1
Lookout for Vision Edge Agent バージョン 1.x 用 Python および Python 仮想環境	3.8 または 3.9

X86 ハードウェア

コアデバイスが x86 ハードウェアを使用している場合は、次のソフトウェアがコアデバイスにインストールされている必要があります。

CPU 推論

ソフトウェア	サポートバージョン
Lookout for Vision Edge Agent バージョン 1.x 用 Python および Python 仮想環境	3.8 または 3.9

GPU 高速化推論

ソフトウェアバージョンは、使用している NVIDIA GPU のマイクロアーキテクチャによって異なります。

Ampere より前のマイクロアーキテクチャを採用した NVIDIA GPU (処理能力 8.0 未満)

Ampere より前のマイクロアーキテクチャの NVIDIA GPU (処理能力 8.0 未満) に必要なソフトウェア。`gpu-code` は `sm_80` より小さくなければなりません。

ソフトウェア	サポートバージョン
NVIDIA CUDA	10.2

ソフトウェア	サポートバージョン
NVIDIA TensorRT	7.1.3 以上 8.0.0 未満
Lookout for Vision Edge Agent バージョン 1.x 用 Python および Python 仮想環境	3.8 または 3.9

Ampere マイクロアーキテクチャを搭載した NVIDIA GPU (処理能力 8.0)

Ampere マイクロアーキテクチャを搭載した NVIDIA GPU (処理能力 8.0) に必要なソフトウェア。`gpu-code` は `sm_80` でなければなりません。

ソフトウェア	サポートバージョン
NVIDIA CUDA	11.2
NVIDIA TensorRT	8.2.0
Lookout for Vision Edge Agent バージョン 1.x 用 Python および Python 仮想環境	3.8 または 3.9

AWS IoT Greengrass Version 2 コアデバイスのセットアップ

Amazon Lookout for Vision は AWS IoT Greengrass Version 2 を使って、AWS IoT Greengrass V2 コアデバイスへの、モデルコンポーネント、Amazon Lookout for Vision Edge Agent コンポーネント、およびクライアントアプリケーションコンポーネントのデプロイメントを簡略化します。デバイスが使用できるデバイスとハードウェアの詳細については「AWS IoT Greengrass Version 2 コアデバイス要件」を参照してください。

コアデバイスのセットアップ

次の情報を使用してコアデバイスをセットアップします。

デバイスをセットアップするには

1. GPU をセットアップします。GPU 高速化推論を使用していない場合は、このステップを実行しないでください。

a. CUDA をサポートする GPU が搭載されていることを確認してください。詳細については、「CUDA 対応 GPU を搭載することを確認する」を参照してください。

b. 次のいずれかを行って、デバイスで CUDA、cuDNN、TensorRT をセットアップします:

- Jetson デバイスを使用している場合は、JetPack バージョン 4.4 ~ 4.6.1 をインストールしてください。詳細については、「JetPack アーカイブ」を参照してください。
- x86 ベースのハードウェアを使用していて、使用している NVIDIA GPU マイクロアーキテクチャが Ampere (処理能力 8.0 未満) 以前である場合、次の手順を行います:
  1. 「Linux 用 NVIDIA CUDA インストールガイド」の手引きに従って、CUDA バージョン 10.2 をセットアップします。
  2. 「NVIDIA cuDNN ドキュメンテーション」の手引きに従って、cuDNN をインストールします。
  3. 「NVIDIA TensorRT ドキュメンテーション」の手引きに従って、TensorRT (バージョン 7.1.3 以降でバージョン 8.0.0 より前) をセットアップします。
- x86 ベースのハードウェアを使用していて、NVIDIA GPU マイクロアーキテクチャが Ampere (処理能力 8.0) の場合、次の手順を実行します。
  1. 「Linux 用 NVIDIA CUDA インストールガイド」の手引きに従って、CUDA (バージョン 11.2) をセットアップします。
  2. 「NVIDIA cuDNN ドキュメンテーション」の手引きに従って、cuDNN をインストールします。
  3. 「NVIDIA TensorRT ドキュメンテーション」の手引きに従って、TensorRT (バージョン 8.2.0) をセットアップします。

2. コアデバイスに AWS IoT Greengrass Version 2 コアソフトウェアをインストールします。詳細については、「AWS IoT Greengrass Version 2 開発者ガイド」の「AWS IoT Greengrass Core ソフトウェアをインストールする」を参照してください。

3. モデルを格納する Amazon S3 バケットから読み取るには、AWS IoT Greengrass Version 2 セットアップ時に作成する IAM ロール (トークン交換ロール) に権限をアタッチします。詳細については、「コンポーネントアーティファクトの S3 バケットへのアクセスを許可する」を参照してください。

4. コマンドプロンプトで次のコマンドを入力して、Python と Python 仮想環境をコアデバイスにインストールします。

```
sudo apt install python3.8 python3-venv python3.8-venv
```

5. 次のコマンドを使用して、Greengrass ユーザーをビデオグループに追加します。これにより、Greengrass でデプロイされたコンポーネントを GPU にアクセスさせられます:

```
sudo usermod -a -G video ggc_user
```

6. (オプション) 別のユーザーから Lookout for Vision Edge Agent API を呼び出したい場合は、必要なユーザーを ggc_group に追加してください。これにより、ユーザーを Unix Domain ソケット経由で Lookout for Vision Edge Agent と通信させられます:

```
sudo usermod -a -G ggc_group $(whoami)
```

Amazon Lookout for Vision モデルのパッケージング

モデルパッケージングジョブでは、Amazon Lookout for Vision モデルをモデルコンポーネントとしてパッケージングします。

モデルパッケージングジョブを作成するには、パッケージングするモデルを選択し、そのジョブによって作成されるモデルコンポーネントの設定を指定します。正常にトレーニングされたモデルのみをパッケージングできます。

Lookout for Vision コンソールまたは AWS SDK を使用して、モデルパッケージングジョブを作成できます。作成したモデルパッケージングジョブに関する情報を取得することもできます。詳細については「モデルパッケージングジョブに関する情報の取得」を参照してください。AWS IoT Greengrass V2 コンソールまたは AWS SDK を使用して、コンポーネントを AWS IoT Greengrass Version 2 コアデバイスにデプロイできます。

トピック

- パッケージング設定
- モデルのパッケージング (コンソール)
- モデルのパッケージング (SDK)
- モデルパッケージングジョブに関する情報の取得

パッケージング設定

次の情報を使用して、モデルパッケージングジョブのパッケージ設定を決定します。

モデルパッケージングジョブを作成するには、「モデルのパッケージング (コンソール)」または「モデルのパッケージング (SDK)」を参照してください。

トピック

- 対象ハードウェア
- コンポーネント設定

対象ハードウェア

モデルに適した対象デバイスまたは対象プラットフォームを選択できますが、両方は選択できません。詳細については「テスト済みのデバイス、チップアーキテクチャー、およびオペレーティングシステム」を参照してください。

対象デバイス

モデルの対象デバイス (NVIDIA® Jetson AGX Xavier など)。コンパイラオプションを指定する必要はありません。

対象プラットフォーム

Amazon Lookout for Vision は次のプラットフォーム構成をサポートします:

- X86_64 (x86 命令セットの 64 ビットバージョン) と Aarch64 (ARMv8 64 ビット CPU) アーキテクチャ。
- Linux オペレーティングシステム。
- NVIDIA または CPU アクセラレータを使用した推論。

対象プラットフォームに適したコンパイラオプションを指定する必要があります。

コンパイラオプション

コンパイラオプションでは、AWS IoT Greengrass Version 2 コアデバイスの対象プラットフォームを指定できます。現在、以下のコマンドオプションを指定できます。

NVIDIA アクセラレーター

- `gpu-code` — モデルコンポーネントを実行するコアデバイスの GPU コードを指定します。
- `trt-ver` — x.y.z. 形式の TensorRT バージョンを指定します。
- `cuda-ver` — CUDA バージョンを x.y 形式で指定します。

CPU アクセラレーター

- (オプション) mcpu — 命令セットを指定します。例えば、「core-avx2」と入力します。値を指定しない場合、Lookout for Vision は値 core-avx2 を使用します。

オプションは JSON 形式で指定します。例:

```
{"gpu-code": "sm_75", "trt-ver": "7.1.3", "cuda-ver": "10.2"}
```

その他の例については、「テスト済みのデバイス、チップアーキテクチャー、およびオペレーティングシステム」を参照してください。

コンポーネント設定

モデルパッケージングジョブは、モデルを含むモデルコンポーネントを作成します。このジョブでは、モデルコンポーネントをコアデバイスにデプロイするために AWS IoT Greengrass V2 が使用するアーティファクトを作成します。

既存のコンポーネントと同じコンポーネント名とコンポーネントバージョンでモデルコンポーネントを作成することはできません。

コンポーネント名

Lookout for Vision がモデルパッケージング中に作成するモデルコンポーネントの名前。指定したコンポーネント名が AWS IoT Greengrass V2 コンソールに表示されます。このコンポーネント名は、クライアントアプリケーションコンポーネント用に作成するレシピで使用します。詳細については「クライアントアプリケーションコンポーネントの作成」を参照してください。

コンポーネントの説明

(オプション) モデルコンポーネントの説明。

コンポーネントのバージョン

モデルコンポーネントのバージョン番号。デフォルトのバージョン番号を使用するか、独自のバージョン番号を選択します。バージョン番号はセマンティックバージョン番号システム（major.minor.patch）に従う必要があります。例えば、バージョン 1.0.0 は、コンポーネントの最初のメジャーリリースを表しています。詳細については、「セマンティックバージョニング 2.0.0」を参照してください。値を指定しない場合、Lookout for Vision はモデルのバージョン番号を使用してバージョンを生成します。

コンポーネントの場所

モデルパッケージングジョブでモデルコンポーネントのアーティファクトを保存する Amazon S3 の場所。Amazon S3 バケットは、使用する同じ AWS リージョンと AWS アカウントに存在する必要がありますAWS IoT Greengrass Version 2。S3 バケットを作成するには、「バケットの作成」を参照してください。

タグ

タグを使用して、コンポーネントを識別、整理、検索、フィルタリングできます。各タグは、ユーザー定義のキーと値で構成されるラベルです。タグは、モデルパッケージングジョブが Greengrass でモデルコンポーネントを作成するときにモデルコンポーネントにアタッチされます。コンポーネントは AWS IoT Greengrass V2 リソースです。タグは、モデルなどの Lookout for Vision リソースにはアタッチされません。詳細については、「AWS リソースのタギング」を参照してください。

モデルのパッケージング (コンソール)

Amazon Lookout for Vision コンソールを使用して、モデルパッケージングジョブを作成できます。

パッケージング設定については、「パッケージング設定」を参照してください。

モデルをパッケージングするには (コンソール)

1. Lookout for Vision がパッケージングジョブのアーティファクト (モデルコンポーネント) を保存するために使用するよう、Amazon S3 バケットを作成するか、既存のバケットを再利用します。
2. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
3. [開始する] を選択します。
4. 左ナビゲーションペインで [プロジェクト] を選択します。
5. [プロジェクト] セクションで、パッケージングしたいモデルを含むプロジェクトを選択します。
6. 左ナビゲーションペインで、プロジェクト名の下にある [エッジモデルパッケージ] を選択します。
7. [モデルパッケージングジョブ] セクションで、[モデルパッケージングジョブを作成] を選択します。
8. パッケージの設定を入力します。詳細については「パッケージング設定」を参照してください。
9. [モデルパッケージングジョブを作成] を選択します。

10. パッケージングジョブが終了するまでお待ちください。ジョブのステータスが「成功」になると、ジョブは終了します。
11. [モデルパッケージングジョブ] セクションでパッケージングジョブを選択します。
12. AWS IoT Greengrass Version 2 でのモデルコンポーネントのデプロイメントを続行するには「Greengrass でのデプロイメントを続行」を選択します。詳細については「デバイスへのコンポーネントのデプロイ」を参照してください。

モデルのパッケージング (SDK)

モデルパッケージングジョブを作成して、モデルをモデルコンポーネントとしてパッケージングします。モデルパッケージングジョブを作成するには、StartModelPackagingJob API を呼び出します。ジョブは完了までに時間がかかる場合があります。現在のステータスを確認するには、DescribeModelPackagingJob を呼び出して、レスポンス内の`Status`フィールドを確認します。

パッケージング設定については、「パッケージング設定」を参照してください。

次の手続きは、AWS CLI を使用してパッケージングジョブを開始する方法を示しています。対象プラットフォームか対象デバイスにモデルをパッケージングできます。Java コードの例については、「StartModelPackagingJob」を参照してください。

モデルをパッケージングするには (SDK)

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. モデルパッケージングジョブを開始するための適切な権限があることを確かめてください。詳細については、「StartModelPackagingJob」を参照してください。
3. 次の CLI コマンドを使用して、対象デバイスまたは対象プラットフォーム用にモデルをパッケージングします。

   Target platform

   次の CLI コマンドは、NVIDIA アクセラレータにより対象プラットフォーム用のモデルをパッケージングする方法を示しています。

   以下の値を変更します:

   - `project_name` を、パッケージングしたいモデルを含むプロジェクトの名前に。
   - `model_version` を、パッケージングしたいモデルのバージョンに。

- (オプション) description を、モデルパッケージングジョブの説明に。
- architecture を、モデルコンポーネントを実行する AWS IoT Greengrass Version 2 コアデバイスのアーキテクチャ (ARM64 または X86_64) に。
- gpu_code を、モデルコンポーネントを実行するコアデバイスの GPU コードに。
- trt_ver を、コアデバイスにインストールした TensorRT バージョンに。
- cuda_verを、コアデバイスにインストールした CUDA バージョンに。
- component_nameを、AWS IoT Greengrass V2 で作成したいモデルコンポーネントの名前に。
- (オプション) component_version を、パッケージングジョブが作成するモデルコンポーネントのバージョンに。形式 major.minor.patch を使用します。例えば、1.0.0 はコンポーネントの最初のメジャーリリースを表します。
- bucket を、パッケージングジョブがモデルコンポーネントのアーティファクトを格納する Amazon S3 バケットに。
- prefix を、パッケージングジョブがモデルコンポーネントのアーティファクトを格納する Amazon S3 バケット内の場所に。
- (オプション) component_description を、モデルコンポーネントの説明に。
- (オプション) tag_key1 および tag_key2 を、モデルコンポーネントにアタッチされているタグのキーに。
- (オプション) tag_value1 および tag_value2 を、モデルコンポーネントにアタッチされているタグのキー値に。

```
aws lookoutvision start-model-packaging-job \
  --project-name project_name \
  --model-version model_version \
  --description="description" \
  --configuration
 "Greengrass={TargetPlatform={Os='LINUX',Arch='architecture',Accelerator='NVIDIA'},Compi
code\": \"gpu_code\", \"trt-ver\": \"trt_ver\", \"cuda-ver\":
 \"cuda_ver\"}',S3OutputLocation={Bucket='bucket',Prefix='prefix'},ComponentName='Compon
 {Key='tag_key2',Value='tag_value2'}]}" \
  --profile lookoutvision-access
```

例:

```
aws lookoutvision start-model-packaging-job \
```

```
  --project-name test-project-01 \
  --model-version 1 \
  --description="Model Packaging Job for G4 Instance using TargetPlatform
 Option" \
  --configuration
 "Greengrass={TargetPlatform={Os='LINUX',Arch='X86_64',Accelerator='NVIDIA'},CompilerOpt
code\": \"sm_75\", \"trt-ver\": \"7.1.3\", \"cuda-ver\":
 \"10.2\"}',S3OutputLocation={Bucket='bucket',Prefix='test-project-01/
folder'},ComponentName='SampleComponentNameX86TargetPlatform',ComponentVersion='0.1.0',C
 is my component',Tags=[{Key='modelKey0',Value='modelValue'},
 {Key='modelKey1',Value='modelValue'}]}" \
  --profile lookoutvision-access
```

Target Device

対象デバイスのモデルをパッケージングするには、次の CLI コマンドを使用します。

以下の値を変更します:

- `project_name` を、パッケージングしたいモデルが含まれているプロジェクトの名前に。
- `model_version` を、パッケージングしたいモデルのバージョンに。
- (オプション) `description` を、モデルパッケージングジョブの説明に。
- `component_name` を、AWS IoT Greengrass V2 で作成したいモデルコンポーネントの名前に。
- (オプション) `component_version` を、パッケージングジョブが作成するモデルコンポーネントのバージョンに。形式 `major.minor.patch` を使用します。例えば、1.0.0 はコンポーネントの最初のメジャーリリースを表します。
- `bucket` を、パッケージングジョブがモデルコンポーネントのアーティファクトを格納する Amazon S3 バケットに。
- `prefix` を、パッケージングジョブがモデルコンポーネントのアーティファクトを格納する Amazon S3 バケット内の場所に。
- (オプション) `component_description` を、モデルコンポーネントの説明に。
- (オプション) `tag_key1` および `tag_key2` を、モデルコンポーネントにアタッチされているタグのキーに。
- (オプション) `tag_value1` および `tag_value2` を、モデルコンポーネントにアタッチされているタグのキー値に。

```
aws lookoutvision start-model-packaging-job \
  --project-name project_name \
  --model-version model_version \
  --description="description" \
  --configuration
 "Greengrass={TargetDevice='jetson_xavier',S3OutputLocation={Bucket='bucket',Prefix='pre
 {Key='tag_key2',Value='tag_value2'}]}" \
  --profile lookoutvision-access
```

例:

```
aws lookoutvision start-model-packaging-job \
  --project-name project_01 \
  --model-version 1 \
  --description="description" \
  --configuration
 "Greengrass={TargetDevice='jetson_xavier',S3OutputLocation={Bucket='bucket',Prefix='com
 model component',Tags=[{Key='tag_key1',Value='tag_value1'},
 {Key='tag_key2',Value='tag_value2'}]}" \
  --profile lookoutvision-access
```

4. レスポンス内の JobName の値に注意してください。それは次のステップで必要です。例:

```
{
    "JobName": "6bcfd0ff-90c3-4463-9a89-6b4be3daf972"
}
```

5. DescribeModelPackagingJob を使ってジョブの現在のステータスを取得します。以下の変更を加えます:

   - project_name を、使用するプロジェクトの名前に。
   - job_name を、前のステップで書き留めたジョブの名前に。

```
aws lookoutvision describe-model-packaging-job \
    --project-name project_name \
    --job-name job_name \
    --profile lookoutvision-access
```

`Status` の値が `SUCCEEDED` であれば、モデルパッケージングジョブは完了です。値が異なる場合は、1 分待ってからやり直してください。

6. AWS IoT Greengrass V2 を使用してデプロイメントを続行します。詳細については「デバイスへのコンポーネントのデプロイ」を参照してください。

モデルパッケージングジョブに関する情報の取得

Amazon Lookout for Vision コンソールと AWS SDK を使用して、作成したモデルパッケージジョブに関する情報を取得できます。

トピック

- モデルパッケージングジョブ情報の取得 (コンソール)
- モデルパッケージングジョブ情報の取得 (SDK)

モデルパッケージングジョブ情報の取得 (コンソール)

モデルパッケージングジョブ情報を取得するには (コンソール)

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [開始する] を選択します。
3. 左ナビゲーションペインで、[プロジェクト] を選択します。
4. [プロジェクト] セクションで、トレーニングしたいモデルパッケージングジョブを含むプロジェクトを選択します。
5. 左ナビゲーションペインで、プロジェクト名の下にある [Edge モデルパッケージ] を選択します。
6. [モデルパッケージングジョブ] セクションで、表示するモデルパッケージングジョブを選択します。モデルパッケージングジョブの詳細ページが表示されます。

モデルパッケージングジョブ情報の取得 (SDK)

AWS SDK を使用して、プロジェクト内のモデルパッケージングジョブをリストし、特定のモデルパッケージングジョブに関する情報を取得できます。

モデルパッケージングジョブをリストする

ListModelPackagingJobs API を呼び出すと、プロジェクト内のモデルパッケージングジョブをリストできます。レスポンスには、各モデルパッケージングジョブに関する情報を提供する ModelPackagingJobMetadata オブジェクトのリストが含まれます。また、リストが不完全な場合、次の結果セットを取得するために使用できるページネーショントークンが含まれます。

モデルパッケージングジョブをリストするには

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次の CLI コマンドを使用します。使用したいプロジェクト名前に project_name を変更します。

```
aws lookoutvision list-model-packaging-jobs \
  --project-name project_name \
  --profile lookoutvision-access
```

モデルパッケージングジョブを説明する

DescribeModelPackagingJob API を使用して、モデルパッケージングジョブに関する情報を取得します。レスポンスは、ジョブの現在のステータスや他の情報を含む ModelPackagingDescription オブジェクトです。

パッケージを説明するには

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次の CLI コマンドを使用します。以下の変更を加えます:

   - project_name を、使用するプロジェクトの名前に。
   - job_name を、ジョブの名前に。StartModelPackagingJob を呼び出すと、ジョブ名 (JobName) が取得されます。

```
aws lookoutvision describe-model-packaging-job \
    --project-name project_name \
    --job-name job_name \
    --profile lookoutvision-access
```

クライアントアプリケーションコンポーネントの記述

クライアントアプリケーションコンポーネントは、ユーザーが作成するカスタム AWS IoT Greengrass Version 2 コンポーネントです。Amazon Lookout for Vision モデルを AWS IoT Greengrass Version 2 コアデバイスで使用するために必要なビジネスロジックを実装します。

モデルにアクセスするために、クライアントアプリケーションコンポーネントは Lookout for Vision Edge Agent コンポーネントを使用します。Lookout for Vision Edge Agent コンポーネントは、モデルを含む画像を分析したり、コアデバイス上でモデルを管理したりするための API を提供します。

Lookout for Vision Edge Agent API は gRPC を使用して実装されます。gRPC は遠隔手続き呼び出しを行うためのプロトコルです。詳細については、「gRPC」を参照してください。コードを記述するには、gRPC がサポートする任意の言語を使用できます。Python コードの例を提供します。詳細については「クライアントアプリケーションコンポーネントでのモデルの使用」を参照してください。

> **Note**
>
> Lookout for Vision Edge Agent コンポーネントは、デプロイするモデルコンポーネントの依存関係です。モデルコンポーネントをコアデバイスにデプロイすると、コアデバイスに自動的にデプロイされます。

クライアントアプリケーションコンポーネントを記述するには、次を行います。

1. gRPC を使用するように環境をセットアップし、サードパーティのライブラリをインストールします。
2. モデルを使用するコードを記述します。
3. コアデバイスにコードをカスタムコンポーネントとしてデプロイします。

カスタム GStreamer パイプラインで異常検出を実行する方法を示すクライアントアプリケーションコンポーネントの例については、https://github.com/awslabs/aws-greengrass-labs-lookoutvision-gstreamer を参照してください。

環境のセットアップ

クライアントコードを記述するために、ご利用の開発環境は Amazon Lookout for Vision モデルコンポーネントおよび依存関係をデプロイした AWS IoT Greengrass Version 2 コアデバイスにリモート

接続します。または、コアデバイスにコードを記述することができます。詳細については、「AWS IoT Greengrass 開発ツール」と「AWS IoT Greengrass コンポーネントを開発する」を参照してください。

Amazon Lookout for Vision Edge Agent にアクセスするには、クライアントコードが gRPC クライアントを使用する必要があります。このセクションでは、gRPC を使用して開発環境をセットアップし、`DetectAnomalies` サンプルコードに必要なサードパーティの依存関係をインストールする方法を示します。

クライアントコードを記述し終えたら、カスタムコンポーネントを作成し、そのカスタムコンポーネントをエッジデバイスにデプロイします。詳細については「クライアントアプリケーションコンポーネントの作成」を参照してください。

トピック

- gRPC のセットアップ
- サードパーティー依存関係の追加

gRPC のセットアップ

開発環境では、Lookout for Vision Edge Agent API を呼び出すためにコード内で使用する gRPC クライアントが必要です。そのためには、Lookout for Vision Edge Agent 用の `.proto` サービス定義ファイルを使用して gRPC スタブを作成します。

> **Note**
>
> Lookout for Vision Edge Agent アプリケーションバンドルからサービス定義ファイルを入手することもできます。アプリケーションバンドルは、Lookout for Vision Edge Agent コンポーネントがモデルコンポーネントの依存関係としてインストールされたときにインストールされます。アプリケーションバンドルは `/greengrass/v2/packages/artifacts-unarchived/aws.iot.lookoutvision.EdgeAgent/`*`edge_agent_version`*`/lookoutvision_edge_agent` にあります。使用している Lookout for Vision Edge Agent のバージョンに `edge_agent_version` を置き換えてください。アプリケーションバンドルを入手するには、Lookout for Vision Edge Agent をコアデバイスにデプロイする必要があります。

gRPC をセットアップするには

1. zip ファイル proto.zip をダウンロードします。zip ファイルには .proto サービス定義ファイル (edge-agent.proto) が含まれています。
2. ファイルを解凍します。
3. コマンドプロンプトを開いて edge-agent.proto を含むフォルダに移動します。
4. 次のコマンドを使用して、Python クライアントインターフェイスを生成します。

   ```
   %%bash
   python3 -m pip install grpcio
   python3 -m pip install grpcio-tools
   python3 -m grpc_tools.protoc --proto_path=. --python_out=. --grpc_python_out=.
    edge-agent.proto
   ```

   コマンドが成功すると、edge_agent_pb2_grpc.py と edge_agent_pb2.py のスタブが working ディレクトリに作成されます。
5. モデルを使用するクライアントコードを記述します。詳細については「クライアントアプリケーションコンポーネントでのモデルの使用」を参照してください。

サードパーティー依存関係の追加

DetectAnomalies サンプルコードは Pillow ライブラリを使用して画像を取り扱います。詳細については「画像バイトによる異常の検出」を参照してください。

次のコマンドを使用して Pillow ライブラリをインストールします。

```
python3 -m pip install Pillow
```

クライアントアプリケーションコンポーネントでのモデルの使用

クライアントアプリケーションコンポーネントからのモデルを使用するステップは、クラウドでホストされているモデルを使用するステップと似ています。

1. モデルの実行を開始します。
2. 画像の異常を検出します。
3. もう必要がなければ、モデルを停止します

Amazon Lookout for Vision Edge Agent には、モデルを開始し、画像内の異常を検出し、モデルを停止するための API が用意されています。API を使用してデバイス上のモデルをリストし、デプロイされたモデルに関する情報を取得することもできます。詳細については「Amazon Lookout for Vision Edge Agent API リファレンス」を参照してください。

gRPC ステータスコードをチェックすることでエラー情報を取得できます。詳細については「エラー情報の取得」を参照してください。

コードを記述するには、gRPC がサポートする任意の言語を使用できます。Python コードの例を提供します。

トピック

- クライアントアプリケーションコンポーネントでのスタブの使用
- モデルの開始
- 異常の検出
- モデルの停止
- デバイス上のモデルの一覧表示
- モデルの記述
- エラー情報の取得

クライアントアプリケーションコンポーネントでのスタブの使用

次のコードを使用して、Lookout for Vision Edge Agent を通じてモデルへのアクセスをセットアップします。

```
import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2

# Creating stub.
 with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    # Add additional code that works with Edge Agent in this block to prevent resources
 leakage
```

モデルの開始

StartModel API を呼び出してモデルを開始します。モデルの開始までに時間がかかる場合があります。DescribeModel を呼び出すことで、現在のステータスをチェックできます。status フィールドの値が「実行中」の場合、モデルは実行中です。

コードの例

component_name はモデルコンポーネントの名前に置き換えてください。

```
import time

import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2


model_component_name = "component_name"


def start_model_if_needed(stub, model_name):
    # Starting model if needed.
    while True:
        model_description_response =
 stub.DescribeModel(pb2.DescribeModelRequest(model_component=model_name))
        print(f"DescribeModel() returned {model_description_response}")
        if model_description_response.model_description.status == pb2.RUNNING:
            print("Model is already running.")
            break
        elif model_description_response.model_description.status == pb2.STOPPED:
            print("Starting the model.")
            stub.StartModel(pb2.StartModelRequest(model_component=model_name))
            continue
        elif model_description_response.model_description.status == pb2.FAILED:
            raise Exception(f"model {model_name} failed to start")
        print(f"Waiting for model to start.")
        if model_description_response.model_description.status != pb2.STARTING:
            break
        time.sleep(1.0)


# Creating stub.
```

```
with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    start_model_if_needed(stub, model_component_name)
```

異常の検出

画像内の異常を検出するには DetectAnomalies API を使用します。

DetectAnomalies オペレーションでは、画像ビットマップが RGB888 圧縮形式で渡されることを想定しています。最初のバイトは赤チャンネル、2 番目のバイトは緑チャンネル、3 番目のバイトは青チャンネルを表します。BGR などの別の形式で画像を提供した場合、DetectAnomalies による予測は正しくありません。

デフォルトでは、OpenCV は画像ビットマップに BGR 形式を使用します。OpenCV を使用して DetectAnomalies により分析用の画像をキャプチャする場合は 、画像を DetectAnomalies に渡す前に画像を RGB888 形式に変換する必要があります。

DetectAnomalies に提供する画像は、モデルのトレーニングに使用した画像と同じ幅と高さである必要があります。

画像バイトによる異常の検出

画像を画像バイトとして提供することで、画像内の異常を検出できます。次の例では、ローカルファイルシステムに保存されている画像から画像バイトが取得されます。

sample.jpg を、分析する画像ファイルの名前に置き換えてください。*component_name* を、モデルコンポーネントの名前に置き換えてください。

```
import time

from PIL import Image
import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2


model_component_name = "component_name"

....
# Detecting anomalies.
```

```
def detect_anomalies(stub, model_name, image_path):
    image = Image.open(image_path)
    image = image.convert("RGB")
    detect_anomalies_response = stub.DetectAnomalies(
        pb2.DetectAnomaliesRequest(
            model_component=model_name,
            bitmap=pb2.Bitmap(
                width=image.size[0],
                height=image.size[1],
                byte_data=bytes(image.tobytes())
            )
        )
    )
    print(f"Image is anomalous -
 {detect_anomalies_response.detect_anomaly_result.is_anomalous}")
    return detect_anomalies_response.detect_anomaly_result


# Creating stub.
with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    start_model_if_needed(stub, model_component_name)
    detect_anomalies(stub, model_component_name, "sample.jpg")
```

共有メモリーセグメントを使った異常検知

画像を POSIX 共有メモリセグメントの画像バイトとして提供することで、画像内の異常を検出できます。最高のパフォーマンスを得るために、DetectAnomalies リクエストに共有メモリを使用することをお勧めします。詳細については「DetectAnomalies」を参照してください。

モデルの停止

モデルを使用しなくなった場合は、StopModel API を使用してモデルの実行を停止します。

```
stop_model_response = stub.StopModel(
    pb2.StopModelRequest(
         model_component=model_component_name
    )
)
print(f"New status of the model is {stop_model_response.status}")
```

デバイス上のモデルの一覧表示

the section called “ListModels” API を使用して、デバイスにデプロイされているモデルを一覧表示できます。

```
models_list_response = stub.ListModels(
    pb2.ListModelsRequest()
)
for model in models_list_response.models:
    print(f"Model Details {model}")
```

モデルの記述

DescribeModel API を呼び出すと、デバイスにデプロイされたモデルに関する情報を取得できます。DescribeModel を使用すると、モデルの現在のステータスを取得するのに便利です。たとえば、DetectAnomalies を呼び出す前にモデルが実行中かどうかを知る必要があります。サンプルコードについては、「モデルの開始」を参照してください。

エラー情報の取得

gRPC ステータスコードは API 結果の報告に使用されます。

次の例に示すように、RpcError 例外をキャッチすることでエラー情報を取得できます。エラーステータスコードの詳細については、API のリファレンストピックを参照してください。

```
# Error handling.
try:
    stub.DetectAnomalies(detect_anomalies_request)
except grpc.RpcError as e:
    print(f"Error code: {e.code()}, Status: {e.details()}")
```

クライアントアプリケーションコンポーネントの作成

gRPC スタブを生成し、クライアントアプリケーションコードの準備ができたら、クライアントアプリケーションコンポーネントを作成できます。作成するコンポーネントは、AWS IoT Greengrass Version 2 コアデバイスに AWS IoT Greengrass V2 によりデプロイするカスタムコンポーネントです。作成するレシピには、カスタムコンポーネントが記述されます。レシピには、デプロイする必要もある依存関係すべてが含まれています。この場合、Amazon Lookout for Vision モデルのパッケー

ジング で作成するモデルコンポーネントを指定します。コンポーネントレシピの詳細については、「AWS IoT Greengrass Version 2 コンポーネントレシピリファレンス」を参照してください。

このトピックの手続きでは、レシピファイルからクライアントアプリケーションコンポーネントを作成し、AWS IoT Greengrass V2 カスタムコンポーネントとして公開する方法を示します。AWS IoT Greengrass V2 コンソールまたは AWS SDK を使用してコンポーネントを公開できます。

カスタムコンポーネントの作成について詳しくは、AWS IoT Greengrass V2 ドキュメンテーションで以下を参照してください。

- デバイス上でコンポーネントを開発およびテストする
- AWS IoT Greengrass コンポーネントを作成する
- コアデバイスにデプロイするコンポーネントを公開する

トピック

- クライアントアプリケーションコンポーネントを公開するための IAM 権限
- レシピの作成
- クライアントアプリケーションコンポーネントの公開 (コンソール)
- クライアントアプリケーションコンポーネントの公開 (SDK)

クライアントアプリケーションコンポーネントを公開するための IAM 権限

クライアントアプリケーションコンポーネントを作成して公開するには、次の IAM 権限が必要です。

- `greengrass:CreateComponentVersion`
- `greengrass:DescribeComponent`
- `s3:PutObject`

レシピの作成

この手続きでは、シンプルなクライアントアプリケーションコンポーネントのレシピを作成します。`lookoutvision_edge_agent_example.py` 内のコードには、デバイスにデプロイされるモデルが一覧表示され、コンポーネントをコアデバイスにデプロイすると自動的に実行されます。出力を確認するには、コンポーネントをデプロイした後にコンポーネントログをチェックします。詳細に

ついては「デバイスへのコンポーネントのデプロイ」を参照してください。準備ができたら、以下の手続きを使用して、ビジネスロジックを実装するコードのレシピを作成します。

レシピは JSON または YAML 形式のファイルとして作成します。また、Amazon S3 バケットにクライアントアプリケーションコードをアップロードします。

クライアントアプリケーションコンポーネントのレシピを作成するには

1. まだ作成していない場合、gRPC スタブコマンドを作成します。詳細については「gRPC のセットアップ」を参照してください。
2. 次のコードを lookoutvision_edge_agent_example.py という名前のファイルに保存します。

```
import grpc
from edge_agent_pb2_grpc import EdgeAgentStub
import edge_agent_pb2 as pb2

# Creating stub.
with grpc.insecure_channel("unix:///tmp/aws.iot.lookoutvision.EdgeAgent.sock") as
 channel:
    stub = EdgeAgentStub(channel)
    # Add additional code that works with Edge Agent in this block to prevent
 resources leakage

    models_list_response = stub.ListModels(
        pb2.ListModelsRequest()
    )
    for model in models_list_response.models:
        print(f"Model Details {model}")
```

3. Amazon S3 バケットを作成 (または既存のバケットを使用) して、クライアントアプリケーションコンポーネントのソースファイルを保存します。バケットは AWS アカウント内にあり、AWS IoT Greengrass Version 2 と Amazon Lookout for Vision を使う同じ AWS リージョンに存在する必要があります。
4. 前のステップで作成した Amazon S3 バケットに lookoutvision_edge_agent_example.py、edge_agent_pb2_grpc.py and edge_agent_pb2.py をアップロードします。各ファイルの Amazon S3 パスをメモします。edge_agent_pb2_grpc.py と edge_agent_pb2.py を gRPC のセットアップ で作成しました。
5. エディタで、次の JSON または YAML レシピファイルを作成します。

- model_component を、モデルコンポーネントの名前に。詳細については「コンポーネント設定」を参照してください。
- URI エントリを lookoutvision_edge_agent_example.py、edge_agent_pb2_grpc.py、edge_agent_pb2. の S3 パスに変更します。

JSON

```
{
  "RecipeFormatVersion": "2020-01-25",
  "ComponentName": "com.lookoutvision.EdgeAgentPythonExample",
  "ComponentVersion": "1.0.0",
  "ComponentType": "aws.greengrass.generic",
  "ComponentDescription": "Lookout for Vision Edge Agent Sample Application",
  "ComponentPublisher": "Sample App Publisher",
  "ComponentDependencies": {
    "model_component": {
      "VersionRequirement": ">=1.0.0",
      "DependencyType": "HARD"
    }
  },
  "Manifests": [
    {
      "Platform": {
        "os": "linux"
      },
      "Lifecycle": {
        "install": "pip3 install grpcio grpcio-tools protobuf Pillow",
        "run": {
          "script": "python3 {artifacts:path}/
lookoutvision_edge_agent_example.py"
        }
      },
      "Artifacts": [
        {
          "Uri": "S3 path to lookoutvision_edge_agent_example.py"
        },
        {
          "Uri": "S3 path to edge_agent_pb2_grpc.py"
        },
        {
```

```
            "Uri": "S3 path to edge_agent_pb2.py"
          }
        ]
      }
    ],
    "Lifecycle": {}
}
```

YAML

```
---
RecipeFormatVersion: 2020-01-25
ComponentName: com.lookoutvison.EdgeAgentPythonExample
ComponentVersion: 1.0.0
ComponentDescription: Lookout for Vision Edge Agent Sample Application
ComponentPublisher: Sample App Publisher
ComponentDependencies:
  model_component:
    VersionRequirement: '>=1.0.0'
    DependencyType: HARD
Manifests:
  - Platform:
      os: linux
    Lifecycle:
      install: |-
        pip3 install grpcio
        pip3 install grpcio-tools
        pip3 install protobuf
        pip3 install Pillow
      run:
        script: |-
          python3 {artifacts:path}/lookout_vision_agent_example.py
    Artifacts:
      - URI: S3 path to lookoutvision_edge_agent_example.py
      - URI: S3 path to edge_agent_pb2_grpc.py
      - URI: S3 path to edge_agent_pb2.py
```

6. JSON または YAML ファイルをコンピュータに保存します。
7. 次のいずれかを実行して、クライアントアプリケーションコンポーネントを作成します:

   - AWS IoT Greengrass コンソールを使用したい場合は、[クライアントアプリケーションコンポーネントの公開 (コンソール)] を行います。

- AWS SDK を使用したい場合は、クライアントアプリケーションコンポーネントの公開 (SDK) を行います。

クライアントアプリケーションコンポーネントの公開 (コンソール)

AWS IoT Greengrass V2 コンソールを使用してクライアントアプリケーションコンポーネントを公開できます。

クライアントアプリケーションコンポーネントを公開するには

1. まだ作成していない場合は、レシピの作成 を実行してクライアントアプリケーションコンポーネントのレシピを作成します。
2. AWS IoT Greengrass コンソール (https://console.aws.amazon.com/es/) を開きます。
3. 左ナビゲーションペインの [Greengrass] で [コンポーネント] を選択します。
4. [マイコンポーネント] で [コンポーネントを作成] を選択します。
5. JSON 形式のレシピを使用する場合は、「コンポーネント作成する」ページで [レシピを JSON として入力] を選択します。YAML 形式のレシピを使用する場合は、[レシピを YAML として入力] を選択します。
6. 「レシピ」で、既存のレシピを、レシピの作成 で作成した JSON または YAML レシピに置き換えます。
7. [コンポーネントを作成] を選択します。
8. 次に、クライアントアプリケーションコンポーネントをデプロイします。

クライアントアプリケーションコンポーネントの公開 (SDK)

CreateComponentVersion API を使用してクライアントアプリケーションコンポーネントを公開できます。

クライアントアプリケーションコンポーネントを公開するには (SDK)

1. まだの場合は、レシピの作成 を行ってクライアントアプリケーションコンポーネントのレシピを作成します。
2. コマンドプロンプトで、次のコマンドを入力してクライアントアプリケーションコンポーネントを作成します。`recipe-file` を レシピの作成 で作成したレシピファイルの名前に置き換えます。

```
aws greengrassv2 create-component-version --inline-recipe fileb://recipe-file
```

リスポンスのコンポーネントの ARN をメモします。それは次の手順で必要となります。

3. 以下のコマンドを使用して、クライアントアプリケーションコンポーネントのステータスを取得します。component-arn を、前のステップでメモした ARN に置き換えます。componentState の値が DEPLOYABLE であれば、クライアントアプリケーションコンポーネントは準備完了です。

```
aws greengrassv2 describe-component --arn component-arn
```

4. 次に、クライアントアプリケーションコンポーネントをデプロイします。

デバイスへのコンポーネントのデプロイ

モデルコンポーネントとクライアントアプリケーションコンポーネントを AWS IoT Greengrass Version 2 コアデバイスにデプロイするには、AWS IoT Greengrass V2 コンソールまたは CreateDeployment API を使用します。詳細については、「デプロイメントを作成する」か「AWS IoT Greengrass Version 2 開発者ガイド」を参照してください。コアデバイスにデプロイされるコンポーネントの更新について詳しくは、「デプロイメントを改訂する」を参照してください。

トピック

- コンポーネントのデプロイのための IAM 権限
- コンポーネントのデプロイ (コンソール)
- コンポーネントのデプロイ (SDK)

コンポーネントのデプロイのための IAM 権限

AWS IoT Greengrass V2 によりコンポーネントをデプロイするには、次の権限も必要です:

- greengrass:ListComponents
- greengrass:ListComponentVersions
- greengrass:ListCoreDevices
- greengrass:CreateDeployment
- greengrass:GetDeployment

- greengrass:ListDeployments

CreateDeployment と GetDeployment には依存アクションがあります。詳細については、「AWS IoT Greengrass V2 によって定義されるアクション」を参照してください。

IAM 権限の変更の詳細については、「ユーザーの権限の変更」を参照してください。

コンポーネントのデプロイ (コンソール)

次の手続きを使用して、クライアントアプリケーションコンポーネントをコアデバイスにデプロイします。クライアントアプリケーションはモデルコンポーネントに依存します (逆にモデルコンポーネントは Lookout for Vision Edge エージェントに依存します)。クライアントアプリケーションコンポーネントをデプロイすると、モデルコンポーネントと Lookout for Vision Edge Agent のデプロイメントも開始されます。

> **Note**
>
> コンポーネントを既存のデプロイメントに追加できます。コンポーネントをモノグループにデプロイすることもできます。

この手続きを実行するには、AWS IoT Greengrass V2 コアデバイスが構成されている必要があります。詳細については「AWS IoT Greengrass Version 2 コアデバイスのセットアップ」を参照してください。

デバイスにコンポーネントをデプロイするには

1. https://console.aws.amazon.com/iot/ で AWS IoT Greengrass コンソールを開きます。
2. 左ナビゲーションペインの [Greengrass] で [デプロイメント] を選択します。
3. [デプロイメント] で[作成]を選択します。
4. 「対象を指定する」ページで、次を実行します:
    1. [デプロイメント情報] で、デプロイの名前を入力または変更して、わかりやすくします。
    2. [デプロイメント対象] で [コアデバイス] を選択し、対象名を入力します。
    3. [次へ] をクリックします。
5. 「コンポーネントを選択する」ページで、次の手順を実行します:

    1. [マイコンポーネント] で、クライアントアプリケーションコンポーネントの名前 (com.lookoutvison.EdgeAgentPythonExample) を選択します。
    2. [次へ] を選択します。
6. 「コンポーネントを構成する」ページで、現行の構成を維持して [次へ] を選択します。
7. 「高度な設定を構成する」ページで、現行の設定を維持して [次へ] を選択します。
8. 「見直しをする」ページで [デプロイ] を選択し、コンポーネントのデプロイを開始します。

デプロイメントステータスのチェック (コンソール)

AWS IoT Greengrass V2 コンソールからデプロイメントのステータスをチェックできます。クライアントアプリケーションコンポーネントが the section called “クライアントアプリケーションコンポーネントの作成” のサンプルレシピおよびコードを使用している場合は、デプロイメント完了後にクライアントアプリケーションコンポーネントのログを確認してください。成功すると、ログにはコンポーネントにデプロイされた Lookout for Vision モデルのリストが含まれます。

AWS SDK を使用したデプロイメントステータスのチェックについては、「デプロイメントステータスをチェックする」を参照してください。

デプロイメントステータスをチェックするには

1. AWS IoT Greengrass コンソール (https://console.aws.amazon.com/es/) を開きます。
2. 左ナビゲーションメニューで、[コアデバイス] を選択します。
3. [Greengrass コアデバイス] でコアデバイスを選択します。
4. [デプロイメント] タブを選択すると、現在のデプロイメントステータスが表示されます。
5. デプロイメントが成功 (ステータスが「完了」になります) したら、コアデバイスでターミナルウィンドウを開き、クライアントアプリケーションコンポーネントのログを /greengrass/v2/logs/com.lookoutvison.EdgeAgentPythonExample.log で確認します。デプロイメントでサンプルのレシピとコードを使用している場合、ログには lookoutvision_edge_agent_example.py からの出力が含まれます。例:

```
Model Details model_component:"ModelComponent"
```

コンポーネントのデプロイ (SDK)

以下の手続きを使用して、クライアントアプリケーションコンポーネント、モデルコンポーネント、および Amazon Lookout for Vision Edge Agent をコアデバイスにデプロイします。

1. deployment.json ファイルを作成して、コンポーネントのデプロイメント構成を定義します。このファイルは、次の例のようになります。

```
{
  "targetArn":"targetArn",
  "components": {
    "com.lookoutvison.EdgeAgentPythonExample": {
      "componentVersion": "1.0.0",
      "configurationUpdate": {
      }
    }
  }
}
```

   - [targetArn] フィールドで *targetArn* をデプロイメントの対象となるモノまたはモノのグループの Amazon リソースネーム (ARN) に置き換えます。形式は以下のとおりです:
     - モノ: arn:aws:iot:*region*:*account-id*:thing/*thingName*
     - モノのグループ: arn:aws:iot:*region*:*account-id:*thinggroup/*thingGroupName*
2. デプロイメント対象に、修正が必要な既存のデプロイメントがあるかどうかをチェックします。以下を行います:

   a. 次のコマンドを実行して、デプロイメント対象のデプロイメントを一覧表示します。targetArn を、対象の AWS IoT モノまたはモノグループの Amazon リソースネーム (ARN) に置き換えます。現在の AWS リージョンにあるモノの ARN を取得するには、aws iot list-things コマンドを使用します。

```
aws greengrassv2 list-deployments --target-arn targetArn
```

      レスポンスには、対象の最新デプロイメントのリストが含まれています。レスポンスが空の場合は、対象に既存のデプロイメントがないため、ステップ 3 にスキップできます。そうでない場合は、次のステップで使用するため、レスポンスから deploymentId をコピーします。

b. 次のコマンドを実行して、デプロイメントの詳細を取得します。これらの詳細には、メタデータ、コンポーネント、ジョブ構成が含まれます。deploymentId を前のステップの ID に置き換えます。

```
aws greengrassv2 get-deployment --deployment-id deploymentId
```

c. 前のコマンドのレスポンスにおける以下の key-value ペアをすべて into deployment.json にコピーします。新しいデプロイメントではこれらの値を変更できます。

- deploymentName - デプロイメントの名前。
- components - デプロイメントのコンポーネント。コンポーネントをアンインストールする場合は、このオブジェクトから削除してください。
- deploymentPolicies - デプロイメントのポリシー。
- tags - デプロイメントのタグ。

3. 次のコマンドを実行して、デバイスにコンポーネントをデプロイします。レスポンス内の deploymentId の値をメモしてください。

```
aws greengrassv2 create-deployment \
    --cli-input-json file://path/to/deployment.json
```

4. 次のコマンドを実行して、デプロイのステータスを取得します。deployment-id を前のステップでメモした値に変更します。deploymentStatus の値が COMPLETED であれば、デプロイメントは正常に完了しています。

```
aws greengrassv2 get-deployment --deployment-id deployment-id
```

5. デプロイメントが成功したら、コアデバイスのターミナルウィンドウを開き、/greengrass/v2/logs/com.lookoutvison.EdgeAgentPythonExample.log でクライアントアプリケーションコンポーネントのログを確認します。デプロイメントでサンプルのレシピとコードを使用している場合、ログには lookoutvision_edge_agent_example.py からの出力が含まれます。例:

```
Model Details model_component:"ModelComponent"
```

Amazon Lookout for Vision Edge Agent API リファレンス

このセクションは、Amazon Lookout for Vision Edge エージェントの API リファレンスです。

モデルによる異常の検出

DetectAnomalies API を使用すると、AWS IoT Greengrass Version 2 コアデバイスで実行中モデルを使用して画像内の異常を検出できます。

モデル情報の取得

コアデバイスにデプロイされたモデルに関する情報を取得する API。

- ListModels
- DescribeModel

モデルの実行

コアデバイスにデプロイされている Amazon Lookout for Vision モデルを開始および停止するための API。

- StartModel
- StopModel

DetectAnomalies

付属画像内の異常を検出します。

`DetectAnomalies` からのレスポンスには、画像に 1 つ以上の異常が含まれているというブーリアン予測と、その予測に対する信頼値が含まれます。モデルがセグメンテーションモデルの場合、レスポンスには以下が含まれます:

- 各異常タイプを異なる色でカバーしたマスク画像。`DetectAnomalies` によりマスク画像を共有メモリに保存することも、マスクを画像バイトとして返すこともできます。
- 画像で異常タイプが占める領域のパーセンテージ。
- マスク画像上の異常タイプの 16 進数カラー。

> **Note**
>
> DetectAnomalies により使用するモデルは実行中の必要があります。DescribeModel を呼び出すことで、現在のステータスを取得できます。モデルの実行を開始するには、「StartModel 」を参照してください。

DetectAnomalies はインターリーブ RGB888 形式のパックビットマップ (画像) をサポートします。最初のバイトは赤チャンネル、2 番目のバイトは緑チャンネル、3 番目のバイトは青チャンネルを表します。BGR などの別の形式で画像を提供した場合、DetectAnomalies による予測は正しくありません。

デフォルトでは、OpenCV は画像ビットマップに BGR 形式を使用します。OpenCV を使用して DetectAnomalies により分析用の画像をキャプチャする場合は、画像を DetectAnomalies に渡す前に画像を RGB888 形式に変換する必要があります。

サポートされている画像の最小サイズは 64x64 ピクセルです。サポートされている画像の最大サイズは 4096x4096 ピクセルです。

画像は protobuf メッセージで送信することも、共有メモリセグメントを介して送信することもできます。サイズの大きい画像を protobuf メッセージにシリアル化すると、DetectAnomalies への呼び出しのレーテンシーが大幅に長くなる可能性があります。レイテンシーを最小限に抑えるため、共有メモリーを使用することをおすすめします。

```
rpc DetectAnomalies(DetectAnomaliesRequest) returns (DetectAnomaliesResponse);
```

DetectAnomaliesRequest

DetectAnomalies 用の入力パラメータ。

```
message Bitmap {
  int32 width = 1;
  int32 height = 2;
  oneof data {
    bytes byte_data = 3;
    SharedMemoryHandle shared_memory_handle = 4;
  }
}
```

```
message SharedMemoryHandle {
  string name = 1;
  uint64 size = 2;
  uint64 offset = 3;
}
```

```
message AnomalyMaskParams {
SharedMemoryHandle shared_memory_handle = 2;
}
```

```
message DetectAnomaliesRequest {
string model_component = 1;
Bitmap bitmap = 2;
AnomalyMaskParams anomaly_mask_params = 3;
}
```

Bitmap

DetectAnomalies により分析したい画像。

width

ピクセルでの画像の幅。

height

ピクセルでの画像の高さ。

byte_data

protobuf メッセージで渡された画像のバイト数。

shared_memory_handle

共有メモリセグメントで渡された画像バイト数。

SharedMemoryHandle

POSIX 共有メモリセグメントを表します。

name

POSIX メモリーセグメントの名前。共有メモリの作成について詳しくは、shm_open を参照してください。

size

オフセットから始まる画像バッファサイズ (バイト単位)。

offset

共有メモリセグメントの先頭から画像バッファの先頭までのオフセット (バイト単位)。

AnomalyMaskParams

アノマリーマスクを出力するためのパラメーター。(セグメンテーションモデル)。

shared_memory_handle

マスクの画像バイトが格納されます (`shared_memory_handle` が指定されていない場合)。

DetectAnomaliesRequest

model_component

使用するモデルを含む AWS IoT Greengrass V2 コンポーネントの名前。

bitmap

`DetectAnomalies` で分析したい画像。

anomaly_mask_params

マスクを出力するためのオプションパラメータ。(セグメンテーションモデル)。

DetectAnomaliesResponse

`DetectAnomalies` からのレスポンス。

```
message DetectAnomalyResult {
bool is_anomalous = 1;
float confidence = 2;
Bitmap anomaly_mask = 3;
repeated Anomaly anomalies = 4;
float anomaly_score = 5;
float anomaly_threshold = 6;
}
```

```
message Anomaly {
string name = 1;
PixelAnomaly pixel_anomaly = 2;
```

```
message PixelAnomaly {
float total_percentage_area = 1;
string hex_color = 2;
}
```

```
message DetectAnomaliesResponse {
  DetectAnomalyResult detect_anomaly_result = 1;
}
```

Anomaly

画像で見つかった異常を表します。(セグメンテーションモデル)。

name

画像で見つかった異常タイプの名前。name がトレーニングデータセットの異常タイプにマップします。サービスはバックグラウンドの異常タイプを DetectAnomalies からのレスポンスに自動的に挿入します。

pixel_anomaly

異常タイプをカバーするピクセルマスクに関する情報。

PixelAnomaly

異常タイプをカバーするピクセルマスクに関する情報。(セグメンテーションモデル)。

total_percentage_area

画像で異常タイプが占める領域のパーセンテージ。

hex_color

画像の異常タイプを表す 16 進数のカラー値。この色は、トレーニングデータセットで使用されている異常タイプの色に対応します。

DetectAnomalyResult

is_anomalous

画像が異常を含むかどうかを示します。画像が異常を含む場合は「true」、画像が正常であれば「false」となります。

confidence

予測の精度における DetectAnomalies の信頼度。 confidence は 0 ~ 1 の浮動小数点値です。

anomaly_mask

shared_memory_handle が指定されていない場合は、マスクの画像バイトが含まれます。(セグメンテーションモデル)。

anomalies

入力画像内で見つかった異常 (0 個以上) のリスト。(セグメンテーションモデル)。

anomaly_score

ある画像について予測される異常値が、異常のない画像とどの程度ずれているかを表す数。anomaly_score は 0.0 (正常な画像からの最小偏差) から 1.0 (正常な画像からの最大偏差) までの浮動小数点値となります。Amazon Lookout for Vision は、画像に対する予測が「正常」であっても、anomaly_score の値を返します。

anomaly_threshold

画像の予測分類が正常か異常かを判断する数 (浮動小数点数)。anomaly_score が anomaly_threshold の値以上である画像は異常とみなされます。anomaly_score 値が anomaly_threshold 未満の場合は正常画像です。モデルが使用する anomaly_threshold の値は、モデルをトレーニングするときに Amazon Lookout for Vision によって計算されます。anomaly_threshold の値は設定も変更もできません。

ステータスコード

コード	数	説明
OK	0	DetectAnomalies が予測に成功しました。

コード	数	説明
UNKNOWN	2	不明なエラーが発生しました。
INVALID_ARGUMENT	3	1 つ以上の入力パラメータが無効です。詳細についてはエラーメッセージをチェックしてください。
NOT_FOUND	5	指定された名前のモデルが見つかりませんでした。
RESOURCE_EXHAUSTED	8	このオペレーションを実行するのに十分なリソースがありません。たとえば、Lookout for Vision EEdge Agent は、`DetectAnomalies` への呼び出しのレートに追いつくことができません。詳細についてはエラーメッセージをチェックしてください。
FAILED_PRECONDITION	9	「実行中」状態ではないモデルに対して `DetectAnomalies` が呼び出されました。
INTERNAL	13	内部エラーが発生しました。

DescribeModel

AWS IoT Greengrass Version 2 コアデバイスにデプロイされる Amazon Lookout for Vision モデルについて説明します。

```
rpc DescribeModel(DescribeModelRequest) returns (DescribeModelResponse);
```

DescribeModelRequest

```
message DescribeModelRequest {
  string model_component = 1;
}
```

model_component

記述するモデルを含む AWS IoT Greengrass V2 コンポーネントの名前。

DescribeModelResponse

```
message ModelDescription {
  string model_component = 1;
  string lookout_vision_model_arn = 2;
  ModelStatus status = 3;
  string status_message = 4;
}
```

```
message DescribeModelResponse {
  ModelDescription model_description = 1;
}
```

ModelDescription

model_component

Amazon Lookout for Vision モデルを含む AWS IoT Greengrass Version 2 コンポーネントの名前。

lookout_vision_model_arn

AWS IoT Greengrass V2 コンポーネントの生成に使用された Amazon Lookout for Vision モデルの Amazon リソースネーム (ARN)。

status

モデルの現在のステータス。詳細については「ModelStatus」を参照してください。

status_message

モデルのステータスメッセージ。

ステータスコード

コード	数	説明
OK	0	呼び出しが成功しました。
UNKNOWN	2	不明なエラーが発生しました。
INVALID_ARGUMENT	3	1 つ以上の入力パラメータが無効です。詳細については エラーメッセージをチェックしてください。
NOT_FOUND	5	付属名を伴うモデルが見つかりませんでした。
INTERNAL	13	内部エラーが発生しました。

ListModels

AWS IoT Greengrass Version 2 コアデバイスにデプロイされたモデルを一覧表示します。

```
rpc ListModels(ListModelsRequest) returns (ListModelsResponse);
```

ListModelsRequest

```
message ListModelsRequest {}
```

ListModelsResponse

```
message ModelMetadata {
  string model_component = 1;
  string lookout_vision_model_arn = 2;
  ModelStatus status = 3;
```

```
  string status_message = 4;
}
```

```
message ListModelsResponse {
  repeated ModelMetadata models = 1;
}
```

ModelMetadata

model_component

Amazon Lookout for Vision モデルを含む AWS IoT Greengrass Version 2 コンポーネントの名前。

lookout_vision_model_arn

AWS IoT Greengrass V2 コンポーネントの生成に使用された Amazon Lookout for Vision モデルの Amazon リソースネーム (ARN)。

status

モデルの現在のステータス。詳細については「ModelStatus」を参照してください。

status_message

モデルのステータスメッセージ。

ステータスコード

コード	数	説明
OK	0	呼び出しが完了しました。
UNKNOWN	2	不明なエラーが発生しました。
INTERNAL	13	内部エラーが発生しました。

StartModel

AWS IoT Greengrass Version 2 コアデバイスで実行中のモデルを開始します。モデルが実行を開始するにはしばらく時間がかかる場合があります。現在のステータスをチェックするに

は、DescribeModel を呼び出します。Status フィールドが RUNNING の場合、モデルは実行中です。

同時に実行できるモデルの数は、コアデバイスのハードウェア仕様によって異なります。

```
rpc StartModel(StartModelRequest) returns (StartModelResponse);
```

StartModelRequest

```
message StartModelRequest {
  string model_component = 1;
}
```

model_component

開始したいモデルを含む AWS IoT Greengrass Version 2 コンポーネントの名前。

StartModelResponse

```
message StartModelResponse {
  ModelStatus status = 1;
}
```

status

モデルの現在のステータス。レスポンスは、呼び出しが成功した場合は STARTING です。詳細については「ModelStatus」を参照してください。

ステータスコード

コード	数	説明
OK	0	モデルが起動中です
UNKNOWN	2	不明なエラーが発生しました。
INVALID_ARGUMENT	3	1 つ以上の入力パラメータが無効です。詳細については工

コード	数	説明
		ラーメッセージをチェックしてください。
NOT_FOUND	5	付属名を伴うモデルが見つかりませんでした。
RESOURCE_EXHAUSTED	8	このオペレーションを実行するのに十分なリソースがありません。たとえば、モデルを読み込むのに十分なメモリがありません。詳細についてはエラーメッセージをチェックしてください。
FAILED_PRECONDITION	9	STOPPED または FAILED 状態ではないモデルに対してメソッドが呼び出されました。
INTERNAL	13	内部エラーが発生しました。

StopModel

AWS IoT Greengrass Version 2 コアデバイスでの実行モデルを停止します。StopModel はモデルが停止すると返されます。レスポンスでの Status フィールドが STOPPED であれば、モデルは停止を完了しました。

```
  rpc StopModel(StopModelRequest) returns (StopModelResponse);
```

StopModelRequest

```
message StopModelRequest {
  string model_component = 1;
}
```

model_component

停止するモデルが含まれている AWS IoT Greengrass Version 2 コンポーネントの名前を入力します。

StopModelResponse

```
message StopModelResponse {
  ModelStatus status = 1;
}
```

status

モデルの現在のステータス。呼び出しが成功した場合、レスポンスは STOPPED です。詳細については「ModelStatus」を参照してください。

ステータスコード

コード	数	説明
OK	0	モデルは停止中です。
UNKNOWN	2	不明なエラーが発生しました。
INVALID_ARGUMENT	3	1 つ以上の入力パラメータが無効です。詳細については エラーメッセージをチェックしてください。
NOT_FOUND	5	付属名を伴うモデルが見つかりませんでした。
FAILED_PRECONDITION	9	RUNNING 状態ではないモデルに対してメソッドが呼び出されました。
INTERNAL	13	内部エラーが発生しました。

ModelStatus

AWS IoT Greengrass Version 2 コアデバイスにデプロイされたモデルのステータス。現在のステータスを取得するには、DescribeModel を呼び出します。

```
enum ModelStatus {
  STOPPED = 0;
  STARTING = 1;
  RUNNING = 2;
  FAILED = 3;
  STOPPING = 4;
}
```

Amazon Lookout for Vision ダッシュボードを使用する

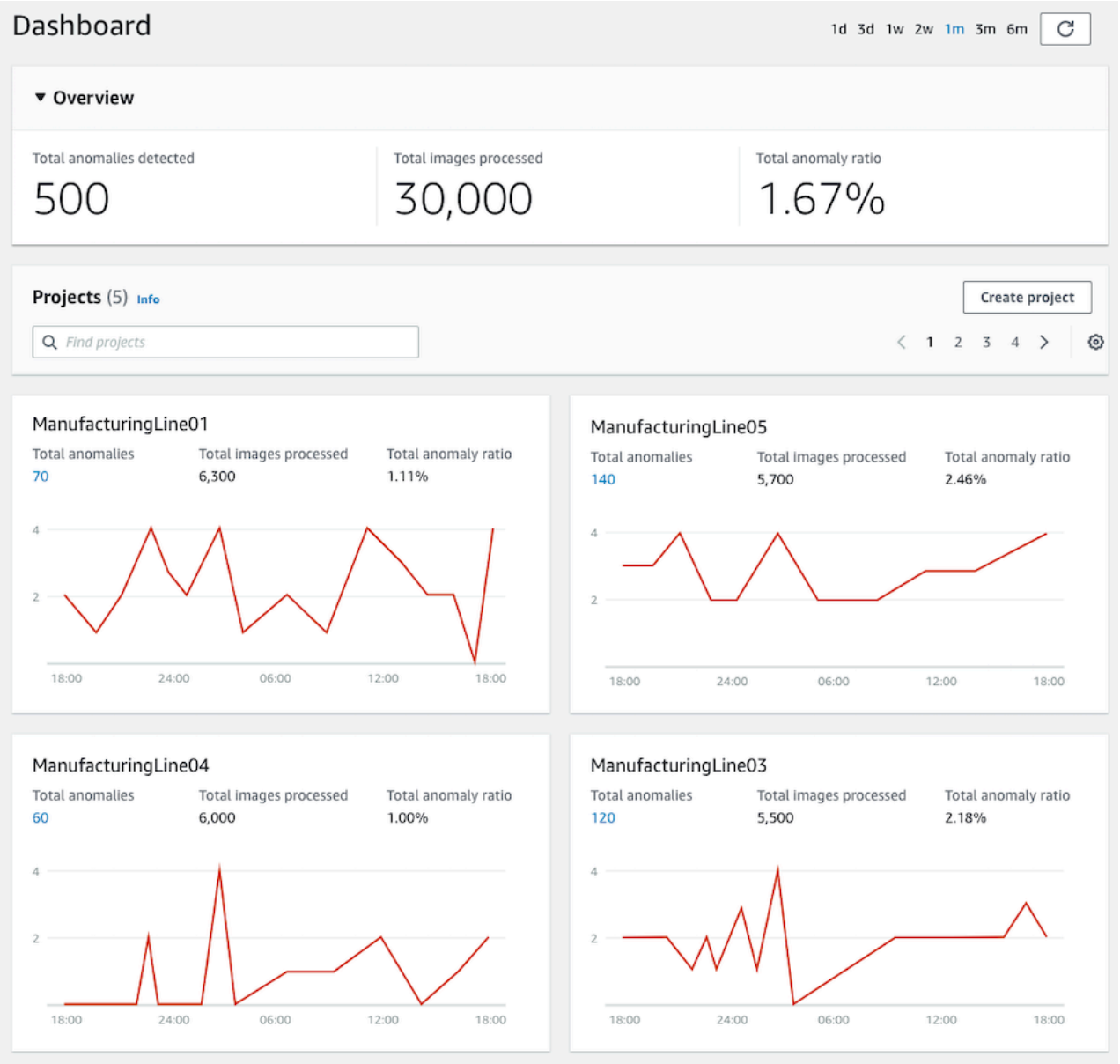
ダッシュボードには、先週に検出された異常の総数など、Amazon Lookout for Vision プロジェクトのメトリクスの概要が表示されます。ダッシュボードを使用すると、すべてのプロジェクトの概要と、個々のプロジェクトの概要が表示されます。メトリクスを表示するタイムラインを選択できます。ダッシュボードを使用して新しいプロジェクトを作成することもできます。

[概要] セクションには、プロジェクトの総数、画像の総数、およびすべてのプロジェクトで検出された画像の総数が表示されます。

[プロジェクト] セクションには、個々のプロジェクトに関する次の概要情報が表示されます。

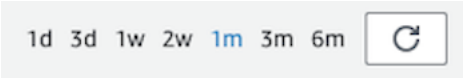
- 検出された異常の総数。
- 処理された画像の合計数。
- 総異常比率 (つまり、異常を検出された画像の割合)。
- グラフは、選択した時間枠における異常検出を示します。

プロジェクトに関する詳しい情報も取得できます。

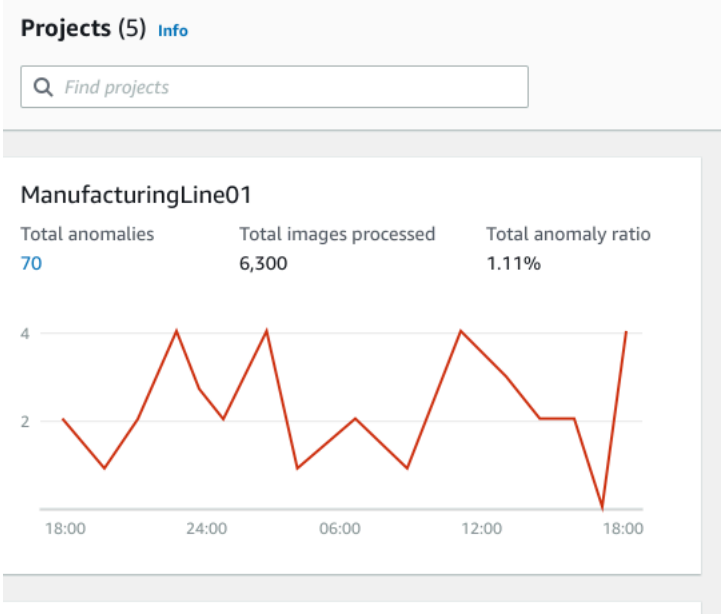


ダッシュボードを使うには

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [始める] を選択します。
3. 左のナビゲーションペインの [ダッシュボード] を選択します。
4. 特定の期間のメトリクスを表示するには、以下の作業を実行します。
   a. ダッシュボードの右上で、時間枠を選択します。
   b. 更新 ボタンを選択して、ダッシュボードに新しいタイムラインを表示します。



5. プロジェクトの詳細を表示するには、[プロジェクト] セクションでプロジェクト名を選択します (例えば、ManufacturingLine01)。



6. プロジェクトを作成するには、[プロジェクト] セクションの [プロジェクトを作成] を選択します。

Amazon Lookout for Vision リソースを管理する

コンソールまたは AWS SDK を使用して Amazon Lookout for Vision リソースを管理できます。Amazon Lookout for Vision には次のリソースがあります。

- プロジェクト
- データセット
- モデル
- トライアル検出

> **Note**
>
> トライアル検出タスクは削除できません。また、AWS SDK を使用してトライアル検出を管理することもできません。

トピック

プロジェクトの表示

Amazon Lookout for Vision プロジェクトのリストと個々のプロジェクトに関する情報は、コンソールから、または AWS SDK を使用して取得することができます。

> **Note**
>
> プロジェクトのリストは結果整合性があります。プロジェクトを作成または削除する場合は、プロジェクトリストが最新になるまでに少し待たなければならない場合があります。

プロジェクトの表示 (コンソール)

次の手順のステップを実行し、コンソールでプロジェクトを表示します。

プロジェクトを表示するには

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [] を選択します。
3. 左側のナビゲーションペインで、[プロジェクト] を選択します。プロジェクトビューが表示されます。
4. プロジェクト名を選択して、プロジェクトの詳細を表示します。

プロジェクトの表示 (SDK)

プロジェクトは、単一のユースケースでデータセットとモデルを管理します。たとえば、機械部品の異常を検出します。次の例では、`ListProjects` を呼び出してプロジェクトのリストを取得します。

プロジェクトを表示するには (SDK)

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のサンプルコードを使用して、プロジェクトを表示します。

   CLI

   `list-projects` コマンドを使用して、アカウントのプロジェクトを一覧表示します。

   ```
   aws lookoutvision list-projects \
     --profile lookoutvision-access
   ```

describe-project コマンドを使用して、プロジェクトに関する情報を取得します。

project-name の値を記述するプロジェクトの名前に変更します。

```
aws lookoutvision describe-project --project-name project_name \
  --profile lookoutvision-access
```

Python

このコードは、AWS ドキュメント SDK サンプル GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
    @staticmethod
    def list_projects(lookoutvision_client):
        """
        Lists information about the projects that are in in your AWS account
        and in the current AWS Region.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        """
        try:
            response = lookoutvision_client.list_projects()
            for project in response["Projects"]:
                print("Project: " + project["ProjectName"])
                print("\tARN: " + project["ProjectArn"])
                print("\tCreated: " + str(["CreationTimestamp"]))
                print("Datasets")
                project_description = lookoutvision_client.describe_project(
                    ProjectName=project["ProjectName"]
                )
                if not project_description["ProjectDescription"]["Datasets"]:
                    print("\tNo datasets")
                else:
                    for dataset in project_description["ProjectDescription"][
                        "Datasets"
                    ]:
                        print(f"\ttype: {dataset['DatasetType']}")
                        print(f"\tStatus: {dataset['StatusMessage']}")

                print("Models")
                response_models = lookoutvision_client.list_models(
                    ProjectName=project["ProjectName"]
```

```
                )
                if not response_models["Models"]:
                    print("\tNo models")
                else:
                    for model in response_models["Models"]:
                        Models.describe_model(
                            lookoutvision_client,
                            project["ProjectName"],
                            model["ModelVersion"],
                        )


 print("-----------------------------------------------------------\n")
            print("Done!")
        except ClientError:
            logger.exception("Problem listing projects.")
            raise
```

Java V2

このコードは、AWS ドキュメント SDK サンプル GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
/**
 * Lists the Amazon Lookout for Vision projects in the current AWS account and
 AWS
 * Region.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that you want to create.
 * @return List<ProjectMetadata> Metadata for each project.
 */
public static List<ProjectMetadata> listProjects(LookoutVisionClient lfvClient)
                throws LookoutVisionException {

        logger.log(Level.INFO, "Getting projects:");
        ListProjectsRequest listProjectsRequest = ListProjectsRequest.builder()
                        .maxResults(100)
                        .build();

        List<ProjectMetadata> projectMetadata = new ArrayList<>();
```

```
        ListProjectsIterable projects =
 lfvClient.listProjectsPaginator(listProjectsRequest);

        projects.stream().flatMap(r -> r.projects().stream())
                        .forEach(project -> {
                                projectMetadata.add(project);
                                logger.log(Level.INFO, project.projectName());
                        });

        logger.log(Level.INFO, "Finished getting projects.");

        return projectMetadata;

}
```

プロジェクトの削除

プロジェクトの削除は、コンソールの「プロジェクトビュー」ページから、または DeleteProject オペレーションによって行うことができます。

プロジェクトの データセットの から参照される画像は削除されません。

プロジェクトの削除 (コンソール)

以下の手順に従って、プロジェクトを削除します。コンソールプロシージャを使用すると、関連するモデルのバージョンとデータセットが削除されます。

プロジェクトを削除するには

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [開始する] を選択します。
3. 左側のナビゲーションペインで、[プロジェクト] を選択します。
4. 「プロジェクト」ページで、削除するプロジェクトを選択します。
5. ページの上部で、[削除] を選択します。
6. [削除] (削除) ダイアログボックスで、[削除] と入力し、プロジェクトを削除することを確認します。
7. 必要であれば、関連する データセットやモデルを削除することを 選択します。

8. [プロジェクトを削除] ( ) を選択します。

プロジェクトの削除 (SDK)

Amazon Lookout for Vision プロジェクトを削除するには、[プロジェクト削除] を呼び出して、削除するプロジェクトの名前を指定します。

プロジェクトを削除するには、まず、プロジェクト内のすべてのモデルを削除する必要があります。詳細については、「モデルの削除 (SDK)」を参照してください。また、モデルに関連付けられているデータセットを削除する必要があります。詳細については、「データセットの削除」を参照してください。

プロジェクトの削除にはしばらくかかることがあります。その間は、プロジェクトのステータスは DELETING です。その後に DeleteProject を呼び出したときに、削除したプロジェクトが含まれていない場合は、プロジェクトが削除されます。

プロジェクトを削除するには (SDK)

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. プロジェクトを削除するには、次のコードを使用します。

   AWS CLI

   project-name の値を削除するプロジェクトの名前に変更します。

   ```
   aws lookoutvision delete-project --project-name project_name \
     --profile lookoutvision-access
   ```

   Python

   このコードは、AWS ドキュメント SDK サンプル GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

   ```
       @staticmethod
       def delete_project(lookoutvision_client, project_name):
           """
           Deletes a Lookout for Vision Model
   ```

```
        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that you want to delete.
        """
        try:
            logger.info("Deleting project: %s", project_name)
            response =
 lookoutvision_client.delete_project(ProjectName=project_name)
            logger.info("Deleted project ARN: %s ", response["ProjectArn"])
        except ClientError as err:
            logger.exception("Couldn't delete project %s.", project_name)
            raise
```

Java V2

このコードは、AWS ドキュメント SDK の例 GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
/**
 * Deletes an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that you want to create.
 * @return String The ARN of the deleted project.
 */
public static String deleteProject(LookoutVisionClient lfvClient, String
 projectName)
                throws LookoutVisionException {

        logger.log(Level.INFO, "Deleting project: {0}", projectName);

        DeleteProjectRequest deleteProjectRequest =
 DeleteProjectRequest.builder()
                        .projectName(projectName)
                        .build();

        DeleteProjectResponse response =
 lfvClient.deleteProject(deleteProjectRequest);

        logger.log(Level.INFO, "Deleted project: {0} ARN: {1}",
                        new Object[] { projectName, response.projectArn() });

        return response.projectArn();
```

```
}
```

データセットの表示

プロジェクトには、モデルのトレーニングとテストに使用する単一のデータセットを使用できます。または、個別のトレーニングデータセットとテストデータセットを使用することもできます。コンソールを使用して、検証テストを表示できます。また、DescribeDataset オペレーション を使用してデータセットに関する情報を取得することができます (トレーニングまたはテスト)。

プロジェクト内のデータセットの表示 (コンソール)

次の手順のステップを実行し、コンソールでプロジェクトのデータセットを表示します。

データセットを表示するには (コンソール)

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [開始する] を選択します。
3. 左側のナビゲーションペインで、[プロジェクト] を選択します。
4. 「プロジェクト」ページで、表示したいデータセットを含むプロジェクトを選択します。
5. 左のナビゲーションペインで [データセット] を選択して、データセットの詳細を表示します。トレーニングデータセットとテストデータセットがある場合は、各データセットのタブが表示されます。

プロジェクト内のデータセットの表示 (SDK)

DescribeDataset オペレーションを使って背景、プロジェクトに関連するトレーニングまたはテストデータセットについての情報を取得できます。

データセット (SDK) を表示するには

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のサンプルコードを使用して、データセットを表示します。

CLI

以下の値を変更します。

- project-name に表示したいモデルを含むプロジェクト名を入力します。
- dataset-type を表示したいデータセットの種類に合わせます。(train または test)

```
aws lookoutvision describe-dataset --project-name project name\
  --dataset-type train or test \
  --profile lookoutvision-access
```

Python

このコードは、AWS ドキュメント SDK の例 GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
    @staticmethod
    def describe_dataset(lookoutvision_client, project_name, dataset_type):
        """
        Gets information about a Lookout for Vision dataset.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the dataset
 that
                             you want to describe.
        :param dataset_type: The type (train or test) of the dataset that you
 want
                             to describe.
        """
        try:
            response = lookoutvision_client.describe_dataset(
                ProjectName=project_name, DatasetType=dataset_type
            )
            print(f"Name: {response['DatasetDescription']['ProjectName']}")
            print(f"Type: {response['DatasetDescription']['DatasetType']}")
            print(f"Status: {response['DatasetDescription']['Status']}")
            print(f"Message: {response['DatasetDescription']['StatusMessage']}")
            print(f"Images: {response['DatasetDescription']['ImageStats']
['Total']}")
```

```
            print(f"Labeled: {response['DatasetDescription']['ImageStats']
['Labeled']}")
            print(f"Normal: {response['DatasetDescription']['ImageStats']
['Normal']}")
            print(f"Anomaly: {response['DatasetDescription']['ImageStats']
['Anomaly']}")
        except ClientError:
            logger.exception("Service error: problem listing datasets.")
            raise
        print("Done.")
```

Java V2

このコードは、AWS ドキュメント SDK の例 GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
/**
 * Gets the description for a Amazon Lookout for Vision dataset.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project in which you want to describe a
 *                    dataset.
 * @param datasetType The type of the dataset that you want to describe (train
 *                    or test).
 * @return DatasetDescription A description of the dataset.
 */
public static DatasetDescription describeDataset(LookoutVisionClient lfvClient,
                String projectName,
                String datasetType) throws LookoutVisionException {

        logger.log(Level.INFO, "Describing {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });

        DescribeDatasetRequest describeDatasetRequest =
 DescribeDatasetRequest.builder()
                        .projectName(projectName)
                        .datasetType(datasetType)
                        .build();

        DescribeDatasetResponse describeDatasetResponse =
 lfvClient.describeDataset(describeDatasetRequest);
```

```
        DatasetDescription datasetDescription =
 describeDatasetResponse.datasetDescription();

        logger.log(Level.INFO, "Project: {0}\n"
                        + "Created: {1}\n"
                        + "Type: {2}\n"
                        + "Total: {3}\n"
                        + "Labeled: {4}\n"
                        + "Normal: {5}\n"
                        + "Anomalous: {6}\n",
                        new Object[] {
                                        datasetDescription.projectName(),
                                        datasetDescription.creationTimestamp(),
                                        datasetDescription.datasetType(),

 datasetDescription.imageStats().total().toString(),

 datasetDescription.imageStats().labeled().toString(),

 datasetDescription.imageStats().normal().toString(),

 datasetDescription.imageStats().anomaly().toString(),
                        });

        return datasetDescription;

}
```

データセットへの画像の追加

データセットを作成したら、データセットに画像を追加することが必要な場合があります。たとえば、モデル評価でモデルの品質が悪いことが示された場合は、画像を追加することでモデルの品質を高めることができます。テストデータセットを作成している場合、画像を追加すると、モデルのパフォーマンス指標の精度が向上します。

データセットを更新した後、モデルを再トレーニングします。

トピック

- 画像をさらに追加する
- 画像の追加 (SDK)

画像をさらに追加する

ローカルコンピュータから画像をアップロードすることで、データセットに画像を追加できます。SDK でラベル付き画像を追加するには、[UpdateDatasetEntries] オペレーションを使用します。

データセットに画像を追加するには (コンソール)

1. [アクション] をクリックし、画像を追加するデータセットを選択します。
2. データセットにアップロードする画像を選択します。画像をドラッグするか、ローカルコンピュータからアップロードする画像を選択できます。同時にアップロードできる画像は、30 枚までです。
3. [画像をアップロード] を選択します。
4. 変更を保存] をクリックします。

画像の追加が終わったら、モデルのトレーニングに使用できるようにラベルを付ける必要があります。詳細については、「画像の分類 (コンソール)」を参照してください。

画像の追加 (SDK)

SDK でラベル付き画像を追加するには、[UpdateDatasetEntries] オペレーションを使用します。追加する画像を含むマニフェストファイルを指定します。マニフェストファイルの JSON Lines の `source-ref` フィールドで画像を指定して、既存の画像を更新することもできます。詳細については、「マニフェストファイルの作成」を参照してください。

データセットに画像を追加するには (SDK)

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のサンプルコードを使用して、画像をデータセットに追加します。

   CLI

   以下の値を変更します。

   - `project-name` を、更新したいデータセットが含まれているプロジェクトの名前に。
   - `dataset-type` を、更新したいデータセットのタイプに (`train` または `test`)。
   - `changes` を、データセットの更新を含むマニフェストファイルの場所に。

```
aws lookoutvision update-dataset-entries\
    --project-name project\
    --dataset-type train or test\
    --changes fileb://manifest file \
    --profile lookoutvision-access
```

Python

このコードは、AWS ドキュメント SDK の例 GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
    @staticmethod
    def update_dataset_entries(lookoutvision_client, project_name, dataset_type,
 updates_file):
        """
        Adds dataset entries to an Amazon Lookout for Vision dataset.
        :param lookoutvision_client: The Amazon Rekognition Custom Labels Boto3
 client.
        :param project_name: The project that contains the dataset that you want
 to update.
        :param dataset_type: The type of the dataset that you want to update
 (train or test).
        :param updates_file: The manifest file of JSON Lines that contains the
 updates.
        """

        try:
            status = ""
            status_message = ""
            manifest_file = ""

            # Update dataset entries
            logger.info(f"""Updating {dataset_type} dataset for project
 {project_name}
with entries from {updates_file}.""")

            with open(updates_file) as f:
                manifest_file = f.read()

            lookoutvision_client.update_dataset_entries(
                ProjectName=project_name,
```

```
                DatasetType=dataset_type,
                Changes=manifest_file,
            )

            finished = False
            while finished == False:

                dataset =
lookoutvision_client.describe_dataset(ProjectName=project_name,

DatasetType=dataset_type)

                status = dataset['DatasetDescription']['Status']
                status_message = dataset['DatasetDescription']['StatusMessage']

                if status == "UPDATE_IN_PROGRESS":
                    logger.info(
                        (f"Updating {dataset_type} dataset for project
{project_name}."))
                    time.sleep(5)
                    continue

                if status == "UPDATE_FAILED_ROLLBACK_IN_PROGRESS":
                    logger.info(
                        (f"Update failed, rolling back {dataset_type} dataset
for project {project_name}."))
                    time.sleep(5)
                    continue

                if status == "UPDATE_COMPLETE":
                    logger.info(
                        f"Dataset updated: {status} : {status_message} :
{dataset_type} dataset for project {project_name}.")
                    finished = True
                    continue

                if status == "UPDATE_FAILED_ROLLBACK_COMPLETE":
                    logger.info(
                        f"Rollback complated after update failure: {status} :
{status_message} : {dataset_type} dataset for project {project_name}.")
                    finished = True
                    continue

                logger.exception(
```

```
                    f"Failed. Unexpected state for dataset update: {status} :
 {status_message} : {dataset_type} dataset for project {project_name}.")
                raise Exception(
                    f"Failed. Unexpected state for dataset update: {status} :
 {status_message} :{dataset_type} dataset for project {project_name}.")

            logger.info(f"Added entries to dataset.")

            return status, status_message

        except ClientError as err:
            logger.exception(
                f"Couldn't update dataset: {err.response['Error']['Message']}")
            raise
```

Java V2

このコードは、AWS ドキュメント SDK の例 GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
/**
 * Updates an Amazon Lookout for Vision dataset from a manifest file.
 * Returns after Lookout for Vision updates the dataset.
 *
 * @param lfvClient    An Amazon Lookout for Vision client.
 * @param projectName  The name of the project in which you want to update a
 *                     dataset.
 * @param datasetType  The type of the dataset that you want to update (train or
 *                     test).
 * @param manifestFile The name and location of a local manifest file that you
 want to
 * use to update the dataset.
 * @return DatasetStatus The status of the updated dataset.
 */

public static DatasetStatus updateDatasetEntries(LookoutVisionClient lfvClient,
 String projectName,
                String datasetType, String updateFile) throws
 FileNotFoundException, LookoutVisionException,
                InterruptedException {

        logger.log(Level.INFO, "Updating {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });
```

```
        InputStream sourceStream = new FileInputStream(updateFile);
        SdkBytes sourceBytes = SdkBytes.fromInputStream(sourceStream);

        UpdateDatasetEntriesRequest updateDatasetEntriesRequest =
UpdateDatasetEntriesRequest.builder()
                        .projectName(projectName)
                        .datasetType(datasetType)
                        .changes(sourceBytes)
                        .build();

        lfvClient.updateDatasetEntries(updateDatasetEntriesRequest);

        boolean finished = false;
        DatasetStatus status = null;

        // Wait until update completes.

        do {

                DescribeDatasetRequest describeDatasetRequest =
DescribeDatasetRequest.builder()
                                .projectName(projectName)
                                .datasetType(datasetType)
                                .build();
                DescribeDatasetResponse describeDatasetResponse = lfvClient
                                .describeDataset(describeDatasetRequest);

                DatasetDescription datasetDescription =
describeDatasetResponse.datasetDescription();

                status = datasetDescription.status();

                switch (status) {

                        case UPDATE_COMPLETE:
                                logger.log(Level.INFO, "{0} Dataset updated for
project {1}.",
                                                new Object[] { datasetType,
projectName });
                                finished = true;
                                break;

                        case UPDATE_IN_PROGRESS:
```

```
                                logger.log(Level.INFO, "{0} Dataset update for
 project {1} in progress.",
                                                new Object[] { datasetType,
 projectName });
                                TimeUnit.SECONDS.sleep(5);

                                break;

                        case UPDATE_FAILED_ROLLBACK_IN_PROGRESS:
                                logger.log(Level.SEVERE,
                                                "{0} Dataset update failed for
 project {1}. Rolling back",
                                                new Object[] { datasetType,
 projectName });

                                TimeUnit.SECONDS.sleep(5);

                                break;

                        case UPDATE_FAILED_ROLLBACK_COMPLETE:
                                logger.log(Level.SEVERE,
                                                "{0} Dataset update failed for
 project {1}. Rollback completed.",
                                                new Object[] { datasetType,
 projectName });
                                finished = true;
                                break;

                        default:
                                logger.log(Level.SEVERE,
                                                "{0} Dataset update failed for
 project {1}. Unexpected error returned.",
                                                new Object[] { datasetType,
 projectName });
                                finished = true;

                }

        } while (!finished);

        return status;

}
```

3. 前のステップを繰り返し、他のデータセットタイプの値を指定します。

データセットからの画像の削除

データセットから画像を直接削除することはできません。代わりに、既存のデータセットを削除し、削除したい画像を含まない新しいデータセットを作成する必要があります。画像を削除する方法は、画像を既存のデータセット (マニフェストファイル、Amazon S3 バケット、またはローカルコンピュータ) にどのようにインポートしたかによります。

AWS SDK を使用して画像を削除することもできます。これは、画像セグメンテーションマニフェストファイルなしで画像セグメンテーションモデルを作成する場合に便利です。これにより、Amazon Lookout for Vision コンソールで画像マスクを再描画する必要がなくなります。

トピック

- データセットからの画像の削除 (コンソール)
- データセットからの画像の削除 (SDK)

データセットからの画像の削除 (コンソール)

Amazon Lookout for Vision コンソールを使用してデータセットから画像を削除するには、以下の手順を使用します。

データセットから画像を削除するには (コンソール)

1. プロジェクトのデータセットギャラリーを開きます。
2. 削除する各画像の名前をメモします。
3. 既存のデータセットを削除します。
4. 次のいずれかを実行します:
    - マニフェストファイルを使用してデータセットを作成した場合は、以下を実行します:
        a. テキストエディタで、データセットの作成に使用したマニフェストファイルを開きます。

b. ステップ 2 でメモした画像ごとに JSON Lines を削除します。画像の JSON Lines は、`source-ref` フィールドをチェックすることで識別できます。

c. マニフェストファイルを保存します。

d. 更新されたマニフェストファイルで新しいデータセットを作成します。

- Amazon S3 バケットからインポートされた画像からデータセットを作成した場合は、以下を行います:

a. ステップ 2 でメモした画像を Amazon S3 バケットから削除します。

b. Amazon S3 バケットの残りの画像で新しいデータセットを作成します。画像をフォルダ名で分類する場合、次のステップで画像を分類する必要はありません。

c. 次のいずれかを実行します。

- 画像分類モデルを作成する場合は、ラベルが付いていない各画像を分類します。
- 画像セグメンテーションモデルを作成する場合は、ラベルが付いていない各画像を分類してセグメント化します。

- ローカルコンピューターからインポートした画像からデータセットを作成した場合は、次の操作を行います:

a. コンピュータで、使用したい画像を含むフォルダを作成します。データセットから除去する画像を含めないでください。詳細については、「ローカルコンピュータに保存されている画像を使用してデータセットを作成する」を参照してください。

b. ステップ 4.a で作成したフォルダ内の画像によりデータセットを作成します。

c. 次のいずれかを行います:

- 画像分類モデルを作成する場合は、ラベルが付いていない各画像を分類します。
- 画像セグメンテーションモデルを作成する場合は、ラベルが付いていない各画像を分類してセグメント化します。

5. モデルをトレーニングします。

データセットからの画像の削除 (SDK)

AWS SDK を使用して、データセットからキーを削除できます。

データセットから画像を削除するには (SDK)

1. プロジェクトのデータセットギャラリーを開きます。
2. 削除する各画像の名前をメモします。

3. ListDataSetEntries オペレーションを使用してデータセットの JSON Lines をエクスポートします。
4. エクスポートされた JSON Lines を含むマニフェストファイルを作成します。
5. テキストエディタで、 マニフェストファイルを開きます。
6. ステップ 2 でメモした画像ごとに JSON Lines を削除します。画像の JSON Lines は、`source-ref` フィールドをチェックすることで識別できます。
7. マニフェストファイルを保存します。
8. 既存のデータセットを削除します。
9. 更新されたマニフェストファイルで新しいデータセットを作成します。
10. モデルをトレーニングします。

データセットの削除

プロジェクトからのデータセットの削除は、コンソールから、または `DeleteDataset` オペレーションによって行うことができます。データセットが参照している画像は削除されません。トレーニングデータセットとテストデータセットがあるプロジェクトからテストデータセットを削除すると、プロジェクトはシングルデータセットプロジェクトに戻ります。残りのデータセットはトレーニング中に分割され、トレーニングデータセットとテストデータセットが作成されます。トレーニングデータセットを削除すると、新しいトレーニングデータセットを作成するまで、プロジェクト内のモデルをトレーニングすることはできません。

データセットの削除 (コンソール)

以下の手順のステップを実行し、データセットを削除します。プロジェクト内のすべてのデータセットを削除すると、「データセットを作成する」ページが表示されます。

データセットを削除するには (コンソール)

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [開始] を選択します。
3. 左側のナビゲーションペインで、[プロジェクト] を選択します。
4. 「プロジェクト」ページで、削除したいデータセットを含むプロジェクトを選択します。
5. 左のナビゲーションペインの [データセット] を選択します。

6. [アクション] をクリックし、削除するデータセットを選択します。
7. [削除] ダイアログボックスで、[削除] と入力し、データセットを削除することを確認します。
8. [トレーニングデータセットを削除] または [テストデータセットを削除] を選択して、データセットを削除します。

データセットの削除 (SDK)

`DeleteDataset` オペレーションを使用してデータセットを削除します。

データセットを削除するには (SDK)

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. モデルを削除するには、次のサンプルコードを使用します。

   CLI

   以下の値を変更します。

   - `project-name` に削除したいモデルを含むプロジェクト名を入力します。
   - 削除したいデータセットに応じて、`dataset-type` を `train` または `test` のいずれかに設定します。シングルデータセットプロジェクトがある場合は、`train` を指定すると、データセットが削除されます

   ```
   aws lookoutvision delete-dataset --project-name project name\
     --dataset-type dataset type \
     --profile lookoutvision-access
   ```

   Python

   このコードは、AWS ドキュメント SDK の例 GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

   ```
       @staticmethod
       def delete_dataset(lookoutvision_client, project_name, dataset_type):
           """
           Deletes a Lookout for Vision dataset
   ```

```
        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the dataset
 that
                             you want to delete.
        :param dataset_type: The type (train or test) of the dataset that you
                             want to delete.
        """
        try:
            logger.info(
                "Deleting the %s dataset for project %s.", dataset_type,
 project_name
            )
            lookoutvision_client.delete_dataset(
                ProjectName=project_name, DatasetType=dataset_type
            )
            logger.info("Dataset deleted.")
        except ClientError:
            logger.exception("Service error: Couldn't delete dataset.")
            raise
```

Java V2

このコードは、AWS ドキュメント SDK の例 GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
/**
 * Deletes the train or test dataset in an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project in which you want to delete a
 *                    dataset.
 * @param datasetType The type of the dataset that you want to delete (train or
 *                    test).
 * @return Nothing.
 */
public static void deleteDataset(LookoutVisionClient lfvClient, String
 projectName, String datasetType)
                throws LookoutVisionException {

        logger.log(Level.INFO, "Deleting {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });
```

```
        DeleteDatasetRequest deleteDatasetRequest =
 DeleteDatasetRequest.builder()
                        .projectName(projectName)
                        .datasetType(datasetType)
                        .build();

        lfvClient.deleteDataset(deleteDatasetRequest);

        logger.log(Level.INFO, "Deleted {0} dataset for project {1}",
                        new Object[] { datasetType, projectName });

}
```

プロジェクトからのデータセットのエクスポート (SDK)

AWS SDK を使用して、Amazon Lookout for Vision プロジェクトから Amazon S3 バケットロケーションにデータセットをエクスポートできます。

データセットをエクスポートすることで、ソースプロジェクトのデータセットのコピーを使用して Lookout for Visionプロジェクトを作成するなどのタスクを実行できます。特定のバージョンのモデルに使用されているデータセットのスナップショットを作成することもできます。

この手続きの Python コードは、プロジェクトのトレーニングデータセット (マニフェストとデータセット画像) を、指定した Amazon S3 の場所にエクスポートします。プロジェクト内に存在する場合、コードはテストデータセットのマニフェストとデータセット画像もエクスポートします。送信先は、ソースプロジェクトと同じ Amazon S3 バケットでも、別の Amazon S3 バケットでもかまいません。このコードは ListDataSetEntries オペレーションを使用してデータセットマニフェストファイルを取得します。Amazon S3 オペレーションは、データセット画像と更新されたマニフェストファイルを宛先の Amazon S3 ロケーションにコピーします。

この手順では、プロジェクトのデータセットをエクスポートする方法を示します。また、エクスポートされたデータセットで新規プロジェクトを作成する方法も示します。

プロジェクトからデータセットをエクスポートするには (SDK)

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. データセットのエクスポート先の Amazon S3 パスを決定します。エクスポート先が Amazon Lookout for Vision のサポートしている AWS リージョンにあることを確かめてください。新規 Amazon S3 バケットを作成する場合は、「バケットの作成」を参照してください。

3. ユーザーがデータセットのエクスポート先の Amazon S3 パスと、ソースプロジェクトデータセット内の画像ファイルの S3 ロケーションへのアクセス権限を持っていることを確かめてください。次のポリシーを使用できます。このポリシーでは、画像ファイルはどの場所にあってもかまいません。*bucket/path* は、データセットのエクスポート先バケットとパスに置き換えてください。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "PutExports",
            "Effect": "Allow",
            "Action": [
                "S3:PutObjectTagging",
                "S3:PutObject"
            ],
            "Resource": "arn:aws:s3:::bucket/path/*"
        },
        {
            "Sid": "GetSourceRefs",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectTagging",
                "s3:GetObjectVersion"
            ],
            "Resource": "*"
        }
    ]
}
```

アクセスを提供するには、ユーザー、グループ、またはロールにアクセス許可を追加します。

- AWS IAM Identity Center のユーザーとグループ:

  アクセス許可セットを作成します。「AWS IAM Identity Center ユーザーガイド」の「権限設定を作成する」の手順に従ってください。

- ID プロバイダーを通じて IAM で管理されているユーザー:

ID フェデレーションのロールを作成する。詳細については、「IAM ユーザーガイド」の「サードパーティー ID プロバイダー (フェデレーション) 用のロールの作成」を参照してください。

- IAM ユーザー:
  - ユーザーが担当できるロールを作成します。手順については、「IAM ユーザーガイド」の「IAM ユーザー用ロールの作成」を参照してください。
  - (非推奨) ポリシーをユーザーに直接アタッチするか、ユーザーをユーザーグループに追加します。「IAM ユーザーガイド」の「ユーザー (コンソール) への権限の追加」の指示に従います。

4. 次のコードを dataset_export.py という名前のファイルに保存します。

```
"""
Purpose

Shows how to export the datasets (manifest files and images)
from an Amazon Lookout for Vision project to a new Amazon
S3 location.
"""

import argparse
import json
import logging

import boto3
from botocore.exceptions import ClientError

logger = logging.getLogger(__name__)


def copy_file(s3_resource, source_file, destination_file):
    """
    Copies a file from a source Amazon S3 folder to a destination
    Amazon S3 folder.
    The destination can be in a different S3 bucket.
    :param s3: An Amazon S3 Boto3 resource.
    :param source_file: The Amazon S3 path to the source file.
    :param destination_file: The destination Amazon S3 path for
    the copy operation.
    """
```

```
    source_bucket, source_key = source_file.replace("s3://", "").split("/", 1)
    destination_bucket, destination_key = destination_file.replace("s3://",
 "").split(
        "/", 1
    )

    try:
        bucket = s3_resource.Bucket(destination_bucket)
        dest_object = bucket.Object(destination_key)
        dest_object.copy_from(CopySource={"Bucket": source_bucket, "Key":
 source_key})
        dest_object.wait_until_exists()
        logger.info("Copied %s to %s", source_file, destination_file)
    except ClientError as error:
        if error.response["Error"]["Code"] == "404":
            error_message = (
                f"Failed to copy {source_file} to "
                f"{destination_file}. : {error.response['Error']['Message']}"
            )
            logger.warning(error_message)
            error.response["Error"]["Message"] = error_message
        raise


def upload_manifest_file(s3_resource, manifest_file, destination):
    """
    Uploads a manifest file to a destination Amazon S3 folder.
    :param s3: An Amazon S3 Boto3 resource.
    :param manifest_file: The manifest file that you want to upload.
    :destination: The Amazon S3 folder location to upload the manifest
    file to.
    """

    destination_bucket, destination_key = destination.replace("s3://",
 "").split("/", 1)

    bucket = s3_resource.Bucket(destination_bucket)

    put_data = open(manifest_file, "rb")
    obj = bucket.Object(destination_key + manifest_file)

    try:
        obj.put(Body=put_data)
```

```
        obj.wait_until_exists()
        logger.info("Put manifest file '%s' to bucket '%s'.", obj.key,
 obj.bucket_name)
    except ClientError:
        logger.exception(
            "Couldn't put manifest file '%s' to bucket '%s'.", obj.key,
 obj.bucket_name
        )
        raise
    finally:
        if getattr(put_data, "close", None):
            put_data.close()


def get_dataset_types(lookoutvision_client, project):
    """
    Determines the types of the datasets (train or test) in an
    Amazon Lookout for Vision project.
    :param lookoutvision_client: A Lookout for Vision Boto3 client.
    :param project: The Lookout for Vision project that you want to check.
    :return: The dataset types in the project.
    """

    try:
        response = lookoutvision_client.describe_project(ProjectName=project)

        datasets = []

        for dataset in response["ProjectDescription"]["Datasets"]:
            if dataset["Status"] in ("CREATE_COMPLETE", "UPDATE_COMPLETE"):
                datasets.append(dataset["DatasetType"])
        return datasets

    except lookoutvision_client.exceptions.ResourceNotFoundException:
        logger.exception("Project %s not found.", project)
        raise


def process_json_line(s3_resource, entry, dataset_type, destination):
    """
    Creates a JSON line for a new manifest file, copies image and mask to
    destination.
    :param s3_resource: An Amazon S3 Boto3 resource.
    :param entry: A JSON line from the manifest file.
```

```
    :param dataset_type: The type (train or test) of the dataset that
    you want to create the manifest file for.
    :param destination: The destination Amazon S3 folder for the manifest
    file and dataset images.
    :return: A JSON line with details for the destination location.
    """
    entry_json = json.loads(entry)

    print(f"source: {entry_json['source-ref']}")

    # Use existing folder paths to ensure console added image names don't clash.
    bucket, key = entry_json["source-ref"].replace("s3://", "").split("/", 1)
    logger.info("Source location: %s/%s", bucket, key)

    destination_image_location = destination + dataset_type + "/images/" + key

    copy_file(s3_resource, entry_json["source-ref"], destination_image_location)

    # Update JSON for writing.
    entry_json["source-ref"] = destination_image_location

    if "anomaly-mask-ref" in entry_json:
        source_anomaly_ref = entry_json["anomaly-mask-ref"]
        mask_bucket, mask_key = source_anomaly_ref.replace("s3://", "").split("/",
 1)

        destination_mask_location = destination + dataset_type + "/masks/" +
 mask_key
        entry_json["anomaly-mask-ref"] = destination_mask_location

        copy_file(s3_resource, source_anomaly_ref, entry_json["anomaly-mask-ref"])

    return entry_json


def write_manifest_file(
    lookoutvision_client, s3_resource, project, dataset_type, destination
):
    """
    Creates a manifest file for a dataset. Copies the manifest file and
    dataset images (and masks, if present) to the specified Amazon S3 destination.
    :param lookoutvision_client: A Lookout for Vision Boto3 client.
    :param project: The Lookout for Vision project that you want to use.
    :param dataset_type: The type (train or test) of the dataset that
```

```
    you want to create the manifest file for.
    :param destination: The destination Amazon S3 folder for the manifest file
    and dataset images.
    """

    try:
        # Create a reusable Paginator
        paginator = lookoutvision_client.get_paginator("list_dataset_entries")

        # Create a PageIterator from the Paginator
        page_iterator = paginator.paginate(
            ProjectName=project,
            DatasetType=dataset_type,
            PaginationConfig={"PageSize": 100},
        )

        output_manifest_file = dataset_type + ".manifest"

        # Create manifest file then upload to Amazon S3 with images.
        with open(output_manifest_file, "w", encoding="utf-8") as manifest_file:
            for page in page_iterator:
                for entry in page["DatasetEntries"]:
                    try:
                        entry_json = process_json_line(
                            s3_resource, entry, dataset_type, destination
                        )

                        manifest_file.write(json.dumps(entry_json) + "\n")

                    except ClientError as error:
                        if error.response["Error"]["Code"] == "404":
                            print(error.response["Error"]["Message"])
                            print(f"Excluded JSON line: {entry}")
                        else:
                            raise
        upload_manifest_file(
            s3_resource, output_manifest_file, destination + "datasets/"
        )

    except ClientError:
        logger.exception("Problem getting dataset_entries")
        raise
```

```
def export_datasets(lookoutvision_client, s3_resource, project, destination):
    """
    Exports the datasets from an Amazon Lookout for Vision project to a specified
    Amazon S3 destination.
    :param project: The Lookout for Vision project that you want to use.
    :param destination: The destination Amazon S3 folder for the exported datasets.
    """
    # Add trailing backslash, if missing.
    destination = destination if destination[-1] == "/" else destination + "/"

    print(f"Exporting project {project} datasets to {destination}.")

    # Get each dataset and export to destination.

    dataset_types = get_dataset_types(lookoutvision_client, project)
    for dataset in dataset_types:
        logger.info("Copying %s dataset to %s.", dataset, destination)

        write_manifest_file(
            lookoutvision_client, s3_resource, project, dataset, destination
        )

    print("Exported dataset locations")
    for dataset in dataset_types:
        print(f"   {dataset}: {destination}datasets/{dataset}.manifest")

    print("Done.")


def add_arguments(parser):
    """
    Adds command line arguments to the parser.
    :param parser: The command line parser.
    """

    parser.add_argument("project", help="The project that contains the dataset.")
    parser.add_argument("destination", help="The destination Amazon S3 folder.")


def main():
    """
    Exports the datasets from an Amazon Lookout for Vision project to a
    destination Amazon S3 location.
    """
```

```
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")
    parser = argparse.ArgumentParser(usage=argparse.SUPPRESS)
    add_arguments(parser)

    args = parser.parse_args()

    try:
        session = boto3.Session(profile_name="lookoutvision-access")
        lookoutvision_client = session.client("lookoutvision")
        s3_resource = session.resource("s3")

        export_datasets(
            lookoutvision_client, s3_resource, args.project, args.destination
        )
    except ClientError as err:
        logger.exception(err)
        print(f"Failed: {format(err)}")


if __name__ == "__main__":
    main()
```

5. コードを実行します。以下のコマンドライン引数を指定します:

   - project — エクスポートしたいデータセットを含むソースプロジェクトの名前。
   - destination — データセットのエクスポート先の Amazon S3 パス。

   例えば、python dataset_export.py *myproject* s3://*bucket/path*/ などです。

6. コードに表示されるマニフェストファイルの場所をメモします。それらはステップ 8 で必要になります。

7. プロジェクトを作成します の手引きに従って、エクスポートされたデータセットで新しい Lookout for Vision プロジェクトを作成します。

8. 次のいずれかを実行します:

   - Lookout for Vision コンソールを使用し、マニフェストファイルを使用したデータセットの作成 (コンソール) の手引きに従って新しいプロジェクト用のデータセットを作成します。ステップ 1 ~ 6 を行う必要はありません。

     ステップ 12 では、以下を行います:

a. ソースプロジェクトにテストデータセットがある場合は [トレーニングデータセットとテストデータセットを分離] を選択し、それ以外の場合は [単一データセット] を選択します。

b. [.manifest ファイルの場所] には、ステップ 6 でメモした適切なマニフェストファイル (train または test) の場所を入力します。

- CreateDataSet オペレーションを使用して、マニフェストファイルを使用したデータセットの作成 (SDK) のコードを使用して新規プロジェクト用のデータセットを作成します。`manifest_file` パラメータには、ステップ 6 でメモしたマニフェストファイルの場所を使用します。ソースプロジェクトにテストデータセットがある場合は、コードを再度使用してテストデータセットを作成します。

9. 準備ができたら、モデルのトレーニング の手引きに従ってモデルをトレーニングします。

モデルの表示

プロジェクトには、モデルの複数のバージョンを含めることができます。また、 コンソールを使用してプロジェクトのモデルを表示することもできます。また、`ListModels` オペレーションを使用することもできます。

> **Note**
>
> モデルのリストには結果整合性があります。モデルを作成する場合は、モデルリストが最新になるまでに少し待たなければならない場合があります。

モデルの表示 (コンソール)

次の手順のステップを実行し、コンソールでプロジェクトのモデルを表示します。

モデルを表示するには (コンソール)

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [開始する] を選択します。
3. 左側のナビゲーションペインで、[プロジェクト] を選択します。
4. 「プロジェクト」ページで、表示したいモデルを含むプロジェクトを選択します。

5. 左のナビゲーションペインで [モデル] を選択して、モデルの詳細を表示します。

モデルの表示 (SDK)

モデルのバージョンを表示するには、ListModels オペレーションを使用します。特定のモデルバージョンに関する情報を取得するには、DescribeModel オペレーションを使用します。次の例では、プロジェクト内のすべてのモデルのバージョンを一覧表示し、個々のモデルバージョンのパフォーマンスおよび出力コンフィギュレーション情報を表示します。

モデルを表示するには (SDK)

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. 次のサンプルコードを使用して、モデルを一覧表示し、モデルに関する情報を取得します。

   CLI

   list-models コマンドを使用すると、プロジェクト内のモデルを一覧表示できます。

   次の値を変更します。

   - [project-name] に表示したいモデルを含むプロジェクト名を入力します。

```
aws lookoutvision list-models --project-name project name \
  --profile lookoutvision-access
```

   モデルについての情報を取得するには、describe-model コマンドを使用します。以下の値を変更します。

   - [project-name] に表示したいモデルを含むプロジェクト名を入力します。
   - [model-version] に記述するモデルのバージョンを入力します。

```
aws lookoutvision describe-model --project-name project name\
    --model-version model version \
   --profile lookoutvision-access
```

Python

このコードは、AWS ドキュメント SDK の例 GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
    @staticmethod
    def describe_models(lookoutvision_client, project_name):
        """
        Gets information about all models in a Lookout for Vision project.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that you want to use.
        """
        try:
            response =
lookoutvision_client.list_models(ProjectName=project_name)
            print("Project: " + project_name)
            for model in response["Models"]:
                Models.describe_model(
                    lookoutvision_client, project_name, model["ModelVersion"]
                )
                print()
            print("Done...")
        except ClientError:
            logger.exception("Couldn't list models.")
            raise
```

Java V2

このコードは、AWS ドキュメント SDK の例 GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
/**
 * Lists the models in an Amazon Lookout for Vision project.
 *
 * @param lfvClient   An Amazon Lookout for Vision client.
 * @param projectName The name of the project that contains the models that
 *                    you want to list.
 * @return List <Metadata> A list of models in the project.
 */
```

```
public static List<ModelMetadata> listModels(LookoutVisionClient lfvClient,
 String projectName)
                throws LookoutVisionException {

        ListModelsRequest listModelsRequest = ListModelsRequest.builder()
                        .projectName(projectName)
                        .build();

        // Get a list of models in the supplied project.
        ListModelsResponse response = lfvClient.listModels(listModelsRequest);

        for (ModelMetadata model : response.models()) {
                logger.log(Level.INFO, "Model ARN: {0}\nVersion: {1}\nStatus:
 {2}\nMessage: {3}", new Object[] {
                                model.modelArn(),
                                model.modelVersion(),
                                model.statusMessage(),
                                model.statusAsString() });
        }

        return response.models();

}
```

モデルの削除

モデルのバージョンを削除するには、コンソールまたは DeleteModel オペレーションを使用します。実行中またはトレーニング中のモデルバージョンを削除することはできません。

モデルがバージョンを実行している場合は、まず StopModel オペレーションを使用して、モデルバージョンを停止します。詳細については、「Amazon Lookout for Vision モデルの停止」を参照してください。モデルがトレーニング中の場合は、モデルを削除する前に終了するまで待ちます。

モデルの削除には数秒かかることがあります。モデルが削除されたかどうかを確認するには、[ListProjects] を呼び出します。そして、モデルのバージョン (ModelVersion) がModels配列にあるかどうかをチェックしてください。

モデルの削除 (コンソール)

コンソールからモデルを削除するには、次の手順を実行します。

モデルを削除するには (コンソール)

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [開始する] を選択します。
3. 左側のナビゲーションペインで、[プロジェクト] を選択します。
4. 「プロジェクト」ページで、削除したいモデルを含むプロジェクトを選択します。
5. 左のナビゲーションペインで [モデル] を選択します。
6. [モデル] ビューで、削除するモデルの横のボタンを選択します。
7. ページの上部で、[削除] を選択します。
8. [削除] ダイアログボックスで、[削除] と入力し、モデルを削除することを確認します。
9. [モデルを削除] を選択してモデルを削除します。

モデルの削除 (SDK)

`DeleteModel` オペレーションでモデルを削除するには、次の手順に従います。

モデルを削除するには (SDK)

1. まだの場合は、AWS CLI と AWS SDK をインストールして構成します。詳細については、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。
2. モデルを削除するには、次のサンプルコードを使用します。

   CLI

   以下の値を変更します。

   - `project-name` に削除したいモデルを含むプロジェクト名を入力します。
   - `model-version` に削除するモデルのバージョンを入力します。

```
aws lookoutvision delete-model --project-name project name\
  --model-version model version \
  --profile lookoutvision-access
```

Python

このコードは、AWS ドキュメント SDK サンプル GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
    @staticmethod
    def delete_model(lookoutvision_client, project_name, model_version):
        """
        Deletes a Lookout for Vision model. The model must first be stopped and
can't
        be in training.

        :param lookoutvision_client: A Boto3 Lookout for Vision client.
        :param project_name: The name of the project that contains the desired
model.
        :param model_version: The version of the model that you want to delete.
        """
        try:
            logger.info("Deleting model: %s", model_version)
            lookoutvision_client.delete_model(
                ProjectName=project_name, ModelVersion=model_version
            )

            model_exists = True
            while model_exists:
                response =
lookoutvision_client.list_models(ProjectName=project_name)

                model_exists = False
                for model in response["Models"]:
                    if model["ModelVersion"] == model_version:
                        model_exists = True

                if model_exists is False:
                    logger.info("Model deleted")
                else:
                    logger.info("Model is being deleted...")
                    time.sleep(2)

            logger.info("Deleted Model: %s", model_version)
        except ClientError:
            logger.exception("Couldn't delete model.")
            raise
```

Java V2

このコードは、AWS ドキュメント SDK サンプル GitHub リポジトリから引用されたものです。詳しい事例はこちらです。

```
/**
* Deletes an Amazon Lookout for Vision model.
*
* @param lfvClient    An Amazon Lookout for Vision client. Returns after the
 model is deleted.
* @param projectName  The name of the project that contains the model that you
 want to delete.
* @param modelVersion The version of the model that you want to delete.
* @return void
*/
public static void deleteModel(LookoutVisionClient lfvClient,
                String projectName,
                String modelVersion) throws LookoutVisionException,
 InterruptedException {

        DeleteModelRequest deleteModelRequest = DeleteModelRequest.builder()
                        .projectName(projectName)
                        .modelVersion(modelVersion)
                        .build();

        lfvClient.deleteModel(deleteModelRequest);

        boolean deleted = false;

        do {

                ListModelsRequest listModelsRequest =
 ListModelsRequest.builder()
                                .projectName(projectName)
                                .build();

                // Get a list of models in the supplied project.
                ListModelsResponse response =
 lfvClient.listModels(listModelsRequest);
```

```
                ModelMetadata modelMetadata = response.models().stream()
                                .filter(model ->
model.modelVersion().equals(modelVersion)).findFirst()
                                .orElse(null);

                if (modelMetadata == null) {
                        deleted = true;
                        logger.log(Level.INFO, "Deleted: Model version {0} of
project {1}.",
                                        new Object[] { modelVersion,
projectName });

                } else {
                        logger.log(Level.INFO, "Not yet deleted: Model version
{0} of project {1}.",
                                        new Object[] { modelVersion,
projectName });
                        TimeUnit.SECONDS.sleep(60);
                }

        } while (!deleted);

}
```

モデルのタグ付け

タグを使用して、Amazon Lookout for Vision モデルを識別、整理、検索、フィルタリングできます。各タグは、ユーザー定義のキーと値で構成されるラベルです。たとえば、モデルの課金を決定しやすくするためには、モデルに `Cost center` キーでタグ付けし、適切なコストセンター番号を値として追加します。詳細については、「AWS リソースのタグ付け」を参照してください。

タグを使用して、次の操作を行います。

- コスト配分タグを使用して、モデルの課金の追跡をします。詳細については、「コスト配分タグの使用」を参照してください。
- IAM (Identity and Access Management) を用いて、モデルへのアクセスを制御します。詳細については、IAM ユーザーガイドの「リソースタグを使用した AWS リソースへのアクセスの制御」を参照してください。

- モデル管理を自動化します。例えば、業務時間外に開発モデルを停止する自動起動または停止スクリプトを実行し、コストを削減することができます。詳細については、「トレーニング済みの Amazon Lookout for Vision モデルの実行」を参照してください。

Amazon Lookout for Visionのコンソールを使用するか、AWS SDK を使用することでモデルにタグ付けすることができます。

トピック

- モデルのタグ付け (コンソール)
- モデルのタグ付け (SDK)

モデルのタグ付け (コンソール)

Amazon Lookout for Vision コンソールを使用して、モデルにタグを追加したり、モデルに追加されたタグを表示したり、タグを削除したりできます。

タグの追加または削除 (コンソール)

この手順では、既存のモデルにタグを追加する方法、または既存のモデルからタグを削除する方法について説明します。トレーニング中に新しいモデルにタグを追加することもできます。詳細については、「モデルのトレーニング」を参照してください。

既存のモデルにタグを追加または削除するには (コンソール)

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [開始する] を選択します。
3. ナビゲーションペインで、[プロジェクト] を選択します。
4. 「プロジェクトリソース」ページで、タグ付けしたいモデルを含むプロジェクトを選択します。
5. ナビゲーションペインの、以前に選択したプロジェクトの下で、[モデル] を選択します。
6. [モデル] セクションで、タグを追加するモデルを選択します。
7. 「モデルの詳細」ページで、[タグ] タブを選択します。
8. [タグ] タブで、[タグを管理] を選択します。
9. 「タグを管理する」ページで、[新規タグを追加] を選択します。
10. キーと値を入力します。

a. [キー] に、キーの名前を入力します。

b. [値] に値を入力します。

11. さらにタグを追加するには、手順 9～10 を繰り返します。

12. (オプション) タグを削除するには、削除したいタグの横にある[削除] を選択します。以前に保存したタグを削除する場合、変更を保存するとそのタグが削除されます。

13. [変更を保存] を選択して、変更を保存します。

モデルタグの表示 (コンソール)

Amazon Lookout for Vision コンソールを使用して、モデルに追加されているタグを表示できます。

プロジェクト内のすべてのモデルに追加されたタグを表示するには、AWS SDK を使用する必要があります。詳細については、「モデルタグの一覧表示 (SDK)」を参照してください。

モデルに追加されたタグを表示するには

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。
2. [開始する] を選択します。
3. ナビゲーションペインで、[プロジェクト] を選択します。
4. 「プロジェクトリソース」ページで、タグを表示したいモデルを含むプロジェクトを選択します。
5. ナビゲーションペインの、以前に選択したプロジェクトの下で、[モデル] を選択します。
6. [モデル] セクションで、タグを表示するモデルを選択します。
7. 「モデルの詳細」ページで、[タグ] タブを選択します。タグは[タグ] セクションに示されています。

モデルのタグ付け (SDK)

AWS SDKを使用することで、以下のことが可能になります。

- 新しいモデルにタグを追加する
- 既存のモデルにタグを追加する
- モデルに追加されたタグを一覧表示する

- モデルからタグを削除する

このセクションには AWS CLI の例が含まれます。AWS CLI をまだインストールしていない場合は、「ステップ 4: AWS CLI と AWS SDKsを設定する」を参照してください。

新しいモデルへのタグの追加 (SDK)

[CreateModel] オペレーションを使用することで、作成時に、モデルにタグを追加することもできます。Tags 配列入力パラメータで 1 つ以上のタグを指定します。

```
aws lookoutvision create-model --project-name "project name"\
  --output-config '{ "S3Location": { "Bucket": "output bucket", "Prefix":  "output
 folder" } }'\
  --tags '[{"Key":"Key","Value":"Value"}]' \
  --profile lookoutvision-access
```

モデルの作成とトレーニングの詳細については、「モデルのトレーニング (SDK) 」 を参照してください。

既存のモデルへのタグの追加 (SDK)

既存のモデルに 1 つ以上のタグを追加するには、[TagResource] オペレーションを使用します。モデルの Amazon リソースネーム (ARN) (ResourceArn) と、追加したいタグ (Tags) を指定します。

```
aws lookoutvision tag-resource --resource-arn "resource-arn"\
  --tags '[{"Key":"Key","Value":"Value"}]' \
  --profile lookoutvision-access
```

Java コードの例については、「TagModel」を参照してください。

モデルタグの一覧表示 (SDK)

モデルに追加されたタグを一覧表示するには、[ListTagsForResource] オペレーションを実行し、モデルの Amazon リソースネーム (ARN)、(ResourceArn) を指定します。応答は、指定されたモデルに追加されているタグキーと値のマップです。

```
aws lookoutvision list-tags-for-resource --resource-arn resource-arn \
  --profile lookoutvision-access
```

プロジェクト内のどのモデルに特定のタグがあるかを確認するには、ListModels でモデルのリストを取得します。次に、ListModels からの応答で各モデルに対して ListTagsForResource を呼び出します。ListTagsForResource からのレスポンスを検査して、必要なタグが存在するかどうかを確認します。

Java コードの例については、「ListModelTags」を参照してください。すべてのプロジェクトでタグの値を検索するサンプル Python コードについては、「find_tag.py」を参照してください。

モデルからタグを削除する (SDK)

1 つ以上のタグを削除するには、[UntagResource] オペレーションを使用します。モデルの Amazon リソースネーム (ARN) (ResourceArn) と、削除したいタグキー (Tag-Keys) を指定します。

```
aws lookoutvision untag-resource --resource-arn resource-arn\
  --tag-keys '["Key"]' \
  --profile lookoutvision-access
```

Java コードの例については、「UntagModel」を参照してください。

トライアル検出タスクの表示

コンソールを使用して、トライアル検出を表示できます。AWS SDK を使用してトライアル検出タスクを表示することはできません。

> **Note**
>
> トライアル検出のリストには結果整合性があります。トライアル検出を作成する場合は、トライアル検出リストが最新になるまでに少し待たなければならない場合があります。

トライアル検出タスクの表示 (コンソール)

以下の手順に従って、トライアル検出を表示します。

トライアル検出タスクを表示するには

1. https://console.aws.amazon.com/lookoutvision/ で Amazon Lookout for Vision コンソールを開きます。

2. [開始する] を選択します。
3. 左のナビゲーションペインで [トライアル検出] を選択します。
4. 「トライアル検出」ページで、トライアル検出タスクを選択して詳細を表示します。

サンプルのコードおよびデータセット

Amazon Lookout for Vision で使用できるサンプルのコードとデータセットを以下に示します。

トピック

サンプルのコード

Amazon Lookout for Vision では、以下のコードサンプルを提供しています。

サンプル	説明
GitHub	Amazon Lookout for Vision モデルのトレーニングとホストを行う Python コードのサンプル。
Amazon Lookout for Vision ラボ	回路基板サンプル画像を使ってモデルを作成するための Python Notebook ノートブック。
Python サンプルコード	Amazon Lookout for Vision ドキュメンテーションで使用されている Python のサンプル。
Java サンプルコード	Amazon Lookout for Vision ドキュメンテーションで使用されている Java サンプル。

サンプルデータセット

以下は Amazon Lookout for Vision で使用できるサンプルデータセットです。

トピック

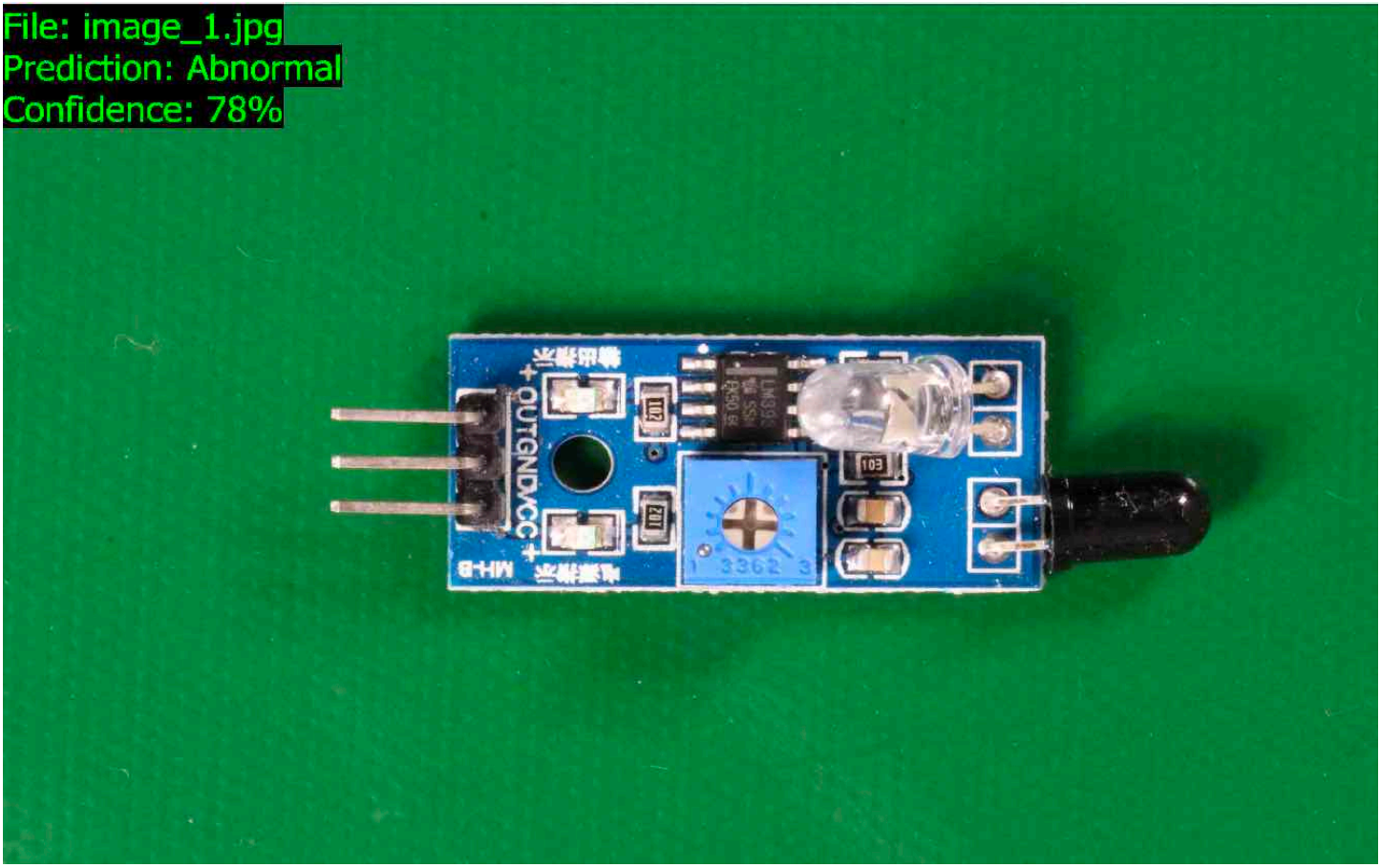
画像セグメンテーションデータセット

Amazon Lookout for Vision コンソールの開始 は、画像セグメンテーションモデルの作成に使用できる、壊れたクッキーのデータセットを提供します。

画像セグメンテーションモデルを作成する別のデータセットについては、「GPU を使用せずに エッジで Amazon Lookout for Vision を使用して異常箇所を特定する」を参照してください。

画像分類データセット

Amazon Lookout for Vision では、画像分類モデルの作成に使用できる回路基板のサンプル画像を提供しています。



https://github.com/aws-samples/amazon-lookout-for-vision GitHub リポジトリから画像をコピーできます。画像は `circuitboard` フォルダにあります。

`circuitboard` フォルダには以下のフォルダがあります。

- `train` — トレーニングデータセットで使用できる画像。

- `test` — テストデータセットで使用できる画像。
- `extra_images` — トライアル検出を実行したり、DetectAnomalies オペレーションでトレーニングしたモデルを試すために使用できる画像です。

`train` と `test` のフォルダには、それぞれ `normal` というサブフォルダ (正常な画像を含む) と `anomaly` というサブフォルダ (異常のある画像を含む) があります。

> **Note**
>
> 後でコンソールでデータセットを作成するときに、Amazon Lookout for Vision は、フォルダ名 (`normal` と `anomaly`) を使って、画像に自動的にラベル付けすることができます。詳細については、「the section called “Amazon S3 バケット”」を参照してください。

データセットイメージを準備するには

1. https://github.com/aws-samples/amazon-lookout-for-vision リポジトリのクローンをコンピュータに作成します。詳細については、「リポジトリのクローン作成」を参照してください。
2. Amazon S3 バケットを作成する。詳細については、「S3 バケット作成方法」を参照してください。
3. コマンドプロンプトで次のコマンドを入力して、コンピュータから Amazon S3 バケットにデータセットイメージをコピーします。

```
aws s3 cp --recursive your-repository-folder/circuitboard s3://your-bucket/
circuitboard
```

イメージをアップロードしたら、モデルを作成できます。以前に回路基板の画像をアップロードした Amazon S3 の場所から画像を追加することで、画像を自動的に分類できます。モデルのトレーニングが完了するたびに、およびモデルが実行 (ホスト) されている時間に対して料金が発生すると覚えてください。

分類モデルを作成するには

1. プロジェクトの作成 (コンソール) を行う
2. Amazon S3 バケットに保存されている画像を使用してデータセットを作成します。 を行う

- ステップ 6 では、[トレーニングデータセットとテストデータセットを分離] タブを選択します。
- ステップ 8a では、「データセット画像を準備するには」でアップロードしたトレーニング画像の S3 URI を入力します。例えば「s3://*your-bucket*/circuitboard/train」のようにです。ステップ 8b では、テストデータセットの S3 URI を入力します。例えば「s3://*your-bucket*/circuitboard/test」のようにです。
- 必ずステップ 9 を行ってください。

3. モデルのトレーニング (コンソール) を行います。
4. モデルの開始 (コンソール) を行います。
5. 画像内の異常を検出する を行います。test_images フォルダの画像を使用できます。
6. モデルで完了したら、モデルの停止 (コンソール) を行ってください。

Amazon Lookout for Vision のセキュリティ

AWS では、クラウドセキュリティを最優先事項としています。AWS のユーザーは、セキュリティを最も重視する組織の要件を満たすように構築されたデータセンターとネットワークアーキテクチャを利用できます。

セキュリティは、AWS とユーザーの間の責任共有です。責任共有モデルでは、これをクラウドのセキュリティおよびクラウド内のセキュリティと説明しています。

- クラウドのセキュリティ - AWS は、AWS クラウドで AWS のサービスを実行するインフラストラクチャを保護する責任を負います。また、AWS は、使用するサービスを安全に提供します。AWS コンプライアンスプログラムの一環として、サードパーティーの監査が定期的にセキュリティの有効性をテストおよび検証しています。Amazon Lookout for Vision に適用するコンプライアンスプログラムの詳細については、「コンプライアンスプログラムによる対象範囲内の AWS のサービス」、をご参照ください。
- クラウド内のセキュリティ - ユーザーの責任は、使用する AWS のサービスに応じて異なります。またお客様は、データの機密性、企業要件、適用法令と規制などのその他の要因に対しても責任を担います。

このドキュメントは、Lookout for Vision を使用する際の責任共有モデルの適用方法を理解するのに役立ちます。以下のトピックでは、セキュリティおよびコンプライアンスの目的を達成するために Lookout for Vision を設定する方法を示します。また、Lookout for Vision リソースのモニタリングや保護に AWS の他のサービスを利用する方法についても説明します。

トピック

- Amazon Lookout for Vision におけるデータ保護
- Amazon Lookout for Vision 用 Identity and Access Management
- Amazon Lookout for Vision のコンプライアンス検証
- Amazon Lookout for Vision での耐障害性
- Amazon Lookout for Vision のインフラストラクチャセキュリティ

Amazon Lookout for Vision におけるデータ保護

責任 AWS 共有モデル、Amazon Lookout for Vision でのデータ保護に適用されます。このモデルで説明されているように、 AWS はすべての を実行するグローバルインフラストラクチャを保護する

責任があります AWS クラウド。お客様は、このインフラストラクチャでホストされているコンテンツに対する管理を維持する責任があります。また、使用する AWS サービス のセキュリティ設定と管理タスクもユーザーの責任となります。データプライバシーの詳細については、「データプライバシーFAQ」を参照してください。欧州におけるデータ保護の詳細については、AWS 「 セキュリティブログ」のAWS 「責任共有モデル」とGDPRブログ記事を参照してください。

データ保護の目的で、認証情報を保護し AWS アカウント 、 AWS IAM Identity Center または AWS Identity and Access Management () を使用して個々のユーザーを設定することをお勧めしますIAM。この方法により、それぞれのジョブを遂行するために必要な権限のみが各ユーザーに付与されます。また、次の方法でデータを保護することもお勧めします:

- 各アカウントで多要素認証 (MFA) を使用します。
- SSL/TLS を使用して AWS リソースと通信します。1TLS.2 が必要で、1.3 TLS をお勧めします。
- を使用して APIとユーザーアクティビティの口グ記録を設定します AWS CloudTrail。
- AWS 暗号化ソリューションと、 内のすべてのデフォルトのセキュリティコントロールを使用します AWS サービス。
- Amazon Macie などの高度なマネージドセキュリティサービスを使用します。これらは、Amazon S3 に保存されている機密データの検出と保護を支援します。
- コマンドラインインターフェイスまたは AWS を介して にアクセスするときに FIPS 140-3 検証済みの暗号化モジュールが必要な場合はAPI、FIPSエンドポイントを使用します。利用可能なFIPSエンドポイントの詳細については、「連邦情報処理規格 (FIPS) 140-3」を参照してください。

お客様の E メールアドレスなどの極秘または機密情報は、タグ、または名前フィールドなどの自由形式のテキストフィールドに配置しないことを強くお勧めします。これは、コンソール、、またはを使用して Lookout for Vision または他の AWS サービス を使用する場合API AWS CLIも同様です AWS SDKs。名前に使用する自由記述のテキストフィールドやタグに入力したデータは、課金や診断口グに使用される場合があります。URL を外部サーバーに提供する場合は、そのサーバーへのリクエストを検証URLするために認証情報を に含めないことを強くお勧めします。

データ暗号化

以下の情報は、Amazon Lookout for Vision がお客様のデータを保護するためにデータ暗号化を使用している箇所を説明するものです。

保管中の暗号化

イメージ

モデルをトレーニングするために、Amazon Lookout for Vision は、ソーストレーニングとテスト画像のコピーを作成します。コピーされたイメージは、Amazon Simple Storage Service (S3) の保管時に、 AWS 所有のキー または指定したキーによるサーバー側の暗号化を使用して暗号化されます。キーは Key Management Service (SSE-) AWS を使用して保存されますKMS。ソースイメージには影響しません。詳細については、「モデルのトレーニング」を参照してください。

Amazon Lookout for Vision モデル

デフォルトでは、トレーニング済みモデルとマニフェストファイルは、 Key Management Service (-) に保存されているキーによるサーバー側の暗号化を使用して Amazon S3 AWS で暗号化されますSSEKMS。 KMSLookout for Vision は AWS 所有のキーを使用します。詳細については、「サーバー側暗号化を使用するデータの保護」を参照してください。トレーニング結果は、`CreateModel` への `output_bucket` 入力パラメータで指定されたバケットに書き込まれます。トレーニング結果は、バケット (`output_bucket`) に構成された暗号化設定を使用して暗号化されます。

Amazon Lookout for Vision コンソールバケット

Amazon Lookout for Vision コンソールは、プロジェクトを管理するために使用できる Amazon S3 バケット (コンソールバケット) を作成します。コンソールバケットは、デフォルトの Amazon S3 暗号化を使用して暗号化されます。詳細については、「S3 バケット用 Amazon Simple Storage Service デフォルト暗号化」を参照してください。独自のKMSキーを使用している場合は、作成後にコンソールバケットを設定します。詳細については、「サーバー側暗号化を使用するデータの保護」を参照してください。Amazon Lookout for Vision は、コンソールバケットへのパブリックアクセスをブロックします。

転送中の暗号化

Amazon Lookout for Vision APIエンドポイントは、 経由の安全な接続のみをサポートしますHTTPS。すべての通信は Transport Layer Security () で暗号化されますTLS。

キー管理

AWS Key Management Service (KMS) を使用して、Amazon S3 バケットに保存する入力イメージの暗号化を管理できます。詳細については、「ステップ 5: (オプション) 独自の AWS Key Management Service キーの使用」を参照してください。

デフォルトでは、イメージは がAWS所有および管理するキーで暗号化されます。独自の Key Management Service (KMS) AWS キーを使用することもできます。詳細については、AWS「 Key Management Service の概念」を参照してください。

インターネットトラフィックのプライバシー

Amazon Lookout for Vision の Amazon Virtual Private Cloud (Amazon VPC) エンドポイントは、Amazon Lookout for Vision への接続のみVPCを許可する 内の論理エンティティです。Amazon VPCはリクエストを Amazon Lookout for Vision にルーティングし、レスポンスを にルーティングしますVPC。詳細については、「Amazon VPCユーザーガイドVPC」の「エンドポイント」を参照してください。Amazon Lookout for Vision で Amazon VPCエンドポイントを使用する方法については、「」を参照してくださいインターフェイス VPC エンドポイント (AWS PrivateLink) を使って Amazon Lookout for Vision にアクセスする。

Amazon Lookout for Vision 用 Identity and Access Management

AWS Identity and Access Management （ IAM) は、管理者が AWS リソースへのアクセスを安全に制御 AWS サービス するのに役立つ です。IAM 管理者は、誰を認証 (サインイン) し、誰に Lookout for Vision リソースの使用を承認する (アクセス許可を付与する) かを制御します。IAM は追加料金なしで AWS サービス 使用できる です。

トピック

- 対象者
- アイデンティティを使用した認証
- ポリシーを使用したアクセスの管理
- Amazon Lookout for Vision と の連携方法 IAM
- Amazon Lookout for Vision の アイデンティティベースポリシー例
- Amazon Lookout for Vision の AWS 管理ポリシー
- Amazon Lookout for Vision アイデンティティとアクセスのトラブルシューティング

対象者

AWS Identity and Access Management （ IAM) の使用方法は、Lookout for Vision で行う作業によって異なります。

サービスユーザー – ジョブを実行するために Lookout for Vision サービスを使用する場合は、管理者から必要な権限と認証情報が与えられます。作業を実行するためにさらに多くの Lookout for Vision の特徴を使用するとき、追加の権限が必要になる場合があります。アクセスの管理方法を理解しておくと、管理者に適切な権限をリクエストするうえで役立ちます。Lookout for Vision の特徴にアクセスできない場合は、「Amazon Lookout for Vision アイデンティティとアクセスのトラブルシューティング」 を参照してください。

サービス管理者 - 社内の Lookout for Vision リソースを担当している場合は、通常、Lookout for Vision へのフルアクセスがあります。サービスユーザーがどの Lookout for Vision の特徴とリソースにアクセスする必要があるかを決定するのは管理者の仕事です。次に、サービスユーザーのアクセス許可を変更するリクエストをIAM管理者に送信する必要があります。このページの情報を確認して、の基本概念を理解してくださいIAM。会社で Lookout for Vision IAMを使用する方法の詳細については、「」を参照してくださいAmazon Lookout for Vision と の連携方法 IAM。

IAM 管理者 – IAM管理者は、Lookout for Vision へのアクセスを管理するポリシーの作成方法の詳細について確認する場合があります。で使用できる Lookout for Vision アイデンティティベースのポリシーの例を表示するにはIAM、「」を参照してくださいAmazon Lookout for Vision の アイデンティティベースポリシー例。

アイデンティティを使用した認証

認証とは、ID 認証情報 AWS を使用して にサインインする方法です。として、IAMユーザーとしてAWS アカウントのルートユーザー、または IAMロールを引き受けることによって認証 (にサインイン AWS) される必要があります。

ID ソースを介して提供された認証情報を使用して、フェデレーティッド ID AWS として にサインインできます。 AWS IAM Identity Center （ IAM Identity Center) ユーザー、会社のシングルサインオン認証、Google または Facebook の認証情報は、フェデレーティッド ID の例です。フェデレーティッド ID としてサインインすると、管理者は以前に IAMロールを使用して ID フェデレーションをセットアップしていました。フェデレーション AWS を使用して にアクセスすると、間接的にロールを引き受けることになります。

ユーザーのタイプに応じて、 AWS Management Console または AWS アクセスポータルにサインインできます。へのサインインの詳細については AWS、「 ユーザーガイド」の「 にサインインする方法 AWS アカウントAWS サインイン 」を参照してください。

AWS プログラムで にアクセスする場合、 はソフトウェア開発キット (SDK) とコマンドラインインターフェイス (CLI) AWS を提供し、認証情報を使用してリクエストに暗号で署名します。 AWS ツールを使用しない場合は、リクエストに自分で署名する必要があります。推奨される方法を使用し

てリクエストを自分で署名する方法の詳細については、「 IAMユーザーガイド」のAWS API「リクエストの署名」を参照してください。

使用する認証方法を問わず、追加セキュリティ情報の提供をリクエストされる場合もあります。例えば、 AWS では、アカウントのセキュリティを高めるために多要素認証 (MFA) を使用することをお勧めします。詳細については、「 ユーザーガイド」の「多要素認証」および「 ユーザーガイド」の「 での多要素認証 (MFA) AWS IAM の使用」を参照してください。 AWS IAM Identity Center

AWS アカウント ルートユーザー

を作成するときは AWS アカウント、アカウント内のすべての およびリソースへの AWS サービス 完全なアクセス権を持つ 1 つのサインインアイデンティティから始めます。この ID は AWS アカウント ルートユーザーと呼ばれ、アカウントの作成に使用した E メールアドレスとパスワードでサインインすることでアクセスできます。日常的なタスクには、ルートユーザーを使用しないことを強くお勧めします。ルートユーザーの認証情報は保護し、ルートユーザーでしか実行できないタスクを実行するときに使用します。ルートユーザーとしてサインインする必要があるタスクの完全なリストについては、「 IAMユーザーガイド」の「ルートユーザーの認証情報を必要とするタスク」を参照してください。

フェデレーティッドアイデンティティ

ベストプラクティスとして、管理者アクセスを必要とするユーザーを含む人間のユーザーに、一時的な認証情報を使用して にアクセスするための ID プロバイダーとのフェデレーションの使用を要求 AWS サービス します。

フェデレーティッド ID は、エンタープライズユーザーディレクトリ、ウェブ ID プロバイダー、AWS Directory Service、アイデンティティセンターディレクトリ、または ID ソースを通じて提供された認証情報 AWS サービス を使用して にアクセスするユーザーです。フェデレーティッド ID が にアクセスすると AWS アカウント、ロールが引き受けられ、ロールは一時的な認証情報を提供します。

アクセスを一元管理する場合は、 AWS IAM Identity Centerを使用することをお勧めします。Identity Center でユーザーとグループを作成することも、独自の IAM ID ソース内のユーザーとグループのセットに接続して同期して、すべての AWS アカウント とアプリケーションで使用することもできます。IAM Identity Center の詳細については、「 ユーザーガイド」のIAM「Identity Center とはAWS IAM Identity Center 」を参照してください。

IAM ユーザーとグループ

IAM ユーザーは、単一のユーザーまたはアプリケーションに対して特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。可能であれば、パスワードやアクセスキーなどの長期的な認証情報を持つIAMユーザーを作成する代わりに、一時的な認証情報を使用することをお勧めします。ただし、IAMユーザーとの長期的な認証情報を必要とする特定のユースケースがある場合は、アクセスキーをローテーションすることをお勧めします。詳細については、「 ユーザーガイド」の「長期的な認証情報を必要とするユースケースでアクセスキーを定期的にローテーションするIAM」を参照してください。

IAM グループは、IAMユーザーのコレクションを指定するアイデンティティです。グループとしてサインインすることはできません。グループを使用して、複数のユーザーに対して一度に権限を指定できます。多数のユーザーグループがある場合、グループを使用することで権限の管理が容易になります。例えば、 という名前のグループを作成しIAMAdmins、そのグループにIAMリソースを管理するアクセス許可を付与できます。

ユーザーは、ロールとは異なります。ユーザーは 1 人の人または 1 つのアプリケーションに一意に関連付けられますが、ロールはそれを必要とする任意の人が引き受けるようになっています。ユーザーには永続的な長期の認証情報がありますが、ロールでは一時認証情報が提供されます。詳細については、「 ユーザーガイド」のIAM「 (ロールの代わりに) ユーザーを作成する場合IAM」を参照してください。

IAM ロール

IAM ロールは、特定のアクセス許可 AWS アカウント を持つ 内のアイデンティティです。ユーザーと似ていますがIAM、特定のユーザーに関連付けられていません。IAM ロール を切り替える AWS Management Console ことで、 で ロールを一時的に引き受けることができます。ロールを引き受けるには、 または AWS API オペレーションを AWS CLI 呼び出すか、カスタム を使用しますURL。ロールの使用方法の詳細については、「 ユーザーガイド」のIAM「ロールの使用IAM」を参照してください。

IAM 一時的な認証情報を持つ ロールは、以下の状況で役立ちます。

- フェデレーションユーザーアクセス – フェデレーティッド ID に許可を割り当てるには、ロールを作成してそのロールの許可を定義します。フェデレーティッド ID が認証されると、その ID はロールに関連付けられ、ロールで定義されている許可が付与されます。フェデレーションのロールの詳細については、「 ユーザーガイド」の「サードパーティー ID プロバイダーのロールの作成IAM」を参照してください。IAM Identity Center を使用する場合は、アクセス許可セットを設定します。ID が認証後にアクセスできる内容を制御するために、IAMIdentity Center はアクセス

許可セットを のロールに関連付けますIAM。アクセス許可セットの詳細については、「AWS IAM Identity Center ユーザーガイド」の「アクセス許可セット」を参照してください。

- 一時的なIAMユーザーアクセス許可 – IAM ユーザーまたはロールは、 IAMロールを引き受けて、特定のタスクに対して異なるアクセス許可を一時的に引き受けることができます。
- クロスアカウントアクセス – IAMロールを使用して、別のアカウントのユーザー (信頼されたプリンシパル) がアカウントのリソースにアクセスすることを許可できます。クロスアカウントアクセスを許可する主な方法は、ロールを使用することです。ただし、一部の では AWS サービス、(ロールをプロキシとして使用する代わりに) ポリシーをリソースに直接アタッチできます。クロスアカウントアクセスのロールとリソースベースのポリシーの違いについては、「 ユーザーガイド」の「 でのクロスアカウントリソースアクセスIAMIAM」を参照してください。
- クロスサービスアクセス — 一部の は、他の の機能 AWS サービス を使用します AWS サービス。例えば、サービスで呼び出しを行うと、そのサービスが Amazon でアプリケーションを実行EC2したり、Amazon S3 にオブジェクトを保存したりするのが一般的です。サービスでは、呼び出し元プリンシパルの許可、サービスロール、またはサービスリンクロールを使用してこれを行う場合があります。
  - 転送アクセスセッション (FAS) – IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS サービス、ダウンストリームサービス AWS サービス へのリクエストのリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS サービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するための権限が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。
  - サービスロール – サービスロールは、ユーザーに代わってアクションを実行するためにサービスが引き受ける IAMロールです。IAM 管理者は、 内からサービスロールを作成、変更、削除できますIAM。詳細については、「 ユーザーガイド」の「 にアクセス許可を委任するロールの作成 AWS サービスIAM」を参照してください。
  - サービスにリンクされたロール – サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS サービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。
- Amazon で実行されているアプリケーション EC2 – IAMロールを使用して、EC2インスタンスで実行され、 AWS CLI または AWS API リクエストを行うアプリケーションの一時的な認証情報を管

理できます。これは、EC2インスタンス内にアクセスキーを保存するよりも望ましいです。 AWS ロールをEC2インスタンスに割り当て、そのすべてのアプリケーションで使用できるようにするには、インスタンスにアタッチされたインスタンスプロファイルを作成します。インスタンスプロファイルには ロールが含まれており、EC2インスタンスで実行されているプログラムが一時的な認証情報を取得できるようにします。詳細については、「 ユーザーガイド」の「 IAMロールを使用して Amazon EC2インスタンスで実行されているアプリケーションにアクセス許可を付与するIAM」を参照してください。

IAM ロールとIAMユーザーのどちらを使用するかについては、「 ユーザーガイド」の「 (ユーザーではなく) IAMロールを作成する場合IAM」を参照してください。

ポリシーを使用したアクセスの管理

でアクセスを制御する AWS には、ポリシーを作成し、 AWS ID またはリソースにアタッチします。ポリシーは、アイデンティティまたはリソースに関連付けられているときにアクセス許可を定義するのオブジェクトです。プリンシパル (ユーザー、ルートユーザー、またはロールセッション) AWS がリクエストを行うと、 はこれらのポリシー AWS を評価します。ポリシーでの権限により、リクエストが許可されるか拒否されるかが決まります。ほとんどのポリシーはJSONドキュメント AWS として に保存されます。JSON ポリシードキュメントの構造と内容の詳細については、「 ユーザーガイド」のJSON「ポリシーの概要IAM」を参照してください。

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

デフォルトでは、ユーザーやロールに権限はありません。必要なリソースに対してアクションを実行するアクセス許可をユーザーに付与するために、IAM管理者はIAMポリシーを作成できます。その後、管理者はIAMポリシーをロールに追加し、ユーザーはロールを引き受けることができます。

IAM ポリシーは、オペレーションの実行に使用するメソッドに関係なく、アクションのアクセス許可を定義します。例えば、`iam:GetRole`アクションを許可するポリシーがあるとします。そのポリシーを持つユーザーは、 AWS Management Console、、 AWS CLIまたは AWS からロール情報を取得できますAPI。

アイデンティティベースのポリシー

ID ベースのポリシーは、IAMユーザー、ユーザーのグループ、ロールなどの ID にアタッチできるJSONアクセス許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行でき

るアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「 ユーザーガイド」のIAM「ポリシーの作成IAM」を参照してください。

アイデンティティベースのポリシーは、さらにインラインポリシーまたはマネージドポリシーに分類できます。インラインポリシーは、単一のユーザー、グループ、またはロールに直接埋め込まれています。管理ポリシーは、 内の複数のユーザー、グループ、ロールにアタッチできるスタンドアロンポリシーです AWS アカウント。管理ポリシーには、 AWS 管理ポリシーとカスタマー管理ポリシーが含まれます。管理ポリシーとインラインポリシーのどちらかを選択する方法については、IAM ユーザーガイドの「管理ポリシーとインラインポリシーの選択」を参照してください。

リソースベースのポリシー

リソースベースのポリシーは、リソースにアタッチするJSONポリシードキュメントです。リソースベースのポリシーの例としては、IAMロール信頼ポリシー や Amazon S3 バケットポリシー などがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS サービス。

リソースベースのポリシーは、そのサービス内にあるインラインポリシーです。リソースベースのポリシーIAMでは、 の AWS 管理ポリシーを使用できません。

アクセスコントロールリスト (ACLs)

アクセスコントロールリスト (ACLs) は、リソースへのアクセス許可を持つプリンシパル (アカウントメンバー、ユーザー、またはロール) を制御します。ACLs はリソースベースのポリシーに似ていますが、JSONポリシードキュメント形式を使用しません。

Amazon S3、 AWS WAF、および Amazon VPCは、 をサポートするサービスの例ですACLs。の詳細についてはACLs、Amazon Simple Storage Service デベロッパーガイドの「アクセスコントロールリスト (ACL) の概要」を参照してください。

その他のポリシータイプ

AWS は、一般的ではない追加のポリシータイプをサポートします。これらのポリシータイプでは、より一般的なポリシータイプで付与された最大の権限を設定できます。

- アクセス許可の境界 – アクセス許可の境界は、アイデンティティベースのポリシーがIAMエンティティ (IAMユーザーまたはロール) に付与できるアクセス許可の上限を設定する高度な機能です。エ

ンティティにアクセス許可の境界を設定できます。結果として得られる権限は、エンティティのアイデンティティベースポリシーとそのアクセス許可の境界の共通部分になります。Principal フィールドでユーザーまたはロールを指定するリソースベースのポリシーでは、アクセス許可の境界は制限されません。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。アクセス許可の境界の詳細については、「 IAMユーザーガイド」の「 IAMエンティティのアクセス許可の境界」を参照してください。

- サービスコントロールポリシー (SCPs) – SCPsは、 の組織または組織単位 (OU) に対する最大アクセス許可を指定するJSONポリシーです AWS Organizations。 AWS Organizations は、 AWS アカウント ビジネスが所有する複数の をグループ化して一元管理するためのサービスです。組織内のすべての機能を有効にすると、サービスコントロールポリシー (SCPs) をアカウントの一部またはすべてに適用できます。は、各 を含むメンバーアカウントのエンティティのアクセス許可SCP を制限します AWS アカウントのルートユーザー。Organizations と の詳細についてはSCPs、「 AWS Organizations ユーザーガイド」の「サービスコントロールポリシー」を参照してください。
- セッションポリシー - セッションポリシーは、ロールまたはフェデレーションユーザーの一時的なセッションをプログラムで作成する際にパラメータとして渡す高度なポリシーです。結果としてセッションの権限は、ユーザーまたはロールのアイデンティティベースポリシーとセッションポリシーの共通部分になります。また、リソースベースのポリシーから権限が派生する場合もあります。これらのポリシーのいずれかを明示的に拒否した場合、権限は無効になります。詳細については、「 ユーザーガイド」の「セッションポリシーIAM」を参照してください。

複数のポリシータイプ

1 つのリクエストに複数のタイプのポリシーが適用されると、結果として作成される権限を理解するのがさらに難しくなります。複数のポリシータイプが関係する場合にリクエストを許可するかどうか AWS を決定する方法については、「 ユーザーガイド」の「ポリシー評価ロジックIAM」を参照してください。

Amazon Lookout for Vision と の連携方法 IAM

IAM を使用して Lookout for Vision へのアクセスを管理する前に、Lookout for Vision で使用できるIAM機能を確認してください。

IAM Amazon Lookout for Vision で使用できる機能

IAM 機能	Lookout for Vision サポート
アイデンティティベースのポリシー	あり
リソースベースのポリシー	なし
ポリシーアクション	あり
ポリシーリソース	はい
ポリシー条件キー (サービス固有)	あり
ACLs	なし
ABAC (ポリシー内のタグ)	部分的
一時的な認証情報	あり
転送アクセスセッション (FAS)	あり
サービスロール	いいえ
サービスリンクロール	なし

Lookout for Vision およびその他の AWS のサービスがほとんどの IAM 機能と連携する方法の概要を把握するには、「 IAMユーザーガイド」のAWS 「 と連携する のサービスIAM」を参照してください。

Lookout for Vision のアイデンティティベースポリシー

アイデンティティベースのポリシーのサポート: あり

ID ベースのポリシーは、IAMユーザー、ユーザーのグループ、ロールなどの ID にアタッチできるJSONアクセス許可ポリシードキュメントです。これらのポリシーは、ユーザーとロールが実行できるアクション、リソース、および条件をコントロールします。アイデンティティベースのポリシーを作成する方法については、「 ユーザーガイド」のIAM「ポリシーの作成IAM」を参照してください。

IAM アイデンティティベースのポリシーでは、許可または拒否されたアクションとリソース、およびアクションが許可または拒否される条件を指定できます。プリンシパルは、それが添付されてい

るユーザーまたはロールに適用されるため、アイデンティティベースのポリシーでは指定できません。JSON ポリシーで使用できるすべての要素については、「 ユーザーガイド」の「 IAMJSONポリシー要素のリファレンスIAM」を参照してください。

Amazon Lookout for Vision の アイデンティティベースポリシー例

Lookout for Vision アイデンティティベースのポリシーの例を表示するには、「Amazon Lookout for Vision の アイデンティティベースポリシー例」 を参照してください。

Lookout for Vision 内のリソースベースポリシー

リソースベースのポリシーのサポート: なし

リソースベースのポリシーは、リソースにアタッチするJSONポリシードキュメントです。リソースベースのポリシーの例としては、IAMロールの信頼ポリシー や Amazon S3 バケットポリシー などがあります。リソースベースのポリシーをサポートするサービスでは、サービス管理者はポリシーを使用して特定のリソースへのアクセスをコントロールできます。ポリシーがアタッチされているリソースの場合、指定されたプリンシパルがそのリソースに対して実行できるアクションと条件は、ポリシーによって定義されます。リソースベースのポリシーでは、プリンシパルを指定する必要があります。プリンシパルには、アカウント、ユーザー、ロール、フェデレーティッドユーザー、または を含めることができます AWS サービス。

クロスアカウントアクセスを有効にするには、リソースベースのポリシーのプリンシパルとして、アカウント全体または別のアカウントのIAMエンティティを指定できます。リソースベースのポリシーにクロスアカウントのプリンシパルを追加しても、信頼関係は半分しか確立されない点に注意してください。プリンシパルとリソースが異なる にある場合 AWS アカウント、信頼されたアカウントのIAM管理者は、プリンシパルエンティティ (ユーザーまたはロール) にリソースへのアクセス許可も付与する必要があります。IAM 管理者は、アイデンティティベースのポリシーをエンティティにアタッチすることで権限を付与します。ただし、リソースベースのポリシーで、同じアカウントのプリンシパルへのアクセス権が付与されている場合は、アイデンティティベースのポリシーをさらに付与する必要はありません。詳細については、「 ユーザーガイド」の「 でのクロスアカウントリソースアクセスIAMIAM」を参照してください。

Lookout for Vision のポリシーアクション

ポリシーアクションのサポート: あり

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

JSON ポリシーの Action要素は、ポリシーでアクセスを許可または拒否するために使用できるアクションを記述します。ポリシーアクションの名前は通常、関連する AWS APIオペレーションと同じです。一致するAPIオペレーションがないアクセス許可のみのアクションなど、いくつかの例外があります。また、ポリシーに複数のアクションが必要なオペレーションもあります。これらの追加アクションは、依存アクションと呼ばれます。

このアクションは、関連付けられたオペレーションを実行するための権限を付与するポリシーで使用されます。

Lookout for Vision アクションのリストを表示するには、「サービス認可リファレンス」の「Amazon Lookout for Vision で定義されているアクション」を参照してください。

Lookout for Vision のポリシーアクションは、アクションの前に以下のプレフィックスを使用します:

```
lookoutvision
```

単一のステートメントで複数のアクションを指定するには、アクションをカンマで区切ります。

```
"Action": [
      "lookoutvision:action1",
      "lookoutvision:action2"
         ]
```

Lookout for Vision アイデンティティベースのポリシーの例を表示するには、「Amazon Lookout for Vision の アイデンティティベースポリシー例」 を参照してください。

Lookout for Vision のポリシーリソース

ポリシーリソースのサポート: あり

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルがどのリソースに対してどのような条件下でアクションを実行できるかということです。

Policy ResourceJSON要素は、アクションが適用されるオブジェクトを指定します。ステートメントには、Resource または NotResource 要素を含める必要があります。ベストプラクティスとして、Amazon リソースネーム (ARN) を使用してリソースを指定します。これは、リソースレベルの許可と呼ばれる特定のリソースタイプをサポートするアクションに対して実行できます。

オペレーションのリスト化など、リソースレベルの権限をサポートしないアクションの場合は、ステートメントがすべてのリソースに適用されることを示すために、ワイルドカード (*) を使用します。

```
"Resource": "*"
```

Lookout for Vision リソースタイプとその のリストを確認するにはARNs、「サービス認証リファレンス」の「Amazon Lookout for Vision で定義されるリソース」を参照してください。各リソースARNの を指定できるアクションについては、「Amazon Lookout for Vision で定義されるアクション」を参照してください。

Lookout for Vision アイデンティティベースのポリシーの例を表示するには、「Amazon Lookout for Vision の アイデンティティベースポリシー例」 を参照してください。

Amazon Lookout for Vision のポリシー条件キー

サービス固有のポリシー条件キーのサポート: あり

管理者はポリシーを使用して AWS JSON、誰が何にアクセスできるかを指定できます。つまり、どのプリンシパルが、どのリソースに対してどのような条件下でアクションを実行できるかということです。

`Condition` 要素 (または `Condition` ブロック) を使用すると、ステートメントが有効な条件を指定できます。`Condition` 要素はオプションです。イコールや未満などの 条件演算子 を使用して条件式を作成することで、ポリシーの条件とリクエスト内の値を一致させることができます。

1 つのステートメントに複数の `Condition` 要素を指定する場合、または 1 つの `Condition` 要素に複数のキーを指定する場合、 AWS では `AND` 論理演算子を使用してそれらを評価します。1 つの条件キーに複数の値を指定すると、 は論理ORオペレーションを使用して条件 AWS を評価します。ステートメントの権限が付与される前にすべての条件が満たされる必要があります。

条件を指定する際にプレースホルダー変数も使用できます。例えば、ユーザー名でタグ付けされている場合にのみ、リソースにアクセスするアクセス許可をIAMユーザーに付与できますIAM。詳細については、「 ユーザーガイド」のIAM「ポリシー要素: 変数とタグIAM」を参照してください。

AWS は、グローバル条件キーとサービス固有の条件キーをサポートします。すべての AWS グローバル条件キーを確認するには、「 ユーザーガイド」のAWS 「 グローバル条件コンテキストキーIAM」を参照してください。

Lookout for Vision の条件キーのリストを確認するには、「サービス認可リファレンス」の「Amazon Lookout for Vision の条件キー」を参照してください。どのアクションやリソースで条件キーを使用できるかについては、「Amazon Lookout for Vision で定義されるアクション」を参照してください。

Lookout for Vision アイデンティティベースのポリシーの例を表示するには、「Amazon Lookout for Vision の アイデンティティベースポリシー例」 を参照してください。

ACLs Lookout for Vision の

をサポートACLs： いいえ

アクセスコントロールリスト (ACLs) は、リソースへのアクセス許可を持つプリンシパル (アカウントメンバー、ユーザー、またはロール) を制御します。ACLs はリソースベースのポリシーに似ていますが、JSONポリシードキュメント形式を使用しません。

ABAC Lookout for Vision を使用する

サポート ABAC (ポリシー内のタグ): 部分的

属性ベースのアクセスコントロール (ABAC) は、属性に基づいてアクセス許可を定義する認可戦略です。では AWS、これらの属性はタグ と呼ばれます。タグは、IAMエンティティ (ユーザーまたはロール) および多くの AWS リソースにアタッチできます。エンティティとリソースのタグ付けは、の最初のステップですABAC。次に、プリンシパルのタグがアクセスしようとしているリソースのタグと一致する場合に、オペレーションを許可するABACポリシーを設計します。

ABAC は、急速に成長している環境や、ポリシー管理が煩雑になる状況に役立ちます。

タグに基づいてアクセスを管理するには、`aws:ResourceTag/key-name`、`aws:RequestTag/key-name`、または `aws:TagKeys` の条件キーを使用して、ポリシーの条件要素でタグ情報を提供します。

サービスがすべてのリソースタイプに対して 3 つの条件キーすべてをサポートする場合、そのサービスの値はありです。サービスが一部のリソースタイプに対してのみ 3 つの条件キーのすべてをサポートする場合、値は「部分的」になります。

の詳細についてはABAC、「 ユーザーガイド」の「 とはABACIAM」を参照してください。のセットアップ手順を含むチュートリアルを表示するにはABAC、「 ユーザーガイド」の「属性ベースのアクセスコントロール (ABAC) を使用するIAM」を参照してください。

Lookout for Vision での一時的な認証情報の使用

一時的な認証情報のサポート: あり

一部の は、一時的な認証情報を使用してサインインすると機能 AWS サービス しません。一時的な認証情報 AWS サービス を使用する などの詳細については、 ユーザーガイドのAWS サービス「 と連携する IAM IAM 」を参照してください。

ユーザー名とパスワード以外の AWS Management Console 方法で にサインインする場合、一時的な認証情報を使用します。例えば、会社のシングルサインオン (SSO) リンク AWS を使用して にアクセスすると、そのプロセスによって一時的な認証情報が自動的に作成されます。また、ユーザーとしてコンソールにサインインしてからロールを切り替える場合も、一時的な認証情報が自動的に作成されます。ロールの切り替えの詳細については、「 IAMユーザーガイド」の「ロールへの切り替え (コンソール) 」を参照してください。

一時的な認証情報は、 AWS CLI または を使用して手動で作成できます AWS API。その後、これらの一時的な認証情報を使用して . AWS recommends にアクセスできます AWS。これは、長期的なアクセスキーを使用する代わりに、一時的な認証情報を動的に生成することを推奨しています。詳細については、「」の「一時的なセキュリティ認証情報IAM」を参照してください。

Lookout for Vision の転送アクセスセッション

転送アクセスセッションをサポート (FAS): はい

IAM ユーザーまたはロールを使用して でアクションを実行すると AWS、プリンシパルと見なされます。一部のサービスを使用する際に、アクションを実行することで、別のサービスの別のアクションがトリガーされることがあります。FAS は、 を呼び出すプリンシパルのアクセス許可を AWS サービス、ダウンストリームサービス AWS サービス へのリクエストのリクエストと組み合わせて使用します。FAS リクエストは、サービスが他の AWS サービス またはリソースとのやり取りを完了する必要があるリクエストを受け取った場合にのみ行われます。この場合、両方のアクションを実行するための権限が必要です。FAS リクエストを行う際のポリシーの詳細については、「転送アクセスセッション」を参照してください。

Lookout for Vision のサービスロール

サービスロールのサポート: なし

サービスロールは、サービスがユーザーに代わってアクションを実行するために引き受ける IAM ロールです。IAM 管理者は、 内からサービスロールを作成、変更、削除できますIAM。詳細につい

ては、「 ユーザーガイド」の「 にアクセス許可を委任するロールの作成 AWS サービスIAM」を参照してください。

> **Warning**
>
> サービスロールの権限を変更すると、Lookout for Vision の機能が中断する可能性があります。Lookout for Vision が指示する場合以外は、サービスロールを編集しないでください。

Lookout for Vision のサービス連動ロール

サービスにリンクされたロールのサポート: なし

サービスにリンクされたロールは、 にリンクされたサービスロールの一種です AWS サービス。サービスは、ユーザーに代わってアクションを実行するロールを引き受けることができます。サービスにリンクされたロールは に表示され AWS アカウント 、サービスによって所有されます。IAM 管理者は、サービスにリンクされたロールのアクセス許可を表示できますが、編集することはできません。

サービスにリンクされたロールの作成または管理の詳細については、「 AWS と連携する のサービスIAM」を参照してください。表の中から、[Service-linked role] (サービスにリンクされたロール) 列に Yes と記載されたサービスを見つけます。サービスにリンクされたロールに関するドキュメントをサービスで表示するには、[はい] リンクを選択します。

Amazon Lookout for Vision の アイデンティティベースポリシー例

デフォルトでは、ユーザーとロールには、Lookout for Vision リソースを作成または変更する権限はありません。また、、 AWS Command Line Interface （ AWS CLI ） AWS Management Console、または を使用してタスクを実行することはできません AWS API。必要なリソースに対してアクションを実行するアクセス許可をユーザーに付与するために、IAM管理者はIAMポリシーを作成できます。その後、管理者はIAMポリシーをロールに追加し、ユーザーはロールを引き受けることができます。

これらのポリシードキュメント例を使用してIAMアイデンティティベースのJSONポリシーを作成する方法については、「 ユーザーガイド」のIAM「ポリシーの作成IAM」を参照してください。

各リソースタイプの の形式など、Lookout for Vision で定義されるアクションとリソースタイプの詳細については、「サービス認証リファレンスARNs」の「Amazon Lookout for Vision のアクション、リソース、および条件キー」を参照してください。

トピック

- ポリシーのベストプラクティス
- Amazon Lookout for Vision のシングルプロジェクトにアクセスする
- タグベースのポリシーの例

ポリシーのベストプラクティス

ID ベースのポリシーは、お使いのアカウントで Lookout for Vision リソースを作成、アクセス、または削除できるのは誰かを決定します。これらのアクションを実行すると、 AWS アカウントに料金が発生する可能性があります。アイデンティティベースポリシーを作成したり編集したりする際には、以下のガイドラインと推奨事項に従ってください:

- AWS 管理ポリシーを開始し、最小特権のアクセス許可に移行する – ユーザーとワークロードにアクセス許可を付与するには、多くの一般的なユースケースにアクセス許可を付与する AWS 管理ポリシーを使用します。これらは で使用できます AWS アカウント。ユースケースに固有の AWS カスタマー管理ポリシーを定義して、アクセス許可をさらに減らすことをお勧めします。詳細については、「 ユーザーガイド」の「 AWS 管理ポリシーAWS 」または「 ジョブ機能の 管理ポリシーIAM」を参照してください。
- 最小特権のアクセス許可を適用する – IAMポリシーでアクセス許可を設定する場合は、タスクの実行に必要なアクセス許可のみを付与します。これを行うには、特定の条件下で特定のリソースに対して実行できるアクションを定義します。これは、最小特権アクセス許可とも呼ばれています。IAM を使用してアクセス許可を適用する方法の詳細については、「 ユーザーガイド」の「 のポリシーとアクセス許可IAMIAM」を参照してください。
- IAM ポリシーの条件を使用してアクセスをさらに制限する – ポリシーに条件を追加して、アクションとリソースへのアクセスを制限できます。例えば、ポリシー条件を記述して、すべてのリクエストを を使用して送信する必要があることを指定できますSSL。条件を使用して、 などの特定の を介してサービスアクションが使用される場合に AWS サービス、サービスアクションへのアクセスを許可することもできます AWS CloudFormation。詳細については、「 ユーザーガイド」のIAMJSON「ポリシー要素: 条件IAM」を参照してください。
- IAM Access Analyzer を使用してIAMポリシーを検証し、安全で機能的なアクセス許可を確保する – IAM Access Analyzer は、ポリシーがポリシー言語 (JSON) とIAMベストプラクティスに準拠するように、新規および既存のIAMポリシーを検証します。IAM Access Analyzer には、安全で機能的なポリシーの作成に役立つ 100 を超えるポリシーチェックと実用的な推奨事項が用意されています。詳細については、「 ユーザーガイド」のIAM「Access Analyzer ポリシーの検証IAM」を参照してください。
- 多要素認証を要求する (MFA) – でIAMユーザーまたはルートユーザーを必要とするシナリオがある場合は AWS アカウント、セキュリティを強化MFAするために をオンにします。API オペレー

ションが呼び出されるMFAタイミングを要求するには、ポリシーにMFA条件を追加します。詳細については、「 IAMユーザーガイド」のMFA「 で保護されたAPIアクセスの設定」を参照してください。

のベストプラクティスの詳細についてはIAM、「 ユーザーガイド」の「 のセキュリティのベストプラクティスIAMIAM」を参照してください。

Amazon Lookout for Vision のシングルプロジェクトにアクセスする

この例では、 AWS アカウントのユーザーに Amazon Lookout for Vision プロジェクトの 1 つへのアクセス権を付与します。

```
{
    "Sid": "SpecificProjectOnly",
    "Effect": "Allow",
    "Action": [
        "lookoutvision:DetectAnomalies"
    ],
    "Resource": "arn:aws:lookoutvision:us-east-1:123456789012:model/myproject/*"
}
```

タグベースのポリシーの例

タグベースのポリシーは、プリンシパルがタグ付けされたリソースに対して実行できるアクションを指定するJSONポリシードキュメントです。

タグを使用したリソースへのアクセス

このポリシー例では、 キーstageと値 でタグ付けされたモデルで DetectAnomaliesオペレーションを使用するアクセス許可をAWSアカウントのユーザーまたはロールに付与しますproduction。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "LookoutVision:DetectAnomalies"
            ],
            "Resource": "*",
```

```
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": "production"
                }
            }
        }
    ]
}
```

タグを使用して、特定の Amazon Lookout for Vision オペレーションへのアクセスを拒否する

このポリシー例では、 キーstageと値 でタグ付けされたモデルで DeleteModel または StopModelオペレーションを呼び出すためのAWS、アカウントのユーザーまたはロールのアクセス許可を拒否しますproduction。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Action": [
                "LookoutVision:DeleteModel",
                "LookoutVision:StopModel"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:ResourceTag/stage": "production"
                }
            }
        }
    ]
}
```

Amazon Lookout for Vision の AWS 管理ポリシー

AWS マネージドポリシーは、AWS が作成および管理するスタンドアロンポリシーです。AWS マネージドポリシーは、多くの一般的なユースケースでアクセス許可を提供できるように設計されているため、ユーザー、グループ、ロールへのアクセス許可の割り当てを開始できます。

AWS マネージドポリシーは、ご利用の特定のユースケースに対して最小特権のアクセス許可を付与しない場合があることにご注意ください。AWS のすべてのお客様が使用できるようになるのを避けるためです。ユースケース別にカスタマー管理ポリシーを定義することで、アクセス許可を絞り込むことをお勧めします。

AWS 管理ポリシーで定義したアクセス権限は変更できません。AWS が AWS マネージドポリシーに定義されているアクセス許可を更新すると、更新はポリシーがアタッチされているすべてのプリンシパルアイデンティティ (ユーザー、グループ、ロール) に影響します。新しい AWS サービスを起動するか、既存のサービスで新しい API オペレーションが使用可能になると、AWS が AWS マネージドポリシーを更新する可能性が最も高くなります。

詳細については、「IAM ユーザーガイド」の「AWS 管理ポリシー」を参照してください。

AWS 管理ポリシー:AmazonLookoutVisionReadOnlyAccess

以下の Amazon Lookout for Vision アクション (SDK オペレーション) で、Amazon Lookout for Vision (およびその依存関係) に対する読み取り専用アクセスをユーザーに許可する `AmazonLookoutVisionReadOnlyAccess` ポリシーを使用します。例えば、`DescribeModel` を使用して、既存のモデルに関する情報を取得します。

- DescribeDataset
- DescribeModel
- DescribeModelPackagingJob
- DescribeProject
- ListDatasetEntries
- ListModelPackagingJobs
- ListModels
- ListProjects
- ListTagsForResource

読み取り専用アクションを呼び出すために、ユーザーは Amazon S3 バケットのアクセス許可を必要としません。ただし、オペレーションレスポンスには Amazon S3 バケットへの参照が含まれる場合があります。例えば、`source-ref` からのレスポンス内の `ListDatasetEntries` エントリは Amazon S3 バケット内の画像への参照です。ユーザーが参照バケットにアクセスする必要があ

る場合は、Amazon S3 バケットのアクセス許可を追加します。例えば、ユーザーは、source-ref フィールドから参照される画像をダウンロードしたいと思うかも知れません。詳細については、「Amazon S3 バケット権限の付与」を参照してください。

AmazonLookoutVisionReadOnlyAccess ポリシーは IAM ID にアタッチできます。

許可の詳細

このポリシーには、以下の許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:DescribeDataset",
                "lookoutvision:DescribeModel",
                "lookoutvision:DescribeProject",
                "lookoutvision:DescribeModelPackagingJob",
                "lookoutvision:ListDatasetEntries",
                "lookoutvision:ListModels",
                "lookoutvision:ListProjects",
                "lookoutvision:ListTagsForResource",
                "lookoutvision:ListModelPackagingJobs"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 管理ポリシー:AmazonLookoutVisionFullAccess

以下の Amazon Lookout for Vision アクション (SDK オペレーション) で、Amazon Lookout for Vision (およびその依存関係) に対するフルアクセスをユーザーに許可する AmazonLookoutVisionFullAccess ポリシーを使用します。例えば、Amazon Lookout for Vision コンソールを使用せずにモデルをトレーニングできます。詳細については、「アクション」を参照してください。

データセットの作成 (CreateDataset) またはモデルの作成 (CreateModel) を行うには、データセットイメージ、Amazon SageMaker Ground Truth マニフェストファイル、およびトレーニング出力を保存するAmazon S3バケットへのフルアクセス権限がユーザーに付与されている必要があります。詳細については、「ステップ 2: 権限をセットアップする」を参照してください。

また、AmazonLookoutVisionConsoleFullAccess ポリシーを使用して Amazon Lookout for Vision SDK アクションにアクセス許可を付与することもできます。

AmazonLookoutVisionFullAccess ポリシーは IAM ID にアタッチできます。

許可の詳細

このポリシーには、以下の許可が含まれています。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionFullAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:*"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 管理ポリシー:AmazonLookoutVisionConsoleFullAccess

AmazonLookoutVisionFullAccess ポリシーを使用して、Amazon Lookout for Vision コンソール、アクション (SDK オペレーション)、およびサービスの依存関係へのフルアクセスをユーザーに許可します。詳細については、「Amazon Lookout for Vision コンソールの開始」を参照してください。

LookoutVisionConsoleFullAccess ポリシーには、Amazon Lookout for Vision コンソールバケットに対するアクセス許可が含まれます。コンソールバケットの詳細については、「ステップ 3: コンソールバケットを作成する」 を参照してください。データセット、画像、および Amazon

SageMaker Ground Truth マニフェストファイルを別の Amazon S3 バケットに格納するには、ユーザーに追加のアクセス許可が必要です。詳細については、「the section called “Amazon S3 バケット許可をセッティングする”」を参照してください。

AmazonLookoutVisionConsoleFullAccess ポリシーは IAM ID にアタッチできます。

アクセス許可のグループ化

このポリシーは、提供された一連のアクセス許可に基づくステートメントごとにグループ化されます。

- LookoutVisionFullAccess — すべての Lookout for Vision アクションを実行するためのアクセスを許可します。
- LookoutVisionConsoleS3BucketSearchAccess — 発信者が所有するすべての Amazon S3 バケットのリストを許可します。Lookout for Vision では、このアクションを使用して AWS リージョン固有の Lookout for Vision コンソールバケットが呼び出し元のアカウントに存在する場合、そのバケットを特定します。
- LookoutVisionConsoleS3BucketFirstUseSetupAccessPermissions — Lookout for Vision コンソールバケットの名前パターンに一致する Amazon S3 バケットの作成と設定を許可します。Lookout for Vision では、これらのアクションを使用して、リージョン固有の Lookout for Vision コンソールバケットが見つからない場合に、Lookout for Vision コンソールバケットを作成および設定します。
- LookoutVisionConsoleS3BucketAccess — Lookout for Vision コンソールバケット名パターンに一致するバケットに対する依存する Amazon S3 アクションを許可します。Lookout for Vision は、Amazon S3 バケットからデータセットを作成する際と、トライアル検出タスクを開始する際に、s3:ListBucket を使用して画像オブジェクトを検索します。Lookout for Vision は s3:GetBucketLocation と s3:GetBucketVersioning を用いて、以下の一環としてバケットの AWS リージョン、所有者、コンフィギュレーションを検証しています。
  - データセットの作成
  - モデルのトレーニング
  - トライアル検出タスクの開始
  - トライアル検出フィードバックの実行

  LookoutVisionConsoleS3ObjectAccess — Lookout for Vision コンソールバケット名パターンに一致するバケット内の Amazon S3 オブジェクトの読み取りと書き込みを許可しま

す。Lookout for Vision では、これらのアクションを使用して、コンソールギャラリービューに画像を表示し、データセットで使用する新しい画像をアップロードします。さらに、これらの権限により、Lookout for Vision は、データセットの作成、モデルのトレーニング、トライアル検出タスクの開始、および試行検出フィードバックの実行中にメタデータを書き出すことができます。

- `LookoutVisionConsoleDatasetLabelingToolsAccess` — 依存する Amazon SageMaker GroundTruth ラベリングアクションを許可します。Lookout for Vision は、これらのアクションを使用して S3 バケットをスキャンして画像を検索し、GroundTruth マニフェストファイルを作成し、試行検出タスクの結果を検証ラベルで注釈付けします。
- `LookoutVisionConsoleDashboardAccess` - Amazon CloudWatch メトリクスの読み取りを許可します。Lookout for Vision では、これらのアクションを使用して、ダッシュボードのグラフと異常が検出された統計情報を入力します。
- `LookoutVisionConsoleTagSelectorAccess` — アカウント固有のタグキーとタグ値の提案の読み取りを許可します。Lookout for Visionは、「タグを管理」コンソールページでタグキーとタグ値の推奨値を提供するために、これらの許可を使用します。
- `LookoutVisionConsoleKmsKeySelectorAccess` — AWS Key Management Service (KMS) のキーとエイリアスのリストアップを可能にします。Amazon Lookout for Vision は、この権限を使用して、暗号化のために顧客が管理する KMS キーをサポートする特定の Lookout for Vision アクションの推奨タグの選択で KMS キーを入力することができます。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionFullAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:*"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleS3BucketSearchAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },
```

```
        {
            "Sid": "LookoutVisionConsoleS3BucketFirstUseSetupAccess",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:PutBucketVersioning",
                "s3:PutLifecycleConfiguration",
                "s3:PutEncryptionConfiguration",
                "s3:PutBucketPublicAccessBlock"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*"
        },
        {
            "Sid": "LookoutVisionConsoleS3BucketAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetBucketLocation",
                "s3:GetBucketAcl",
                "s3:GetBucketVersioning"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*"
        },
        {
            "Sid": "LookoutVisionConsoleS3ObjectAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion",
                "s3:PutObject",
                "s3:AbortMultipartUpload",
                "s3:ListMultipartUploadParts"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*/*"
        },
        {
            "Sid": "LookoutVisionConsoleDatasetLabelingToolsAccess",
            "Effect": "Allow",
            "Action": [
                "groundtruthlabeling:RunGenerateManifestByCrawlingJob",
                "groundtruthlabeling:AssociatePatchToManifestJob",
                "groundtruthlabeling:DescribeConsoleJob"
            ],
            "Resource": "*"
```

```
        },
        {
            "Sid": "LookoutVisionConsoleDashboardAccess",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleTagSelectorAccess",
            "Effect": "Allow",
            "Action": [
                "tag:GetTagKeys",
                "tag:GetTagValues"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleKmsKeySelectorAccess",
            "Effect": "Allow",
            "Action": [
                "kms:ListAliases"
            ],
            "Resource": "*"
        }
    ]
}
```

AWS 管理ポリシー:AmazonLookoutVisionConsoleReadOnlyAccess

AmazonLookoutVisionConsoleReadOnlyAccess ポリシーを使用して、Amazon Lookout for Vision コンソール、アクション (SDK オペレーション)、およびサービスの依存関係への読み取り専用アクセスをユーザーに許可します。

AmazonLookoutVisionConsoleReadOnlyAccess ポリシーには、Amazon Lookout for Vision コンソールバケットに対する Amazon S3 アクセス許可が含まれます。データセットイメージまたは Amazon SageMaker Ground Truth マニフェストファイルが別の Amazon S3 バケットにある場合、ユーザーに追加の権限が必要です。詳細については、「the section called “Amazon S3 バケット許可をセッティングする”」を参照してください。

AmazonLookoutVisionConsoleReadOnlyAccess ポリシーは IAM ID にアタッチできます。

アクセス許可のグループ化

このポリシーは、提供された一連のアクセス許可に基づくステートメントごとにグループ化されます。

- LookoutVisionReadOnlyAccess — Lookout for Vision 読み取り専用アクションを実行するためのアクセスを許可します。
- LookoutVisionConsoleS3BucketSearchAccess — 発信者が所有するすべての Amazon S3 バケットのリストを許可します。Lookout for Vision では、このアクションを使用して AWS リージョン固有の Lookout for Vision コンソールバケットが発信者のアカウントに存在する場合、そのバケットを特定します。
- LookoutVisionConsoleS3ObjectReadAccess — Lookout for Vision コンソールバケットで Amazon S3 オブジェクトと Amazon S3 オブジェクトのバージョンを読み込むことができます。Lookout for Vision では、これらのアクションを使用して、データセット、モデル、およびトライアル検出で画像を表示します。
- LookoutVisionConsoleDashboardAccess — Amazon CloudWatch メトリクスの読み取りを許可します。Lookout for Vision では、これらのアクションを使用して、検出されたダッシュボードグラフおよび異常の統計情報を入力します。

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "LookoutVisionReadOnlyAccess",
            "Effect": "Allow",
            "Action": [
                "lookoutvision:DescribeDataset",
                "lookoutvision:DescribeModel",
                "lookoutvision:DescribeProject",
                "lookoutvision:DescribeTrialDetection",
                "lookoutvision:DescribeModelPackagingJob",
                "lookoutvision:ListDatasetEntries",
                "lookoutvision:ListModels",
                "lookoutvision:ListProjects",
                "lookoutvision:ListTagsForResource",
                "lookoutvision:ListTrialDetections",
```

```
                "lookoutvision:ListModelPackagingJobs"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleS3BucketSearchAccess",
            "Effect": "Allow",
            "Action": [
                "s3:ListAllMyBuckets"
            ],
            "Resource": "*"
        },
        {
            "Sid": "LookoutVisionConsoleS3ObjectReadAccess",
            "Effect": "Allow",
            "Action": [
                "s3:GetObject",
                "s3:GetObjectVersion"
            ],
            "Resource": "arn:aws:s3:::lookoutvision-*/*"
        },
        {
            "Sid": "LookoutVisionConsoleDashboardAccess",
            "Effect": "Allow",
            "Action": [
                "cloudwatch:GetMetricData",
                "cloudwatch:GetMetricStatistics"
            ],
            "Resource": "*"
        }
    ]
}
```

Lookout for Vision AWS 管理ポリシーの更新について

Lookout for Vision の AWS 管理ポリシーの更新に関する詳細を、このサービスがこれらの変更の追跡を開始した以降の分について表示します。このページの変更に関する自動通知については、Lookout for Visionドキュメント履歴ページの RSS フィードを購読してください。

	とAmazonLookoutvisionConsoleFullAccess ポリシーに以下のモデルパッケージオペレーションを追加しました: • DescribeModelPackagingJob • ListModelPackagingJobs • StartModelPackagingJob Amazon Lookout for Vision は、AmazonLookoutVisionReadOnlyAccess ポリシーとAmazonLookoutVisionConsoleReadOnlyAccess ポリシーに以下のモデルパッケージオペレーションを追加しました: • DescribeModelPackagingJob • ListModelPackagingJobs	
新しいポリシーが追加されました	Amazon Lookout for Vision では、以下のポリシーを追加しました。 • AmazonLookoutVisionReadOnlyAccess • AmazonLookoutVisionFullAccess • AmazonLookoutVisionConsoleFullAccess • AmazonLookoutVisionConsoleReadOnlyAccess	2021 年 5 月 11 日
Lookout for Vision は変更の追跡を開始しました	Amazon Lookout for Vision が AWS 管理ポリシーの変更のトラッキングを開始しました。	2021 年 3 月 1 日

Amazon Lookout for Vision アイデンティティとアクセスのトラブルシューティング

以下の情報は、Lookout for Vision および の使用時に発生する可能性がある一般的な問題の診断と修正に役立ちますIAM。

トピック

Lookout for Vision でアクションを実行する権限がない

「I am not authorized to perform an action in Amazon Bedrock」というエラーが表示された場合、そのアクションを実行できるようにポリシーを更新する必要があります。

次の例のエラーは、mateojacksonIAMユーザーが コンソールを使用して架空の*my-example-widget*リソースの詳細を表示しようとしているが、架空のlookoutvision:*GetWidget*アクセス許可がない場合に発生します。

```
User: arn:aws:iam::123456789012:user/mateojackson is not authorized to perform:
 lookoutvision:GetWidget on resource: my-example-widget
```

この場合、lookoutvision:*GetWidget* アクションを使用して *my-example-widget*リソースへのアクセスを許可するように、mateojackson ユーザーのポリシーを更新する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

iam を実行する権限がありません。PassRole

iam:PassRole アクションを実行する権限がないというエラーが表示された場合は、ポリシーを更新して Lookout for Vision にロールを渡すことができるようにする必要があります。

一部の AWS サービス では、新しいサービスロールまたはサービスにリンクされたロールを作成する代わりに、そのサービスに既存のロールを渡すことができます。そのためには、サービスにロールを渡す権限が必要です。

次の例のエラーは、 というIAMユーザーがコンソールを使用して Lookout for marymajor Vision でアクションを実行しようする場合に発生します。ただし、このアクションをサービスが実行するには、サービスロールから付与された権限が必要です。メアリーには、ロールをサービスに渡す許可がありません。

```
User: arn:aws:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

この場合、Mary のポリシーを更新してメアリーに iam:PassRole アクションの実行を許可する必要があります。

サポートが必要な場合は、 AWS 管理者にお問い合わせください。サインイン認証情報を提供した担当者が管理者です。

自分の 以外のユーザーに Lookout for Vision リソース AWS アカウント へのアクセスを許可したい

他のアカウントのユーザーや組織外の人が、リソースにアクセスするために使用できるロールを作成できます。ロールの引き受けを委託するユーザーを指定できます。リソースベースのポリシーまたはアクセスコントロールリスト (ACLs) をサポートするサービスでは、これらのポリシーを使用して、ユーザーにリソースへのアクセスを許可できます。

詳細については、以下を参照してください。

- Lookout for Vision がこれらの機能をサポートしているかどうかを確認するには、「Amazon Lookout for Vision と の連携方法 IAM」 を参照してください。
- 所有している のリソースへのアクセスを提供する方法については、 AWS アカウント 「 IAMユーザーガイド」の「所有 AWS アカウント している別の のIAMユーザーへのアクセスを提供する」を参照してください。
- リソースへのアクセスをサードパーティーの に提供する方法については AWS アカウント、「 ユーザーガイド」の「サードパーティー AWS アカウント が所有する へのアクセスを提供するIAM」を参照してください。
- ID フェデレーションを介してアクセスを提供する方法については、 IAMユーザーガイドの「外部認証されたユーザーへのアクセスの提供 (ID フェデレーション) 」を参照してください。
- クロスアカウントアクセスでのロールとリソースベースのポリシーの使用の違いについては、「 ユーザーガイド」の「 でのクロスアカウントリソースアクセスIAMIAM」を参照してください。

Amazon Lookout for Vision のコンプライアンス検証

サードパーティーの監査者は、複数の コンプライアンスプログラムの一環として Amazon Lookout for Vision のセキュリティと AWS コンプライアンスを評価します。Amazon Lookout for Vision は、一般データ保護規則 (GDPR) に準拠しています。

AWS サービス が特定のコンプライアンスプログラムの範囲内にあるかどうかを確認するには、コンプライアンスプログラムAWS サービス による対象範囲内のコンプライアンスプログラムを参照し、関心のあるコンプライアンスプログラムを選択します。一般的な情報については、AWS「コンプライアンスプログラム」を参照してください。

を使用して、サードパーティーの監査レポートをダウンロードできます AWS Artifact。詳細については、「 でのレポートのダウンロード AWS Artifact」の」を参照してください。

を使用する際のお客様のコンプライアンス責任 AWS サービス は、お客様のデータの機密性、貴社のコンプライアンス目的、適用される法律および規制によって決まります。 では、コンプライアンスに役立つ以下のリソース AWS を提供しています。

- セキュリティとコンプライアンスのクイックスタートガイド – これらのデプロイガイドでは、アーキテクチャ上の考慮事項について説明し、セキュリティとコンプライアンスに重点を置いたベースライン環境 AWS を にデプロイする手順について説明します。
- アマゾン ウェブ サービスHIPAAのセキュリティとコンプライアンスのためのアーキテクチャ – このホワイトペーパーでは、企業が AWS を使用して HIPAA対象アプリケーションを作成する方法について説明します。

> **Note**
>
> すべての AWS サービス がHIPAA対象となるわけではありません。詳細については、HIPAA「対象サービスリファレンス」を参照してください。

- AWS コンプライアンスリソース – このワークブックとガイドのコレクションは、お客様の業界や地域に適用される場合があります。
- AWS カスタマーコンプライアンスガイド — コンプライアンスの観点から責任共有モデルを理解します。このガイドでは、ガイダンスを保護し AWS サービス 、複数のフレームワーク (米国国立標準技術研究所 (NIST) 、Payment Card Industry Security Standards Council ()、PCI国際標準化機構 (ISO) など) のセキュリティコントロールにマッピングするためのベストプラクティスをまとめています。

- 「 デベロッパーガイド」の「 ルールによるリソースの評価」 – この AWS Config サービスは、リソース設定が社内プラクティス、業界ガイドライン、および規制にどの程度準拠しているかを評価します。 AWS Config
- AWS Security Hub – これにより AWS サービス 、 内のセキュリティ状態を包括的に把握できます AWS。Security Hub では、セキュリティコントロールを使用して AWS リソースを評価し、セキュリティ業界標準とベストプラクティスに対するコンプライアンスをチェックします。サポートされているサービスとコントロールのリストについては、Security Hub のコントロールリファレンスを参照してください。
- Amazon GuardDuty – これにより AWS アカウント、疑わしいアクティビティや悪意のあるアクティビティがないか環境を監視することで、、、ワークロード、コンテナ、データに対する潜在的な脅威 AWS サービス を検出します。 GuardDuty は、特定のコンプライアンスフレームワークで義務付けられている侵入検知要件を満たすことでDSS、 PCI などのさまざまなコンプライアンス要件への対応に役立ちます。
- AWS Audit Manager – これにより AWS サービス 、 AWS 使用状況を継続的に監査し、リスクの管理方法と規制や業界標準への準拠を簡素化できます。

Amazon Lookout for Vision での耐障害性

AWS のグローバルインフラストラクチャは AWS リージョンとアベイラビリティーゾーンを中心に構築されます。AWSリージョンには、低レイテンシー、高いスループット、そして高度の冗長ネットワークで Connect されている複数の物理的に独立· 隔離されたアベイラビリティーゾーンがあります。アベイラビリティーゾーンでは、ゾーン間で中断することなく自動的にフェイルオーバーするアプリケーションとデータベースを設計および運用することができます。アベイラビリティーゾーンは、従来の単一または複数のデータセンターインフラストラクチャよりも可用性、耐障害性、および拡張性が優れています。

AWS リージョンとアベイラビリティーゾーンの詳細については、AWS グローバルインフラストラクチャを参照してください。

Amazon Lookout for Vision のインフラストラクチャセキュリティ

マネージドサービスである Amazon Lookout for Vision は、 AWS グローバルネットワークセキュリティで保護されています。 AWS セキュリティサービスと がインフラストラクチャ AWS を保護する方法については、AWS 「 クラウドセキュリティ」を参照してください。インフラストラクチャセキュリティのベストプラクティスを使用して AWS 環境を設計するには、「 Security Pillar AWS Well-Architected Framework」の「Infrastructure Protection」を参照してください。

が AWS 公開したAPI呼び出しを使用して、ネットワーク経由で Lookout for Vision にアクセスします。クライアントは以下をサポートする必要があります:

- Transport Layer Security (TLS) 。1TLS.2 が必要で、1.3 TLS をお勧めします。
- (Ephemeral Diffie-HellmanPFS) や DHE (Elliptic Curve Ephemeral Diffie-Hellman) などの完全前方秘匿性 ECDHE () を備えた暗号スイート。これらのモードは、Java 7 以降など、ほとんどの最新システムでサポートされています。

さらに、 リクエストは、 IAMプリンシパルに関連付けられたアクセスキー ID とシークレットアクセスキーを使用して署名する必要があります。または、AWS Security Token Service (AWS STS) を使用して、一時セキュリティ認証情報を生成し、リクエストに署名することもできます。

Amazon Lookout for Vision のモニタリング

モニタリングは、Amazon Lookout for Vision と他の AWS ソリューションの信頼性、可用性、パフォーマンスを管理する上で重要です。AWS には、Lookout for Vision を監視し、異常を検出した場合に報告し、必要に応じて自動的に対処するために、次のモニタリングツールが用意されています。

- Amazon CloudWatch は、AWS リソースと、AWS で実行するアプリケーションをリアルタイムでモニタリングします。メトリクスの収集と追跡、カスタマイズしたダッシュボードの作成、および指定したメトリクスが指定したしきい値に達したときに通知またはアクションを実行するアラームの設定を行うことができます。例えば、CloudWatch で Amazon EC2 インスタンスの CPU 使用率などのメトリクスを追跡し、必要に応じて新しいインスタンスを自動的に起動できます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。
- Amazon CloudWatch Logs では、Amazon EC2 インスタンス、CloudTrail、その他ソースから得たログファイルのモニタリング、保存、およびアクセスが可能です。CloudWatch Logs は、ログファイル内の情報をモニタリングし、特定のしきい値が満たされたときに通知します。高い耐久性を備えたストレージにログデータをアーカイブすることも可能です。詳細については、「Amazon CloudWatch Logs ユーザーガイド」を参照してください。
- Amazon EventBridge を使用して、AWS サービスを自動化し、アプリケーションの可用性問題やリソース変更といったシステムイベントに自動的に対応できます。AWS のサービスからのイベントは、ほぼリアルタイムに EventBridge に提供されます。簡単なルールを記述して、注目するイベントと、イベントがルールに一致した場合に自動的に実行するアクションを指定できます。詳細については、Amazon EventBridge ユーザーガイドを参照してください。
- AWS CloudTrail は、AWSアカウントにより、またはそのアカウントに代わって行われた API コールや関連イベントを取得し、指定した Amazon S3 バケットにログファイルを配信します。AWS を呼び出したユーザーとアカウント、呼び出し元の IP アドレス、および呼び出しの発生日時を特定できます。詳細については、AWS CloudTrailユーザーガイドを参照してください。

Amazon CloudWatch による Lookout for Vision のモニタリング

CloudWatch を使用して Lookout for Vision をモニタリングし、ほぼリアルタイムで、raw データの収集とその読み取り可能化を処理します。これらの統計は 15 か月間保持されるため、履歴情報にアクセスし、ウェブアプリケーションまたはサービスの動作をより的確に把握できます。また、特定のしきい値を監視するアラームを設定し、これらのしきい値に達したときに通知を送信したりアクションを実行したりできます。詳細については、「Amazon CloudWatch ユーザーガイド」を参照してください。

Lookout for Vision サービスは、AWS/LookoutVision 名前空間の以下のメトリクスをレポートします。

メトリクス	説明
DetectedAnomalyCount	プロジェクトで検出された異常の数 有効な統計:Sum,Average 単位: 個
ProcessedImageCount	異常検出を実行した画像の合計数 有効な統計:Sum,Average 単位: 個
InvalidImageCount	無効で結果を返せなかった画像の数 有効な統計:Sum,Average 単位: 個
SuccessfulRequestCount	成功した API 呼び出しの数 有効な統計:Sum,Average 単位: 個
ErrorCount	API エラーの数 有効な統計:Sum,Average 単位: 個
ThrottledCount	スロットリングが原因で発生した API エラーの数 有効な統計:Sum,Average 単位: 個

メトリクス	説明
Time	Lookout for Vision が異常検出をコンピューティングする時間 (ミリ秒) 有効な統計:Data Samples,Average 単位:Average 統計の場合はミリ秒、Data Samples 統計の場合はカウント値
MinInferenceUnits	StartModel リクエスト中に指定された推論ユニットの最小数 有効な統計: Average 単位: 個
MaxInferenceUnits	StartModel リクエスト中に指定された推論ユニットの最大数 有効な統計: Average 単位: 個
DesiredInferenceUnits	Lookout for Vision のスケールアップまたはスケールダウンの対象となる推論ユニットの数。 有効な統計: Average 単位: 個
InServiceInferenceUnits	モデルが使用している推論ユニットの数。 有効な統計: Average 平均統計を使用して、使用されたインスタンス数の 1 分間平均を取得することをお勧めします。 単位: 個

Lookout for Vision メトリクスでサポートされるディメンションは以下のとおりです。

ディメンション	説明
ProjectName	プロジェクトごとにメトリクスを分割して、どのプロジェクトに問題があるのか、更新が必要なのかを確認することができます。
ModelVersion	モデルのバージョンごとにメトリクスを分割し、どのモデルに問題があるか、更新が必要かを確認することができます。

AWS CloudTrail による Lookout for Vision API コールのロギング

Amazon Lookout for Vision は、ユーザー、ロール、または Lookout for Vision の AWS サービスによって実行されたアクションを記録するサービスである AWS CloudTrail と統合されています。CloudTrail は、Lookout for Vision のすべての API コールをイベントとしてキャプチャします。キャプチャされた呼び出しには、Lookout for Vision コンソールからの呼び出しと、Lookout for Vision API オペレーションへのコード呼び出しが含まれます。追跡を作成する場合は、Lookout for Vision のイベントを含む、Amazon S3 バケットへの CloudTrail イベントの継続的な配信を有効にすることができます。証跡を設定しない場合でも、CloudTrail コンソールの [イベント履歴] で最新のイベントを表示できます。CloudTrail で収集された情報を使用して、Lookout for Vision に対するリクエスト、リクエスト元の IP アドレス、リクエスト者、リクエスト日時などの詳細を確認できます。

CloudTrail の詳細については、「AWS CloudTrail ユーザーガイド」を参照してください。

CloudTrail での Lookout for Vision 情報

AWS アカウントを作成すると、そのアカウントに対して CloudTrail が有効になります。Lookout for Visionでアクティビティが発生すると、そのアクティビティはイベント履歴の他の AWS サービスイベントと一緒にCloudTrailのイベントに記録されます。AWS アカウントで最近のイベントを表示、検索、ダウンロードできます。詳細については、「CloudTrail イベント履歴でのイベントの表示」を参照してください。

Lookout for Vision のイベントを含む、AWS アカウントのイベントの継続的な記録については、追跡を作成します。追跡により、CloudTrail はログファイルを Amazon S3 バケットに配信できます。デフォルトでは、コンソールで作成した追跡がすべての AWS リージョンに適用されます。追跡は、AWS パーティションのすべてのリージョンからのイベントを記録し、指定した Simple Storage Service (Amazon S3) バケットにログファイルを配信します。さらに、CloudTrail ログで収

集したイベントデータをより詳細に分析し、それに基づく対応するためにその他の AWS サービスを設定できます。詳細については、次を参照してください。

- 追跡の作成のための概要
- CloudTrail のサポート対象サービスと統合
- CloudTrail 用の Amazon SNS 通知の構成
- 複数リージョンからの CloudTrail ログファイルの受け取り、および複数アカウントからの CloudTrail ログファイルの受け取り

Lookout for Vision の全てのアクションは CloudTrail によって記録され、Lookout for Vision API リファレンスドキュメントに記載されます。例えば、`CreateProject`、`DetectAnomalies`、`StartModel` といったアクションを呼び出すと、CloudTrail ログファイルにエントリが生成されます。

Amazon Lookout for Vision API コールをモニタリングすると、次の API コールが表示される場合があります。

- lookoutvision:StartTriallDetection
- lookoutvision:ListTriallDetection
- lookoutvision:DescribeTrialDetection

これらの特別なコールは、Amazon Lookout for Vision でトライアル検出に関連するさまざまなオペレーションをサポートするために使用されます。詳細については、「トライアル検出タスクでモデルを検証する」を参照してください。

各イベントまたはログエントリには、リクエストの生成者に関する情報が含まれます。同一性情報は次の判断に役立ちます。

- リクエストが、ルートと AWS Identity and Access Management ユーザー認証情報のどちらを使用して送信されたか。
- リクエストがロールまたはフェデレーションユーザーの一時的なセキュリティ認証情報を使用して行われたかどうか。
- リクエストが、別の AWS サービスによって送信されたかどうか。

詳細については、「CloudTrail userIdentity エレメント」を参照してください。

Lookout for Vision ログファイルのエントリについて

「トレイル」は、指定した Simple Storage Service (Amazon S3) バケットにイベントをログファイルとして配信するように設定できます。CloudTrail のログファイルには、単一か複数のログエントリがあります。イベントはあらゆるソースからの単一のリクエストを表し、リクエストされたアクション、アクションの日時、リクエストのパラメータなどの情報が含まれます。CloudTrail ログファイルは、パブリック API コールの順序付けられたスタックトレースではないため、特定の順序では表示されません。

次の例は、`CreateDataset` アクションを示す CloudTrail ログエントリです。


```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAYN4CJAYDEXAMPLE:user",
    "arn": "arn:aws:sts::123456789012:assumed-role/Admin/MyUser",
    "accountId": "123456789012",
    "accessKeyId": "ASIAYN4CJAYEXAMPLE",
    "sessionContext": {
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAYN4CJAYDCGEXAMPLE",
        "arn": "arn:aws:iam::123456789012:role/Admin",
        "accountId": "123456789012",
        "userName": "Admin"
      },
      "webIdFederationData": {},
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2020-11-20T13:15:09Z"
      }
    }
  },
  "eventTime": "2020-11-20T13:15:43Z",
  "eventSource": "lookoutvision.amazonaws.com",
  "eventName": "CreateDataset",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "128.0.0.1",
  "userAgent": "aws-cli/3",
  "requestParameters": {
```

```
    "projectName": "P1",
    "datasetType": "train",
    "datasetSource": {
      "groundTruthManifest": {
        "s3Object": {
          "bucket": "myuser-bucketname",
          "key": "training.manifest"
        }
      }
    },
    "clientToken": "EXAMPLE-0526-47dd-a5d3-2ca975820a34"
  },
  "responseElements": {
    "status": "CREATE_IN_PROGRESS"
  },
  "requestID": "EXAMPLE-15e1-4bc9-be38-cda2537c75bf",
  "eventID": "EXAMPLE-c5e7-43e0-8449-8d9b87e15acb",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "eventCategory": "Management",
  "recipientAccountId": "123456789012"
}
```

AWS CloudFormation による Amazon Lookout for Vision プロジェクトの作成

Amazon Lookout for Vision は AWS CloudFormation と統合されています。これは、リソースとインフラストラクチャの作成と管理の所要時間を短縮できるように AWS リソースをモデル化して設定するためのサービスです。必要なすべての AWS リソースを記述するテンプレートを作成すると、これらのリソースが AWS CloudFormation で自動的にプロビジョニングおよび設定されます。

AWS CloudFormation を使用して、Amazon Lookout for Vision をプロビジョニングおよび設定することができます。

AWS CloudFormation を使用すると、テンプレートを再利用してLookout for Vision プロジェクトをいつでも繰り返しセットアップできます。プロジェクトを一度記述するだけで、同じプロジェクトを複数の AWS アカウントとリージョンで何度でもプロビジョニングできます。

Lookout for Vision と AWS CloudFormation テンプレート

Lookout for Visionと関連サービスのためのプロジェクトをプロビジョニングし設定するためには、AWS CloudFormation テンプレートを理解する必要があります。テンプレートは、JSON またはYAMLでフォーマットされたテキストファイルです。これらのテンプレートには、AWS CloudFormation スタックにプロビジョニングしたいリソースを記述します。JSON や YAML に不慣れな方は、AWS CloudFormation Designer を使えば、AWS CloudFormation テンプレートを使いこなすことができます。詳細については、「AWS CloudFormation ユーザーガイド」の「AWS CloudFormation Designer とは」を参照してください。

Lookout for Vision プロジェクトのリファレンス (JSON テンプレートと YAML テンプレートの例を含む) については、「Lookout Vision リソースタイプリファレンス」を参照してください。

AWS CloudFormation の詳細はこちら

AWS CloudFormation の詳細については、以下のリソースを参照してください。

- AWS CloudFormation
- AWS CloudFormation ユーザーガイド
- AWS CloudFormation API リファレンス
- AWS CloudFormation コマンドラインインターフェイスユーザーガイド

インターフェイス VPC エンドポイント (AWS PrivateLink) を使って Amazon Lookout for Vision にアクセスする

AWS PrivateLink を使って、VPC と Amazon Lookout for Vision 間にプライベート接続を作成できます。インターネットゲートウェイ、NAT デバイス、VPN 接続、AWS Direct Connect 接続のいずれかを使用せずに、VPC 内にあるかのように Lookout for Vision にアクセスできます。VPC のインスタンスは、パブリック IP アドレスがなくても Lookout for Vision にアクセスできます。

このプライベート接続を確立するには、AWS PrivateLink を利用したインターフェイスエンドポイントを作成します。インターフェイスエンドポイントに対して有効にする各サブネットにエンドポイントネットワークインターフェイスを作成します。これらは、Lookout for Vision 宛てのトラフィックのエントリポイントとして機能するリクエスタ管理型ネットワークインターフェイスです。

詳細については、「AWS PrivateLink ガイド」の「AWS PrivateLink を通して AWS サービス にアクセスする」を参照してください。

Lookout for Vision VPC エンドポイントに関する考慮事項

Lookout for Vision のインターフェイスエンドポイントをセットアップする前に、「ガイド」の「AWS PrivateLink 考慮事項」を確認してください。

Lookout for Vision は、インターフェイスエンドポイントを介したその API アクションすべてへの呼び出しをサポートしています。

VPC エンドポイントポリシーは Lookout for Vision でサポートされています。デフォルトでは、エンドポイントを通じた Lookout for Vision へのフルアクセスが許可されています。または、セキュリティグループをエンドポイントのネットワークインターフェイスに関連付けて、インターフェイスエンドポイントを介して Lookout for Vision へのトラフィックを制御することもできます。

Lookout for Vision 用のインターフェイス VPC エンドポイントの作成

Amazon VPC コンソールまたは AWS Command Line Interface (AWS CLI) を使用して、Lookout for Vision 用のインターフェイスエンドポイントを作成できます。詳細については、「AWS PrivateLink ガイド」の「インターフェイスエンドポイントを作成する」を参照してください。

次のサービス名を使って Lookout for Vision 用インターフェースエンドポイントを作成します:

```
com.amazonaws.region.lookoutvision
```

インターフェースエンドポイント用プライベート DNS を有効にすると、デフォルトのリージョン DNS 名を使用して Lookout for Vision への API リクエストを行えます。例えば、lookoutvision.us-east-1.amazonaws.com のようにです。

Lookout for Vision 用の VPC エンドポイントポリシーの作成

エンドポイントポリシーは、インターフェイスエンドポイントにアタッチできる IAM リソースです。デフォルトのエンドポイントポリシーは、インターフェイスエンドポイント経由で Lookout for Vision へのフルアクセスを許可します。VPC から Lookout for Vision に許可されたアクセス権を管理するには、カスタムエンドポイントポリシーをインターフェイスエンドポイントにアタッチします。

エンドポイントポリシーは、以下の情報を指定します。

- アクションを実行できるプリンシパル (AWS アカウント、IAM ユーザー、IAM ロール)。
- 実行可能なアクション。
- このアクションを実行できるリソース。

詳細については、「AWS PrivateLink ガイド」の「Control access to services using endpoint policies (エンドポイントポリシーを使用してサービスへのアクセスをコントロールする)」を参照してください。

例: Lookout for Vision アクションの VPC エンドポイントポリシー

Lookout for Vision のカスタムエンドポイントポリシーの例を次に示します。このポリシーはインターフェイスエンドポイントにアタッチされると、VPC インターフェースエンドポイントにアクセスできるすべてのユーザーが、プロジェクト myProject に関連付けられた Lookout for Vision モデル myModel の DetectAnomalies API オペレーションの呼び出しを許可されるように指定します。

```
{
   "Statement":[
      {
         "Principal":"*",
         "Effect":"Allow",
         "Action":[
            "lookoutvision:DetectAnomalies"
```

```
         ],
         "Resource": "arn:aws:lookoutvision:us-west-2:123456789012:model/myProject/
myModel"
      }
   ]
}
```

Amazon Lookout for Vision でのクォータ

次の表では、Amazon Lookout for Vision 内の現在のクォータについて説明します。変更可能なクォータの詳細については、[AWS サービスクォータ] を参照してください。

モデルクォータ

次のクォータは、モデルのテスト、トレーニング、および機能に適用されます。

リソース	クォータ
サポートされるファイル形式	PNG および JPEG 画像形式
Amazon S3 バケット内の画像ファイルの最小画像寸法	64 ピクセル×64 ピクセル
Amazon S3 バケット内の画像ファイルの最大画像寸法	4096 ピクセル × 4096 ピクセルが最大です。小さいサイズは速くアップロードできます。
プロジェクトで使用する画像ファイルの画像寸法が異なる場合	データセット内の画像はすべて同じ寸法でなければなりません。
Amazon S3 バケット内の画像ファイルの最大ファイルサイズ	8 MB
ラベルの欠如	トレーニングの前に、画像に正常または異常のラベルを付ける必要があります。ラベルのない画像は、トレーニング中に無視されます。
トレーニングデータセットで正常とラベル付けされた画像の最小数	トレーニングデータセットとテストデータセットが別々のプロジェクトの場合は 10。シングルデータセットを持つプロジェクトの場合は 20。
トレーニングデータセット内の異常とラベル付けされた画像の最小数	トレーニングデータセットとテストデータセットが別々のプロジェクトの場合は 0。シングルデータセットを持つプロジェクトの場合は 10。

リソース	クォータ
分類トレーニングデータセット内の画像の最大数	16,000
分類テストデータセット内の画像の最大数。	4,000
テストデータセットで正常とラベル付けされた画像の最小数	10
テストデータセット内の異常とラベル付けされた画像の最小数	10
異常ローカライゼーショントレーニングデータセット内の画像の最大数	8000
異常ローカライゼーションテストデータセット内の画像の最大数。	800
トライアル検出データセット内のイメージの最大数	2,000
データセットマニフェストファイルの最大サイズ	1 GB
モデル内の最大トレーニングデータセット数	1
最大トレーニング時間	24 時間
最大テスト時間	24 時間
プロジェクト内の異常ラベルの最大数	100
マスク画像上の異常ラベルの最大数	20
異常ラベル用の画像の最小数。カウントするには、画像に 1 タイプの異常ラベルだけが含まれている必要があります。	シングルデータセットプロジェクトにおいては 20。トレーニングデータセットとテストデータセットが別々のプロジェクトでの各データセットにおいては 10。

Amazon Lookout for Vision のドキュメント履歴

次の表は、「Amazon Lookout for Vision 開発者ガイド」の各リリースにおける重要な変更点を説明したものです。このドキュメントの更新に関する通知については、RSS フィードでサブスクライブできます。

- 最新のドキュメント更新: 2023 年 2 月 20 日

変更	説明	日付
サンプル Lambda 関数を追加	AWS Lambda 関数で異常を検出する方法をサンプルで示します。詳細については、「AWS Lambda 関数による異常の検出」を参照してください。	2023 年 2 月 20 日
AWS WAF の IAM ガイダンスを更新しました	IAM ベストプラクティスに沿ってガイドを更新しました。詳細については、「IAM のセキュリティベストプラクティス」を参照してください。	2023 年 2 月 8 日
データセットのエクスポートサンプルを追加	`ListDatasetEntries` オペレーションを使用して Amazon Lookout for Vision プロジェクトからデータセットをエクスポートする方法を示す Python のサンプルを追加しました。詳細については、「プロジェクト (SDK) からのデータセットのエクスポート」を参照してください。	2022 年 12 月 2 日
「開始」トピックを更新	「開始」を更新し、サンプルデータセットを使って画像	2022 年 10 月 20 日

	セグメンテーションモデルの作成について説明しました。詳細については、「Amazon Lookout for Vision の開始」を参照してください。	
トラブルシューティングトピックを追加	モデルトレーニングのトラブルシューティングトピックを追加しました。モデルのトレーニングの詳細については、「モデルトレーニングのトラブルシューティング」を参照してください。	2022 年 10 月 17 日
Amazon SageMaker Ground Truth ジョブの使用に関するトピックを追加	自分で画像にラベリングする代わりに、Amazon SageMaker Ground Truth ジョブを使用して分類モデルや画像セグメンテーションモデル用の画像にラベリングすることができます。詳細については、「Amazon SageMaker Ground Truth ジョブの使用」を参照してください。	2022 年 8 月 19 日
Amazon Lookout for Vision で、異常ローカライゼーションを提供するようになりました。	画像上でさまざまな種類の異常 (傷、へこみ、裂け目など) が存在する箇所、異常のラベル、および異常のサイズを検出するセグメンテーションモデルを作成できます。詳細については、「トレーニング済み Amazon Lookout for Vision モデルの実行」を参照してください。	2022 年 8 月 16 日

Amazon Lookout for Vision では、エッジデバイスで CPU 推論を提供するようになりました。	Amazon Lookout for Vision モデルをデプロイして、GPU アクセラレータを必要とせずに、CPU のみにより Linux を実行している x86 コンピューティングプラットフォームで推論をローカル実行できるようになりました。詳細については、「CPU アクセラレーター」を参照してください。	2022 年 8 月 16 日
Amazon Lookout for Vision で推論ユニットを自動的にスケーリングできるようになりました。	需要の急増に対応するため、Amazon Lookout for Vision ではモデルが使用する推論ユニットの数をスケーリングできるようになりました。詳細については「トレーニング済み Amazon Lookout for Vision モデルの実行」を参照してください。	2022 年 8 月 16 日
Java サンプルを追加	Amazon Lookout for Vision 開発者ガイドに Java サンプルが含まれるようになりました。詳細については「AWS SDK の開始」を参照してください。	2022 年 5 月 2 日
エッジデバイスへのモデルデプロイの一般提供開始	AWS IoT Greengrass Version 2 が管理するエッジデバイスへのモデルデプロイが一般に利用可能になりました。詳細については「エッジデバイスでの Amazon Lookout for Vision モデルの使用」を参照してください。	2022 年 3 月 14 日

コンソールバケット情報を更新	コンソールバケットの内容とコンソールバケット作成の代替方法に関する情報を更新しました。詳細については、「ステップ 4: コンソールバケットを作成する」を参照してください。	2022 年 3 月 7 日
CSV ファイルからマニフェストファイルを作成	CSV ファイルから分類情報を読み取るスクリプトを使用して、マニフェストファイルの作成を簡略化できるようになりました。詳細については、「CSV ファイルからのマニフェストファイルの作成」を参照してください。	2022 年 2 月 10 日
エッジデバイスへのモデルデプロイをプレビューリリース	AWS IoT Greengrass Version 2 が管理するエッジデバイスへのモデルデプロイのプレビューリリースが利用可能になりました。詳細については、「エッジデバイスでの Amazon Lookout for Vision モデルの使用」を参照してください。	2021 年 12 月 7 日
新しい Python と Java 2 のサンプルが追加されました	`DetectAnomalies` で画像を解析するための Python と Java 2 のサンプルを追加しました。詳細については、「画像内の異常の検出」を参照してください。	2021 年 9 月 7 日

新規 AWS 管理ポリシーを追加しました。	Amazon Lookout for Vision は、AWS 管理ポリシーをサポートするようになりました。詳細については、「Amazon Lookout for Vision の AWS 管理ポリシー」を参照してください。	2021 年 5 月 11 日
推論ユニット情報を更新しました。	推論ユニットとその請求方法について説明した情報を追加しました。詳細については、「トレーニング済み Amazon Lookout for Vision モデルの実行」を参照してください。	2021 年 3 月 15 日
Amazon Lookout for Vision の一般利用が可能になりました。	Amazon Lookout for Vision が一般利用が可能になりました。Python のコードサンプルを更新し、モデルのトレーニングなどの非同期タスクを扱えるようにしました。	2021 年 2 月 17 日
タグ付けと AWS CloudFormation サポートを追加しました。	Amazon Lookout for Vision のモデルへのタグ付けと、AWS CloudFormation とのプロジェクト作成が可能になりました。詳細については、「モデルのタグ付け」と「AWS CloudFormation を使用した Amazon Lookout for Vision プロジェクトの作成」を参照してください。	2021 年 1 月 31 日
新しい特徴とガイド	本書は、「Amazon Lookout for Visionサービス Amazon Lookout for Vision 開発者ガイド」の初回リリースです。	2020 年 12 月 1 日

AWS 用語集

AWS の最新の用語については、「AWS の用語集リファレンス」の「AWS 用語集」を参照してください。

翻訳は機械翻訳により提供されています。提供された翻訳内容と英語版の間で齟齬、不一致または矛盾がある場合、英語版が優先します。