AWS DeepLens
Developer Guide
# Table of Contents

What Is AWS DeepLens? .......................................................................................................................... 1
Hardware .............................................................................................................................................. 1
Supported Frameworks ......................................................................................................................... 2
How AWS DeepLens Works ................................................................................................................. 2
For First-time Users .............................................................................................................................. 2
More Info ........................................................................................................................................... 3

Getting Started .................................................................................................................................. 4
Prerequisites ........................................................................................................................................ 4
Create an AWS Account ....................................................................................................................... 4
Create an IAM User ............................................................................................................................. 6
Understand Required IAM Roles ........................................................................................................ 6

Register Your Device ............................................................................................................................ 8
Connect to Your Device ......................................................................................................................... 13
Set Up Your Device ............................................................................................................................ 15
Verify Device Status ............................................................................................................................ 17
Deploy a Project .................................................................................................................................. 18
View Device Output ............................................................................................................................ 19

Working with Projects ......................................................................................................................... 21
Supported Frameworks ......................................................................................................................... 21
MXNet Models and Layers ..................................................................................................................... 21
TensorFlow Models and Layers ............................................................................................................ 23
Caffe Models and Layers ....................................................................................................................... 25

Viewing Project Output ....................................................................................................................... 28
View Output Streams on Your Device ............................................................................................... 28
Creating a Lambda Function for Viewing the Project Stream ............................................................ 30

Working with Sample Projects ........................................................................................................... 33
Sample Projects Overview .................................................................................................................. 33
Creating and Deploying a Sample Project ......................................................................................... 35
Extending Functionality ...................................................................................................................... 37
Editing an Existing Model with Amazon SageMaker ......................................................................... 41

Working with Custom Projects ........................................................................................................... 52
Importing Your Amazon SageMaker Trained Model .......................................................................... 52
Importing an Externally Trained Model ............................................................................................. 52
Optimizing a Custom Model ............................................................................................................... 54
Creating an Inference Lambda ........................................................................................................... 55

Managing Your Device ......................................................................................................................... 61
Securely Boot Your Device .................................................................................................................... 61
Update Your Device ............................................................................................................................. 61
Deregistering Your AWS DeepLens Device ....................................................................................... 62
Deleting AWS DeepLens Resources ................................................................................................. 62
Deleting Resources Using the Console ............................................................................................... 62

Logging and Troubleshooting ............................................................................................................ 64
Project Logs ......................................................................................................................................... 64
CloudWatch Logs Log Groups .............................................................................................................. 64
File System Logs .................................................................................................................................. 65
Troubleshooting Guide .......................................................................................................................... 66
Troubleshooting Software on Device .................................................................................................. 66
Troubleshooting Registration of Device ............................................................................................ 70
Troubleshooting Model Deployments to Device ................................................................................... 74

Device Library ...................................................................................................................................... 77
awscam Module ................................................................................................................................. 77
ggetLastFrame Function .................................................................................................................... 77
Model Object ..................................................................................................................................... 77
mo Module ......................................................................................................................................... 82
optimize Function ............................................................................................................ 82
Troubleshooting the Model Optimizer ................................................................................ 85
DeepLens_Kinesis_Video Module .................................................................................. 91
createProducer Function .............................................................................................. 92
Producer Object ................................................................................................................ 93
Stream Object .................................................................................................................. 93
Document History ........................................................................................................................ 96
AWS Glossary ...................................................................................................................... 98
What Is AWS DeepLens?

Welcome to the AWS DeepLens Developer Guide. AWS DeepLens is a wireless video camera and API. It shows you how to use the latest Artificial Intelligence (AI) tools and technology to develop computer vision applications. Through examples and tutorials, AWS DeepLens gives you hands-on experience using a physical camera to run real-time computer vision models.

The AWS DeepLens camera, or device, uses deep convolutional neural networks (CNNs) to analyze visual imagery. You use the device as a development environment to build computer vision applications.

AWS DeepLens works with the following AWS services:

- Amazon SageMaker, for model training and validation
- AWS Lambda, for running inference against CNN models
- AWS Greengrass, for deploying updates and functions to your device

Get started with AWS DeepLens by using any of the pretrained models that come with your device. As you become proficient, you can develop, train, and deploy your own models.

Topics

- AWS DeepLens Hardware (p. 1)
- Supported Frameworks (p. 2)
- How AWS DeepLens Works (p. 2)
- Are You an AWS DeepLens First-time User? (p. 2)
- More Info (p. 3)

AWS DeepLens Hardware

The AWS DeepLens camera includes the following:

- A 4-megapixel camera with MJPEG (Motion JPEG)
- 8 GB of on-board memory
- 16 GB of storage capacity
- A 32-GB SD (Secure Digital) card
- WiFi support for both 2.4 GHz and 5 GHz standard dual-band networking
- A micro HDMI display port
- Audio out and USB ports

The AWS DeepLens camera is powered by an Intel® Atom processor, which can process 100 billion floating-point operations per second (GFLOPS). This gives you all of the compute power that you need to perform inference on your device. The micro HDMI display port, audio out, and USB ports allow you to attach peripherals, so you can get creative with your computer vision applications.

You can use AWS DeepLens as soon as you register it. Begin by deploying a sample project, and use it as an example for developing your own computer vision applications.
Supported Frameworks

Currently, AWS DeepLens supports only the Apache MXNet framework and Gluon models. For more information, see Machine Learning Frameworks Supported by AWS DeepLens (p. 21).

How AWS DeepLens Works

The following diagram illustrates how AWS DeepLens works.

1. When turned on, the AWS DeepLens captures a video stream.
2. Your AWS DeepLens produces two output streams:
   - **Device stream**—The video stream passed through without processing
   - **Project stream**—The results of the model's processing video frames
3. The Inference Lambda function receives unprocessed video frames.
4. The Inference Lambda function passes the unprocessed frames to the project's deep learning model, where they are processed.
5. The Inference Lambda function receives the processed frames from the model and passes the processed frames on in the project stream.

For more information, see Viewing AWS DeepLens Output Streams (p. 28).

Are You an AWS DeepLens First-time User?

If you are a first-time AWS DeepLens user, we recommend that you do the following in order:

1. **Read** How AWS DeepLens Works (p. 2)—Explains AWS DeepLens and how to use it to develop computer vision applications.
2. **Explore Amazon SageMaker**—Explains some of the basic functionality of Amazon SageMaker. AWS DeepLens uses Amazon SageMaker to build and train CNN models. You use Amazon SageMaker to create your own AWS DeepLens models and projects.

3. **Learn about the AWS DeepLens Device Library (p. 77)**—The device library describes the classes and methods that you can use in your Lambda functions.

4. **Perform the tasks in Getting Started with AWS DeepLens (p. 4)**—Set up your AWS account, create the AWS Identity and Access Management (IAM) permissions and roles that you need to run AWS DeepLens, and register and set up your AWS DeepLens device.

After you've set up your AWS DeepLens environment and device, begin using it by trying these exercises:

a. **Creating and Deploying an AWS DeepLens Sample Project (p. 35)**—Walks you through creating sample AWS DeepLens project, which is included with your device.

b. **Editing an Existing Model with Amazon SageMaker (p. 41)**—Walks you through creating and training a model using Amazon SageMaker.

c. **Extending any Project's Functionality (p. 37)**—Walks you through taking output from your AWS DeepLens and using it to trigger an action.

---

**More Info**

- AWS DeepLens Forum
Getting Started with AWS DeepLens

Before using AWS DeepLens, register your device, connect your work computer to its Wi-Fi network, set it up, and verify that it’s connected. The following graphic shows where you perform each step.

After the device is successfully registered, its status become online. The next basic tasks involve deploying a project and viewing device output.

Topics
- Prerequisites (p. 4)
- Register Your AWS DeepLens Device (p. 8)
- Connect to Your AWS DeepLens Device’s Wi-Fi Network (p. 13)
- Set Up Your AWS DeepLens Device (p. 15)
- Verify Your AWS DeepLens Device Status (p. 17)
- Deploy a Project to Your AWS DeepLens Device (p. 18)
- View Your AWS DeepLens Device Output in a Browser (p. 19)

Prerequisites

Before you can begin using AWS DeepLens, you need an AWS account and an IAM user. In addition, you should also understand the required IAM roles and policies for performing AWS DeepLens-related operations.

Topics
- Create an AWS Account (p. 4)
- Create an IAM User (p. 6)
- Understanding Required IAM Roles (p. 6)

Create an AWS Account

To use AWS services, you need an AWS account. If you don't have one, create one now.

The AWS account is free. You pay only for the AWS services that you use.

To create an AWS account

2. Choose Create a Free Account.
3. Follow the instructions on the page.
Part of the sign-up process involves receiving a phone call and entering a PIN using the phone keypad.
Create an IAM User

You use an AWS Identity and Access Management (IAM) user to specify to whom the IAM policies and roles apply.

**Important**
Record your access key and secret key. You need them to make calls with the AWS CLI.

To create an IAM user

1. Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.
2. In the navigation pane, choose Users, then choose Add user.
3. For Access type, choose both Programmatic Access and AWS Management Console Access.
4. For Console password, choose Autogenerated password or Custom password. If you choose Custom password, type a password.
5. Choose whether to require the user to reset the password at the next sign-in, then choose Next: Permissions.
6. For Set permissions for <user name>, choose Attach existing policies directly, AdministrativeAccess, and Next: Review.
7. Review the settings. To return to the previous page to make changes, choose Previous. To create the user, choose Create user.

Now that you have an AWS account and IAM user, continue to Register Your AWS DeepLens Device (p. 8).

Understanding Required IAM Roles

Before you can build and run your AWS DeepLens-based computer vision application, you must give AWS DeepLens permissions to create the project for your application and deploy it to your device. Because AWS DeepLens uses Lambda functions to make inference calls and uses AWS Greengrass as the underlying infrastructure to connect your AWS DeepLens device to the AWS Cloud, AWS DeepLens also needs permissions to these services and others. To control access to AWS resources, you use AWS Identity and Access Management (IAM). With IAM, you control who is authenticated (signed in) and authorized (has permissions) to use resources with roles and permissions policies.

When you register your first device using the AWS DeepLens console, AWS DeepLens creates the following required IAM roles with predefined IAM policies:

- **AWSDeepLensServiceRole**—An IAM role that AWS DeepLens uses to access the AWS services that it uses, including AWS IoT, Amazon Simple Storage Service (Amazon S3), AWS Greengrass, and AWS Lambda. The AWSDeepLensServiceRolePolicy policy is attached to this role by default. You don't need to customize it.

- **AWSDeepLensLambdaRole**—An IAM role that is passed to AWS Lambda for creating Lambda functions and accessing other AWS services on your behalf. The AWSLambdaFullAccess policy is attached to this role by default. Customization of the default policy is not necessary.

- **AWSDeepLensGreengrassRole**—An IAM role that is passed to AWS Greengrass to allow AWS Greengrass to create needed AWS resources and to access other required AWS services. This role allows you to deploy Lambda inference functions to your AWS DeepLens device for on-device execution. The AWSGreengrassResourceAccessRolePolicy policy is attached to this role by default. You don't need to customize it.

- **AWSDeepLensGreengrassGroupRole**—An IAM role that is passed to AWS Greengrass device groups, which gives AWS DeepLens administrative Lambda functions to access other AWS services. The AWSDeepLensLambdaFunctionAccessPolicy policy is attached to this role by default. You might need to customize this role (and the attached policy) for individual devices.
Note
Depending on the specification for your AWS DeepLens, you might need to customize the default AWSDeepLensLambdaFunctionAccessPolicy policy. To customize the permissions in this policy, copy the AWSDeepLensLambdaFunctionPolicy policy to create a new policy document. Make necessary changes, attach the new policy document to this or another IAM role, and then reference the modified role when registering your AWS DeepLens device.

Create IAM Roles for Your AWS DeepLens Project

If your AWS account doesn't already have required IAM roles, use the AWS DeepLens console to create them using a single command button. If you already have these roles, AWS DeepLens uses them when you register your device. In either case, you have the option to customize device-dependent permissions. You can also create these roles before registering your device for the first time. To create these roles, you use the IAM console.

To create the required IAM roles in the IAM console

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. Create the IAM role for AWS DeepLens.
   
   From the list, choose AWSDeepLensServiceRole. If AWSDeepLensServiceRole isn't listed, choose Create role in IAM and follow these steps:
   
   a. Accept the DeepLens service and DeepLens use case by choosing Next: Permissions.
   b. Accept the AWSDeepLensServiceRolePolicy policy by choosing Next: Review.
   c. Accept the role name AWSDeepLensServiceRole and the provided description by choosing Create role. Don't change the role name.
   d. Close the IAM window.
3. Create the IAM role for AWS Lambda.
   
   From the list, choose AWSDeepLensLambdaRole. If AWSDeepLensLambdaRole isn't listed, choose Create role in IAM and follow these steps:
   
   a. Accept the Lambda service and the Lambda use case by choosing Next: Permissions.
   b. Accept the AWSLambdaFullAccess policy by choosing Next: Review.
   c. Accept the role name AWSDeepLensLambdaRole and the provided description by choosing Create role. Don't change the role name.
   d. Close the IAM window.
4. Create the IAM role for AWS Greengrass.
   
   From the list, choose AWSDeepLensGreengrassRole. If AWSDeepLensGreengrassRole isn't listed, choose Create role in IAM and follow these steps:
   
   a. Accept the Greengrass service and Greengrass use case by choosing Next: Permissions.
   b. Accept the AWSGreengrassResourceAccessRolePolicy policy by choosing Next: Review.
   c. Accept the role name AWSDeepLensGreengrassRole and the provided description by choosing Create role. Don't change the role name.
   d. Close the IAM window.
5. Create the IAM role for AWS Greengrass device groups.
   
   From the list, choose AWSDeepLensGreengrassGroupRole. If AWSDeepLensGreengrassGroupRole isn't listed, choose Create role in IAM and follow these steps:
Register Your Device

Register Your AWS DeepLens Device

Before you can use your AWS DeepLens device, you need to register the device with the AWS DeepLens service.

To create a device profile, you specify the name of the device and the AWS Identity and Access Management (IAM) roles and policies that give the device permissions to create and call required resources. The IAM roles and policies also allow your device to access other required AWS services, such as AWS Lambda and AWS Greengrass.

Before registering your device, complete the Prerequisites (p. 4).

To register your AWS DeepLens device and download the required certificate, use the AWS DeepLens console.

To register your AWS DeepLens device

2. Choose Register device.
3. On the Name device page, for Device name, type a name for your AWS DeepLens device, and choose Next.

   The device name can have up to 100 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen).


   c. Accept the role name AWSDeepLensGreengrassGroupRole and the provided description by choosing Create role. Don't change the role name.

   d. Close the IAM window.
4. On the Set permissions page, do the following:

   a. If this is your first time registering an AWS DeepLens device and you haven't created the required IAM roles (p. 6) for AWS DeepLens and the AWS services that AWS DeepLens uses in your account, for Permissions, choose Create roles. AWS DeepLens creates the required IAM roles for you.
b. If you want to customize the device-specific permissions needed to execute Lambda functions, override the default `AWSDeepLensGreengrassGroupRole` IAM role. Choose **Customize device configuration**, and then do one of the following:

i. If you've already created a customized IAM role of the `AWSDeepLensGreengrassGroupRole` type, choose it from the list.

ii. Otherwise, choose **Create a role in IAM** to open the IAM console. Follow the on-screen instructions to create an IAM role of the `DeepLens` type and attach a policy to the role, using the default `AWSDeepLensLambdaFunctionAccessPolicy` as a template.
5. Choose Next.

6. On the Download certificate page, choose Download certificate to save the device certificate in the default or selected directory. Then choose Continue.

7. Save File. Note where you save the certificate because you need it later.
Important

The downloaded device certificate is a .zip file. You attach the certificate's ZIP file to AWS DeepLens. Don't unzip the file.
Certificates aren't reusable. You generate a new certificate every time you register your device.

8. After downloading the certificate, choose Register. Your device is now registered.

After registering your AWS DeepLens device, connect to your device's Wi-Fi network (p. 13) to launch the device setup application that's included on your device.
Connect to Your AWS DeepLens Device's Wi-Fi Network

Before using your AWS DeepLens device, you need to connect it to the AWS Cloud, through either a wired or wireless network connection. To enable the connection, upload the certificate generated during device registration (p. 8) to the device. Your AWS DeepLens device includes a web application for uploading the certificate and finishing setting up the device.

To launch the setup application, connect your work computer to the device's Wi-Fi network through the device's softAP access point. SoftAP is software that enables a computer that hasn't been specifically made to be a router to function as a wireless access point.

**To connect to your AWS DeepLens device's Wi-Fi network**

1. Start your AWS DeepLens device by plugging the power cord into an outlet and the other end into the back of your device. Turn the device on by pressing the On/Off button on the front of the device.
2. Wait until the device's Wi-Fi indicator (the middle indicator on the front of the device) starts to flash. This indicates that the Wi-Fi network is ready.
3. Connect your computer to the device's Wi-Fi network. On your computer, choose the SSID for your device from the list of available networks and type the password for the device's network. The SSID and password are printed on the bottom of your device. The device's Wi-Fi network's SSID has the AMDC-nnnn format.

![Device Wi-Fi setup](image)

**Note**

To enter your device's Wi-Fi password on a computer running Windows, you may need to choose **Connecting using a security key instead** over **Enter the PIN from the router label** (usually 8 digits).
After successfully connecting to the device's Wi-Fi network, follow the next step to set up your device (p. 15).
Set Up Your AWS DeepLens Device

After connecting to your AWS DeepLens device's Wi-Fi network, you must set up the device to complete the device registration. To set up your device, you:

1. Configure device connection to the AWS Cloud through your home/office Wi-Fi network or wired Ethernet
2. Upload the device certificate that you downloaded in Register Your AWS DeepLens Device (p. 8)” to the device. Your device will present this device certificate to the AWS Cloud to access the AWS DeepLens service and other AWS services.
3. Download the streaming certificate from your device. You must upload this streaming certificate to each supported web browser in order to view device output using the web browser.
4. Create a password for you and an administrator. You use this password to log in to the device to manage updates and troubleshoot any device issues.
5. Enable or disable SSH connection. Enabling SSH lets you log into the device without using the AWS DeepLens console.
6. Enable automatic software updates to keep your device's software up-to-date.

The set up procedure uses the web application that you enabled when you connected your computer to your device's Wi-Fi network. If you haven't connected your work computer to your device's Wi-Fi network (AMDC–NNNN), do that now.

To set up your device

1. On the AWS DeepLens device console, on the Setup guide page, choose complete the setup to open the device setup application.
Alternatively, you can follow the steps below to open the device setup application manually:

a. On the computer connected to the device's (AMDC-NNNN) Wi-Fi network, open a browser window.

b. If your AWS DeepLens device has awscam software version 1.3.6 or later installed, in the browser, navigate to http://deeplens.config to launch the device setup app. Otherwise, navigate to http://192.168.0.1. For more information, see Device Setup URL (p. 73).

2. On the **Device setup** page, complete the following steps.

   a. Connect to the network.
   
   Type the your home/office Wi-Fi network name (i.e., SSID) and the password, then choose **Next**. If you're using Ethernet to connect to AWS DeepLens, choose the Ethernet option.

   b. Upload the device certificate.

   Locate and choose the certificate that you downloaded from the AWS DeepLens console when registering the device (p. 8), then choose **Upload certificate**.

   The downloaded device certificate is saved as a .zip file. Upload the unzipped device certificate file.

   c. Download the streaming certificate

   d. Configure device access by completing the following steps.
i. For **Create a password for the device**, type a password. You need this password to log in to your device to maintain and update your AWS DeepLens device. You can log in remotely using SSH, if enabled, or locally using a hardwired monitor, a USB mouse and/or a USB keyboard.

ii. For **SSH server**, we recommend that you choose **Disable**. SSH allows you to log in without using the AWS DeepLens console.

iii. For **Automatic updates**, we recommend that you choose **Enable**. Automatic updates keeps the most recent software deployed to your device. You will need to restart the device to have the updates installed.

e. Review the settings and choose **Finish** to finish setting up the device and to terminate your connection to the device's Wi-Fi network.

To change settings, choose **Edit** for the setting that you want to change, and then choose **Finish**.

This completes the device registration. To verify the device status, go to the next (p. 17) step.

### Verify Your AWS DeepLens Device Status

After you finish setting up your device, your computer automatically disconnects from your device's Wi-Fi network and reconnects to the internet through your home or office network. This can take a few seconds.

When your device is set up successfully, you see the following message:
The above message indicates that your registered device is now connected to and authenticated by the AWS Cloud. You are now ready to deploy an AWS DeepLens project. To do this, follow the instructions given in the section called “Deploy a Project” (p. 18). For more information, see Working with Projects (p. 21).

If your device is not online, verify that the device is connected to the internet and the device certificate is the one you downloaded when the device is registered. Then, return to Connect to Your AWS DeepLens Device's Wi-Fi Network (p. 13) and repeat the steps for setting up your device and connecting it to the network.

**Deploy a Project to Your AWS DeepLens Device**

In this walkthrough, you deploy the Object Detection project. The Object Detection project analyzes images within a video stream on your AWS DeepLens device to identify objects. It can recognize as many as 20 types of objects. The steps to deploying any AWS DeepLens project are the same as we use here to deploy the Object Detection project.

Your web browser is the interface between you and your AWS DeepLens device. You perform all of the following activities on your browser after logging on to AWS:

1. On the Projects screen, choose the radio button to the left of your project name, then choose Deploy to device.
2. On the **Target device** screen, from the list of AWS DeepLens devices, choose the radio button to the left of the device that you want to deploy this project to. An AWS DeepLens device can have only one project deployed to it at a time.

3. Choose **Review**.

   If a project is already deployed to the device, you will see an error message that deploying this project will overwrite the project that is already running on the device. Choose **Continue project**.

   This will take you to the **Review and deploy** screen.

4. On the **Review and deploy** screen, review your project and choose either **Previous** to go back and make changes, or **Deploy** to deploy the project.

   **Important**
   
   Deploying a project incurs costs for the AWS services that are used to run the project.

   After the project deployment completes, follow the instructions in **Viewing AWS DeepLens Output Streams (p. 28)** to view your project's output.

---

**View Your AWS DeepLens Device Output in a Browser**

**Note**

To view the output streams in a supported browser, your device must have the **awscam** software version 1.3.9 or higher installed. For information to update the device software, see the section called "Update Your Device" (p. 61).

**To view unprocessed device streams or processed project streams in a supported web browser**

1. Download the streaming certificate.
   a. **Connect to your device's ADMC-NNNN Wi-Fi network (p. 13).**
   b. **Connect to the device's web server at http://deeplens.config.**
   c. **Download the streaming certificate for viewing device output streams, following the on-screen instructions.**

2. **Import into your supported web browser the streaming certificate you downloaded during the device registration (p. 15).**
   a. **For FireFox (Windows and Mac Sierra or higher), follow these steps:**
      i. Choose **Preferences** in FireFox.
      ii. Choose **Privacy & Security**.
      iii. Choose **View certificate**.
      iv. Choose the **Your Certificates** tab.
      v. Choose **Import**.
      vi. Choose the downloaded streaming certificate to load into FireFox.
      vii. When prompted to enter your computer's system password, leave the password field blank and follow the on-screen instruction to finish importing the streaming certificate.
   b. **For Chrome (Mac Sierra or higher), follow these steps:**
      i. On your Mac, double-click the downloaded streaming certificate to add it to **System** under **Keychains** and **My Certificate** under **Category**.
Alternatively, open and unlock the Keychain Access app, choose System under Keychains on the left and My Certificate under Category. Drag and drop the downloaded streaming certificate into the window.

ii. Enter your computer's system password.

iii. On the next screen leave the Password field blank and then choose OK.

c. For Chrome (Windows and Linux), follow these steps:

i. In your Chrome browser, open Settings and choose Advanced settings.

ii. In Privacy and security, choose Manage certificates.

iii. Choose Import.

iv. Navigate to the folder containing the downloaded streaming certificate and choose it to load into Chrome.

v. When prompted for your computer's system password, leave the password field blank and follow the on-screen instructions to finish importing the streaming certificate.

3. View output streams:


b. In the navigation pane, choose Devices.

c. For Devices, choose your AWS DeepLens device.

d. From the device details page, choose View output.

e. On the View output page, choose View stream.

This will open your AWS DeepLens project's output viewer in a separate browser tab.

Alternatively, you can open the output viewer by navigating to https://<your-device-ip-address>:4000. You can find your device's IP address on the device details page in the AWS DeepLens console.

When prompted, choose OK to confirm the selection of the imported streaming certificate, choose to add security exception to the newly opened output stream viewer URL, and follow the on-screen instructions to finish opening the stream viewer.

f. Choose Live stream to view the unprocessed device stream and choose Project stream to view the processed project stream.
Working with AWS DeepLens Projects

When your AWS DeepLens device is registered and connected, you can begin using it. To use your device for deep learning, you create a project and deploy it to your AWS DeepLens device. An AWS DeepLens project is made up of deep learning models and associated AWS Lambda functions.

AWS DeepLens comes with several sample projects that you can deploy and use right out of the box.

Topics
- Machine Learning Frameworks Supported by AWS DeepLens (p. 21)
- Viewing AWS DeepLens Output Streams (p. 28)
- Working with AWS DeepLens Sample Projects (p. 33)
- Working with AWS DeepLens Custom Projects (p. 52)

Machine Learning Frameworks Supported by AWS DeepLens

AWS DeepLens supports deep learning models trained using the Apache MXNet (including support for Gluon API), TensorFlow, and Caffe frameworks. This section lists the models and modeling layers that AWS DeepLens supports for each framework.

Topics
- Supported Apache MXNet Models and Supporting MXNet Layers (p. 21)
- Supported TensorFlow Models and Supporting TensorFlow Layers (p. 23)
- Supported Caffe Models and Supporting Caffe Layers (p. 25)

Supported Apache MXNet Models and Supporting MXNet Layers

AWS DeepLens supports Apache MXNet deep learning models that are exposed by the Gluon API and MXNet layers. This section lists the supported models and layers.

Topics
- Supported MXNet Models Exposed by the Gluon API (p. 21)
- Supporting MXNet Layers (p. 22)

Supported MXNet Models Exposed by the Gluon API

AWS DeepLens supports the following Apache MXNet deep learning models from the Gluon model zoo that are exposed by the Gluon API.
**AWS DeepLens Developer Guide**

**MXNet Models and Layers**

<table>
<thead>
<tr>
<th>Supported Gluon Models</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AlexNet</strong></td>
</tr>
<tr>
<td>Image classification model trained on the ImageNet dataset imported from the Open Neural Network Exchange (ONNX).</td>
</tr>
<tr>
<td><strong>MobileNet</strong></td>
</tr>
<tr>
<td>Image classification model trained in TensorFlow using the RMSprop optimizer.</td>
</tr>
<tr>
<td><strong>ResNet</strong></td>
</tr>
<tr>
<td>Image classification model trained on the ImageNet dataset imported from MXNet.</td>
</tr>
<tr>
<td><strong>SqueezeNet</strong></td>
</tr>
<tr>
<td>Image classification model trained on the ImageNet dataset imported from ONNX.</td>
</tr>
<tr>
<td><strong>VGG</strong></td>
</tr>
<tr>
<td>Image classification model trained on the ImageNet dataset imported from MXNet or ONNX.</td>
</tr>
</tbody>
</table>

**Example**

The following example shows how to export a SqueezeNet version 1 model using the Gluon API. The output is a symbol and parameters file. The filename has the ‘squeezenet’ prefix.

```python
import mxnet as mx
from mxnet.gluon.model_zoo import vision
squeezenet = vision.squeezenet_v1(pretrained=True, ctx=mx.cpu())

# To export, you need to hybridize your gluon model,
squeezenet.hybridize()

# SqueezeNet’s input pattern is 224 pixel X 224 pixel images. Prepare a fake image,
fake_image = mx.nd.random.uniform(shape=(1,3,224,224), ctx=mx.cpu())

# Run the model once.
result = squeezenet(fake_image)

# Now you can export the model. You can use a path if you want ‘models/squeezenet’.
squeezenet.export('squeezenet')
```

For a complete list of models and more information, see the Gluon Model Zoo.

**Supporting MXNet Layers**

You can use the following Apache MXNet modeling layers to train deep learning model for AWS DeepLens.

<table>
<thead>
<tr>
<th>Supported MXNet Modeling Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Activation</strong></td>
</tr>
<tr>
<td>Applies an activation function to the input</td>
</tr>
<tr>
<td><strong>BatchNorm</strong></td>
</tr>
<tr>
<td>Applies batch normalization</td>
</tr>
<tr>
<td><strong>Concat</strong></td>
</tr>
<tr>
<td>Joins input arrays along a given axis</td>
</tr>
<tr>
<td><strong>_contrib_MultiBoxDetection</strong></td>
</tr>
<tr>
<td>Converts a multibox detection prediction</td>
</tr>
<tr>
<td><strong>_contrib_MultiBoxPrior</strong></td>
</tr>
<tr>
<td>Generates prior boxes from data, sizes, and ratios</td>
</tr>
<tr>
<td><strong>Convolution</strong></td>
</tr>
<tr>
<td>Applies a convolution layer on input</td>
</tr>
</tbody>
</table>
### Supported MXNet Modeling Layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deconvolution</td>
<td>Applies a transposed convolution on input</td>
</tr>
<tr>
<td>elemwise_add</td>
<td>Applies element-wise addition of arguments</td>
</tr>
<tr>
<td>Flatten</td>
<td>Collapses the higher dimensions of an input into an 2-dimensional array</td>
</tr>
<tr>
<td>FullyConnected</td>
<td>Applies a linear transformation of ( Y = WX + b ) on input ( X )</td>
</tr>
<tr>
<td>InputLayer</td>
<td>Specifies the input to a neural network</td>
</tr>
<tr>
<td>L2Norm</td>
<td>Applies L2 normalization to the input array</td>
</tr>
<tr>
<td>LRN (Local Response Normalization)</td>
<td>Applies local response normalization to the input array</td>
</tr>
<tr>
<td>Pooling</td>
<td>Performs pooling on the input</td>
</tr>
<tr>
<td>Reshape</td>
<td>Reshapes the input array with a different view without changing the data</td>
</tr>
<tr>
<td>ScaleShift</td>
<td>Applies scale and shift operations on input elements</td>
</tr>
<tr>
<td>SoftmaxActivation</td>
<td>Applies Softmax activation to the input</td>
</tr>
<tr>
<td>SoftmaxOutput</td>
<td>Computes the gradient of cross-entropy loss with respect to Softmax output</td>
</tr>
<tr>
<td>transpose</td>
<td>Permutes the dimensions of an array</td>
</tr>
<tr>
<td>UpSampling</td>
<td>Performs nearest-neighbor or bilinear upsampling to input</td>
</tr>
<tr>
<td>_mul</td>
<td>Performs multiplication</td>
</tr>
<tr>
<td>_Plus</td>
<td>Performs an element-wise sum of the input arrays with broadcasting</td>
</tr>
</tbody>
</table>

For more information about MXNet layers, see [MXNet Gluon Neural Network Layers](#).

### Supported TensorFlow Models and Supporting TensorFlow Layers

AWS DeepLens supports the following TensorFlow models and layers for deep learning.

**Topics**

- Supported TensorFlow Models (p. 23)
- Supporting TensorFlow Layers (p. 24)

### Supported TensorFlow Models

AWS DeepLens supports the following deep learning models that have been trained with TensorFlow.
### Supported TensorFlow Models

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inception</td>
<td>An image classification model trained on the ImageNet dataset using TensorFlow</td>
</tr>
<tr>
<td>MobileNet</td>
<td>An image classification model trained on the ImageNet dataset using TensorFlow</td>
</tr>
<tr>
<td>NasNet</td>
<td>An image classification model trained on the ImageNet dataset using TensorFlow</td>
</tr>
<tr>
<td>VGG</td>
<td>An image classification model trained on the ImageNet dataset using TensorFlow</td>
</tr>
</tbody>
</table>

For more information about TensorFlow models, see tensorflow/models on GitHub.

### Supporting TensorFlow Layers

You can use the following TensorFlow layers to train deep learning models that are supported by AWS DeepLens.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add</td>
<td>Computes element-wise addition</td>
</tr>
<tr>
<td>AvgPool</td>
<td>Performs average pooling on the input</td>
</tr>
<tr>
<td>BatchToSpaceND</td>
<td>Rearranges data from batch into blocks of spatial data</td>
</tr>
<tr>
<td>BiasAdd</td>
<td>Adds bias</td>
</tr>
<tr>
<td>Const</td>
<td>Creates a constant tensor</td>
</tr>
<tr>
<td>Conv2D</td>
<td>Computes a 2-D convolution</td>
</tr>
<tr>
<td>Conv2DBackpropInput</td>
<td>Computes the gradients of convolution with respect to the input</td>
</tr>
<tr>
<td>Identity</td>
<td>Returns a tensor with the same shape and contents as input</td>
</tr>
<tr>
<td>Maximum</td>
<td>Computes element-wise maximization.</td>
</tr>
<tr>
<td>MaxPool</td>
<td>Performs the max pooling on the input</td>
</tr>
<tr>
<td>Mean</td>
<td>Computes the mean of elements across dimensions of a tensor</td>
</tr>
<tr>
<td>Mul</td>
<td>Computes element-wise multiplication</td>
</tr>
<tr>
<td>Neg</td>
<td>Computes numerical negative value element-wise</td>
</tr>
<tr>
<td>Pad</td>
<td>Pads a tensor</td>
</tr>
<tr>
<td>Placeholder</td>
<td>Inserts a placeholder for a tensor that will be always fed</td>
</tr>
<tr>
<td>Prod</td>
<td>Computes the product of elements across dimensions of a tensor</td>
</tr>
<tr>
<td>RandomUniform</td>
<td>Outputs random values from a uniform distribution</td>
</tr>
</tbody>
</table>
## Supported TensorFlow Layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>Creates a sequence of numbers</td>
</tr>
<tr>
<td>Relu</td>
<td>Computes rectified linear activations</td>
</tr>
<tr>
<td>Reshape</td>
<td>Reshapes a tensor</td>
</tr>
<tr>
<td>Rsqrt</td>
<td>Computes reciprocal of square root</td>
</tr>
<tr>
<td>Shape</td>
<td>Returns the shape of a tensor</td>
</tr>
<tr>
<td>Softmax</td>
<td>Computes Softmax activations</td>
</tr>
<tr>
<td>SpaceToBatchND</td>
<td>Zero-pads and then rearranges blocks of spatial data into batch</td>
</tr>
<tr>
<td>Square</td>
<td>Computes element-wise square</td>
</tr>
<tr>
<td>Squeeze</td>
<td>Removes dimensions of size 1 from the shape of a tensor</td>
</tr>
<tr>
<td>StopGradient</td>
<td>Stops gradient computation</td>
</tr>
<tr>
<td>Sub</td>
<td>Computes element-wise subtraction</td>
</tr>
<tr>
<td>Sum</td>
<td>Computes the sum of elements across dimensions of a tensor</td>
</tr>
<tr>
<td>Tile</td>
<td>Constructs a tensor by tiling a given tensor</td>
</tr>
</tbody>
</table>

For more information about TensorFlow layers, see [TensorFlow Layers](#).

## Supported Caffe Models and Supporting Caffe Layers

AWS DeepLens supports the following deep learning models trained with Caffe and Caffe modeling layers.

**Topics**
- Supported Caffe Models (p. 25)
- Supporting Caffe Layers (p. 26)

### Supported Caffe Models

AWS DeepLens supports the following deep learning models trained with Caffe.

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlexNet</td>
<td>An image classification model trained on the ImageNet dataset using Caffe</td>
</tr>
<tr>
<td>DenseNet</td>
<td>An image classification model from the original Torch model</td>
</tr>
<tr>
<td>Inception</td>
<td>An image classification model converted from the original Apache MXNet model</td>
</tr>
<tr>
<td>MobileNet</td>
<td>An image classification model trained on the ImageNet dataset using Caffe</td>
</tr>
</tbody>
</table>
### Supported Caffe Models

<table>
<thead>
<tr>
<th>Model</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet</td>
<td>An image classification model trained on the ImageNet dataset using Caffe</td>
</tr>
<tr>
<td>SqueezeNet</td>
<td>An image classification model trained on the ImageNet dataset using Caffe</td>
</tr>
<tr>
<td>VGG</td>
<td>An image classification model trained on the ImageNet dataset using Caffe</td>
</tr>
</tbody>
</table>

For more information about Caffe models, see Caffe Model Zoo.

### Supporting Caffe Layers

You can use the following Caffe layers to train deep learning models supported by AWS DeepLens.

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BatchNorm</td>
<td>Normalizes the input to have 0-mean and/or unit variance across the batch</td>
</tr>
<tr>
<td>Concat</td>
<td>Concatenates input blobs</td>
</tr>
<tr>
<td>Convolution</td>
<td>Convolves the input with a bank of learned filters</td>
</tr>
<tr>
<td>Deconvolution</td>
<td>Performs in the opposite sensor of the Convolution layer</td>
</tr>
<tr>
<td>Dropout</td>
<td>Performs dropout</td>
</tr>
<tr>
<td>Eltwise</td>
<td>Performs element-wise operations, such as product and sum, along multiple input blobs</td>
</tr>
<tr>
<td>Flatten</td>
<td>Reshapes the input blob into flat vectors</td>
</tr>
<tr>
<td>InnerProduct</td>
<td>Computes an inner product</td>
</tr>
<tr>
<td>Input</td>
<td>Provides input data to the model</td>
</tr>
<tr>
<td>LRN (Local Response Normalization)</td>
<td>Normalizes the input in a local region across or within feature maps</td>
</tr>
<tr>
<td>Permute</td>
<td>Permutes the dimensions of a blob</td>
</tr>
<tr>
<td>Pooling</td>
<td>Pools the input image by taking the max, average, etc.,. within regions</td>
</tr>
<tr>
<td>Power</td>
<td>Computes the output as $(shift + scale * x)^{power}$ for each input element $x$</td>
</tr>
<tr>
<td>ReLU</td>
<td>Computes rectified linear activations</td>
</tr>
<tr>
<td>Reshape</td>
<td>Changes the dimensions of the input blob, without changing its data</td>
</tr>
<tr>
<td>ROIPooling</td>
<td>Applies pooling for each region of interest</td>
</tr>
<tr>
<td>Scale</td>
<td>Computes the element-wise product of two input blobs</td>
</tr>
</tbody>
</table>
### Supported Caffe Layers

<table>
<thead>
<tr>
<th>Layer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slice</td>
<td>Slices an input layer to multiple output layers along a given dimension</td>
</tr>
<tr>
<td>Softwax</td>
<td>Computes the Softmax activations</td>
</tr>
<tr>
<td>Tile</td>
<td>Copies a blob along specified dimensions</td>
</tr>
</tbody>
</table>

For more information about Caffe layers, see [Caffe Layers](#).
Viewing AWS DeepLens Output Streams

An AWS DeepLens device produces two output streams: the device stream and a project stream. The device stream is an unprocessed video stream. The project stream is the result of the processing that the model performs on the video frames.

You can view the output in a supported web browser when your AWS DeepLens device is online. For more information, see the section called “View Device Output” (p. 19). In addition, you can also view the output on the device that is connected to a monitor, a keyboard and a mouse. The instructions are given in this section.

**Topics**

- Viewing Output Streams on Your AWS DeepLens Device (p. 28)
- Creating a Lambda Function for Viewing the Project Stream (p. 30)

Viewing Output Streams on Your AWS DeepLens Device

In addition to viewing your AWS DeepLens output streams in a browser (p. 19), you can use `mplayer` to view the streams directly from your AWS DeepLens device after connecting it to a monitor, a keyboard, and a mouse. This is especially useful when your AWS DeepLens device is not online.

Instead of connecting to the device directly using a monitor, a keyboard, and a mouse, you can also use `ssh` to connect to the device, if SSH access is enabled on the device when it is registered. You can then use `mplayer` on your work computer to view the streams. Make sure that `mplayer` is installed on your computer before viewing the output streams from your AWS DeepLens device. For more information about the installation, see `mplayer` download.

**Topics**

- Viewing a Device Stream on Your AWS DeepLens Device (p. 28)
- Viewing a Project Stream on Your AWS DeepLens Device (p. 29)

Viewing a Device Stream on Your AWS DeepLens Device

To view an unprocessed device stream on your AWS DeepLens device

1. Plug your AWS DeepLens device into a power outlet and turn it on.
2. Connect a USB mouse and keyboard to your AWS DeepLens.
3. Use the micro HDMI cable to connect your AWS DeepLens to a monitor. A login screen appears on the monitor.
4. Sign in to the device using the SSH password that you set when you registered the device.
5. To see the video stream from your AWS DeepLens, start your terminal and run the following command:

   ```bash
   mplayer -demuxer lavf /opt/awsyum/out/ch1_out.h264
   ```

6. To stop viewing the video stream and end your terminal session, press Ctrl+C.
Viewing a Project Stream on Your AWS DeepLens Device

To view a project stream on your AWS DeepLens device

1. Plug your AWS DeepLens device to a power outlet and turn it on.
2. Connect a USB mouse and keyboard to your AWS DeepLens.
3. Use the micro HDMI cable to connect your AWS DeepLens to a monitor. A login screen appears on the monitor.
4. Sign in to the device using the SSH password you set when you registered the device.
5. To see the video stream from your AWS DeepLens, start your terminal and run the following command:

   ```bash
   mplayer -demuxer lavf -lavfdopts format=mjpeg:probesize=32 /tmp/results.mjpeg
   ```

6. To stop viewing the video stream and end your terminal session, press Ctrl+C.
Creating a Lambda Function for Viewing the Project Stream

To view the project stream, you need an AWS Lambda function that interacts with the mjpeg stream on your device and the deep learning model. For the sample projects included with AWS DeepLens, the code is included in the inference Lambda function for the project. For your custom projects, you need to create a Lambda function that performs this task.

Create a Lambda function for your custom projects

Add the following sample code to your projects and change the model name and the dimensions as appropriate.

```python
# Copyright Amazon AWS DeepLens, 2017

import os
import greengrasssdk
from threading import Timer
import time
import awscam
import cv2
from threading import Thread

# Create an AWS Greengrass core SDK client.
client = greengrasssdk.client('iot-data')

# The information exchanged between AWS IoT and the AWS Cloud has
# a topic and a message body.
# This is the topic that this code uses to send messages to the Cloud.
iotTopic = '${aws/things/{}//infer}'.format(os.environ['AWS_IOT_THING_NAME'])

ret, frame = awscam.getLastFrame()
ret,jpeg = cv2.imencode('.jpg', frame)
Write_To_FIFO = True

class FIFO_Thread(Thread):
    def __init__(self):
        ''' Constructor. '''
        Thread.__init__(self)

    def run(self):
        fifo_path = '/tmp/results.mjpeg'
        if not os.path.exists(fifo_path):
            os.mkfifo(fifo_path)
        f = open(fifo_path,'w')
        client.publish(topic=iotTopic, payload="Opened Pipe")
        while Write_To_FIFO:
            try:
                f.write(jpeg.tobytes())
            except IOError as e:
                continue

def greengrass_infinite_infer_run():
    try:
        modelPath = '/opt/awscam/artifacts/mxnet_deploy_ssd_resnet50_300_FP16_FUSED.xml'
        modelType = 'ssd'
        input_width = 300
        input_height = 300
        max_threshold = 0.25
```
```
AWS DeepLens Developer Guide
Creating a Lambda Function for Viewing the Project Stream

        15 : 'person', 16 : 'pottedplant', 17 : 'sheep',
        18 : 'sofa', 19 : 'train', 20 : 'tvmonitor'}

results_thread = FIFO_Thread()
results_thread.start()

    # Send a starting message to the AWS IoT console.
    client.publish(topic=iotTopic, payload="Object detection starts now")

    # Load the model to the GPU (use {"GPU": 0} for CPU).
    mcfg = {"GPU": 1}
    model = awscam.Model(modelPath, mcfg)
    client.publish(topic=iotTopic, payload="Model loaded")
    ret, frame = awscam.getLastFrame()
    if ret == False:
        raise Exception("Failed to get frame from the stream")

    yscale = float(frame.shape[0]/input_height)
xscale = float(frame.shape[1]/input_width)

doInfer = True
while doInfer:
    # Get a frame from the video stream.
    ret, frame = awscam.getLastFrame()

    # If you fail to get a frame, raise an exception.
    if ret == False:
        raise Exception("Failed to get frame from the stream")

    # Resize the frame to meet the model input requirement.
    frameResize = cv2.resize(frame, (input_width, input_height))

    # Run model inference on the resized frame.
    inferOutput = model.doInference(frameResize)

    # Output the result of inference to the fifo file so it can be viewed with
    mplayer.
    parsed_results = model.parseResult(modelType, inferOutput)['ssd']
    label = '{
        for obj in parsed_results:
            if obj['prob'] > max_threshold:
                xmin = int( xscale * obj['xmin'] ) + int(obj['xmin'] - input_width/2)
            + input_width/2)
            ymin = int( yscale * obj['ymin'] )
            xmax = int( xscale * obj['xmax'] ) + int(obj['xmax'] - input_width/2)
            + input_width/2)
            ymax = int( yscale * obj['ymax'] )
            cv2.rectangle(frame, (xmin, ymin), (xmax, ymax), (255, 165, 20), 4)
            label += "{}": {:.2f},
            label_show = "{}:    {:.2f}%".format(outMap[obj['label']],
            label += "null": 0.0
            label += '}'
            client.publish(topic=iotTopic, payload = label)
            global jpeg
            ret, jpeg = cv2.imencode('.jpg', frame)

    except Exception as e:
        msg = "Test failed: " + str(e)
        client.publish(topic=iotTopic, payload=msg)

    # Asynchronously schedule this function to be run again in 15 seconds.
    Timer(15, greengrass_infinite_infer_run).start()
# Execute the function.
greengrass_infinite_infer_run()

# This is a dummy handler and will not be invoked.
# Instead, the code is executed in an infinite loop for our example.
def function_handler(event, context):
    return

After you've created and deployed the Lambda function, follow the instructions in the section called “View Device Output” (p. 19) or Viewing a Project Stream on Your AWS DeepLens Device (p. 29) to view the processed project stream.
When your AWS DeepLens device is registered and connected, you can begin using it. To use your device for deep learning, you create a project and deploy it to your AWS DeepLens device. An AWS DeepLens project is made up of deep learning models and associated AWS Lambda functions.

AWS DeepLens comes with several sample projects that you can deploy and use right out of the box.

Topics

- AWS DeepLens Sample Projects Overview (p. 33)
- Creating and Deploying an AWS DeepLens Sample Project (p. 35)
- Extending any Project’s Functionality (p. 37)
- Editing an Existing Model with Amazon SageMaker (p. 41)

AWS DeepLens Sample Projects Overview

To get started with AWS DeepLens, use the sample project templates. AWS DeepLens sample projects are projects where the model is pre-trained so that all you have to do is create the project, import the model, deploy the project, and run the project. Other sections in this guide teach you to extend a sample project’s functionality so that it performs a specified task in response to an event, and train a sample project to do something different than the original sample.

Artistic Style Transfer

This project transfers the style of an image, such as a painting, to an entire video sequence captured by AWS DeepLens.

This project shows how a Convolutional Neural Network (CNN) can apply the style of a painting to your surroundings as it’s streamed with your AWS DeepLens device. The project uses a pretrained optimized model that is ready to be deployed to your AWS DeepLens device. After deploying it, you can watch the stylized video stream.

You can also use your own image. After fine tuning the model for the image, you can watch as the CNN applies the image’s style to your video stream.

- Project model: deeplens-artistic-style-transfer
- Project function: deeplens-artistic-style-transfer

Object Recognition

This project shows you how a deep learning model can detect and recognize objects in a room.

The project uses the Single Shot MultiBox Detector (SSD) framework to detect objects with a pretrained resnet_50 network. The network has been trained on the Pascal VOC dataset and is capable of recognizing 20 different kinds of objects. The model takes the video stream from your AWS DeepLens device as input and labels the objects that it identifies. The project uses a pretrained optimized model that is ready to be deployed to your AWS DeepLens device. After deploying it, you can watch your AWS DeepLens model recognize objects around you.

The model is able to recognize the following objects: airplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train, and TV monitor.
• **Project model:** deeplens-object-detection  
• **Project function:** deeplens-object-detection

**Face Detection and Recognition**

With this project, you use a face detection model and your AWS DeepLens device to detect the faces of people in a room.

The model takes the video stream from your AWS DeepLens device as input and marks the images of faces that it detects. The project uses a pretrained optimized model that is ready to be deployed to your AWS DeepLens device.

• **Project model:** deeplens-face-detection  
• **Project function:** deeplens-face-detection

**Hot Dog Recognition**

Inspired by a popular television show, this project classifies food as either a hot dog or not a hot dog.

It uses a model based on the SqueezeNet deep neural network. The model takes the video stream from your AWS DeepLens device as input, and labels images as a hot dog or not a hot dog. The project uses a pretrained, optimized model that is ready to be deployed to your AWS DeepLens device. After deploying the model, you can use the Live View feature to watch as the model recognizes hot dogs.

You can edit this model by creating Lambda functions that are triggered by the model's output. For example, if the model detects a hot dog, a Lambda function could send you an SMS message. To learn how to create this Lambda function, see [Editing an Existing Model with Amazon SageMaker](#) (p. 41)

**Cat and Dog Recognition**

This project shows how you can use deep learning to recognize a cat or a dog.

It is based on a convolutional neural network (CNN) architecture and uses a pretrained Resnet-152 topology to classify an image as a cat or a dog. The project uses a pretrained, optimized model that is ready to be deployed to your AWS DeepLens device. After deploying it, you can watch as AWS DeepLens uses the model to recognize your pets.

• **Project model:** deeplens-cat-and-dog-recognition  
• **Project function:** deeplens-cat-and-dog-recognition

**Action Recognition**

This project recognizes more than 30 kinds of activities.

It uses the Apache MXNet framework to transfer learning from a SqueezeNet trained with ImageNet to a new task. The network has been tuned on a subset of the UCF101 dataset and is capable of recognizing more than 30 different activities. The model takes the video stream from your AWS DeepLens device as input and labels the actions that it identifies. The project uses a pretrained, optimized model that is ready to be deployed to your AWS DeepLens device.

After deploying the model, you can watch your AWS DeepLens use the model to recognize 37 different activities, such as applying makeup, applying lipstick, participating in archery, playing basketball, bench
pressing, biking, playing billiards, blowing drying your hair, blowing out candles, bowling, brushing teeth, cutting things in the kitchen, playing a drum, getting a haircut, hammering, handstand walking, getting a head massage, horseback riding, hula hooping, juggling, jumping rope, doing jumping jacks, doing lunges, using nunchucks, playing a cello, playing a flute, playing a guitar, playing a piano, playing a sitar, playing a violin, doing pushups, shaving, skiing, typing, walking a dog, writing on a board, and playing with a yo-yo.

- **Project model:** deeplens-action-recognition
- **Project function:** deeplens-action-recognition

### Head Pose Detection

This sample project uses a deep learning model generated with the TensorFlow framework to accurately detect the orientation of a person’s head.

This project uses the ResNet-50 network architecture to detect the orientation of the head. The network has been trained on the Prima HeadPose dataset, which comprises 2,790 images of the faces of 15 people, with variations of pan and tilt angles from -90 to +90 degrees. We categorized these head pose angles to 9 head pose classes: down right, right, up right, down, middle, up, down left, left, and up left.

To help you get started, we have provided a pretrained, optimized model ready to deploy to your AWS DeepLens device. After deploying the model, you can watch AWS DeepLens recognize various head poses.

- **Project model:** deeplens-head-pose-detection
- **Project function:** deeplens-head-pose-detection

### Creating and Deploying an AWS DeepLens Sample Project

To help you get started with AWS DeepLens, we provide a number of sample AWS DeepLens project templates that you can use to create projects and get you up and going quickly. For more information, see AWS DeepLens Sample Projects Overview (p. 33).

In this walkthrough, you create the Object Detection project. The Object Detection project analyzes images within a video stream on your AWS DeepLens device to identify objects. It can recognize as many as 20 types of objects.

Though the instructions here are specific to the Object Detection project, you can follow the same steps to create and deploy any of the sample projects. When creating a sample project, the fields in the console are pre-populated for you so you can accept the defaults. In the **Project content** portion of the screen, you need to know the project’s model and function. That information is available for the individual projects in the AWS DeepLens Sample Projects Overview (p. 33) topic.

Your web browser is the interface between you and your AWS DeepLens device. You perform all of the following activities on the AWS DeepLens console using your browser.
Step 1: Create Your Project

The following procedure creates the Object Detection sample project.

To create an AWS DeepLens project using a sample project

The steps for creating a project encompass two screens. On the first screen you select your project. On the second screen, you specify the project's details.

2. Choose Projects, then choose Create new project.
3. On the Choose project type screen
   a. Choose Use a project template, then choose the sample project you want to create. For this exercise, choose Object detection.
   b. Scroll to the bottom of the screen, then choose Next.
4. On the Specify project details screen
   a. In the Project information section:
      i. Either accept the default name for the project, or type a name you prefer.
      ii. Either accept the default description for the project, or type a description you prefer.
   b. In the Project content section:
      i. Model—make sure the model is the correct model for the project you're creating. For this exercise it should be deeplens-object-detection. If it isn't, remove the current model then choose Add model. From the list of models, choose deeplens-object-detection.
      ii. Function—make sure the function is the correct function for the project you're creating. For this exercise it should be deeplens-object-detection. If it isn't, remove the current function then choose Add function. From the list of functions, choose deeplens-object-detection.
Step 2: Deploy Your Project

In this walkthrough, you deploy the Object Detection project.

Your web browser is the interface between you and your AWS DeepLens device. You perform all of the following activities on your browser after logging on to AWS:

1. On the Projects screen, choose the radio button to the left of your project name, then choose Deploy to device.
2. On the Target device screen, from the list of AWS DeepLens devices, choose the radio button to the left of the device that you want to deploy this project to. An AWS DeepLens device can have only one project deployed to it at a time.
3. Choose Review.

   If a project is already deployed to the device, you will see an error message that deploying this project will overwrite the project that is already running on the device. Choose Continue project.

   This will take you to the Review and deploy screen.
4. On the Review and deploy screen, review your project and choose either Previous to go back and make changes, or Deploy to deploy the project.

   Important
   Deploying a project incurs costs for the AWS services that are used to run the project.

   For instructions on viewing your project's output, see Viewing AWS DeepLens Output Streams (p. 28).

Extending any Project's Functionality

In this section, you take the "Hotdog recognition" sample project and add some rule-based functionality to it to make AWS DeepLens send an SMS notification whenever it detects a hot dog. Though we use the "Hotdog recognition" sample project in this topic, this process could be used for any project, sample or custom.

This section demonstrates how to extend your AWS DeepLens projects to interact with other AWS services. For example, you could extend AWS DeepLens to create:

- A dashboard and search interface for all objects and faces detected by AWS DeepLens with timelines and frames using Amazon Elasticsearch Service.
- Anomaly detection models to detect the number of people walking in front of your store using Kinesis Data Analytics.
- A face detection and celebrity recognition application to identity VIPs around you using Amazon Rekognition.

In this exercise, you modify the project you previously created and edited (see Editing an Existing Model with Amazon SageMaker (p. 41)) to use the AWS IoT rules engine and an AWS Lambda function.

Topics
- Create and Configure the Lambda Function (p. 38)
Create and Configure the Lambda Function

Create and configure an AWS Lambda function that runs in the Cloud and filters the messages from your AWS DeepLens device for those that have a high enough probability (>0.5) of being a hot dog. You can also change the probability threshold.

Create a Lambda Function

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
   Make sure you have selected the US East (N. Virginia) AWS Region.
2. Choose Create function.
3. Choose Author from scratch.
4. Type a name for the Lambda function, for example, \texttt{<your name>_hotdog_notifier}.
5. For Role, keep Create a new Role from template(s).
6. Type a name for the role; for example, \texttt{<your name>_hotdog_notifier}.
7. For Policy Templates, choose SNS Publish policy.
8. Choose Create function.

Add an AWS IoT Rule

This AWS IoT rule specifies the source of the data that triggers the action you specify in your Lambda function (the next step).

1. Scroll down to \texttt{aws-iot}.
2. For IoT type, choose Custom IoT rule.
3. For Rule, choose Create new rule.
4. Type a name (\texttt{<your-name>_search_hotdogs}) and a description for the rule.
5. Paste a AWS IoT topic into the Rule query statement box. Replace the red text with the AWS IoT topic for your AWS DeepLens. To find the AWS IoT topic, navigate to Devices on your AWS DeepLens, choose your device, then scroll to the bottom of the device detail page.

```
Select Hotdog from '/$aws/deeplens/$aws/things/deepensa_5e6d406g-2bf4-4444-9d4f-4668f7366855/infer'
```

This query captures messages from your AWS DeepLens in JSON format:

```
{ "Hotdog" : "0.5438" }
```

6. Choose Enable trigger.
7. Scroll to the bottom of the page and choose Add.
8. Choose Save to save the IoT rule.

Configure the Lambda Function

Configure the Lambda function by replacing the default code with custom code and adding an environmental variable. For this project, you also need to modify the custom code that we provide.
1. In AWS Lambda, choose **Functions**, then choose the name of your function.

2. On the **your-name_hotdog_notifier** page, choose **Configuration**.

3. In the function code box, delete all of the code.

4. Paste the following code in the function code box. You need to change one line in the code to indicate how you want to get notifications. You do that in the next step.

```javascript
/**
 * This is a sample Lambda function that sends an SMS notification when your
 * AWS DeepLens device detects a hot dog.
 * Follow these steps to complete the configuration of your function:
 * Update the phone number environment variable with your phone number.
 */

const AWS = require('aws-sdk');

/* Be sure to add email and phone_number to the function's environment variables */
const email = process.env.email;
const phone_number = process.env.phone_number;
const SNS = new AWS.SNS({ apiVersion: '2010-03-31' });

exports.handler = (event, context, callback) => {
  console.log('Received event:', event);
  // publish message
  const params = {
    Message: 'Your AWS DeepLens device just identified a hot dog.
    Congratulations!',
    PhoneNumber: phone_number,
  }
  if (event.label.includes("Hotdog")) {
    SNS.publish(params, callback);
  }
};
```

5. Add one of the following lines of code in the location indicated in the code block. In the next step, you add an environmental variable that corresponds to the code change you make here.

- To receive email notifications: `const email=process.env.email;`
- To receive phone notifications: `const phone_number=process.env.phone_number;`

6. Choose **Environmental variables** and add one of the following:

<table>
<thead>
<tr>
<th>Notification by</th>
<th>Key</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Email</td>
<td>email</td>
<td>Your complete email address.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: <strong><a href="mailto:YourAlias@ISP.com">YourAlias@ISP.com</a></strong></td>
</tr>
<tr>
<td>Phone</td>
<td>phone_number</td>
<td>Your phone number with country code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Example: <strong>US +1 8005551212</strong></td>
</tr>
</tbody>
</table>

The key value must match the `const` name in the line of code that you added in the previous step.

7. Choose **Save** and **Test** (on the upper right).
Test Your Configuration

**To test your configuration**

2. Choose **Test**.
3. Publish the following message to the topic that you defined in your rule: `{ "Hotdog" : "0.6382" }.

   You should receive the SMS message that you defined in your Lambda function: Your AWS DeepLens device just identified a hot dog. Congratulations!

Test Using the Hot Dog Project

If you haven't already deployed the Hot Dog project, do the following.

1. Navigate to https://console.aws.amazon.com/deeplens/home?region=us-east-1#firstrun/ and choose **Projects/Create a project template/Hotdog or Not Hotdog**.
2. Deploy the project to your device.
   
   For more information, see Creating and Deploying an AWS DeepLens Sample Project (p. 35).
3. Show your AWS DeepLens a hot dog to see if it detects it and sends you the confirmation message.

   To experiment, change the probability threshold for triggering the Lambda function and see what happens.

Disable the AWS IoT Rule

Unless you want AWS DeepLens to keep notifying you when it sees a hot dog, disable the AWS IoT rule.

1. Log in to the AWS IoT console at **https://console.aws.amazon.com/iot**.
2. Choose **Act**, then choose the rule that you created for this exercise, `<your-name>_search_hotdogs`.
3. In the upper-right corner, choose **Actions**, then choose **Disable**.
Editing an Existing Model with Amazon SageMaker

In this example, you start with a SqueezeNet object detection model and use Amazon SageMaker to train it to perform binary classification to determine whether an object is a hot dog. The example shows you how to edit a model to perform binary classification, and explains learning rate and epochs. We have provided a Jupyter notebook instance, which is open source software for interactive computing. It includes the editing code to execute and explanations for the entire process.

After training the model, you import its artifacts into AWS DeepLens, and create a project. You then watch as your AWS DeepLens detects and identifies hot dogs.

Topics

• Step 1: Create an Amazon S3 Bucket (p. 42)
• Step 2: Create an Amazon SageMaker Notebook Instance (p. 43)
• Step 3: Edit the Model in Amazon SageMaker (p. 45)
• Step 4: Optimize the Model (p. 46)
• Step 5: Import the Model (p. 47)
• Step 6: Create an Inference Lambda Function (p. 48)
• Step 7: Create a New AWS DeepLens Project (p. 49)
• Step 8: Review and Deploy the Project (p. 50)
• Step 9: View Your Model’s Output (p. 51)
Step 1: Create an Amazon S3 Bucket

Before you begin, be sure that you have created an AWS account, and the required IAM users and roles.

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.
   Make sure you are in the US East (N. Virginia) region.
2. Choose Create bucket.
3. On the Name and region screen:
   a. Name the bucket `deeplens-sagemaker-<your-name>`. The bucket name must begin with `deeplens-sagemaker-` or the services will not be able to access it.
   b. Verify that you are in the US East (N. Virginia) region.
   c. Choose Next.
4. On the Set properties screen choose Next.
5. On the Set permissions screen, verify that both Objects and Object permissions have both the Read and Write permissions set, then choose Next.
6. On the Review screen, review your settings then choose Create bucket which creates your Amazon S3 bucket and returns you to the Amazon S3 screen.
7. On the Amazon S3 screen, locate and choose your bucket's name.
8. On your bucket's screen, choose Permissions, then under Public access choose Everyone.
9. On the Everyone popup, under Access to objects enable List objects and Write objects. Under Access to this bucket's ACL enable Read bucket permissions and Write bucket permissions, then choose Save.
10. After you return to your bucket's page, choose Overview then choose Create folder.
11. Name the folder test then choose Save.
Step 2: Create an Amazon SageMaker Notebook Instance

Create an Amazon SageMaker notebook instance.

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
   Make sure that you have chosen the us-east-1 — US East (N. Virginia) Region.
2. Choose Create notebook instance.
3. On the Create notebook instance page, then do the following:

   a. For Notebook instance name, type a name; for example, `<your-name>-hotdog`.
   b. For Instance type, choose ml.t2.medium.
   c. For IAM role choose Enter a custom IAM role ARN, paste the Amazon Resource Name (ARN) of your Amazon SageMaker role in the Custom IAM role ARN box.

   To find the ARN of your Amazon SageMaker role:
   
   i. Open the IAM console at https://console.aws.amazon.com/iam/.
   ii. In the navigation pane, choose Roles.
   iii. Find the AWSDeepLensSagemakerRole and choose its name. This takes you to the role's Summary page.
   iv. On the Summary page, locate and copy the Role ARN. The ARN will look something like this:

   
   ```
   arn:aws:iam::account_id:role/AWSDeepLensSagemakerRole
   ```

   d. Both VPC and Encryption key are optional. Skip them.

   **Tip**
   
   If you want to access resources in your VPC from the notebook instance, choose a VPC and a SubnetId. For Security Group, choose the default security group of the VPC.
The inbound and outbound rules of the default security group are sufficient for the exercises in this guide.

e. Choose Create notebookinstance.

Your new notebook instance is now available on the Notebooks page.
Step 3: Edit the Model in Amazon SageMaker

In this step, you open the `<your-name>-hotdog` notebook and edit the object detection model so it recognizes a hot dog. The notebook contains explanations to help you through each step.

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
2. Choose the US East (N. Virginia) Region is chosen.
3. In the navigation pane, choose Notebook instances.
4. On the Notebooks page, choose the radio button to the left of the notebook instance that you just created (`<your-name>-hotdog`). When the notebook's status is InService, choose Open.
6. Download the .zip file or clone the Git repository with the following command. If you downloaded the .zip file, locate it and extract all.

   ```
   git clone git@github.com:aws-samples/reinvent-2017-deeplens-workshop.git
   ```

   If you downloaded the .zip file, locate it and extract all.

You now upload the training file and use it to edit the model.

1. On the Jupyter tab, choose Upload.
2. Navigate to the extracted file `deeplens-hotdog-or-not-hotdog.ipynb` then choose Open, then choose Upload.
3. Locate and choose the `deeplens-hotdog-or-not-hotdog` notebook.
4. In the upper right corner of the Jupyter screen, verify that the kernal is `conda_mxnet_p36`. If it isn't, change the kernal.
5. In the `deeplens-hotdog-or-not-hotdog.ipynb` file, search for `bucket= 'your S3 bucket name here'`. Replace `your S3 bucket name here` with the name of your S3 bucket, for example `deeplens-sagemaker-<your-name>`.

   Return to the top of the file.

   For each step (In `[#]:`) in the file:

   a. Read the step's description.
   b. If the block has code in it, place your cursor in the code block and run the code block. To run a code block in Jupyter, use `Ctrl+<Enter>` (macOS `Cnd_<Enter>`) or choose the run icon (▶).

   **Important**

   Each step is numbered in a fashion such as `In [1]:`. While the block is executing, that changes to `In [*]:`. When the block finishes executing it returns to `In [1]:`. Do not move on to the next code block while the current block is still running.

6. After you finish editing the model, return to the Amazon S3 console, choose your bucket name, choose the `test` folder, and then verify that the following artifacts of the edited model are stored in your S3 bucket's test folder.

   - Hotdog_or_not_model-0000.params
   - Hotdog_or_not_model-symbol.json
Step 4: Optimize the Model

Now that you have a trained mxNet model there is one final step that is required before you run the model on the AWS DeepLens’s GPU. The trained mxNet model does not come in a computationally optimized format. If we deploy the model in the original format it will run on the CPU via mxNet at sub optimal speed. In order to run the model at optimal speed on the GPU we need to perform model optimization. For instructions on how to optimize your MXNet model, see Optimizing a Custom Model (p. 54).
**Step 5: Import the Model**

Import the edited model into AWS DeepLens.

2. Choose **Models**, then choose **Import model**.
3. For **Import model to AWS DeepLens**, choose **Externally trained model**.
4. For **Model settings**, do the following:
   a. For **Model artifact**, type the path to the artifacts that you uploaded to the Amazon S3 bucket in the previous step. The path begins with `s3://deeplens-`. For example, if you followed the naming in Step 1, the path will be `s3://deeplens-sagemaker-<your-name>/<dir>`.
   b. For **Model name**, type a name for the model.
   c. For **Model description**, type a description.
5. Choose **Import model**.
Step 6: Create an Inference Lambda Function

Use the AWS Lambda console to create a Lambda function that uses your model. For specific instructions with sample code, see Creating an AWS DeepLens Inference Lambda Function (p. 55).
Step 7: Create a New AWS DeepLens Project

Now create a new AWS DeepLens project and add the edited model to it.

2. Choose Projects.
3. Choose Create new project.
4. For Choose project type, choose Create a new blank project, then choose Next.
5. For Project information, type a name and description for this project.
6. For Project content, choose Add model.
7. Search for and choose the model that you just created, then choose Add model.
8. Choose Add function, then choose deplens-hotdog-o-hotdog, then choose Add function.
9. Choose Create.
Step 8: Review and Deploy the Project

2. From the list of projects, choose the project that you just created, then choose **Deploy to device**.
3. Choose your AWS DeepLens as your target device, then choose **Review**.
4. Review the settings, then choose **Deploy**.

**Important**

Deploying a project incurs costs for the various AWS services that are used.
Step 9: View Your Model's Output

To view your model's output, follow the instructions at Viewing AWS DeepLens Output Streams (p. 28).
Working with AWS DeepLens Custom Projects

When you are ready, you can create and deploy your own projects. The topics in this section cover the tasks you need to perform to create, optimize, and deploy your personal projects.

Topics

- Importing Your Amazon SageMaker Trained Model (p. 52)
- Importing an Externally Trained Model (p. 52)
- Optimizing a Custom Model (p. 54)
- Creating an AWS DeepLens Inference Lambda Function (p. 55)

Importing Your Amazon SageMaker Trained Model

To use your Amazon SageMaker trained module you must import it into AWS DeepLens.

To import your Amazon SageMaker trained model into AWS DeepLens

2. From the navigation pane, choose Models then choose Import model.
3. For Import source choose Amazon SageMaker trained model.
4. In the Model settings area:
   a. From the list of completed jobs, choose the Amazon SageMaker training job ID for the model you want to import.
   The ID of the job must begin with `deeplens-`, unless you've customized the AWSDeepLensServiceRolePolicy (used for registering the device) to extend the policy to allow AWS DeepLens to access Amazon SageMaker's training jobs not starting with `deeplens*`.
   If you do not find the job you're looking for in the list, go to the Amazon SageMaker console and check the status of the jobs to verify that it has successfully completed.
   b. For the Model name, type the name you want for the model. Model names can contain alphanumeric characters and hypens, and be no longer than 100 characters.
   c. For the Description you can optionally type in a description for your model.
   d. Choose Import model.

Importing an Externally Trained Model

To use an externally trained model, import it.

To import an externally trained model

2. In the navigation pane, choose Models, then choose Import model.
3. For Import source, choose Externally trained model.
4. For Model settings, provide the following information.
   a. For Model artifact path, type the full path to the Amazon Simple Storage Service (Amazon S3) location of the artifacts created when you trained your model. The path begins with `s3://`
deeplens-. For example, if you followed the naming conventions we use throughout this documentation, the path will be s3://deeplens-sagemaker-your_name/<dir>.
b. For Model name, type a name. A model name can have a maximum of 100 alphanumeric characters and hypens.
c. Optional. For Description, type a description of the model.
d. Choose Import model.
Optimizing a Custom Model

To access the GPU for inference, AWS DeepLens uses the clDNN, Compute Library for Deep Neural Networks. To run your own models on AWS DeepLens, you have to convert them into clDNN format. The model optimizer converts the format with the following code:

```python
error, model_path = mo.optimize(model_name,input_width,input_height)
```

You include the model optimizer code in an inference Lambda function, which is required to allow AWS DeepLens to access deployed models.

For information on how to create an inference Lambda function that includes the model optimizer, see Creating an AWS DeepLens Inference Lambda Function (p. 55).

For more information about the model optimizer, see Model Optimization (mo) Module (p. 82).
Creating an AWS DeepLens Inference Lambda Function

In this topic, you create an inference Lambda function that performs three key functions: preprocessing, inference, and postprocessing. Each step is accompanied by the associated code. For the complete function code, see The Completed Lambda Function (p. 59).

To create an AWS DeepLens inference Lambda function

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose Create function, then choose Blueprints.
3. Choose the Blueprints box then from the dropdown, choose Blueprint name and type greengrass-hello-world.
4. When the greengrass-hello-world blueprint appears, choose it, then choose Configure.
5. In the Basic information section:
   a. Type a name for your Lambda function.
   b. From the Role list, choose Choose an existing role.
   c. Choose AWSDeepLensLambdaRole, which you created when you registered your device.
6. Scroll to the bottom of the page and choose Create function.
7. In the function code box, makes sure that the handler is greengrassHelloWorld.function_handler. The name of the function handler must match the name of the Python script that you want to run. In this case, you are running the greengrassHelloWorld.py script, so the name of the function handler is greengrassHelloWorld.function_handler.
8. Delete all of the code in the GreengrassHelloWorld.py box and replace it with the code that you generate in the rest of this procedure.
9. Add the following code to your Lambda function.
   a. Import these required packages:
      • os—Allows your Lambda function to access the AWS DeepLens operating system.
      • awscam—Allows your Lambda function to use the AWS DeepLens Device Library. For more information, see Model Object (p. 77).
      • mo — Allows your Lambda function to access the AWS DeepLens model optimizer. For more information, see Model Optimization (mo) Module (p. 82).
      • cv2—Allows your Lambda function to access the Open CV library, which contains tools for image preprocessing.
      • thread—Allows your Lambda function to access Python’s multi-threading library.

To import these packages, copy and paste the following code into the GreengrassHello file.

```python
import os
import awscam
import mo
import cv2
from threading import Thread
```

b. Create a Greengrass SDK client. You will use this client to send messages to the cloud.

```python
client = greengrasssdk.client('iot-data')
```
c. Create an AWS IoT topic for your Lambda function’s messages. You can access this topic in the AWS IoT console.

```python
iot_topic = '#aws/things/{}infer'.format(os.environ['AWS_IOT_THING_NAME'])
```

d. To view the output locally with mplayer, declare a global variable that contains the .jpeg image that you send to the FIFO file `results.mjpeg`.

```python
jpeg = None
Write_To_FIFO = True
# making Write_To_FIFO = False kills the thread so you cannot view your output over mplayer
```

e. To publish the output images to the FIFO file and view them with mplayer, create a class that runs on its own thread.

```python
class FIFO_Thread(Thread):
    def __init__(self):
        '''Constructor.''
        Thread.__init__(self)

    def run(self):
        fifo_path = '/tmp/results.mjpeg'
        if not os.path.exists(fifo_path):
            os.mkfifo(fifo_path)
        f = open(fifo_path,'w')
        client.publish(topic=iot_topic, payload="Opened Pipe")

        while Write_To_FIFO:
            try:
                f.write(jpeg.tobytes())
            except IOError as e:
                continue
```

f. Define an inference class.

i. Define an AWS Greengrass inference function. `input_width` and `input_height` define the width and height of the input in pixels. To perform inference, the model expects frames of this size. You can customize these values for the model that you are deploying to AWS DeepLens.

```python
def greengrass_infinite_infer_run():
    input_width  = 224
    input_height = 224
```

ii. Name the model. The name is the prefix of the trained model’s `params` and `json` files. For example, if the files are named `squeezenet_v1.1-0000.params` and `squeezenet_v1.1-0000.json`, the model name is the prefix `squeezenet_v1.1`.

   **Important**
   The model name must match the prefix. Otherwise, the model can’t perform inference, and generates an error.

```python
model_name = 'squeezenet_v1'
```

iii. Initialize the model optimizer. The model optimizer converts the deployed model to clDNN format, which is accessible to the AWS DeepLens GPU. The model optimizer returns the path to the post-optimized artifacts.
iv. Load the model into the inference engine. To use the CPU, instead of the GPU, specify "GPU":0. The CPU is much less efficient, so we don't recommend using it.

```python
model = awscam.Model(model_path,{"GPU":1})
# You can send a message to AWS IoT to show that the model is loaded.
client.publish(topic=iot_topic, payload="Model loaded.")
```

v. Define the type of model that you are running. The options are:

- **segmentation**—For neural style transfer.
- **ssd**—Single shot detector. For object localization it includes a definition of the locale in the frame that the object occupies by drawing a bounding box around the object.
- **classification**—For image classification.

Because you are deploying a SqueezeNet model that classifies images, define the model type as `classification`.

```python
model_type = "classification"
```

vi. Map the numeric label generated by the model to a human-readable label. Because squeezenet_v1.1 has 1,000 classifiers, it's unrealistic to create the mapping in code. Instead, add a text file to the Lambda .zip file. You can then load the labels into a list where the index of the list represents the label returned by the network.

```python
with open('sysnet.txt', 'r') as f:
    labels = [l.rstrip() for l in f]
```

vii. Define the number of classifiers that you want to see in the output.

```python
topk = 5
```

The value 5 specifies that the top 5 values with the highest probability are output, in descending order. You can specify any value as long as it's supported by the model.

viii. Start the FIFO thread so you can view the output with the mplayer.

```python
results_thread = FIFO_Thread()
results_thread.start()
# You can publish an "Inference starting" message to the AWS IoT console.
client.publish(topic = iot_topic, payload = "Inference starting")
```

ix. Get the most recent frame from the AWS DeepLens camera. If the latest frame is not returned, raise an exception.

```python
ret, frame = awscam.getLastFrame()
if ret == False:
    raise Exception("Failed to get frame from the stream")
```

x. Preprocess the input frame from the camera by making sure that its dimensions match the dimensions of the frame that the model was trained on. To resize the input frame, specify the input dimensions defined earlier, `input_width` and `input_height`. Depending on the model that you trained, you might need to perform other preprocessing steps, such as image normalization.
frame_resize = cv2.resize(frame, (input_width, input_height))

xi. Perform inference on the resized frame.

infer_output = model.doInference(frame_resize)

xii. Parse the results.

parsed_results = model.parseResult(model_type, infer_output)

xiii. Display only the $n$ results that have the highest probability.

top_k = parsed_results[model_type][0:topk]

xiv. Send the results to the cloud.

First, put the message in JSON format. This allows other Lambda functions in the cloud to subscribe to the AWS IoT topic and perform actions when they detect an interesting event.

```
msg = "\
    prob_num = 0
    for obj in top_k:
        if prob_num == topk-1:
            msg += "{}":{:.2f}".format(labels[obj["label"]], obj["prob"])
        else:
            msg += "{}":{:.2f}".format(labels[obj["label"]], obj["prob"])
        prob_num += 1
    msg += ""
```

Then send it to the cloud.

```
client.publish(topic="iot_topic", payload = msg)
```

xv. Postprocess the image. In this case add a line of text to the image: a label of the most likely results.

```
cv2.putText(frame, labels[top_k[0]["label"]], (0,22), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 165, 20), 4)
```

xvi. Update the global jpeg variable so you can view the results with mplayer.

```
global jpeg
ret, jpeg = cv2.imencode('.jpg', frame)
  # If you want, you can add exception handling as follows.
  # Don't forget to put the preceding code in a try block.
except Exception as e:
    msg = "Lambda function failed: " + str(e)
    client.publish(topic=iot_topic, payload = msg)
```

xvii. Run the function and view the results.

```
greengrass_infinite_infer_run()
```

Make sure that you save and publish the function code. If you don't, you can't view the inference Lambda function that you just created in the AWS DeepLens console.
The Completed Lambda Function

The following code creates the Lambda function that allows AWS DeepLens to access deployed models.

```python
# Copyright Amazon AWS DeepLens, ©2018
import os                      # access to operating system for AWS DeepLens
import awscam                  # access to AWS DeepLens Device Library
import mo                      # access to AWS DeepLens model optimizer
import cv2                     # access to Open CV library
from threading import Thread   # access to Python's multi-threading library

# create a Greengrasscore SDK client
client = greengrasssdk.client('iot-data')

# create AWS IoT for the Lambda function to send messages
iot_topic = '$aws/things/{}/infer'.format(os.environ['AWS_IOT_THING_NAME'])

# global variable to contain jpeg image
jpeg = None

# create a simple class that runs on its own thread so we can publish output images
# to the FIFO file and view using mplayer
class FIFO_Thread(Thread):
    def __init__(self):
        '''Constructor.''
        Thread.__init__(self)

    def run(self):
        fifo_path = '/tmp/results.mjpeg'
        if not os.path.exists(fifo_path):
            os.mkfifo(fifo_path)
        f = open(fifo_path, 'w')
        client.publish(topic=iot_topic, payload="Opened Pipe")

        while Write_To_FIFO:
            try:
                f.write(jpeg.tobytes())
            except IOError as e:
                continue

# define inference class within the Lambda function
def greengrass_infinite_infer_run():
    input_width  = 224
    input_height = 224

    # define the name of the model
    model_name = 'squeezenet_v1'

    # optimize the model into Cl-DNN format
    error, model_path = mo.optimize(model_name, input_width, input_height)

    # load the model into the inference engine
    model = awscam.Model(model_path, {"GPU":1})
    # You can send a message to AWS IoT to show that the model is loaded.
    client.publish(topic=iot_topic, payload="Model loaded.")

    # define the type of model
    # possibilities are:
```
AWS DeepLens Developer Guide
Creating an Inference Lambda

# segmentation - for neural style transfers
# ssd - (single shot detector) for object localization
# classification - for image classification

model_type = "classification"

# load the labels into a list where the index represents the label returned by the
# network
with open('sysnet.txt', 'r') as f:
    labels = [l.rstrip() for l in f]

# define the number of classifiers to see
topk = 5

# start the FIFO thread to view the output locally
results_thread = FIFO_Thread()
results_thread.start()

# you can publish an "Inference starting" message to the AWS IoT console
client.publish(topic = iot_topic, payload = "Inference starting")

# access the latest frame on the mjpeg stream
ret, frame = awscam.getLastFrame()
if ret == False:
    raise Exception("Failed to get frame from the stream")

# resize the frame to the size expected by the model
frame_resize = cv2.resize(frame, (input_width, input_height))

# do inference on the frame
infer_output = mdel.doInference(frame_resize)

# parse the results and keep the top topk results
parsed_results = model.parseResult(model_type, infer_output)
top_k = parsed_results[model_type][0:topk]

# format the results as JSON and send to the cloud
msg = "{""""prob_num = 0
for obj in top_k:
    if prob_num == topk-1:
        msg += "{}":{:.2f}'.format(labels[obj["label"]], obj["prob"])
    else:
        msg += "{}":{:.2f}'.format(labels[obj["label"]], obj["prob"])
    prob_num += 1
msg += "}"
client.publish(topic = iot_topic, payload = msg)

# post-process the image to view it on the mplayer
# add a line of text to the image: a label of the most likely results
cv2.putText(frame, labels[top_k[0]["label"]], (0,22), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 165, 20), 4)

# define a global variable so results can be viewed using mplayer
global jpeg
ret, jpeg = cv2.imencode('.jpg', frame)

# Catch an exception in case something went wrong
except Exception as e:
    msg = "Lambda function failed: " + str(e)
    client.publish(topic=iot_topic, payload = msg)

# run the function and view the results
greengrass_infinite_infer_run()
Managing Your AWS DeepLens Device

The following topics explain how to manage your AWS DeepLens device:

Topics
- Securely Boot Your AWS DeepLens Device (p. 61)
- Update Your AWS DeepLens Device (p. 61)
- Deregistering Your AWS DeepLens Device (p. 62)
- Deleting AWS DeepLens Resources (p. 62)

Securely Boot Your AWS DeepLens Device

To protect the AWS DeepLens device from malicious attacks, it is configured to boot securely. The secure boot settings prevent the device from loading unauthorized operating systems, including versions that are under mainstream support. If an attacker bypasses the secure boot settings and loads other operating systems or other versions, the device could become unstable and the warranty will be void.

Update Your AWS DeepLens Device

When you set up your device, you had the option to enable automatic updates (see Set Up Your AWS DeepLens Device (p. 15)). If you enabled automatic updates, you need only to reboot the device to get the software updated on your device. If you didn't enable automatic updates, you need to manually update your device periodically.

To manually update your AWS DeepLens using your password

1. Plug in your AWS DeepLens and turn it on.
2. Use a micro HDMI cable to connect your AWS DeepLens to a monitor.
3. Connect a USB mouse and keyboard to your AWS DeepLens.
4. When the login screen appears, sign in to the device using the SSH password you set when you registered it.
5. Start your terminal and run each of the following commands:

   ```
   sudo apt-get update
   sudo apt-get install awscam
   sudo reboot
   ```

To manually update your AWS DeepLens using your IP address

1. Find your IP address by either logging into Ubuntu, or looking at your Wi-Fi router.
2. Start a terminal and type:

   ```
   ssh aws_cam@IP-address
   ```

3. Run each of the following commands:
Deregistering Your AWS DeepLens Device

Deregistering your AWS DeepLens disassociates your AWS account and credentials from the device. Before you deregister your device, delete the photos or videos that are stored on it.

To deregister your AWS DeepLens

2. From the primary navigation pane, choose Devices, then choose the device that you want to deregister.
3. Next to the Current project on the selected device page, choose Remove current project from device.

   **Important**
   Delete the photos or videos that are stored on the device, using SSH and the SSH password that you set when you registered the device to log on to the device. Navigate to the folder where the photos or videos are stored and delete them.

4. When prompted, choose Yes to confirm.
5. Next to Device Settings, choose Deregister.
6. When prompted, choose Deregister to confirm.

Your AWS DeepLens is now deregistered. To use it again, repeat each of these steps:

- Register Your AWS DeepLens Device (p. 8)
- Connect to Your AWS DeepLens Device's Wi-Fi Network (p. 13)
- Set Up Your AWS DeepLens Device (p. 15)
- Verify Your AWS DeepLens Device Status (p. 17)

Deleting AWS DeepLens Resources

When you build your computer vision application and deploy it to your AWS DeepLens device, you store AWS DeepLens resources—such as a device profile, a deployed project, and a model—in the AWS Cloud. When you are done with your AWS DeepLens project, you can use the AWS console to delete the model, project, and AWS DeepLens logs from CloudWatch Logs. To delete a registered device profile, see Deregistering Your AWS DeepLens Device (p. 62).

Topics

- Deleting AWS DeepLens Resources Using the AWS Console (p. 62)
• Deleting an AWS DeepLens Project (p. 63)
• Deleting an AWS DeepLens Model (p. 63)
• Deleting an AWS DeepLens Device Profile (p. 63)
• Deleting AWS DeepLens Logs from CloudWatch Logs (p. 63)

Deleting an AWS DeepLens Project

2. In the navigation pane, choose Projects.
3. On the Projects page, choose the button next to the project that you want to delete.
4. In the Actions list, choose Delete.
5. In the Delete project-name dialog box, choose Delete to confirm that you want to delete the project.

Deleting an AWS DeepLens Model

2. In the navigation pane, choose Models.
3. On the Models page, choose the button next to the model that you want to delete.
4. In the Action list, choose Delete.
5. In the Delete model-name dialog box, choose Delete to confirm that you want to delete the model.

Deleting an AWS DeepLens Device Profile

To deleting an unused device profile, you deregister the device. For more information, see Deregistering Your AWS DeepLens Device (p. 62).

Deleting AWS DeepLens Logs from CloudWatch Logs

If you exposed personally identifiable information in the AWS Greengrass user logs that were created in Amazon CloudWatch Logs when you ran your AWS DeepLens project, use the CloudWatch Logs console to delete them.

1. Sign in to the AWS Management Console and open the CloudWatch console at https://console.aws.amazon.com/cloudwatch/.
2. In the navigation pane, choose Logs.
3. On the Log Groups page, choose the button next to the AWS Greengrass user log that you want to delete. The log begins with /aws/greengrass/Lambda/....
4. In the Actions list, choose Delete log group.
5. In the Delete log group dialog box, choose Yes, Delete to confirm that you want to delete the logs.

Note
When the Lambda function for your project runs again, the deleted log group will be recreated.
Logging and Troubleshooting Your AWS DeepLens Project

Topics
- AWS DeepLens Project Logs (p. 64)
- Troubleshooting AWS DeepLens (p. 66)

AWS DeepLens Project Logs

When you create a new project, AWS DeepLens automatically configures AWS Greengrass Logs. AWS Greengrass Logs writes logs to Amazon CloudWatch Logs and to the local file system of your device. When a project is running, AWS DeepLens sends diagnostic messages to CloudWatch Logs as AWS Greengrass log streams and to your AWS DeepLens device as local file system logs. The messages sent to CloudWatch Logs and your local file system logs are identical, except that the crash.log file is available only in file system logs.

Topics
- CloudWatch Logs Log Groups for AWS DeepLens (p. 64)
- File System Logs for AWS DeepLens (p. 65)

CloudWatch Logs Log Groups for AWS DeepLens

CloudWatch Logs log events for AWS DeepLens are organized into the following log groups.

<table>
<thead>
<tr>
<th>Log Group</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>aws/greengrass/GreengrassSystem/certmanager</td>
<td>Diagnostic logs for certificate management</td>
</tr>
<tr>
<td>aws/greengrass/GreengrassSystem/connection_manager</td>
<td>Diagnostic logs for device connection management, including Lambda function configuration</td>
</tr>
<tr>
<td>aws/greengrass/GreengrassSystem/ip_detector</td>
<td>Diagnostic logs for IP address detection</td>
</tr>
<tr>
<td>aws/greengrass/GreengrassSystem/python_runtime</td>
<td>Diagnostic logs related to the Python runtime</td>
</tr>
<tr>
<td>aws/greengrass/GreengrassSystem/router</td>
<td>Diagnostic logs that are created when the results of inference are forwarded to AWS IoT</td>
</tr>
<tr>
<td>aws/greengrass/GreengrassSystem/runtime</td>
<td>Diagnostic logs related to the AWS Greengrass runtime, for example, for (Message Queuing Telemetry Transport (MQTT) event subscription</td>
</tr>
</tbody>
</table>
File System Logs for AWS DeepLens

AWS Greengrass Logs for AWS DeepLens are also stored in the local file system on your AWS DeepLens device. The local file system logs include the crash log, which is not available in CloudWatch Logs.

You can find the local file system logs in the following directories on your AWS DeepLens device.

<table>
<thead>
<tr>
<th>Log Directory</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/opt/awscam/greengrass/gcc/var/log</td>
<td>The top-level directory for the AWS Greengrass logs for AWS DeepLens. The crash logs are in the crash.log file in this directory.</td>
</tr>
<tr>
<td>/opt/awscam/greengrass/gcc/var/log/system</td>
<td>The subdirectory that contains the AWS Greengrass system component logs.</td>
</tr>
<tr>
<td>/opt/awscam/greengrass/gcc/var/log/user</td>
<td>The subdirectory that contains logs for user-defined Lambda inference functions.</td>
</tr>
</tbody>
</table>

For more information about AWS Greengrass file system logs for AWS DeepLens, including access requirements, see File System Logs in the AWS Greengrass Developer Guide.
Troubleshooting AWS DeepLens

Use the following guidelines to troubleshoot issues with AWS DeepLens.

Topics
- Troubleshooting AWS DeepLens Software Issues (p. 66)
- Troubleshooting Device Registration Issues (p. 70)
- Troubleshooting Issues with Model Deployments to the AWS DeepLens Device (p. 74)

Troubleshooting AWS DeepLens Software Issues

The following troubleshooting tips explain how to resolve issues that you might encounter with AWS DeepLens software.

Topics
- Determine Which Version of AWS DeepLens Software You Have (p. 66)
- Check the Latest Version of awscam (p. 67)
- Reactivate Wi-Fi Hotspots After Restarting the Device (p. 67)
- Ensure That a Successfully Deployed Lambda Inference Function Can Be Called (p. 68)
- Update awscam to the Latest Version (p. 68)
- Upgrade awscam Dependencies (p. 69)
- Install Security Updates on the Device (p. 70)
- Handle Unsupported Architecture During a Device Update (p. 70)
- Reset the Device Software to the Factory Settings (p. 70)

Determine Which Version of AWS DeepLens Software You Have

The software on the AWS DeepLens device is a Debian package named awscam. To find the version of awscam that you have, you can use the AWS DeepLens console or the terminal on your AWS DeepLens device,

To use the AWS DeepLens console
1. Sign in to the AWS DeepLens console.
2. In the navigation pane, choose Devices.
3. In the Devices list, choose your registered device.

To use the terminal on your AWS DeepLens device
1. Connect your device to a monitor using a micro-HDMI cable, mouse, and keyboard or connect to the device using an ssh command.
2. In a terminal on the device, run the following command:

```
dpkg -l awscam
```
3. Look for the version number in the version column. For example, in the following example, the installed version of awscam is 1.2.3.

```
Desired=Unknown/Install/Remove/Purge/Hold
```
Check the Latest Version of awscam

You can use the terminal on your AWS DeepLens device to see what the latest version of awscam is.

To see what the latest version of awscam is using your AWS DeepLens device
1. Connect to your AWS DeepLens device.
2. Open a terminal on your device.
3. Run the following commands:

```
sudo apt-get update
```

```
sudo apt-cache policy awscam
```

The following example shows an example output, where 1.2.3 would be the latest version of awscam on your device:

```
awscam | 1.2.3 | https://awsdeep-update.us-east-1.amazonaws.com awscam/main amd64 Packages
```

Reactivate Wi-Fi Hotspots After Restarting the Device

If your device loses Wi-Fi connection and restarting your device doesn't reactivate Wi-Fi hotspots, you might be affected by the Linux kernel upgrade from Ubuntu. To resolve this, roll back the kernel.

To roll back the Linux kernel on your device
1. Open a terminal on your device.
2. Run the following command:

```
ifconfig
```

3. Inspect the output of the command. If the output lists only the localhost network interface, turn off the device and then turn it back on to restart the system.
4. If you still don't have Wi-Fi access, run the following command to check which version of the Linux kernel is installed.

```
uname -r
```

5. If you have the 4.13.x-xx version of the kernel, you are affected by the kernel upgrade from Ubuntu. To resolve this, roll back the kernel using the following commands. Replace x–xx with your version number:

```
sudo apt remove linux-image-4.13.x-xx-generic linux-headers-4.13.x-xx-generic
sudo reboot
```

After the device has restarted, the kernel version should be 4.10.17+ or 4.13.x–xx-deeplens.
If the version of your `awscam` software is 1.2.0 or later, you shouldn't have this issue.

**Ensure That a Successfully Deployed Lambda Inference Function Can Be Called**

If you successfully deployed an inference function to Lambda before January 10, 2018, but the Lambda function doesn't appear to be called to perform inference, and your Linux kernel version is 4.13.0-26, you are affected by the Linux kernel upgrade from Ubuntu. To resolve this, follow the procedure in *Determine Which Version of AWS DeepLens Software You Have (p. 66)* to roll back the Linux kernel to Ubuntu 4.10.17+.

**Update `awscam` to the Latest Version**

If you enabled automatic updates when you set up your device, AWS DeepLens automatically updates `awscam` every time you turn on your device. If you find the software is out-of-date, restart the AWS DeepLens device and wait for a few minutes after the system is running.

The update might get delayed or fail under the following conditions:

- **Scenario 1:** The Ubuntu system is updated every time you restart the device. This blocks concurrent `awscam` automatic updates. To check if an Ubuntu update is running, run the following command from a terminal:

  ```
  sudo lsof /var/lib/dpkg/lock
  ```

  If the output looks similar to the following, a concurrent Ubuntu update is running:

  ```
  COMMAND   PID  USER FD  TYPE DEVICE SIZE/OFF NODE NAME
  unattende 2638 root 4uW REG  179,2  0        21   /var/lib/dpkg/lock
  ```

  In this case, wait for the Ubuntu update to finish. If the installed version of `awscam` is earlier than 1.1.11, restart the system after Ubuntu finishes updating. Otherwise, wait for the `awscam` update to start automatically.

- **Scenario 2:** `awscam` installation can be interrupted by a system restart or other events. If this happens, `awscam` might be only partially configured. Run the following command to check the package version:

  ```
  dpkg -l awscam
  ```

  If the upgrade has not completed successfully, you see output similar to the following:

  ```
  Desired=Unknown/Install/Remove/Purge/Hold
  Status=Not/Inst/Conf-files/Unpacked/half-inst/trig-aWait/Trig-pend
  |/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
  ||/ Name           Version     Architecture   Description
  ++++-==============-===========-==============-===============
  iF  awscam         1.1.10      amd64          awscam
  ```

  A status of iF prevents an `awscam` update. To resolve this issue, run the following command:

  ```
  sudo dpkg --configure -a
  ```

  Verify the status again and restart the device to let the automatic update of `awscam` proceed.

- **Scenario 3:** If you've tried the suggestions for Scenario 1 and 2, and `awscam` still hasn't been updated, manually start the update by running the following commands:
sudo apt-get update
sudo apt-get upgrade awscam

Allow several minutes for the update to finish. If AWS DeepLens reports an error during the upgrade process, see Upgrade awscam Dependencies (p. 69).

**Upgrade awscam Dependencies**

When upgrading `awscam` on your device, you might get an error message about unmet dependencies. An example of such error message is shown in the following output:

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
You might want to run 'apt-get -f install' to correct these.
The following packages have unmet dependencies:
pulseaudio : Depends: libpulse0 (= 1:8.0-0ubuntu3.3) but 1:8.0-0ubuntu3.4 is installed
pulseaudio-module-bluetooth : Depends: pulseaudio (= 1:8.0-0ubuntu3.4)
pulseaudio-module-x11 : Depends: pulseaudio (= 1:8.0-0ubuntu3.4)
E: Unmet dependencies. Try using -f.
```

To resolve such an error, run the following command from a terminal on your device:

```
sudo apt-get -f install
```

If the error persists or you get another error message similar to the following, you might need to force installation:

```
Errors were encountered while processing:
/var/cache/apt/archives/pulseaudio_1%3a8.0-0ubuntu3.4_amd64.deb
E: Sub-process /usr/bin/dpkg returned an error code (1)
```

To force installation, run the following commands:

```
sudo dpkg --configure -a
sudo dpkg -i --force-overwrite /var/cache/apt/archives/pulseaudio_1%3a8.0-0ubuntu3.4_amd64.deb
```

If you have the Linux kernel version 4.13-32, you might need to change the path of the pulse audio package to `/var/cache/apt/archives/pulseaudio_1%3a8.0-0ubuntu3.7_amd64.deb` in the preceding command.

Make sure that the package is installed by inspecting the output of the following command:

```
sudo apt-get -f install
```

If multiple packages show the `Errors were encountered while processing` error message, run the following command on each of them:

```
dpkg -i --force-overwrite
```

After forced installation has finished, run the following command to complete the upgrade:
Install Security Updates on the Device

When using `ssh` to connect to your device, you might see messages like `245 packages can be updated and 145 updates are security updates.`

In general, security updates don't affect the operation of your device. However, if you have security concerns, you can manually apply security updates with the following command:

```bash
sudo apt-get update
sudo apt-get upgrade
```

To update all packages at once, run the following command:

```bash
sudo apt-get dist-upgrade
```

If you get `unmet dependencies` error messages, follow the instruction in Upgrade awscam Dependencies (p. 69) to resolve the errors.

Handle Unsupported Architecture During a Device Update

If you run the `sudo apt-get update` command and get the error `The https://awsdeepupdate.us-east-1.amazonaws.com packages repository is not set to public (access denied)` causing error: "doesn't support architecture i386" error message, you can safely ignore it. The awscam package is supported only for x86_64 architecture.

Reset the Device Software to the Factory Settings

To reset your AWS DeepLens device to its factory settings, follow the AWS DeepLens system restore instructions. After the reset, you may need to update the device (awscam) software to the latest version. You will be asked to update the software when registering the device using the AWS DeepLens console. You should update the software as soon as possible after resetting the device.

Troubleshooting Device Registration Issues

Use the following guidelines to troubleshoot issues with AWS DeepLens device registration.

Topics
- Complete Device Registration When the Wi-Fi Connection Is Down (p. 71)
- Connect Your DeepLens Device to Your Home or Office Wi-Fi Network When the Wi-Fi SSID or Password Contains Special Characters (p. 71)
- Choose an IAM Role When None Is Available (p. 72)
- Fix an Incomplete Certificate (.zip) File (p. 72)
- Fix an Incorrectly Uploaded Certificate File (p. 72)
- Resolve Exceeded Maximum User Limits for Devices, Projects, or Models (p. 72)
- Fix Failed Deregistration of the AWS DeepLens Device (p. 72)
- Resolve Failed Registration of the AWS DeepLens Device (p. 72)
- Launch the Device Setup App When the Device's Local Address (192.168.0.1) Is Not Accessible (p. 73)
- See the Device's Wi-Fi Network on a Laptop When the Device Indicates It Is On or Off (p. 73)
Complete Device Registration When the Wi-Fi Connection Is Down

To connect to your AWS DeepLens device from your laptop with Wi-Fi, enable the soft access (softAP) mode on the device. If you've registered the device, the softAP mode is disabled. Re-enable it by inserting a paper clip into the pin hole on the back of the device. After you feel a click, wait for about 10 seconds. When the middle LED light on the device blinks, the device is back in softAP mode. You can then connect to your device through Wi-Fi and complete the device registration.

Connect Your DeepLens Device to Your Home or Office Wi-Fi Network When the Wi-Fi SSID or Password Contains Special Characters

If your home or office network Wi-Fi name (SSID) or password contains special characters (including single quotes, backslashes, or white spaces), you might not be able to connect your AWS DeepLens device to the Wi-Fi network after choosing Complete the setup on the Setup guide page in the AWS DeepLens console. (See Step 1 of the To set up your device procedure (p. 15).) You'll be blocked in Step 1: Connect to network on the Device setup page and, hence, cannot complete the device registration.

To work around this, use SSH commands in a terminal to connect the device to your home or office Wi-Fi network.

1. Connect to your device's Wi-Fi network. For instructions, see the section called “Connect to Your Device” (p. 13).

   **Note**
   If the middle LED light on your device is not flashing, use a pin to press the REST button on the back of the device until you hear a click. The front LED light will start to flash in about 10 seconds. If the LED light doesn't flash, reboot the device.

2. After the Wi-Fi connection is established, open a web browser on your working computer and type http://deeplens.amazon.net/#deviceAccess in the address bar.

3. In the Configure device access page, choose Enable under SSH server. Create a device password if you haven't already done so. Choose Save.

4. Instead of choosing Finish on the Device setup page, which closes the device (AMDC-NNNN) network Wi-Fi connection, open a terminal on your working computer and run the following SSH command to log in to your device:

```
ssh aws_cam@deeplens.amazon.net
```

Type the device password.

**Note**
On a computer running Windows, you can use PuTTY or a similar SSH client.

5. After you have logged in, run the following shell command in the terminal:

```
sudo nmcli device wifi con '<wifi_ssid>' password '<wifi_password>' ifname mlan0
```

Replace `<wifi_ssid>` and `<wifi_password>` with the SSID and the password for the Wi-Fi connection, respectively.
You must enclose the SSID and password in single quotes. If the SSID or password contains a single quote, escape each instance by prefixing a back slash (\) and enclosing them within a pair of single quotes. For example, if a single quote appears in the original `<wifi_ssid>` as Jane’s Home Network, the escaped version is Jane ‘s Home Network.

After the command has run, verify that your device is connected to the internet by pinging any public website.

6. Complete device registration by following the instructions at the section called “Set Up Your Device” (p. 15).

When prompted, update the device software to version 1.3.10 or higher. The update ensures that your AWS DeepLens device can connect to your home or office Wi-Fi network from the AWS DeepLens console so you won’t have to redo this workaround.

Choose an IAM Role When None Is Available

You must create the required IAM roles before choosing one. To create an IAM role, follow the instructions in Step 4 in Register Your AWS DeepLens Device (p. 8).

Fix an Incomplete Certificate (.zip) File

If you download a certificate .zip file and forget to choose or tap the Finish button, you get an incomplete and, therefore, unusable certificate file. To fix this, delete the downloaded zip file and reregister your device.

Fix an Incorrectly Uploaded Certificate File

Upload the correct certificate (.zip) file to overwrite the incorrect one.

Resolve Exceeded Maximum User Limits for Devices, Projects, or Models

Deregister the device, delete a project, or delete a model before adding new one.

Fix Failed Deregistration of the AWS DeepLens Device

Stop deregistration and reregister the device on the same camera.

To deregister and reregister an AWS DeepLens device

1. Reset the camera to softAP mode by inserting a small paper clip into the reset pinhole at the back of the device.
2. Reregister the device with a new device name.
3. Upload the new certificate to replace the old one.

Resolve Failed Registration of the AWS DeepLens Device

Device registration often fails because you have incorrect or insufficient permissions. Make sure that the correct IAM roles and permissions are set for the AWS services you use.

For example, when setting the permissions in the AWS DeepLens console, you must already have created an IAM role with the required permissions for AWS Greengrass and associated this role with the IAM role for AWS Greengrass option. If registration fails, make sure that you've created and specified the correct roles, then reregister the device.
Specifying an existing Lambda role for the **IAM role for AWS Lambda** can also cause registration to fail if the role name doesn't start with `AWSDeepLens`. Specifying a role whose name doesn't begin with `AWSDeepLens` doesn't affect the deployment of Lambda functions.

To set up the roles correctly, follow the instructions in Register Your AWS DeepLens Device (p. 8) and check with the AWS DeepLens console tips for each role.

**Launch the Device Setup App When the Device's Local Address (192.168.0.1) Is Not Accessible**

If your AWS DeepLens device is running software (`awscam`) version 1.3.6 or later, open the device setup app by navigating to `http://deeplens.config` in a browser window on your working computer. The working computer must be connected to the device's (`AMDC-nnnn`) Wi-Fi network (p. 13).

If your device is running an earlier version of the `awscam` package, update to the latest version by running the following commands from a terminal on the device:

```
sudo apt update
sudo apt install awscam
```

If you don't want to update the `awscam` package to the latest version yet, navigate to `http://192.168.0.1` to launch the device setup app. If your home or office Wi-Fi network router also uses 192.168.0.1 as the default IP address, reset the router's IP address, for example, to 192.168.1.0. For instructions on resetting your router's IP address, see How do I change the IP Address of my router?

If your device's Wi-Fi network is unresponsive, reset the device and connect to it again. To reset the device, press a pin on the **RESET** button on the back of the device, and wait until the Wi-Fi indicator (on the front of the device) blinks.

**See the Device's Wi-Fi Network on a Laptop When the Device Indicates It Is On or Off**

When you can't see your device's Wi-Fi network (also known as the softAP network) on your laptop even though the middle LED light on the device is blinking or if you still see the softAP network on the laptop after the middle LED light stops blinking, refresh the available Wi-Fi networks on your laptop. Usually, this involves turning off the laptop Wi-Fi and turning it back on.

Rebooting a device takes up to 2 minutes after the softAP network is ready to accept connection requests.

**Fix an Unsuccessful Device Wi-Fi Reset**

If the device's middle LED light isn't blinking after you reset the device by inserting a pin into the reset hole on the back of it, connect the device to the internet with an Ethernet cable, open a terminal window on the device, and run the following commands to reset the `awscam` package:

```
sudo systemctl status softap.service
```

If the command returns error code 203, reinstall `awscam-webserver` by running the following commands on your AWS DeepLens device:

```
sudo apt-get install --reinstall awscam-webserver
```
The successful reboot sets up the device's Wi-Fi network, and the middle LED light on the device will start to blink.

**Resolve an Access Denied Error Encountered While Downloading a Model After Device Registration Went through without Errors**

When setting permissions during device registration, make sure that you have the IAM role for AWS Greengrass created and the role is associated with the IAM group role for AWS Greengrass option on the console when setting permissions during device registration. If you didn't, reregister the device with the correct roles.

**Troubleshooting Issues with Model Deployments to the AWS DeepLens Device**

Use the following information to troubleshoot issues that occur when deploying models to an AWS DeepLens device.

**Topics**

- Resolve the ModelDownloadFailed Error (p. 74)
- Resolve Model Optimization Failure Reported As std::bad_alloc() (p. 74)
- Resolve Model Optimization Failure Caused by a Missing stride Attribute (p. 74)
- Resolve Model Optimization Failure When the List Object Is Missing the shape Attribute (p. 75)
- Ensure Reasonable Inference for a Model (p. 76)
- Determine Why AWS DeepLens Classifies Data Incorrectly When the Model Performs Well on the Validation Set (p. 76)

**Resolve the ModelDownloadFailed Error**

When using the AWS DeepLens console, you must provide two IAM roles for AWS Greengrass: IAM Role for AWS Greengrass and IAM group role for AWS Greengrass. If you specify the same IAM role for both, you get this error. To fix it, specify AWSDeepLensGreengrassRole for the IAM Role for AWS Greengrass and AWSDeepLensGreengrassGroupRole for the IAM group role for AWS Greengrass.

**Resolve Model Optimization Failure Reported As std::bad_alloc()**

This occurs because the version of MXNet on the device doesn't match the version used to train the model. Upgrade the version of MXNet on the device by running the following command from a terminal on the device:

```
sudo pip3 install mxnet==1.0.0
```

We assume that the version of MXNet used for training is 1.0.0. If you use a different version for training, change the version number accordingly.

**Resolve Model Optimization Failure Caused by a Missing stride Attribute**

If you haven't specified the stride argument in the hyper parameter list for pooling layers, the model optimizer fails and reports the following error message:
AttributeError: 'MxNetPoolingLayer' object has no attribute 'stride_x'

This occurs because the `stride` argument is removed from the latest MXNet pooling layers' hyper parameter list.

To fix the problem, add "stride": "(1, 1)" to the symbol file. In a text editor, edit your symbol file so that the pooling layer looks like this:

```json
{
    "op": "Pooling",
    "name": "pool1",
    "attr": {
        "global_pool": "True",
        "kernel": "(7, 7)",
        "pool_type": "avg",
        "stride": "(1, 1)"
    },
    "inputs": <your input shape>
}
```

After the `stride` property is added to all the pooling layers, you should no longer get the error and everything should work as expected. For more information, see this forum post.

### Resolve Model Optimization Failure When the List Object Is Missing the shape Attribute

You receive the following error message during model optimization when the layer names for non-null operators have suffixes:

File "/opt/intel/deeplearning_deploymenttoolkit_2017.1.0.5852/deployment_tools/model_optimizer/model_optimizer_mxnet/mxnet_convertor/mxnet_layer_utils.py", line 75, in transform_convolution_layer
res = np.ndarray(l.weights.shape)
AttributeError: 'list' object has no attribute 'shape'

In a text editor, open your model's symbol file (<model_name>-symbol.json). Remove suffixes from all of the layer names for non-null operators. The following is an example of how a non-null operator is defined:

```json
{
    "op": "Convolution",
    "name": "deep_dog_conv0_fwd",
    "attr": {
        "dilate": "(1, 1)",
        "kernel": "(3, 3)",
        "layout": "NCHW",
        "no_bias": "False",
        "num_filter": "64",
        "num_group": "1",
        "pad": "(0, 0)",
        "stride": "(2, 2)"
    }
}
```

Change "name": "deep_dog_conv0_fwd" to "name": "deep_dog_conv0".
For a softmax layer, if the "attr": "{}" property is missing, add it to the JSON file. For example:

```
{
    "op": "SoftmaxOutput",
    "name": "softmax",
    "attr": {},
    "inputs": [[192,0,0]]
}
```

Finally, make sure that you use only the supported layers in your model. For a list of supported MXNet layers, see MXNet Models (p. 21).

**Ensure Reasonable Inference for a Model**

To ensure reasonable inference based on a given model, the height and width of the model optimizer must match the height and width of images in the training set. For example, if you trained your model using images that are 512 pixels X 512 pixels, you must set your model optimizer's height and width to 512 X 512. Otherwise, the inference won't make sense.

**Determine Why AWS DeepLens Classifies Data Incorrectly When the Model Performs Well on the Validation Set**

AWS DeepLens classifies data incorrectly even if the model performs well on a validation set when your training data wasn't normalized. Retrain the model with normalized training data.

If the training data has been normalized, also normalize the input data before passing it to the inference engine.
AWS DeepLens Device Library

The AWS DeepLens device library consists of a set of Python modules that provide objects and methods for various device operations:

- The `awscam` module for running inference code based on a project's model.
- The `mo` module for converting your Caffe, Apache MXNet or TensorFlow deep learning model artifacts into AWS DeepLens model artifacts and performing necessary optimization.
- The `DeepLens_Kinesis_Video` module for integrating with Kinesis Video Streams to manage streaming from the AWS DeepLens device to a Kinesis Video Streams stream.

**Topics**
- `awscam` Module for Inference (p. 77)
- Model Optimization (mo) Module (p. 82)
- `DeepLens_Kinesis_Video` Module for Amazon Kinesis Video Streams Integration (p. 91)

`awscam` Module for Inference

**Topics**
- `awscam.getLastFrame` Function (p. 77)
- Model Object (p. 77)

**awscam.getLastFrame Function**

Retrieves the latest frame from the video stream. The video streaming runs constantly when the AWS DeepLens is running.

**Request Syntax**

```python
import awscam
ret, video_frame = awscam.getLastFrame()
```

**Parameters**

- None

**Return Type**

- `ret`—A Boolean value (true or false) that indicates whether the call was successful.
- `video_frame`—A `numpy.ndarray` that represents a video frame.

**Model Object**

```python
class awscam.Model
```

Represents an AWS DeepLens machine learning model.
import awscam
model = awscam.Model(model_topology_file, loading_config)

Methods
- Model Class Constructor (p. 79)
- model.doInference Method (p. 80)
- model.parseResult Method (p. 81)
Model Class Constructor

The constructor for a `awscam.Model`

Request Syntax

```python
import awscam
model = awscam.Model(model_topology_file, loading_config)
```

Parameters

- **model_topology_file**—Required. A neural network topology file (.xml) from the Intel model optimizer.
  
  Models supported: Classification and Single Shot MultiBox Detector (SSD)

- **loading_config** (dict)—Required. Specifies whether the model should be loaded into the GPU or CPU. The format of this parameter is a dictionary.

  **Permitted values:**
  
  - `{ "GPU": 1 }`—Loads the model into the GPU
  - `{ "GPU": 0 }`—Loads the model into the CPU
model.doInference Method

Runs inference on a video frame (image file) by applying the loaded model. The method returns the result of the inference.

Request Syntax

```python
import awscam
model = awscam.Model(model_topology_file, loading_config)
result = model.doInference(video_frame)
```

Parameters

- **video_frame**—Required. The model runs its inference on a video frame (image file) and returns the result of the model inference, which is a dictionary.

Return Type

- **dict list**

Returns

Returns a dict with a single key-value pair. The key is the name of the model's output layer, which is defined by the model you use. The value is a list of dicts in which each element is an object that the model identified and its associated probability. The user who applies the model should know how to parse the result.

Example

Sample output:

```json
{
   'SoftMax_67': array(
      [
         2.41881448e-08,
         3.57339691e-09,
         1.00263861e-07,
         5.40415579e-09,
         4.37702547e-04,
         6.16787545e-08
      ],
      dtype=float32)
}
```
**Model Object**

**model.parseResult Method**

Parses the results of some commonly used models, such as classification, SSD, and segmentation models. For customized models, you need to write your own parse functions.

**Request Syntax**

```python
import awscam
model = awscam.Model(model_topology_file, loading_config)
raw_infer_result = model.doInference(video_frame)
result = model.parseResult(model_type, raw_infer_result)
```

**Parameters**

- **model_type**—String that identifies the model type to use to generate the inference. Required.
  - Valid values: classification, ssd, and segmentation

- **raw_infer_result**—The output of the function `model.doInference(video_frame)`. Required.

**Return Type**

- **dict**

**Returns**

Returns a dict with a single key-value pair. The key is the model type. The value is a list of dicts, in which each element is an object label and probability calculated by the model.

**Example**

The output of a classification model might look like the following:

```json
{
  "output": [
    {"label": "318", "prob": 0.5},
    {"label": "277", "prob": 0.3},
    ...
    {"label": "433", "prob": 0.001}  
  ]
}
```

The output of an SDD model contains bounding box information, similar to the following:

```json
{
  "output": [
    {"label": "318", "xmin": 124, "xmax": 245, "ymin": 10, "ymax": 142, "prob": 0.5},
    {"label": "277", "xmin": 89, "xmax": 166, "ymin": 233, "ymax": 376, "prob": 0.3},
    ...
    {"label": "433", "xmin": 355, "xmax": 468, "ymin": 210, "ymax": 266, "prob": 0.001}  
  ]
}
```
Model Optimization (mo) Module

The `mo` Python module is a AWS DeepLens model optimization library that you can use to convert your Caffe, Apache MXNet, or TensorFlow (p. 21) model artifacts to AWS DeepLens model artifacts and perform necessary optimization.

To optimize a model, call the `mo.optimize` function and specify the appropriate input parameters.

**Topics**
- `mo.optimize` Function (p. 82)
- Troubleshooting the Model Optimizer (p. 85)

**mo.optimize Function**

Converts AWS DeepLens model artifacts from a Caffe, MXNet, or TensorFlow representation to an AWS DeepLens representation and performs necessary optimization.

**Syntax**

```python
import mo
res = mo.optimize(model_name, input_width, input_height, platform, aux_inputs)
```

**Request Parameters**

- `model_name`: The name of the model to optimize.
  
  Type: string
  
  Required: Yes

- `input_width`: The width of the input image in pixels. The value must be a non-negative integer less than or equal to 1024.

  Type: integer.
  
  Required: Yes

- `input_height`: The height of the input image in pixels. The value must be a non-negative integer less than or equal to 1024.

  Type: integer.
  
  Required: Yes

- `platform`: The source platform for the optimization. For valid values, see the following table.

  Type: string
  
  Required: No

**Valid platform Values:**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caffe or caffe</td>
<td>The optimization converts Caffe model artifacts to AWS DeepLens model artifacts.</td>
</tr>
</tbody>
</table>
The optimize function converts Apache MXNet model artifacts to AWS DeepLens model artifacts. This is the default option.

TensorFlow or tensorflow or tf

The optimization converts TensorFlow model artifacts to AWS DeepLens model artifacts.

- aux_inputs: A Python dictionary object that contains auxiliary inputs, including entries common to all platforms and entries specific to individual platforms.

  Type: Dict

  Required: No

**Valid aux_inputs dictionary Entries**

<table>
<thead>
<tr>
<th>Item Name</th>
<th>Applicable Platforms</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--img-format</td>
<td>All</td>
<td>Image format. The default value is BGR.</td>
</tr>
<tr>
<td>--img-channels</td>
<td>All</td>
<td>Number of image channels. The default value is 3.</td>
</tr>
<tr>
<td>--precision</td>
<td>All</td>
<td>Image data type. The default value is FP16.</td>
</tr>
<tr>
<td>--fuse</td>
<td>All</td>
<td>A switch to turn on (ON) or off (OFF) fusing of linear operations to convolution. The default value is ON.</td>
</tr>
<tr>
<td>--models-dir</td>
<td>All</td>
<td>Model directory. The default directory is /opt/awscam/artifacts.</td>
</tr>
<tr>
<td>--output-dir</td>
<td>All</td>
<td>Output directory. The default directory is /opt/awscam/artifacts.</td>
</tr>
<tr>
<td>--input_proto</td>
<td>Caffe</td>
<td>The prototxt file path. The default value is an empty string (&quot;&quot;&quot;).</td>
</tr>
<tr>
<td>--epoch</td>
<td>MXNet</td>
<td>Epoch number. The default value is 0.</td>
</tr>
<tr>
<td>--input_model_is_text</td>
<td>TensorFlow</td>
<td>A Boolean flag that indicates whether the input model file is in text protobuf format (True) or not (False). The default value is False.</td>
</tr>
</tbody>
</table>

**Returns**

The optimize function returns a result that contains the following:
• **model_path**: Path of the optimized model artifacts when they are successfully returned.

  Type: string

• **status**: Operational status of the function. If a function fails, for possible cause of failures and corrective actions, see the status table below.

  Type: integer

<table>
<thead>
<tr>
<th>status</th>
<th>Cause</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Model optimization succeeded.</td>
<td>No action needed.</td>
</tr>
<tr>
<td>1</td>
<td>Model optimization failed because the requested platform is not supported.</td>
<td>• Choose a supported platform.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Make sure that the platform name is spelled correctly.</td>
</tr>
<tr>
<td>2</td>
<td>Model optimization failed because you are using inconsistent platform versions.</td>
<td>• Make sure that you are running the latest version of the platform. To check your version, at a command prompt, run <code>pip install mxnet</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Make sure that there are no unsupported layers in the model for the target platform.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Make sure that your awscam software is up-to-date.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• See <a href="#">Troubleshooting the Model Optimizer</a> for recommended actions for error messages reported in the CloudWatch Logs for AWS DeepLens and on your AWS DeepLens device.</td>
</tr>
</tbody>
</table>

To load the optimized model for inference, call the `awscam.Model` API and specify the `model_path` returned from this function.
Troubleshooting the Model Optimizer

In this section, you'll find a list of frequently asked questions and answers about errors reported by the model optimizer. The AWS DeepLens model optimizer depends on the Intel Computer Vision SDK, which is installed with the AWS DeepLens software. For errors not covered here, see Intel Computer Vision SDK Support and ask a question on the forum.

Topics

- How to handle the "Current caffe.proto does not contain field." error? (p. 86)
- How to create a bare Caffe model with only prototxt? (p. 86)
- How to handle the "Unable to create ports for node with id." error? (p. 86)
- How to Handle "Invalid proto file: there is neither 'layer' nor 'layers' top-level messages." error? (p. 86)
- How to handle the "Old-style inputs (via 'input_dims') are not supported. Please specify inputs via 'input_shape'" error? (p. 86)
- How to handle the "Invalid prototxt file: value error." error? (p. 88)
- How to handle the "Error happened while constructing caffe.Net in the Caffe fallback function." error? (p. 88)
- How to handle the "Cannot infer shapes due to exception in Caffe." error? (p. 88)
- How to handle the "Cannot infer shape for node {} because there is no Caffe available. Please register python infer function for op or use Caffe for shape inference." error? (p. 88)
- How to handle the "Input shape is required to convert MXNet model. Please provide it with --input_width and --input_height" error? (p. 88)
- How to handle the "Cannot find prototxt file: for Caffe please specify --input_proto - a protobuf file that stores topology and --input_model that stores pretrained weights." error? (p. 89)
- How to handle the "Specified input model does not exist." error? (p. 89)
- How to handle the "Failed to create directory ... Permission denied!" error? (p. 89)
- How to handle the "Discovered data node without inputs and value." error? (p. 89)
- How to handle the "Placeholder node doesn't have input port, but input port was provided." error? (p. 89)
- How to handle the "No or multiple placeholders in the model, but only one shape is provided, cannot set it." error? (p. 89)
- How to handle the "Cannot write an event file for the tensorboard to directory." error? (p. 89)
- How to handle the "Stopped shape/value propagation at node." error? (p. 89)
- How to handle the "Module tensorflow was not found. Please install tensorflow 1.2 or higher." error? (p. 90)
- How to handle the "Cannot read the model file: it is incorrect TensorFlow model file or missing." error? (p. 90)
- How to handle the "Cannot pre-process TensorFlow graph after reading from model file. File is corrupt or has unsupported format." error? (p. 90)
- How to handle the "Input model name is not in an expected format, cannot extract iteration number." error? (p. 90)
- How to handle the "Cannot convert type of placeholder because not all of its outputs are 'Cast' to float operations." error? (p. 90)
- How to handle the "Data type is unsupported." error? (p. 90)
- How to handle the "No node with name ..." error? (p. 90)
- How to handle the "Module mxnet was not found. Please install mxnet 1.0.0" error? (p. 90)
- How to handle the "The following error happened while loading mxnet model ..." error? (p. 90)
• How to handle the "... elements of ... were clipped to infinity while converting a blob for node [...] to ..." error? (p. 91)
• How to handle the "... elements of ... were clipped to zero while converting a blob for node [...] to ..." error? (p. 91)
• How to handle the "The topology contains no "input" layers." error? (p. 91)

How to handle the "Current caffe.proto does not contain field." error?

Your model has custom layers that are not supported by the model optimizer.

How to create a bare Caffe model with only prototxt?

To create a bare Caffe model with only prototxt, import the Caffe module and run the following:

```python3
import caffe
net = caffe.Net('PATH_TO_PROTOTXT/my_net.prototxt', caffe.TEST)
net.save('PATH_TO_PROTOTXT/my_net.caffemodel')
```

How to handle the "Unable to create ports for node with id." error?

Your model has custom layers that aren't supported by the model optimizer. Remove the custom layers.

How to Handle "Invalid proto file: there is neither 'layer' nor 'layers' top-level messages." error?

The structure of a Caffe topology is described in the caffe.proto file. For example, in the model optimizer, you might find the following default proto file: /opt/awscam/intel/deeplearning_deploymenttoolkit/deployment_tools/model_optimizer/mo/front/caffe/proto/my_caffe.proto. This file contains the structure, for example:

```protobuf
message NetParameter {
  // ... some other parameters

  // The layers that make up the net. Each of their configurations, including
  // connectivity and behavior, is specified as a LayerParameter.
  repeated LayerParameter layer = 100;  // Use ID 100 so layers are printed last.

  // DEPRECATED: use 'layer' instead.
  repeated V1LayerParameter layers = 2;
}
```

This means that any topology should contain layers as top-level structures in prototxt. For more information, see LeNet topology.

How to handle the "Old-style inputs (via 'input_dims') are not supported. Please specify inputs via 'input_shape'." error?

The structure of a Caffe topology is described in the caffe.proto file. For example, in the model optimizer, you can find the following default proto file: /opt/awscam/intel/
deeplearning_deploymenttoolkit/deployment_tools/model_optimizer/mo/front/caffe/proto/my_caffe.proto. This file contains the structure, for example:

```proto
message NetParameter {
  optional string name = 1; // Consider giving the network a name.
  // DEPRECATED. See InputParameter. The input blobs to the network.
  repeated string input = 3;
  // DEPRECATED. See InputParameter. The shape of the input blobs.
  repeated BlobShape input_shape = 8;

  // 4D input dimensions are deprecated. Use "input_shape" instead.
  // If specified, for each input blob, there should be four
  // values specifying the num, channels, and height and width of the input blob.
  // Therefore, there should be a total of (4 * #input) numbers.
  repeated int32 input_dim = 4;
  // ... other parameters
}
```

Specify the input layer of the provided model using one of the following formats:

1. Input layer format 1:

   ```proto
   input: "data"
   input_shape
   {
    dim: 1
    dim: 3
    dim: 227
    dim: 227
   }
   ```

2. Input layer format 2:

   ```proto
   input: "data"
   input_shape
   {
    dim: 1
    dim: 3
    dim: 600
    dim: 1000
   }
   input: "im_info"
   input_shape
   {
    dim: 1
    dim: 3
   }
   ```

3. Input layer format 3:

   ```proto
   layer
   {
    name: "data"
    type: "Input"
    top: "data"
    input_param {shape: {dim: 1 dim: 3 dim: 600 dim: 1000}}
   }
   layer
   {
    name: "im_info"
    type: "Input"
   }
   ```
4. Input layer format 4:

```
  input: "data"
  input_dim: 1
  input_dim: 3
  input_dim: 500
```

However, if your model contains more than one input layer, the model optimizer can convert the model with an input layer of format 1, 2, and 3. You can't use format 4 in any multi-input topology.

**How to handle the "Invalid prototxt file: value error." error?**

See the section called "How to Handle "Invalid proto file: there is neither 'layer' nor 'layers' top-level messages." error? " (p. 86) and the section called "How to handle the "Cannot find prototxt file: for Caffe please specify --input_proto - a protobuf file that stores topology and --input_model that stores pretrained weights." error? " (p. 89).

**How to handle the "Error happened while constructing caffe.Net in the Caffe fallback function." error?**

The model optimizer can't use the Caffe module to construct a net, while it is trying to infer a specified layer in the Caffe framework. Make sure that your caffemodel and prototxt files are correct. To make sure that the problem is not in the prototxt file, see the section called “How to create a bare Caffe model with only prototxt?” (p. 86).

**How to handle the "Cannot infer shapes due to exception in Caffe." error?**

Your model has custom layers that are not supported by the model optimizer. Remove any custom layers.

**How to handle the "Cannot infer shape for node {} because there is no Caffe available. Please register python infer function for op or use Caffe for shape inference." error?**

Your model has custom layers that are not supported by the model optimizer. Remove any custom layers.

**How to handle the "Input shape is required to convert MXNet model. Please provide it with --input_width and --input_height." error?**

To convert a MXNet model to an IR, you must specify the input shape, because MXNet models do not contain information about input shapes. Use the --input_shape flag to specify the input_width and input_height.
How to handle the "Cannot find prototxt file: for Caffe please specify --input_proto - a protobuf file that stores topology and --input_model that stores pretrained weights." error?

The model optimizer can't find a prototxt file for a specified model. By default, the prototxt file must be located in the same directory as the input model with the same name (except for the extension). If any of these conditions is not satisfied, use --input_proto to specify the path to the prototxt file.

How to handle the "Specified input model does not exist." error?

Most likely, you have specified an incorrect path to the model. Make sure that the path is correct and the file exists.

How to handle the "Failed to create directory ... Permission denied!" error?

The model optimizer can't create a directory specified as the --output_dir parameter. Make sure that you have permissions to create the specified directory.

How to handle the "Discovered data node without inputs and value." error?

One of the layers in the specified topology might not have inputs or values. Make sure that the provided caffemodel and protobuf files contain all required inputs and values.

How to handle the "Placeholder node doesn't have input port, but input port was provided." error?

You might have specified an input node for a placeholder node, but the model does not contain the input node. Make sure that the model has the input or use a model-supported input node.

How to handle the "No or multiple placeholders in the model, but only one shape is provided, cannot set it." error?

You might have provided only one shape for the placeholder node for a model that contains no inputs or multiple inputs. Make sure that you have provided the correct data for placeholder nodes.

How to handle the "Cannot write an event file for the tensorboard to directory." error?

The model optimizer failed to write an event file in the specified directory because that directory doesn't exist or you don't have permissions to write to it. Make sure that the directory exists and you have the required write permission.

How to handle the "Stopped shape/value propagation at node." error?

The model optimizer can't infer shapes or values for the specified node because: the inference function has a bug, the input nodes have incorrect values or shapes, or the input shapes are invalid. Verify that the inference function, input values, or shapes are correct.
How to handle the "Module tensorflow was not found. Please install tensorflow 1.2 or higher." error?

To use the model optimizer to convert TensorFlow models, install TensorFlow 1.2 or higher.

How to handle the "Cannot read the model file: it is incorrect TensorFlow model file or missing." error?

The model file should contain a TensorFlow graph snapshot of the binary format. Ensure that --input_model_is_text is provided for the model in the text format, by specifying the --input_model_is_text parameter as false. By default, a model is interpreted a binary file.

How to handle the "Cannot pre-process TensorFlow graph after reading from model file. File is corrupt or has unsupported format." error?

Make sure that you've specified the correct model file, the file has the correct format, and the content of the file isn't corrupted.

How to handle the "Input model name is not in an expected format, cannot extract iteration number." error?

MXNet model files must be in .json or .para format. Use the correct format.

How to handle the "Cannot convert type of placeholder because not all of its outputs are 'Cast' to float operations." error?

Use the FP32 data type for the model's placeholder node. Don't use UINT8 as the input data type for the placeholder node.

How to handle the "Data type is unsupported." error?

The model optimizer can't convert the model to the specified data type. Set the data type with the --precision parameter with FP16, FP32, half, or float values.

How to handle the "No node with name ..." error?

The model optimizer can't access the node that doesn't exist. Make sure to specify the correct placeholder, input node name, or output node name.

How to handle the "Module mxnet was not found. Please install mxnet 1.0.0" error?

To ensure that the model optimizer can convert MXNet models, install MXNet 1.0.0 or higher.

How to handle the "The following error happened while loading mxnet model ..." error?

Make sure that the specified path is correct, that the model exists and is not corrupted, and that you have sufficient permissions to work with the model.
How to handle the "... elements of ... were clipped to infinity while converting a blob for node [...] to ..." error?

When you use the --precision=FP16 command line option, the model optimizer converts all of the blobs in the node to the FP16 type. If a value in a blob is out of the range of valid FP16 values, the model optimizer converts the value to positive or negative infinity. Depending on the model, this can produce incorrect inference results or it might not cause any problems. Inspect the number of these elements and the total number of elements in the used blob that is printed out with the name of the node.

How to handle the "... elements of ... were clipped to zero while converting a blob for node [...] to ..." error?

When you use the --precision=FP16 command line option, the model optimizer converts all of the blobs in the node to the FP16 type. If a value in the blob is so close to zero that it can't be represented as a valid FP16 value, the model optimizer converts it to true zero. Depending on the model, this might produce incorrect inference results or it might not cause any problem. Inspect the number of these elements and the total number of elements in the used blob that is printed out with the name of the node.

How to handle the "The topology contains no "input" layers." error?

The prototxt file for your Caffe topology might be intended for training when the model optimizer expects a deploy-ready prototxt file. To prepare a deploy-ready prototxt file, follow the instructions provided on GitHub. Typically, preparing a deploy-ready topology involves removing data layers, adding input layers, and removing loss layers.

DeepLens_Kinesis_Video Module for Amazon Kinesis Video Streams Integration

The Amazon Kinesis Video Streams for AWS DeepLens Video library is a Python module that encapsulates the Kinesis Video Streams Producer SDK for AWS DeepLens devices. Use this module to send video feeds from an AWS DeepLens device to Kinesis Video Streams, and to control when to start and stop video streaming from the device. This is useful if you need to train your own deep-learning model. In this case, you can send video feeds of given time intervals from your AWS DeepLens device to Kinesis Video Streams and use the data as an input for the training.

The module, DeepLens_Kinesis_Video, is already installed on your AWS DeepLens device. You can call this module in a Lambda function that is deployed to your AWS DeepLens device as part of a AWS DeepLens project.

The following Python example shows how to import the DeepLens_Kinesis_Video module and call its objects and methods.

```python
import DeepLens_Kinesis_Video as dkv
import time

aws_access_key = "Your IAM access key"
aws_secret_key = "Your IAM secret key"
region = "us-east-1"
stream_name = "Your stream name, for example, deeplens-kvs"
retention = 1 # in minutes.
```
wait_time_sec = 60*2 # The number of seconds to stream the data
producer = dkv.createProducer(aws_access_key, aws_secret_key, "", region) # No session token is needed.
my_stream = producer.createStream(stream_name, retention)
my_stream.start()
time.sleep(wait_time_sec)
my_stream.stop()

Topics
- DeepLens_Kinesis_Video.createProducer Function (p. 92)
- Producer Object (p. 93)
- Stream Object (p. 93)

DeepLens_Kinesis_Video.createProducer Function

Creates an instance of the Kinesis Video Streams Producer SDK client object for AWS DeepLens. Use the instance to connect your AWS DeepLens device to the AWS Cloud and to manage video streams from the device to Kinesis Video Streams.

Syntax

```python
import DeepLens_Kinesis_Video as dkv
producer = dkv.createProducer(aws_access_key, aws_secret_key, session_token, aws_region)
```

Parameters

- **aws_access_key**
  Type: "string"
  Required: Yes
  The IAM access key of your AWS account.

- **aws_secret_key**
  Type: "string"
  Required: Yes
  The IAM secret key of your AWS account.

- **session_token**
  Type: "string"
  Required: No
  A session token, if any. An empty value means an unspecified token.

- **aws_region**
  Type: "string"
  Required: Yes
  The AWS Region to stream data to. For AWS DeepLens applications, the only valid value is us-east-1.

Returns
• A **Producer object.** *(p. 93)*

**Example**

```python
import DeepLens_Kinesis_Video as dkv
producer = dkv.createProducer("ABCDEF...LMNO23PU", "abcDEFGHi...bc8defGxhiJ", ",", "us-east-1")
```

**Producer Object**

An object that represents the Kinesis Video Streams Producer SDK client that is used to create streams to which to send video feeds from the AWS DeepLens device.

**Topics**

• **Producer.createStream Method** *(p. 93)*

**Producer.createStream Method**

Creates a stream object to send video feeds from the AWS DeepLens device to.

**Syntax**

```python
stream = producer (p. 93).createStream(stream_name, retention)
```

**Parameters**

• **stream_name**

  Type: "string"

  Required: Yes

  The name of a Kinesis stream that your AWS DeepLens device sends video feeds to. If the stream doesn't exist, Kinesis Video Streams creates it. Otherwise, it uses the existing stream to receive the video feeds. You can have multiple streams as long as they have unique names and handles.

• **retention**

  Type: int

  Required: Yes

  The time, in hours, to retain the streamed data. To view the data in the **Stream Preview** on the Kinesis Video Streams console, set this parameter to greater than or equal to 2 hours.

**Returns**

• A **Stream** *(p. 93)* object.

**Stream Object**

An object that encapsulates a Kinesis stream for video feeds from your AWS DeepLens device and exposes methods for controlling when to start and stop streaming data.
Stream.isActive Property

Returns a boolean flag to indicate whether the producer is sending data to this stream.

Syntax

```
stream (p. 93).isActive
```

Parameters
- None

Returns
- Type: Boolean
  - Value: true if the stream is active. Otherwise, false.

Stream.start Method

Notifies the producer to start sending video feeds from your AWS DeepLens device to the Kinesis stream.

Syntax

```
stream (p. 93).start()
```

Parameters
- None

Returns
- None

Stream.stop Method

Notifies the producer to stop sending video feeds from your AWS DeepLens device to the Kinesis stream.

Syntax

```
stream (p. 93).stop()
```

Parameters
- None

Returns
• None
### Document History for AWS DeepLens

- **API version:** 2017-08-31

The following table describes the additions and changes to the AWS DeepLens Developer Guide documentation.

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Support for models created with TensorFlow and Caffe</td>
<td>Adds TensorFlow and Caffe as supported modeling frameworks. For more information see <a href="#">Supported TensorFlow Models and Layers</a> and <a href="#">Supported Caffe Models and Layers</a>.</td>
<td>June 14, 2018</td>
</tr>
<tr>
<td>Simplified IAM roles creation</td>
<td>Enables creation of required IAM roles to run your AWS DeepLens app project with a single click of a button in the AWS DeepLens console. For more information, see <a href="#">Register Your Device</a>.</td>
<td>June 14, 2018</td>
</tr>
<tr>
<td>Secure boot</td>
<td>Adds secure booting. The AWS DeepLens device securely boots to prevent the installation of unauthorized operating systems. For more information see <a href="#">Securely Boot Your Device</a>.</td>
<td>June 14, 2018</td>
</tr>
<tr>
<td>Ability to view device output streams in a browser</td>
<td>Adds support for using a browser to view unprocessed device streams or processed project streams. For more information, see <a href="#">View Your AWS DeepLens Device Output in a Browser</a>.</td>
<td>June 14, 2018</td>
</tr>
<tr>
<td>Integration with Amazon Kinesis Video Streams</td>
<td>Uses the Kinesis Video Streams for AWS DeepLens Video library to send video feeds from an AWS DeepLens device to Kinesis Video Streams for a specified period. The feeds can be used as input for additional vision analysis or as training data for your computer vision deep-learning model. For more information, see <a href="#">AWS DeepLens Kinesis Video Integration Library Module</a>.</td>
<td>June 14, 2018</td>
</tr>
<tr>
<td>Feature</td>
<td>Description</td>
<td>Date</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>Named device setup URL</td>
<td>Makes the AWS DeepLens device setup app accessible through the URL of <a href="http://deeplens.config">http://deeplens.config</a>. For more information see Set up Your Device.</td>
<td>June 5, 2018</td>
</tr>
<tr>
<td>Troubleshooting Guide</td>
<td>Provides a list of commonly asked questions and answers about troubleshooting common AWS DeepLens issues. For more information see Troubleshooting Guide.</td>
<td>May 3, 2018</td>
</tr>
<tr>
<td>Gluon support</td>
<td>Adds support for Gluon models. For more information, see Supported MXNet Models Exposed by the Gluon API.</td>
<td>March 13, 2018</td>
</tr>
<tr>
<td>Importing from Amazon SageMaker</td>
<td>Simplifies the process for importing a model trained with Amazon SageMaker. For more information, see Importing Your Amazon SageMaker-Trained Model.</td>
<td>February 22, 2018</td>
</tr>
<tr>
<td>Model optimization</td>
<td>Adds support for optimizing your custom model so that it runs on the GPU instead of the CPU. For more information, see Optimize a Custom Model and Model Optimization (mo) Module.</td>
<td>January 31, 2018</td>
</tr>
<tr>
<td>AWS DeepLens Developer Guide</td>
<td>The first release of the programmer’s guide to building edge-optimized computer vision applications using AWS DeepLens with inference models trained in common deep learning frameworks, including MXNet.</td>
<td>November 29, 2017</td>
</tr>
</tbody>
</table>
AWS Glossary

For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.