
AWS DeepLens

Developer Guide



AWS DeepLens: Developer Guide

Copyright © 2019 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

The AWS Documentation website is getting a new look!

Try it now and let us know what you think. [Switch to the new look >>](#)

You can return to the original look by selecting English in the language selector above.

Table of Contents

What Is AWS DeepLens?	1
Device Versions	1
The AWS DeepLens Device	1
The AWS DeepLens 2019 Edition Device	2
Hardware	2
Project Workflow	5
Modeling Frameworks	5
Application Development	6
More Info	6
Getting Started	7
Set Up Your Development Environment	8
Sign Up for an AWS Account	8
Create an IAM User	9
Set Up Required Permissions	9
Register Your Device	12
Register AWS DeepLens 2019 Edition Device	13
Register AWS DeepLens Device	23
Test Using Your Registered Device	35
Create and Deploy a Project	35
View Project Output	35
Building Projects	38
Supported Frameworks	38
MXNet Models and Layers	38
TensorFlow Models and Layers	40
Caffe Models and Layers	42
Viewing Project Output	45
View Video Streams from AWS DeepLens 2019 Edition Device in Browser	45
View Video Stream from AWS DeepLens Device in Browser	46
View Video Streams on Your AWS DeepLens Device	48
Creating a Lambda Function for Viewing the Project Stream	50
Working with Sample Projects	53
Sample Projects Overview	53
Create and Deploy a Sample Project in the Console	55
Relay Project Output through SMS	57
Use Amazon SageMaker to Provision a Model for a Project	63
Working with Custom Projects	73
Import Your Amazon SageMaker Trained Model	73
Import an Externally Trained Model	73
Optimize a Custom Model	75
Create and Publish an Inference Lambda Function	76
Create and Deploy a Custom Project	83
Use Neo to Optimize Inference	84
Building Project Tutorials	86
Build and Run the Head Pose Detection Project	86
Managing Your Device	105
Securely Boot Your Device	105
Update Your Device	105
Deregistering Your AWS DeepLens Device	106
Deleting AWS DeepLens Resources	107
Delete Resources Using the Console	107
Logging and Troubleshooting	109
Project Logs	109
CloudWatch Logs Log Groups	109
File System Logs	110

Troubleshooting Guide	111
Troubleshooting Software on Device	111
Troubleshooting Registration of Device	115
Troubleshooting Model Deployments to Device	121
Device Library	125
awscam Module	125
getLastFrame Function	125
Model Object	126
mo Module	131
optimize Method	131
Troubleshooting the Model Optimizer	134
DeepLens_Kinesis_Video Module	140
createProducer Function	141
Producer Object	142
Stream Object	143
Security	145
Data Protection	145
Authentication and Access Control	145
Incident Response	145
Update Management	145
Document History	146
AWS Glossary	149

What Is AWS DeepLens?

AWS DeepLens is a deep learning-enabled video camera. It is integrated with the Amazon Machine Learning ecosystem and can perform local inference against deployed models provisioned from the AWS Cloud. It lets you learn and explore the latest Artificial Intelligence (AI) tools and technology for developing computer vision applications based on a deep learning model.

As a beginner to machine learning, you can use AWS DeepLens to explore deep learning through hands-on tutorials based on ready-deployed deep learning sample projects. Each sample project contains a pre-trained model and a pedagogically straightforward inference function.

As a seasoned practitioner, you can leverage the AWS DeepLens development platform to train a convolutional neural network (CNN) model and deploy to the AWS DeepLens device your computer vision application project containing the model. You can train the model in any of the [supported deep learning frameworks](#) (p. 38), including Caffe, MXNet and TensorFlow.

To create and run an AWS DeepLens-based computer vision application project, you typically use the following AWS services:

- Use the Amazon SageMaker service to train and validate a CNN model or import a pre-trained model.
- Use the AWS Lambda service to create a project function to make inferences of video frames off the camera feeds against the model.
- Use the AWS DeepLens service to create a computer vision application project that consists of the model and inference function.
- Use the AWS IoT Greengrass service to deploy the application project and a Lambda runtime to your AWS DeepLens device, as well as the software or configuration updates.

This means that you must grant appropriate permissions to access these AWS services.

Topics

- [AWS DeepLens Device Versions](#) (p. 1)
- [AWS DeepLens Hardware](#) (p. 2)
- [AWS DeepLens Project Workflow](#) (p. 5)
- [Supported Modeling Frameworks](#) (p. 5)
- [Learning AWS DeepLens Application Development](#) (p. 6)

AWS DeepLens Device Versions

The following topics discuss salient features of and differences between the AWS DeepLens and AWS DeepLens 2019 Edition devices.

Topics

- [The AWS DeepLens Device](#) (p. 1)
- [The AWS DeepLens 2019 Edition Device](#) (p. 2)

The AWS DeepLens Device

First introduced at AWS re:Invent 2017, the AWS DeepLens device became available in the US market on June 14, 2018 with the support of a limited set of pre-trained models for

- [Artistic style transfer \(p. 53\)](#)
- [Object recognition \(p. 53\)](#)
- [Face detection and recognition \(p. 54\)](#)
- [Hot dog recognition \(p. 54\)](#)
- [Cat and dog recognition \(p. 54\)](#)
- [Action recognition \(p. 54\)](#)

Later, more pre-trained models were added to the support:

- [Head pose detection \(p. 55\)](#)
- [Bird classification \(p. 55\)](#)

For detailed hardware specification, see [AWS DeepLens Hardware \(p. 2\)](#).

The AWS DeepLens 2019 Edition Device

The AWS DeepLens 2019 Edition device became generally available on July 10, 2019 in the [UK](#), [Germany](#), [France](#), [Spain](#), [Italy](#), [Japan](#), [Canada](#) as well as [US](#) markets.

The AWS DeepLens 2019 Edition device includes the following hardware and software improvements:

- Simplified initial device setup through the USB-USB cable connection, instead of the SoftAP wireless connection.
- Improved (>2x) model optimization with [Amazon SageMaker Neo](#).
- [A sample application demonstrating how to improve work safety.](#)
- [A sample application demonstrating how to count number of coffee drinkers.](#)
- [A sample application demonstrating how to decipher facial sentiments.](#)

For detailed hardware specification, see [AWS DeepLens Hardware \(p. 2\)](#).

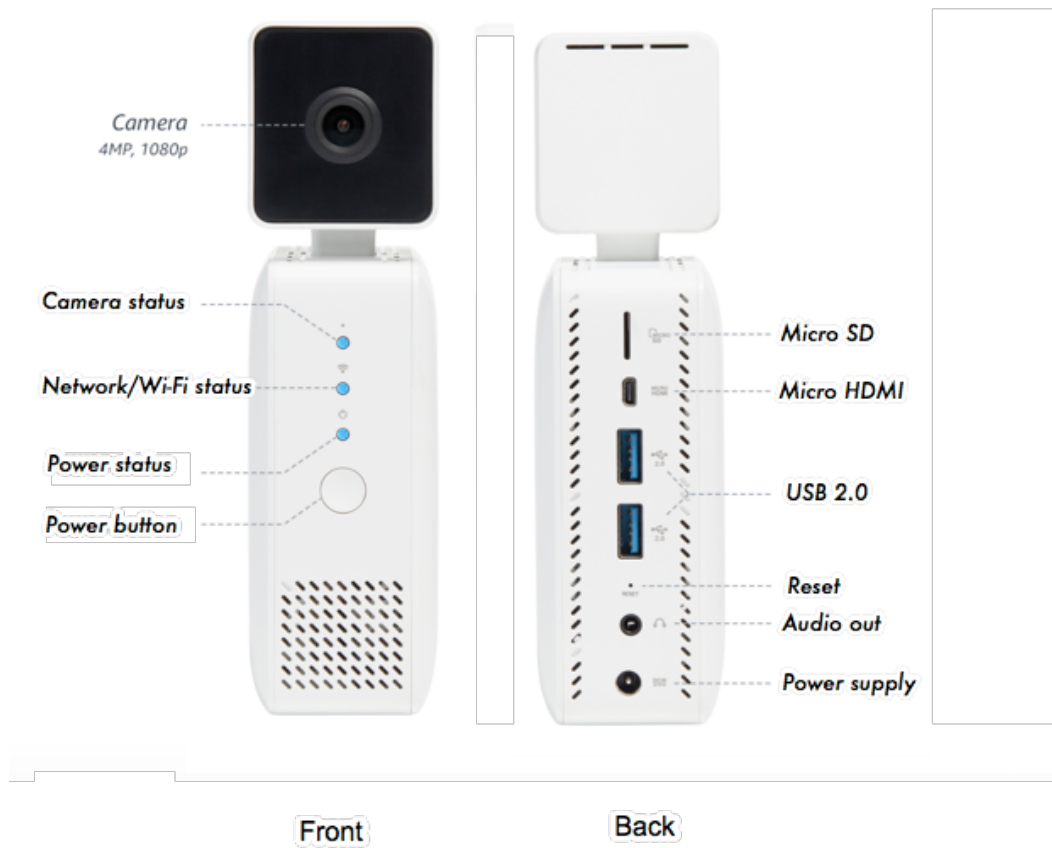
AWS DeepLens Hardware

The AWS DeepLens device has the following specs:

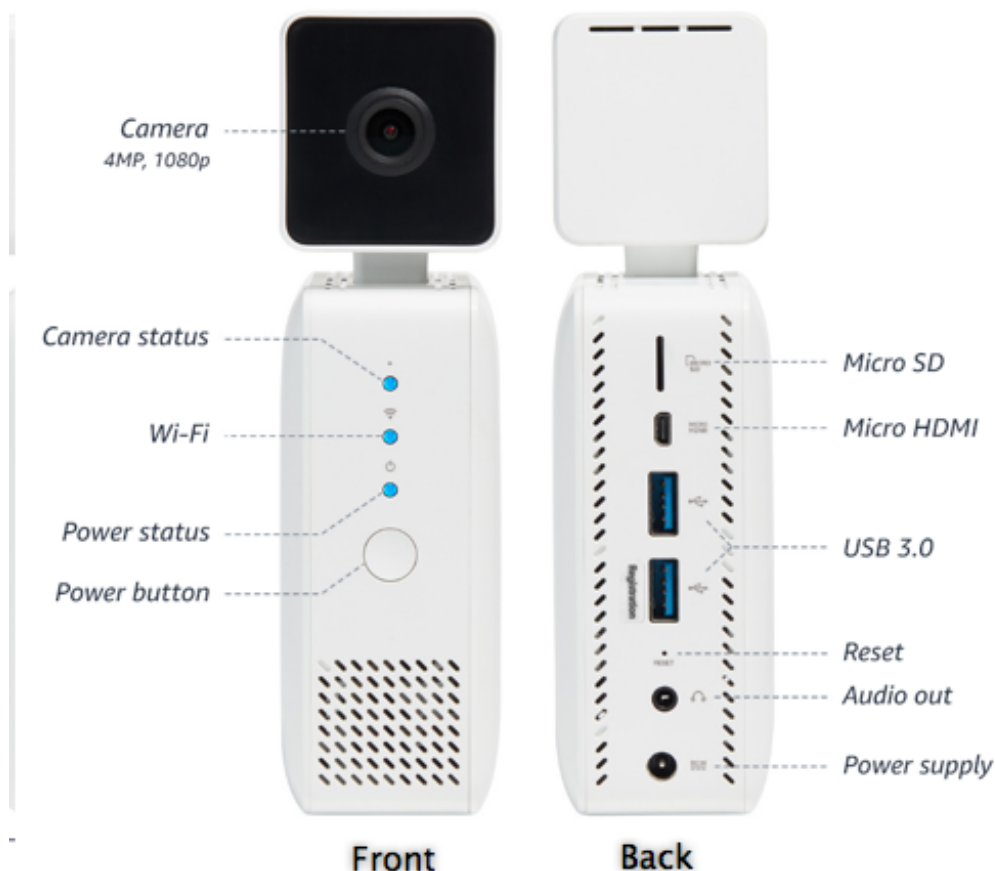
- A 4-megapixel camera with MJPEG (Motion JPEG)
- 8 GB of on-board memory
- 16 GB of storage capacity
- A 32-GB SD (Secure Digital) card
- Wi-Fi support for both 2.4 GHz and 5 GHz standard dual-band networking
- A micro HDMI display port
- Audio out and USB ports
- Power consumption: 20 W
- Power input: 5V and 4Amps

deeplens_2019_edition_device_specs

The following image shows the front and back of the AWS DeepLens hardware.



The following image shows the front and back of the AWS DeepLens 2019 Edition hardware.



On the front of the device, the power button is located at the bottom. Above it are three LED indicators show the device statuses:

- **Power status** indicates if the device is powered on or off.
- **Wi-Fi** shows if the device is in the setup mode (blinking), connected to your home or office network (steady) or not connected to the Internet (off).
- **Camera status**: when it is on, it indicates that a project is deployed successfully to the device and the project is running. Otherwise, the LED light is off.

On the back of the device, you have access to the following:

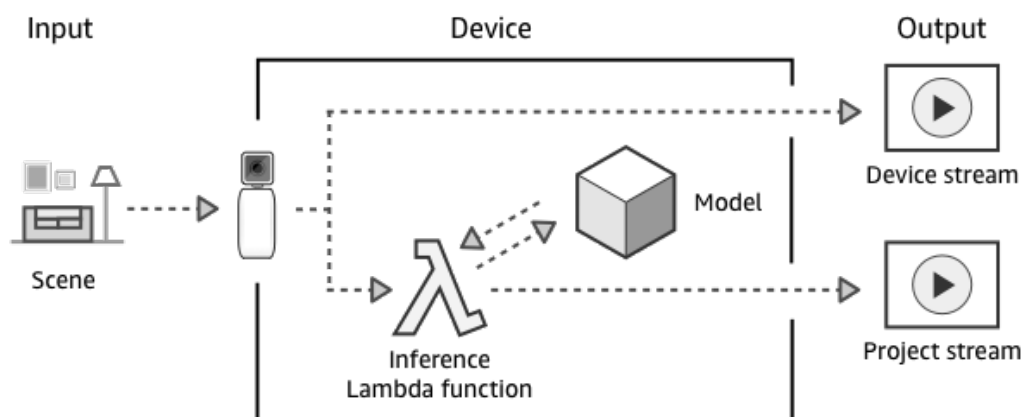
1. A micro SD card slot for storing and transferring data.
2. A micro HDMI slot for connecting a display monitor to the device using a micro-HDMI-to-HDMI cable.
3. Two USB 2.0 (AWS DeepLens Edition)/3.0 (AWS DeepLens 2019 Edition) slots for connecting USB mouse and keyboard, or any other USB-enabled accessories, to the device.
4. A reset pinhole for turning the device into the setup mode that allows you to make the device as an access point and to connect your laptop to the device to configure the device.
5. An audio outlet for connecting to a speaker or earphones.
6. A power supply connector for plugging in the device to an AC power source.

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 computing 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 a template for developing your own computer vision applications.

AWS DeepLens Project Workflow

The following diagram illustrates the basic workflow of a deployed AWS DeepLens project.



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.

```
Infinite inference is running. Sample result of the last frame is
label    prob
794      0.681035220623
669      0.0477042198181
905      0.0444286055863
564      0.0426452308893
706      0.0312749966979
```

For more information, see [Viewing AWS DeepLens Output Streams \(p. 45\)](#).

Supported Modeling Frameworks

With AWS DeepLens, you train a project model using a supported deep learning modeling framework. You can train the model on the AWS Cloud or elsewhere. Currently, AWS DeepLens supports Caffe,

TensorFlow and Apache MXNet frameworks, as well Gluon models. For more information, see [Machine Learning Frameworks Supported by AWS DeepLens \(p. 38\)](#).

Learning AWS DeepLens Application Development

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

1. **Read [AWS DeepLens Project Workflow \(p. 5\)](#)**—Explains the basic workflow of a AWS DeepLens project running inference on the device against the deployed deep learning model.
2. **Explore [Amazon SageMaker](#)**—You use Amazon SageMaker to create and train your own AWS DeepLens CNN model or import a pre-trained one.
3. **Learn about the [AWS DeepLens Device Library \(p. 125\)](#)**—The device library encapsulates the classes and methods that you can call in your Lambda functions to optimize the deployed model artifacts to grab a frame from the device video feeds, and to run inference on the video frame against the model.
4. **Register your device as prescribed in [Getting Started with AWS DeepLens \(p. 7\)](#)**—Register your AWS device, create the AWS Identity and Access Management (IAM) roles and policies to grant necessary permissions to run AWS DeepLens and its dependent AWS services.

After you've registered your AWS DeepLens device and configured the development environment, follow the exercises below:

- a. **[Create and Deploy an AWS DeepLens Sample Project in the AWS DeepLens Console \(p. 55\)](#)**—Walks you through creating sample AWS DeepLens project, which is included with your device.
- b. **[Use Amazon SageMaker to Provision a Pre-trained Model for a Sample Project \(p. 63\)](#)**—Walks you through creating and training a model using Amazon SageMaker.
- c. **[Relay an AWS DeepLens Project Output through AWS SMS \(p. 57\)](#)**—Walks you through taking output from your AWS DeepLens and using it to trigger an action.

More Info

- AWS DeepLens is available in the `us-east-1` region.
- [AWS DeepLens Forum](#)

Getting Started with AWS DeepLens

Before building a computer vision application project and then deploying the project to your AWS DeepLens device, you must first accomplish the following:

1. Set up your AWS DeepLens application development environment:

For this task, you sign up for an AWS account, if you have not already got one. You should also create an IAM user for building and deploying your AWS DeepLens project. Additionally, you should have some understanding of the permissions required to use AWS DeepLens. You can leverage the AWS DeepLens console to define the required permissions for you. Alternatively, you can use the IAM console or call the IAM API to define them yourself.

2. Register your AWS DeepLens device:

For this task, you name your device for the AWS DeepLens service to identify it; grant IAM permissions for you to create and deploy an AWS DeepLens project; call AWS IoT to generate a security certificate for the device; and request AWS IoT Greengrass to create an AWS IoT thing representation for your device. You do these on the AWS Cloud. To complete the registration, you must call the setup app on the device after connecting your computer to the device's local Wi-Fi network, also known as its `AMDC-NNNN` network. To set up the device, you choose to use a home or office network to connect to the internet, upload the security certificate generated by AWS IoT for the device, and set the password for device logins.

3. Verify the device registration status.

You can deploy your AWS DeepLens project to your device only when the device is successfully registered and online. Here, online means that the device is connected to the Internet and authenticated by the AWS Cloud. Hence, verifying the registration status amounts to ensuring that the internet connection remains open and the correct security certificate is uploaded to the device.

After you've successfully registered your AWS DeepLens device and connected the device to the AWS cloud, you typically proceed to explore AWS DeepLens by performing the following:

1. Building an AWS DeepLens computer vision application project, including training a deep learning computer vision model and developing an inference Lambda function.
2. Deploying the project to run on the device.
3. Inspecting the project output to verify the inference result.

To introduce you to this process end-to-end, we will walk you through all the above-mentioned tasks in this section, focusing on AWS DeepLens development environment setup and the AWS DeepLens device registration, while highlighting project creation, deployment and result viewing with a sample project containing a pre-trained model and a tested inference Lambda function.

Topics

- [Set Up Your AWS DeepLens Development Environment \(p. 8\)](#)
- [Register Your AWS DeepLens Device \(p. 12\)](#)
- [Test Using Your Registered AWS DeepLens Device \(p. 35\)](#)

Set Up Your AWS DeepLens Development Environment

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

- [Sign Up for an AWS Account](#) (p. 8)
- [Create an IAM User](#) (p. 9)
- [Set Up Required Permissions](#) (p. 9)

Sign Up for an AWS Account

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

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

To sign up for an AWS account

1. Go to <https://portal.aws.amazon.com>.
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

The following procedure walks you through the steps to create an [AWS Identity and Access Management \(IAM\) user](#), which is preferred to using the root user of your AWS account, to use AWS DeepLens for your deep learning computer vision applications.

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** to call APIs of permitted AWS services and **AWS Management Console Access** to launch the AWS Management Console for permitted services.
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**, select the check box next to a listed policy, e.g., **AdministratorAccess** if you want the user to have the administrator's privileges, and then choose **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. 12).

Set Up Required Permissions

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 IoT Greengrass as the underlying infrastructure to connect your AWS DeepLens device to the AWS Cloud, AWS DeepLens also needs permissions for these dependent AWS services to execute the Lambda functions and manage the device on your behalf.

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 device using the AWS DeepLens console for the first time, the AWS DeepLens console can create the following required IAM roles with predefined IAM policies, with a single command:

- **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 IoT Greengrass, and AWS Lambda. The AWS DeepLens console attaches the AWS-managed policy of [AWSDeepLensServiceRolePolicy](#) to this role. 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 AWS DeepLens console attaches the AWS-managed policy of [AWSLambdaFullAccess](#) to this role. Customization of this default policy is not necessary.
- **AWSDeepLensGreengrassRole**:—An IAM role that is passed to AWS IoT Greengrass to allow AWS IoT 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

AWS DeepLens console attaches the AWS-managed policy of [AWSGreengrassResourceAccessRolePolicy](#) to this role. You don't need to customize it.

- **AWSDeepLensGreengrassGroupRole:**—An IAM role that is passed to AWS IoT Greengrass device groups, which gives AWS DeepLens administrative Lambda functions to access other AWS services. The AWS DeepLens console attaches the AWS-managed policy of [AWSDeepLensLambdaFunctionAccessPolicy](#) to this role. The [AWS IoT Greengrass group](#) defines how your AWS DeepLens device communicates with the AWS IoT Greengrass core devices.

The managed **AWSDeepLensLambdaFunctionAccessPolicy** has predefined permissions to allow the project's Lambda function to call certain operations on Amazon S3 objects with the `deeplens` prefixes. It also supports AWS DeepLens logging operations to CloudWatch Logs and permits the function to send video feeds to Kinesis Video Streams. If your project's Lambda function makes use of other AWS services, you need to customize the **AWSDeepLensLambdaFunctionAccessPolicy** policy to add new policy statements specific to the additional services.

For example, suppose that you have a device installed in a warehouse and another one at your office. The device in the warehouse needs to access your inventory system built upon DynamoDB, whereas the one in the office does not. You must then create a new IAM role of the **AWSDeepLensGreengrassGroupRole** type and attach to the new role the **AWSDeepLensLambdaFunctionAccessPolicy** and additional policy statements that permit the Lambda function on the device in the warehouse to access DynamoDB.

Alternatively, you can create these IAM roles yourself. For information about how to create the required IAM roles and permissions yourself, see [the section called “Create IAM Roles for Your Project” \(p. 10\)](#).

Create IAM Roles for Your AWS DeepLens Project

If your AWS account doesn't already have required IAM roles, you can use the AWS DeepLens console to create them with a single command or use the AWS Identity and Access Management console to create them individually.

If you already have these roles, AWS DeepLens uses them when you register your device.

In either case, you have the option to customize the [AWSDeepLensGreengrassGroupRole](#) to grant different permissions to different devices.

Topics

- [Create AWSDeepLensServiceRole Using the IAM Console \(p. 10\)](#)
- [Create AWSDeepLensLambdaRole Using the IAM Console \(p. 11\)](#)
- [Create AWSDeepLensGreengrassGroupRole Using the IAM Console \(p. 11\)](#)
- [Create AWSDeepLensGreengrassRole Using the IAM Console \(p. 10\)](#)
- [Create AWSDeepLensSagemakerRole Using the IAM Console \(p. 12\)](#)

Create **AWSDeepLensServiceRole** Using the IAM Console

To create **AWSDeepLensServiceRole** Using the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Create role**.
3. Under **Select type of trusted entity**, choose **AWS service**.
4. Under **Choose the service that will use this role**, choose **DeepLens**.
5. Under **Select your use case**, choose **DeepLens**

6. Choose **Next: Permissions**.
7. In the **Create role** page, make sure that the **AWSDeepLensServiceRolePolicy** is listed under **Attached permissions policies** and then choose **Next: Review**.
8. For **Role name**, type **AWSDeepLensServiceRole** and then choose **Create role**.

Create **AWSDeepLensLambdaRole** Using the IAM Console

To create **AWSDeepLensLambdaRole** in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Create role**.
3. Under **Select type of trusted entity**, choose **AWS service**.
4. Under **Choose the service that will use this role**, choose **Lambda**.
5. Choose **Next: Permissions**.
6. In the **Create role** page, type **AWSLambdaFullAccess** in the search query input field next to **Filter policies** under **Attached permissions policies**. Choose the checkmark next to the **AWSLambdaFullAccess** policy entry and then choose **Next: Review**.
7. For **Role name**, type **AWSDeepLensLambdaRole** and then choose **Create role**.

Create **AWSDeepLensGreengrassGroupRole** Using the IAM Console

To create **AWSDeepLensGreengrassGroupRole** in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Create role**.
3. Under **Select type of trusted entity**, choose **AWS service**.
4. Under **Choose the service that will use this role**, choose **DeepLens**.
5. Under **Select your use case**, choose **DeepLens - Greengrass Lambda**.
6. Choose **Next: Permissions**.
7. In the **Create role** page, make sure that the **AWSDeepLensLambdaFunctionAccessPolicy** is listed under **Attached permissions policies** and then choose **Next: Review**.
8. For **Role name**, type **AWSDeepLensGreengrassGroupRole** and then choose **Create role**.

Create **AWSDeepLensGreengrassRole** Using the IAM Console

To create **AWSDeepLensGreengrassRole** in the IAM console

1. Open the IAM console at <https://console.aws.amazon.com/iam/>.
2. Choose **Create role**.
3. Under **Select type of trusted entity**, choose **AWS service**.
4. Under **Choose the service that will use this role**, choose **Greengrass**.
5. Under **Select your use case**, choose **Greengrass**.
6. Choose **Next: Permissions**.
7. In the **Create role** page, type **AWSGreengrassResourceAccessRolePolicy** in the filter query input field. Choose the checkmark next to the **AWSGreengrassResourceAccessRolePolicy** listed under **Attached permissions policies** and then choose **Next: Review**.

- For **Role name**, type **AWSDeepLensGreengrassRole** and then choose **Create role**.

Create **AWSDeepLensSageMakerRole** Using the IAM Console

If you use Amazon SageMaker to train a custom deep learning model for your AWS DeepLens project, you must also create an IAM role to grant Amazon SageMaker permissions to access required AWS resource on your behalf. To grant the permissions, follow the step below to create the **AWSDeepLensSageMakerRole** in the IAM console.

To create **AWSDeepLensSageMakerRole** in the IAM console

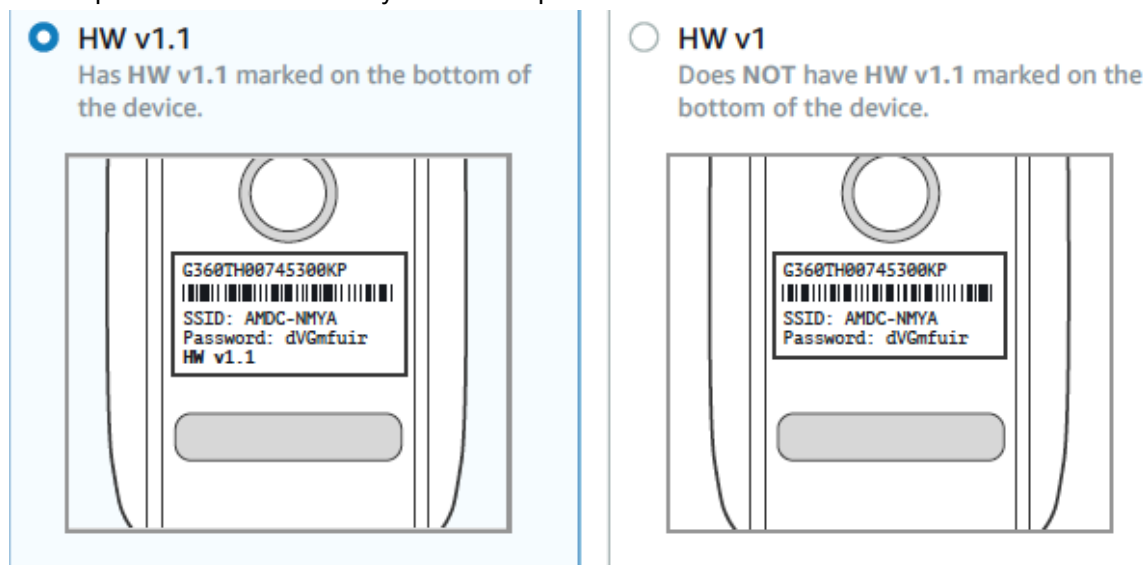
- Open the IAM console at <https://console.aws.amazon.com/iam/>.
- Choose **Create role**.
- Under **Select type of trusted entity**, choose **AWS service**.
- Under **Choose the service that will use this role**, choose **SageMaker**.
- Under **Select your use case**, choose **SageMaker - Execution**
- Choose **Next: Permissions**.
- In the **Create role** page, type **AmazonSageMakerFullAccess** in the filter query input field. Choose the checkmark next to the **AmazonSageMakerFullAccess** listed under **Attached permissions policies** and then choose **Next: Review**.
- For **Role name**, type **AWSDeepLensSageMakerRole** and then choose **Create role**.

Register Your AWS DeepLens Device

To run your deep learning computer vision application on your AWS DeepLens device, you must first register the device with AWS. After it's registered, you can create an AWS DeepLens project in the AWS Cloud. You'll also be able to deploy the project to the registered device and update the device software and settings.

Before registering your device, make sure that you have met all the requirements discussed in [Set Up Your AWS DeepLens Development Environment](#) (p. 8).

The registration process is different for each version of the AWS DeepLens hardware. Check the hardware version printed on the bottom of your AWS DeepLens device



and then follow the version-specific instructions provided below.

Topics

- [Register AWS DeepLens 2019 Edition Device \(p. 13\)](#)
- [Register Your AWS DeepLens Device \(p. 23\)](#)

Register AWS DeepLens 2019 Edition Device

Registering your AWS DeepLens 2019 Edition device involves performing the following tasks:

1. Connect the device to your computer.
2. Verify the device serial number.
3. Set up the device's internet connection.
4. Create device representation in the AWS Cloud.
5. Update the device settings.

Follow the sections below for detailed instructions for each of the tasks.

Note

These instructions are for only AWS DeepLens 2019 Edition device. You can find out the hardware version on the bottom of your AWS DeepLens device. If you see **HW v1.1** printed there, your device is of AWS DeepLens 2019 Edition. You can then proceed as follows. Otherwise, see [the section called "Register AWS DeepLens Device" \(p. 23\)](#).

Topics

- [Connect Your AWS DeepLens 2019 Edition Device to Your Computer \(p. 13\)](#)
- [Validate Your AWS DeepLens 2019 Edition Device Serial Number \(p. 16\)](#)
- [Connect Your AWS DeepLens 2019 Edition Device to the Internet \(p. 17\)](#)
- [Name Your AWS DeepLens 2019 Edition Device and Complete the Registration \(p. 19\)](#)
- [View or Update Your AWS DeepLens 2019 Edition Device Settings \(p. 22\)](#)

Connect Your AWS DeepLens 2019 Edition Device to Your Computer

To use a computer to register your AWS DeepLens 2019 Edition device, you must first connect the device to your computer. To connect your device to your computer, follow the steps below.

Note

In addition, you can also [connect to the device directly using a monitor, a mouse and a keyboard \(p. 16\)](#).

To connect your AWS DeepLens 2019 Edition device to your computer

1. Sign in to the AWS Management Console for AWS DeepLens at <https://console.aws.amazon.com/deeplens/home?region=us-east-1#firstrun>.
2. Choose **Register a device**.

Note

If you don't see a **Register a device** button, choose **Devices** from the main navigation pane.

3. On the **Choose a hardware version** dialog box, choose **HW v1.1** for your AWS DeepLens 2019 Edition device. **Start** to begin the registration.

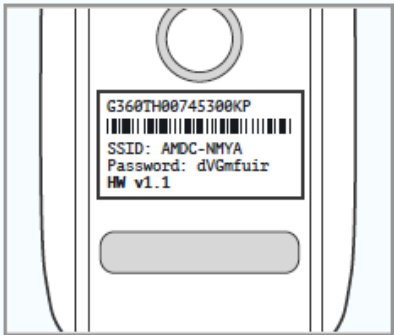
Choose a hardware version

Registration differs depending on the hardware version of the AWS DeepLens device.
The hardware version is printed at the bottom of the device.

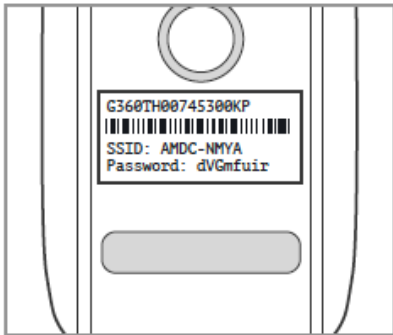
Choose a version

☒ **HW v1.1**
Has HW v1.1 marked on the bottom of the device.


☐ **HW v1**
Does NOT have HW v1.1 marked on the bottom of the device.



G360TH00745300KP
SSID: AMDC-NMYA
Password: dVGmfuir
HW v1.1



G360TH00745300KP
SSID: AMDC-NMYA
Password: dVGmfuir

 **Your computer will lose internet connection during registration.**
This is an expected behavior, click **Start** to continue.

Cancel

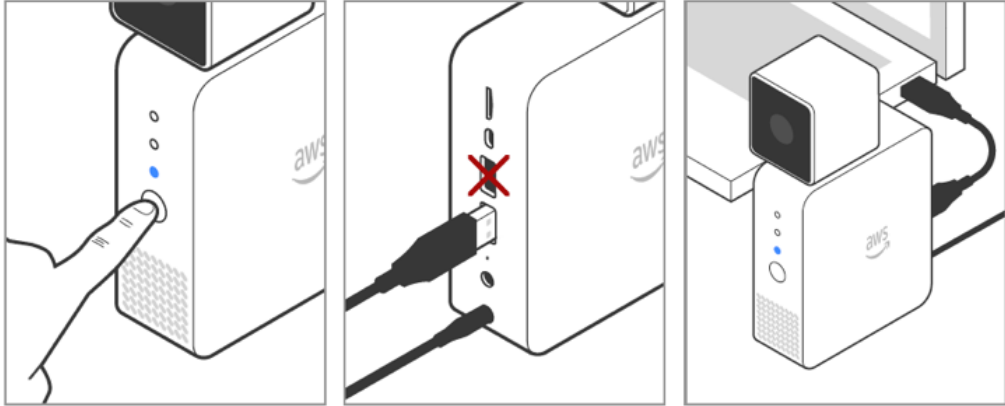
Start

- On the **Connect your device to your computer** page, follow the instructions to power on your device and then connect the device to your computer by plugging one end of the USB-to-USB cable to a USB port into your computer and the other end into the device's **REGISTRATION** USB port on the back of the device. Then, choose **Next**.

Connect your device to your computer

Instructions

1. **Power on your device:** Connect your AWS DeepLens to a power outlet using the included adapter, and turn it on. The power LED should turn blue.
2. **Connect it your computer:** Use the included USB cable to connect your computer to the device **REGISTRATION** USB port on the back of your AWS DeepLens.



Device USB connection

🔄 Detecting your device, it may take about 2 minutes

Cancel

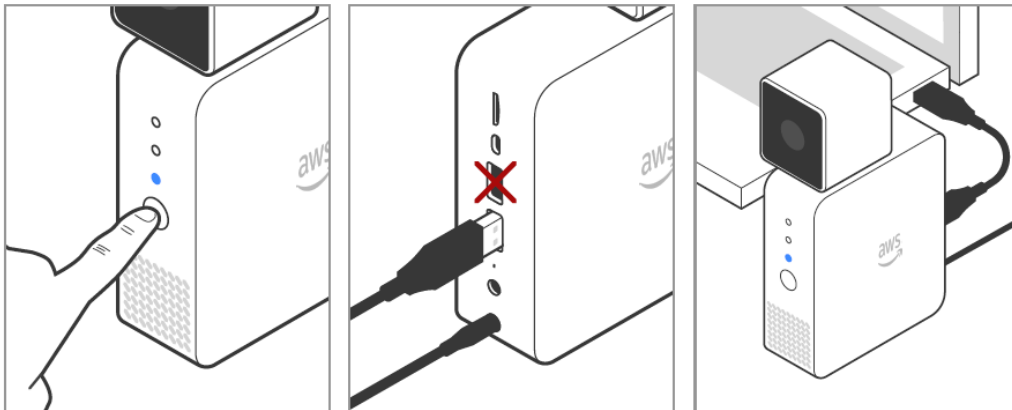
Next

5. Wait for the connection to complete when the **Device USB connection** status becomes **Connected** and then choose **Next**.

Connect your device to your computer

Instructions

1. **Power on your device:** Connect your AWS DeepLens to a power outlet using the included adapter, and turn it on. The power LED should turn blue.
2. **Connect it your computer:** Use the included USB cable to connect your computer to the device **REGISTRATION** USB port on the back of your AWS DeepLens.



Device USB connection
✔ Connected



Your computer has lost internet connection.

This is an expected behavior. Do not refresh your browser during registration. Click Next to continue.

Cancel

Next

Connect to AWS DeepLens 2019 Edition Device Using Monitor, Mouse and Keyboard

As an alternative to using a computer to connect to your AWS DeepLens 2019 Edition device, you can also connect to the device directly using a monitor with a μ HDMI-to-HDMI cable, a USB keyboard and a USB mouse. You'll be prompted for the Ubuntu login credentials. If this is your first time to log in to the device this way, enter `aws_cam` for **Username** and `aws_cam` for **Password**. You'll then be asked to reset the password. For security reasons, use a strong password of a complex phrase. For any subsequent logins, enter your new password in **Password**.

Validate Your AWS DeepLens 2019 Edition Device Serial Number

After your device is connected to your computer, the AWS DeepLens console will validate the device serial number (DSN) to verify that the device is version 1.1.

To validate your AWS DeepLens 2019 Edition device

1. Make note of the last four digits of your device serial number. It's printed on the bottom of the device.
2. On the **Validate your device** page, enter the last four digits of your device serial number.
3. Choose **Next**.

Validate your device

Serial number validation

Enter the last 4 digits of your device's serial number located below the device.

- - - -

Valid characters are 0-9.

Cancel Back Next

Connect Your AWS DeepLens 2019 Edition Device to the Internet

After the device has been validated, you must get your AWS DeepLens 2019 Edition device connected to the internet. You can use a wireless (Wi-Fi) network or a wired (Ethernet) network. The latter requires that you have an Ethernet-to-USB adapter. The internet connection ensures that the device receives necessary software updates during registration.

Here, we demonstrate setting up the internet connection over Wi-Fi.

To set up the internet connection to your AWS DeepLens 2019 Edition device over Wi-Fi

1. On the **Set up your device's internet connection** page, choose **Wi-Fi network** under **Network type**.
2. Choose your network from the **Wi-Fi network name (SSID)**, type your network password under **Wi-Fi password**, and then choose **Connect**.

Setup your device's internet connection

Network type

Choose a network type to connect your device to the AWS cloud.


☒ Wi-Fi network

☐ Ethernet

Wi-Fi network details

Specify your Wi-Fi network details.


Wi-Fi network name (SSID)

 my_network

WEP ▼

Wi-Fi password

☐ Show password

 Connecting to Wi-Fi network...

Cancel

Back

Next

- Wait for the device to be connected to the internet, when **Status** under **Wi-Fi network details** becomes **Online**. Choose **Install software update** under **Required software update**.

Setup your device's internet connection

Wi-Fi network details

Edit

Status
 Online

Wireless network SSID
my_network

Required software update

You must install a software update on your AWS DeepLens.

Install software update

Cancel

Back

Next

- Wait for the software update to complete. Choose **Next**.

Setup your device's internet connection

Wi-Fi network details Edit

Status	Wireless network SSID
✓ Online	my_network

Required software update

✓ Software update installed successfully. Click **Next** to continue.

Cancel Back Next

Name Your AWS DeepLens 2019 Edition Device and Complete the Registration

After your AWS DeepLens 2019 Edition device is connected to the internet and its software updated, name the device for future reference and agree to the AWS access permissions that are granted on your behalf in order to complete the registration.

To name your AWS DeepLens 2019 Edition device, agree to AWS permissions, and complete the registration

1. Under **Name your device**, give the device a name.
2. Under **Permissions**, check the box to confirm **I agree that required IAM roles containing necessary permissions be created on my behalf.**
3. Choose **Register device** to complete the registration.

The screenshot shows a web form titled "Name your device". It has two main sections: "Device details" and "Permissions". In the "Device details" section, there is a label "Device name" with a description: "This is a unique and uneditable name that identifies the device that is being registered." Below this is a text input field containing the word "Dippy". A note below the field states: "The device name can have up to 100 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen). No spaces." The "Permissions" section has a sub-header "Permissions" with a link "Info". Below it is a checkbox that is checked, with the text: "I agree that the required AWS permissions be created on my behalf. You can always revoke the permissions by logging into the IAM console." At the bottom right of the form are three buttons: "Cancel", "Back", and "Register device".

The device registration is now underway. The asynchronous process involves the following tasks:

- The device communicates with the AWS DeepLens service on the cloud through the specified Wi-Fi network.
- The AWS DeepLens service has necessary IAM roles with the required AWS permissions created for you.
- The AWS Cloud creates device representation with AWS IoT and other AWS services.
- The device downloads its AWS certificate of your account from the AWS Cloud.

To verify that the registration is successfully completed, follow the [next step \(p. 20\)](#).

Verify Your AWS DeepLens 2019 Edition Device Registration

After the registration is complete, verify the device registration is successful by checking the device's registration status.

To verify your AWS DeepLens 2019 Edition device registration

1. Wait for the device registration to complete. Make sure the **Registration status** under **Device Status** shows **Registered**.

✔ Device software update was successful.
No packages are upgradable.

✔ Your device has been registered successfully.
You may disconnect your device from your computer.

DeepLens > Devices > my-deeplens-2019 Share your feedback

my-deeplens-2019

Edit logsDeregisterEdit device settings

Device status

Registration status ✔ Registered	Device status ⓘ Update available	Software Version 1.4.0.b200442
-------------------------------------	-------------------------------------	-----------------------------------

Current project

Remove projectDeploy a project

You are now ready to deploy a machine learning project. Choose **Deploy a project** to get started.

Project output

You can use AWS IoT console to view a JSON-formatted output of the project deployed to your AWS DeepLens device. [Info](#)

To view MQTT messages:

- Copy the topic that is unique to your registered device: `$aws/things/deeplens_vITHDwzXQDuMghZOCi` Copy
- Go to [AWS IoT console](#), paste the topic in the Subscription topic input field and choose **Subscribe to topic**.

Video streaming

- Open your [Device settings](#) and follow the instructions to install the streaming certificate.
- Then, click on **View video stream**.

View video stream [↗](#)

You can also view your device's video output over a monitor. [Learn more](#).

Device details

ⓘ Update your device software

Update

Last updated -	IP address 192.168.14.78	Greengrass system logs Logs Info
Creation time 7/12/2019, 11:49:12 AM	ARN arn:aws:deeplens:us-east-1:738575810317:device/my-deeplens-2019	Lambda logs Logs Info
Hardware version 1.1		

2. Optionally, note the MQTT topic Id (e.g., `$aws/things/deeplens_cTsBnnQxT0mMx45HGLJ4DA/infer`) displayed under **Project Output**. You'll need this Id to [view a deployed project output in the AWS IoT console](#) (p. 35).

View or Update Your AWS DeepLens 2019 Edition Device Settings

You can view or update the device settings once the device is successfully registered.

To view or update your AWS DeepLens 2019 Edition device settings

1. On the successfully registered device details page in the AWS DeepLens console, choose **Edit device settings**.
2. Make sure your device is connected to your computer via the USB-to-USB cable and then choose **Next** in the **Connect your device to your computer** dialog box.
3. After the **Device USB connection** status becomes **Connected**, choose **Next**, again, to open the **Device settings** page.

DeepLens > Devices > my-deeplens-2019 > Edit device settings [Share your feedback](#)

Edit device settings

Network details [Edit](#)
Wi-Fi network name (SSID) Status
Mobile Online

SSH Server [Edit](#)
Enable the SSH server on your device to enable login via CLI to execute commands.
Status Password
Enabled set

Video streaming
Install a streaming certificate to your browser to stream video from your device.
Select your browser
Each browser has specific instructions to install the streaming certificate.
[Select browser](#)

4. To use a different network for the device's internet connection, choose **Edit** in **Network details** and then select a network type, choose a Wi-Fi network, type the password, and choose **Connect**.
5. To enable SSH connection to the device and set the SSH password, choose **Edit** in **SSH server** and then select the **Enable** option, create a password, confirm the password, and choose **Save changes**.
6. To enable viewing video output from your device in a browser, install required video streaming certificates for one or more supported browsers.
 - a. Choose a supported browser from the **Select your browser** drop-down list. As an example, we choose **Firefox (Windows, MacOS Sierra or higher, and Linux)**.
 - b. Choose **Download streaming certificate** to save the certificate to your computer.
 - c. Follow the remaining instructions thereafter to import the downloaded streaming certificate into the browser.

Note

Be sure to use DeepLens for the certificate password, when prompted.

To verify the certificate is installed properly, follow the instructions given in [View Video Streams from AWS DeepLens 2019 Edition Device in Browser \(p. 45\)](#).

When done with viewing or updating the device settings, you can disconnect the device from your computer and use AWS DeepLens.

Register Your AWS DeepLens Device

The instructions presented thereafter apply to the original (also known as v1) AWS DeepLens device. You can find out the hardware version at the bottom of your AWS DeepLens device. If you don't see **HW v1.1** printed there, your device is of the original AWS DeepLens edition. You can then proceed as instructed as follows. Otherwise, see [the section called "Register AWS DeepLens 2019 Edition Device" \(p. 13\)](#).

The registration process involves performing the following tasks. Some of the tasks are carried out on the AWS Cloud and others are on the AWS DeepLens device.

- Name your device so that you can identify it within the AWS DeepLens service.
- Grant IAM permissions to build and deploy AWS DeepLens projects for deep learning computer vision applications.
- Download a security certificate for the device. This certificate is generated by AWS IoT upon request by the AWS DeepLens service. You must upload it to the device when setting up the device later.
- Create an AWS IoT thing representation for your AWS DeepLens device, which is carried out by AWS IoT Greengrass upon request by the AWS DeepLens service.
- Turn on the device's setup mode and join your computer in the device's local Wi-Fi (also referred to as **AMDC-NNNN**) network. This lets you call the device setup app as a web application hosted by the local web server of the device.
- Start the device setup application on the device to configure device access to internet; to upload the AWS-generated security certificate to the device for the AWS Cloud to authenticate the device; and to create a device login password for signing in to the device using a hardwired monitor, mouse and keyboard or using an SSH client from a computer within the same home or office network.

Your AWS DeepLens device has a default password of `aws_cam`. You can use this default device password to log on to the device connected to a monitor using a μ USB-to-USB cable, a USB mouse and possibly a USB keyboard even before registration. For security reasons, you should reset this password with a more complex or strong password phrase.

Topics

- [Configure Your AWS Account for AWS DeepLens Device \(p. 23\)](#)
- [Connect to Your AWS DeepLens Device's Wi-Fi Network \(p. 26\)](#)
- [Set Up Your AWS DeepLens Device \(p. 30\)](#)
- [Verify Your AWS DeepLens Device Registration Status \(p. 34\)](#)

Configure Your AWS Account for AWS DeepLens Device

Configuring your AWS account for your AWS DeepLens device involves naming the device, grant AWS access permissions, and download a certificate for the device to be authenticated by AWS.

To configure your AWS account for AWS DeepLens device

1. Sign in to the AWS Management Console for AWS DeepLens at <https://console.aws.amazon.com/deeplens/home?region=us-east-1#firstrun>.
2. Choose **Register a device**. If you don't see a **Register a device** button, choose **Devices** on the main navigation pane.
3. On the **Choose a hardware version** dialog window, choose the **HW v1** radio button for your AWS DeepLens device. Then choose **Start**.

4. In the **Name your device** section on the **Configure your AWS account** page, type a name (e.g., `My_DeepLens_1`) for your AWS DeepLens device in the **Device name** text field .

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

DeepLens > Devices > Register a DeepLens device

Step 1
Configure your AWS account

Step 2
Connect to your device

Step 3
Configure your device

Configure your AWS account

Let's start by configuring your AWS account to work with your device. You will name your device, grant resource access permissions to deploy ML projects, and download the security certificate needed to add DeepLens to your AWS account.

Name your device

Device name
Choose a friendly name to identify your device in the AWS DeepLens console.

The device name can have up to 100 characters. Valid characters are a-z, A-Z, 0-9, and - (hyphen). No spaces.

Permissions

DeepLens requires additional roles and permissions to deploy ML projects. Choose Create roles to automatically set them up and grant permissions for your DeepLens (you can remove or change them later). You can also manually grant the permissions using the IAM console. [info](#)

[Create roles](#)

Certificate

DeepLens connects securely to the AWS Cloud using a certificate which is unique to your account. Download your security certificate now. Then, you will upload it to your device later during the device configuration step. For security, we recommend you do not share this certificate with others.

[Download certificate](#)

You must download the security certificate to continue with registration. Do not unzip the file.

[Cancel](#) [Next](#)

5. In the **Permissions** section, choose **Create roles** for the AWS DeepLens console to create [the required IAM roles with relevant permissions \(p. 9\)](#) on your behalf.

After the roles are successfully created, you'll be informed that you have the necessary permissions for setting the AWS DeepLens device. If the roles already exist in your account, the same message will be displayed.

6. In the **Certificate** section, choose **Download certificate** to save the device certificate.

Important

The downloaded device certificate is a .zip file. Don't unzip it.

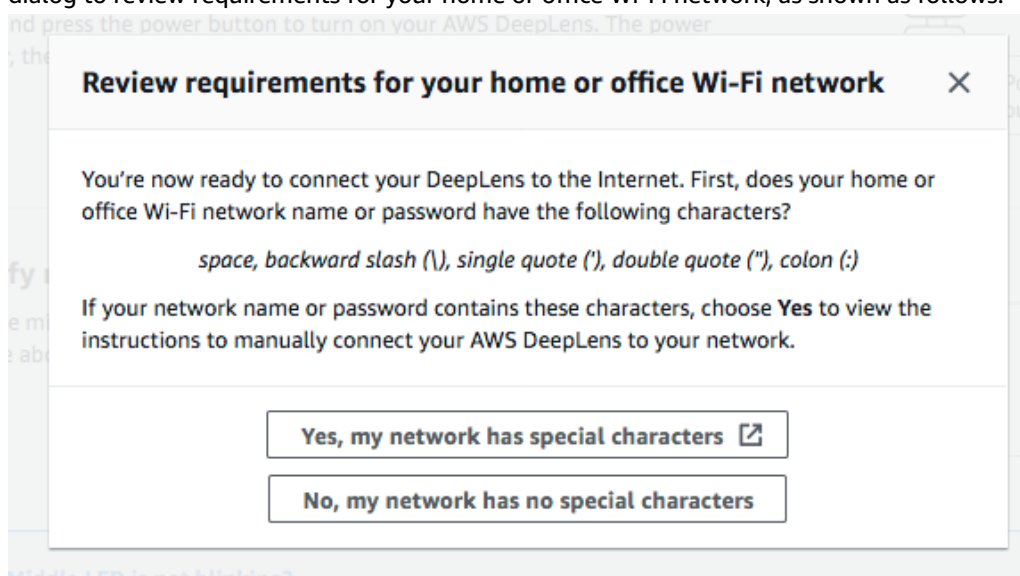
Certificates aren't reusable. You must generate a new certificate every time you register your device.

7. After the certificate is downloaded, choose **Next** to proceed to joining your computer to your device's (AMDC-~~NNNN~~) Wi-Fi network in order to start the device setup application hosted on the device.

Note

On devices of certain earlier versions, the legacy device setup app assumes that your home or office Wi-Fi network's name (SSID) and password do not contain special characters, including space, backward slash (\), single quote ('), double quote (") or colon (;). If you've got such a device and your home or office Wi-Fi network name or password contains such special characters, the legacy device setup app will be blocked when you attempt to configure its network connection.

If you have such a legacy device, after choosing **Next**, you will be prompted with a modal dialog to review requirements for your home or office Wi-Fi network, as shown as follows:



- If your home or office network name or password contains any special character, choose **Yes, my network has special characters** and follow [this troubleshooting guide \(p. 117\)](#) to establish an SSH session to configure the Wi-Fi network connection between your device and the internet and, then, to move on to configuring the rest of the device settings. After that, you should update the device software to its latest version.
- If your home or office network name or password do not contain any special character, choose **No, my network has no special characters** to be directed to [the Connect to your device page \(p. 26\)](#).

If you have a more recent device or your device software is updated, you will be directed to [the Connect to your device page \(p. 26\)](#) without the modal dialog after choosing **Next**.

Connect to Your AWS DeepLens Device's Wi-Fi Network

To set up your AWS DeepLens device, you must first connect your computer to the device's local Wi-Fi network, also known as the device's `AMDC-NNNN` network. When the Wi-Fi indicator (the middle LED light) blinks on the front of the device, this network is active and the device is in setup mode. You're then ready to connect a computer to the device.

Note

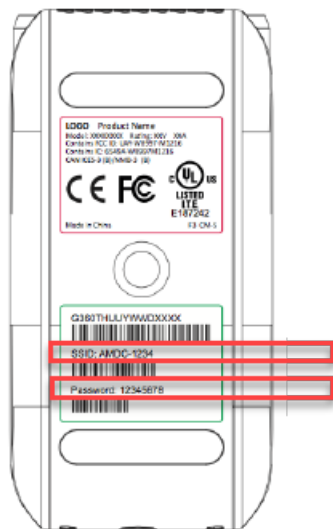
In addition, you can also [connect to the device directly using an external monitor, mouse and keyboard \(p. 30\)](#).

When you set up your device for the first time, the device is automatically booted into setup mode and ready for your computer to join its `AMDC-NNNN` network. To update device settings after the initial setup, you must explicitly turn on the device setup mode (instructions given below) and then have your computer rejoin the device's Wi-Fi network. If the device exits setup mode before you can finish the setup, you must reconnect your computer to the device in the same way.

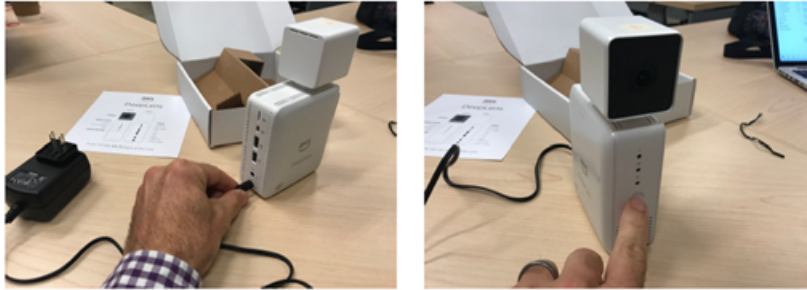
While your AWS DeepLens device is in its setup mode and your computer is a member of its Wi-Fi network, you can open the device setup application, an HTML page hosted on the device's local web server, to configure the device settings. The device remains in setup mode for up to 2 hours, giving you enough time to finish configuring the device. When the setup is complete or has exceeded 2 hours, the device will exit setup mode and the Wi-Fi indicator stops blinking. The `AMDC-NNNN` network is then deactivated and your computer disconnected from that network and reconnected to its home or office network.

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

1. Look at the bottom of your device and make note of the device's `AMDC-NNNN` network SSID and password.



2. Plug in your AWS DeepLens device to an AC power outlet. Press the power button on the front of the device to turn the device on.

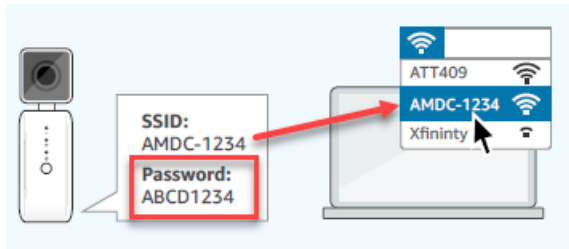


3. Wait until the device has entered into setup mode when the Wi-Fi indicator (middle LED) on the front of the device starts to flash.

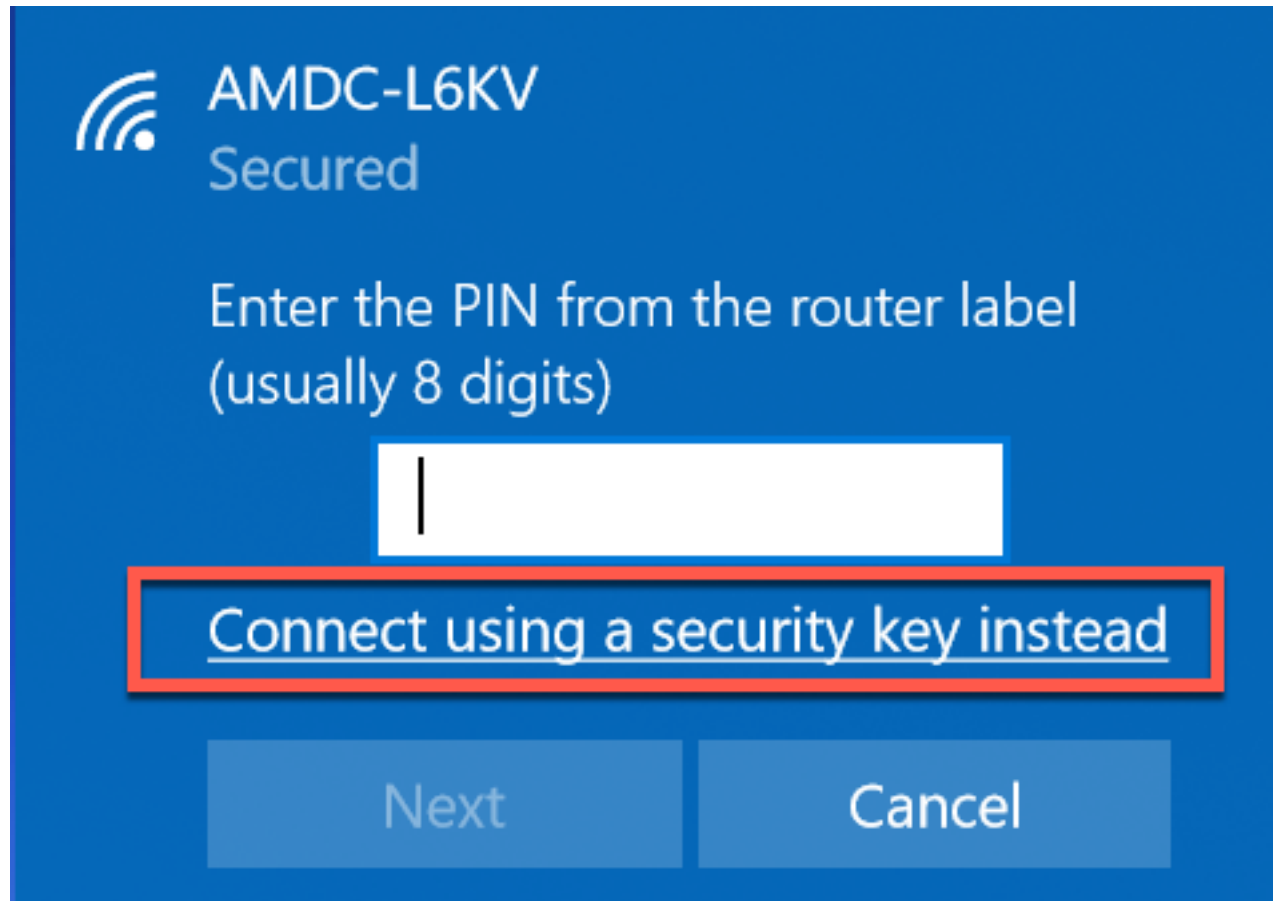
Note

If Wi-Fi indicator does not flash, the device is no longer in the setup mode. To turn on the device's setup mode again, press a paper clip into the reset pinhole on the back of the device. After you hear a click, wait about 20 seconds for the Wi-Fi indicator to blink.

4. Open the network management tool on your computer. Choose your device's SSID from the list of available Wi-Fi 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.



On a computer running Windows, choose **Connecting using a security key instead** instead of **Enter the PIN from the router label (usually 8 digits)** and then enter your device's Wi-Fi password.



After successfully connecting your computer to the device's Wi-Fi network, you're now ready to launch the device setup application to [configure your device \(p. 30\)](#).

To launch the device setup app, do one of the following:

- For the initial registration using the AWS DeepLens console, go back to the **Connect to your device** page and choose **Next**.

Step 1
Configure your AWS
account

Step 2
Connect to your device

Step 3
Configure your device

Connect to your device

Now that you have configured your AWS account, you will connect to network access and upload the security certificate to finish registration.

Connect your computer to your device's Wi-Fi

1. Plug in and power on

Plug in and press the power button to turn on your AWS DeepLens. The power indicator, the bottom LED, will turn on.

2. Verify middle LED is blinking

When the middle LED is blinking, AWS DeepLens is in setup mode. Note- this may take about 20 seconds after power on.



Middle LED is not blinking?

Make sure your AWS DeepLens is plugged in and powered on.

If the middle LED is still not blinking, you can manually reset the device by inserting a paperclip into the small hole marked 'reset' on the back of the device for about 20 seconds and the wireless network status will change to 'Setup'.

3. Connect your computer to the AWS DeepLens Wi-Fi

- For updating the device settings after the initial registration, open a web browser tab and enter `http://deeplens.amazon.net` or `http://deeplens.config` in the address bar.

Note

If the above URL doesn't work, your AWS DeepLens device may have the awscam software version 1.3.5 or earlier installed. In this case, update the device software and try it again. Alternatively, instead of `http://deeplens.amazon.net` or `http://deeplens.config`, you can open the device setup page by using one of the following URLs, depending on the software version on your AWS DeepLens device.

- `http://192.168.0.1`, if the AWS DeepLens software package (awscam) version is less than 1.2.4
- `http://10.105.168.217`, if the AWS DeepLens software package (awscam) version is greater than or equal to 1.2.4

For more information, see [Device Setup URL](#) (p. 120).

Connect to AWS DeepLens Device Using Monitor, Mouse and Keyboard

In addition to using a computer to connect to your AWS DeepLens device, you can also connect to the device directly using a monitor with a μ HDMI-to-HDMI cable, a USB keyboard and/or a USB mouse. You'll be prompted for the Ubuntu login credentials. Enter `aws_cam` in **username**. When logging to the device for the first time, use the default password of `aws_cam` in **password**. You'll then be asked to reset the password. For security reasons, use a strong password of a complex phrase. For any subsequent logins, use the reset password.

Set Up Your AWS DeepLens Device

When setting up your AWS DeepLens device, you perform the following tasks while your computer is connected to your device's `AMDC-NNNN` network:

- Enable the device's internet connection through your home or office wireless (Wi-Fi) or wired (Ethernet) network.
- Attach to your device the AWS-provisioned security certificate as discussed in [the section called "Register Your Device"](#) (p. 12).
- Set up login access to the device, including with an SSH connection or a wired connection.
- Optionally, download the streaming certificate from the device to enable viewing project video output in a supported browser.

If your computer is no longer a member of the `AMDC-NNNN` network because your AWS DeepLens device has exited setup mode, follow the instructions in [the section called "Connect to Device"](#) (p. 26) to establish a connection and to open the device setup page, again, before proceeding further.

To set up your AWS DeepLens device


1. If your AWS DeepLens device has a more recent version of the software installed, you won't be prompted with the following tasks. Skip to Step 2 below. Otherwise, proceed to the **Device setup** page, which you opened after [connecting to the device](#) (p. 26), to set up your home or office network to connect your device to the internet:
 - a. Under **Step 1: Connect to network**, choose your home or office Wi-Fi network SSID from the **Wi-Fi network ID** drop-down list. Type the Wi-Fi network password

Device setup

Step 1: Connect to network

Connect to your Wi-Fi network or use Ethernet via an Ethernet-USB adapter.

Wi-Fi network ID

 Haymuto WEP ▼

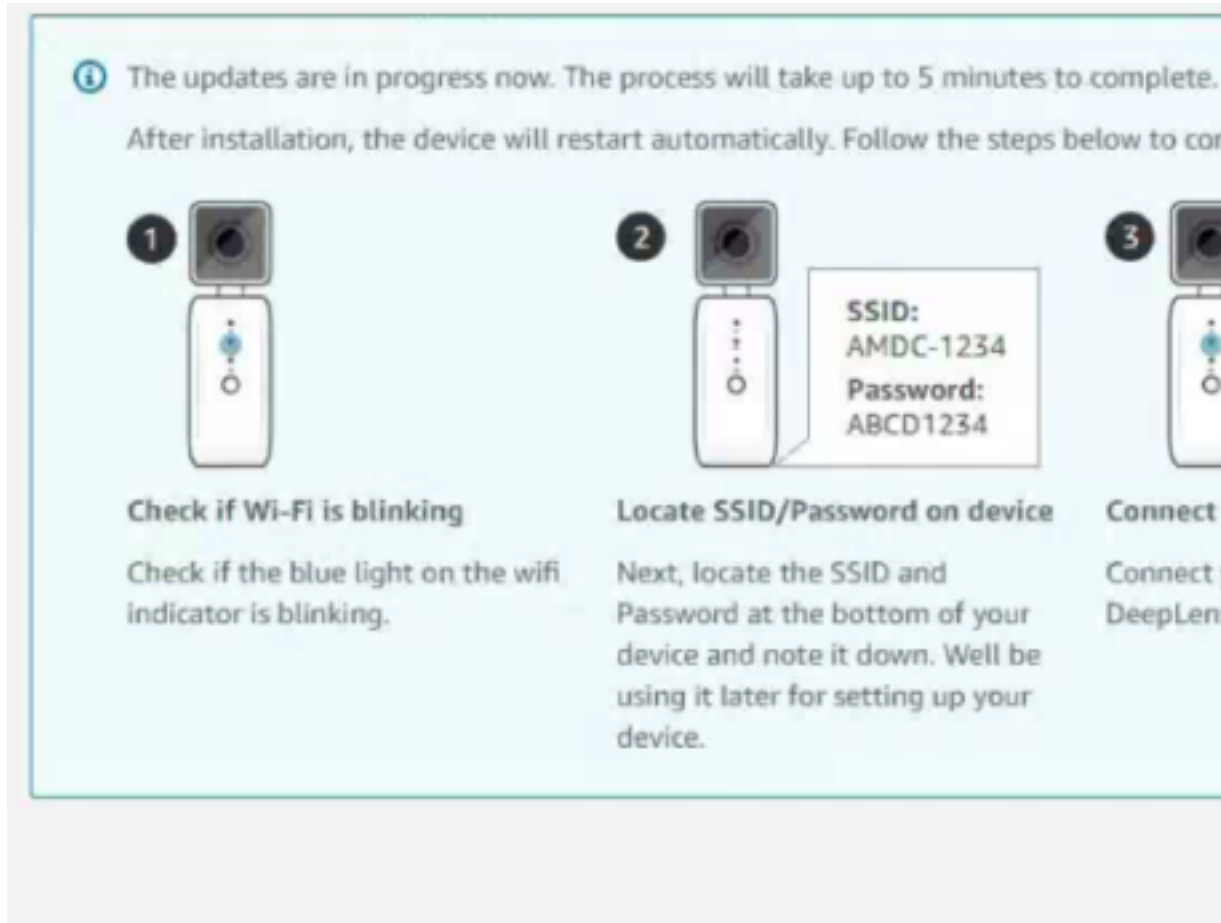
Wi-Fi password

☐ Show password

[Use Ethernet](#)

Alternatively, You can choose **Use Ethernet** to connect your AWS DeepLens device to the internet.


- b. Choose **Next** to connect the device to the internet.
- c. On the next page, Choose **Install and reboot** to install the latest software on the device and to restart the device.



After the software update is installed and the device restarted, reconnect your computer to your device's **AMDC-~~NNNN~~** network, navigate to <http://deeplens.config> in a web browser to open the updated device setup application, and to complete the rest of the device setup as shown in Step 2.

2. When your device has an updated software installed, starting the device setup application (e.g., <http://deeplens.config>) opens a browser page similar to the following:

deeplens.config/



Configure your AWS DeepLens

You are now connected to your AWS DeepLens. In this step, you will connect the device to your home or office network, set a password to access your device.

Connect your device to your home or office network

In order to complete device registration, you need to connect your device to the internet. You can use Wi-Fi or wired (adapter). If connecting to your office network, make sure AWS DeepLens can access the Internet through ports 8883

Wireless network SSID	Status
Haymuto	✔ Online
Wired connection	Status
Ethernet-USB Adapter	✔ Online

Edit

Upload security certificate to associate your AWS DeepLens to your AWS account

To enable your device to connect to AWS, browse to the security certificate that you downloaded previously. By default, the certificate is located in the Downloads directory. Upload the .zip file.

Attached certificate

✔ certificates-deeplens_h0_PUOccTzG4k2G26Aaw5Q.zip

Edit

Set a password to control how you access and update the device

You are required to setup a password to protect login access to your device. If you forget your password, you will lose access to your device. You can re-enable access by performing a factory reset, but you will lose any data stored on your device.

Device password	SSH server
*****	Enabled

Edit

The **Connect your device to your home or office network** section shows the device's internet connection status as **Online**. You can choose **Edit** to update the internet connection by choosing a different Wi-Fi network or a wired network with a micro-USB-to-Ethernet adaptor. Otherwise, continue to the next step to complete the rest of the device configuration.

3. Under **Upload security certificate to associate your AWS DeepLens to your AWS account** on the updated **Configure your AWS DeepLens** page, do the following:
 - a. Choose **Browse** to open a file picker.
 - b. Locate and choose the security certificate that you downloaded when [preparing your AWS account for AWS DeepLens \(p. 12\)](#),
 - c. Choose **Upload zip file** to attach the certificate to the device.

Note

The downloaded security certificate for the device is a .zip file. Upload the unzipped certificate file as-is.

For device setup update after the initial registration, your device has a previous certificate installed. In this case, choose **Edit** and follow the instructions above in this step to upload the new certificate.

4. Under **Set a password to control how you access and update the device**, complete the following steps to configure device access.
 - a. For the initial registration, type a password in **Create a password**. The password must be at minimum eight characters long and contain at least one number, an uppercase letter, and a special character (e.g., '*', '&', '#', '\$', '%', '@', or '!'). You need this password to log in to your device either using an SSH connection (if enabled below) or using a hardwired monitor, a USB mouse and/or a USB keyboard.
 - b. For **SSH server**, choose **Enable** or **Disable**. If enabled, SSH allows you to log in to your device using an SSH terminal on your Mac or Linux computer or using PuTTY or another SSH client on your Windows computer.

For subsequent configuration updates after the initial registration, you can choose **Edit** and follow the ensuing instructions to update the password.

5. Optionally, on the upper-right corner of the device setup page, choose **Enable video streaming** to enable viewing project video output using a supported browser. For the initial registration, we recommend that you skip this option and enable it later when updating the device configuration after you've become more familiar with AWS DeepLens.
6. Review the settings. Then, choose **Finish** to complete setting up the device and to terminate your connection to the device's Wi-Fi network. Make sure to connect your computer back to your home or office network.

Note

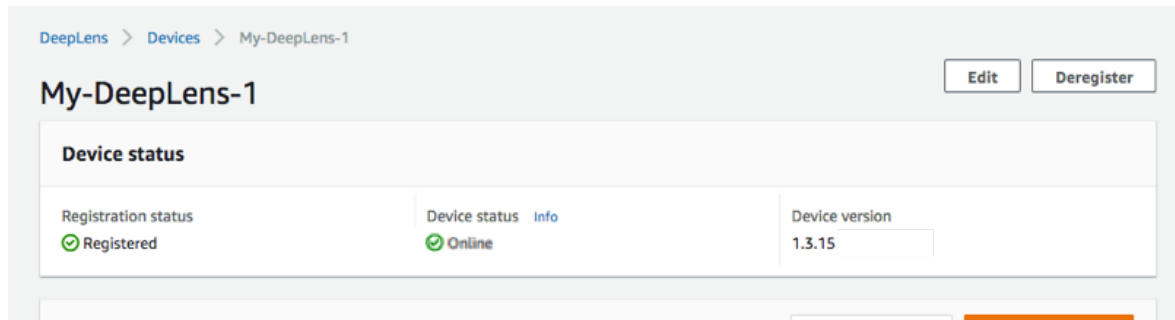
To ensure the setup completes successfully, make sure that AWS DeepLens device has access to ports 8883 and 8443 and is not blocked by your network firewall policy.

If the AWS DeepLens device's connection to the internet repeated turns on and off after you chose **Finish**, restart your home or office Wi-Fi network.

This completes the device registration. [Verify your device registration status \(p. 34\)](#) before moving on to create and deploy a project to your AWS DeepLens device.

Verify Your AWS DeepLens Device Registration Status

After the device setup is complete, choose **Open AWS DeepLens console** to verify the registration status.



Device registration takes a few minutes to process. When the registration completes successfully, **Registration status** changes to **Registered**. If the process fails, **Registration status** becomes **Failed**. In this case, choose **Deregister** and then start the registration all over again. When the registration is interrupted or otherwise incomplete, **Registration status** remains **Awaiting credentials**. In this case, [reconnect to the device's AMDC-NNNN Wi-Fi network \(p. 26\)](#) to resume the registration.

When the device's **Registration status** and **Device status** becomes **Registered** and **Online**, respectively, you're ready to create and deploy a project to your AWS DeepLens device. For example, you can follow these [instructions \(p. 35\)](#) to test your device by creating and deploying a sample project and inspecting a project output.

Test Using Your Registered AWS DeepLens Device

After you have registered and set up your AWS DeepLens device successfully, you can test it by creating and deploying an AWS DeepLens sample project to the device, and verifying the project's JSON output in the AWS IoT console.

Topics

- [Create and Deploy an AWS DeepLens Project \(p. 35\)](#)
- [View Your AWS DeepLens Project Output in the AWS IoT Console \(p. 35\)](#)

Create and Deploy an AWS DeepLens Project

An AWS DeepLens project consists of a deep learning model and a Lambda function that makes inferences of an input image based on the deep learning model.

For the testing purpose, you can deploy to your registered device a sample project with a pre-trained deep learning model and a tested inference Lambda function created by AWS DeepLens. For more information, see [Building Projects \(p. 38\)](#).

To use a sample project, follow the instructions given in [the section called "Create and Deploy a Sample Project in the Console" \(p. 55\)](#).

View Your AWS DeepLens Project Output in the AWS IoT Console

After a successful project deployment, your deployed AWS DeepLens project starts to run on your registered AWS DeepLens device. If the Lambda function of your project uses the AWS IoT Greengrass SDK (`greengrasssdk`) to publish text-based messages to AWS IoT, including JSON-formatted inference

results as is done in all of the AWS DeepLens sample projects, you can view such project output using the [AWS IoT Core console](#).

If the inference Lambda function uses the [OpenCV Library](#) (`cv2`) module to output inference results as video feeds, you can view the video output using a web browser or connecting to the device directly a monitor, mouse and keyboard. However, viewing text-based project output is more economical to verify that your deployed project runs as expected.

To view your AWS DeepLens project out in the AWS Management Console for AWS IoT Core

1. Open the AWS DeepLens console and make sure to choose the **US N. Virginia** region.
2. From the AWS DeepLens primary navigation pane, choose **Devices** and then choose your AWS DeepLens project.
3. From the **Device details** page of selected device, copy the **MQTT** topic value, of the `$aws/things/deeplens_<uuid>/infer` format. You will need to paste this AWS IoT topic ID in the AWS IoT console that you'll open next.
4. Open the [AWS Management Console for IoT Core](#) and make sure to choose **US North Virginia** region.
5. From the primary navigation pane, choose **Test**.
6. Under **Subscription topic** on the **MQTT client** page, paste the AWS IoT topic ID (copied in **Step 3** above) into the **Subscription topic** input field. Then, choose **Subscribe to topic**.
7. Inspect the display of the results that your inference Lambda function publishes by calling the `greengrasssdk.publish` method. The following AWS IoT console display shows example results published by the [Cat-and-dog-recognition sample project function](#) (p. 76):

\$aws/things/deeplens_OE1qzPjbTkuUYXS-ubal... Aug 9, 20

```
{  
  "dog": 0.10010205209255219,  
  "cat": 0.8998979926109314  
}
```

\$aws/things/deeplens_OE1qzPjbTkuUYXS-ubal... Aug 9, 20

```
{  
  "dog": 0.10232298076152802,  
  "cat": 0.8976770639419556  
}
```

\$aws/things/deeplens_OE1qzPjbTkuUYXS-ubal... Aug 9, 20

```
{  
  "dog": 0.10075829178094864,  
  "cat": 0.8992417454719543  
}
```

Building AWS DeepLens Projects

When your AWS DeepLens device is registered with and connected to the AWS Cloud, you can begin to create an AWS DeepLens project on the AWS Cloud and deploy it to run on the device. An AWS DeepLens project is a deep learning-based computer vision application. It consists of a deep learning model and a Lambda function to perform inference based on the model.

Before creating an AWS DeepLens project, you must have trained or have someone else trained a deep learning model using one of [the supported machine learning frameworks \(p. 38\)](#). The model can be trained using Amazon SageMaker or another machine learning environment. In addition, you must also have created and published an inference function in AWS Lambda. In this chapter, you'll learn how to train a computer vision deep learning model in Amazon SageMaker and how to create an inference Lambda function to make inferences and to implement other application logic.

To help you learn building your AWS DeepLens project, you have access to a set of AWS DeepLens sample projects. You can use the sample projects as-is to learn programming patterns for building a AWS DeepLens project. You can also use them as templates to extend their functionality. In this chapter, you'll learn more about these sample projects and how to employ one to run on your device, as well.

Topics

- [Machine Learning Frameworks Supported by AWS DeepLens \(p. 38\)](#)
- [Viewing AWS DeepLens Output Streams \(p. 45\)](#)
- [Working with AWS DeepLens Sample Projects \(p. 53\)](#)
- [Working with AWS DeepLens Custom Projects \(p. 73\)](#)
- [Building AWS DeepLens Project Tutorials \(p. 86\)](#)

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 \(p. 38\)](#)
- [Supported TensorFlow Models and Supporting TensorFlow Layers \(p. 40\)](#)
- [Supported Caffe Models and Supporting Caffe Layers \(p. 42\)](#)

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 \(p. 39\)](#)
- [Supporting MXNet Layers \(p. 39\)](#)

Topics

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.

Supported Gluon Models	
AlexNet	Image classification model trained on the ImageNet dataset imported from the Open Neural Network Exchange (ONNX).
MobileNet	Image classification model trained in TensorFlow using the RMSprop optimizer.
ResNet	Image classification model trained on the ImageNet dataset imported from MXNet.
SqueezeNet	Image classification model trained on the ImageNet dataset imported from ONNX.
VGG	Image classification model trained on the ImageNet dataset imported from MXNet or ONNX.

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.

```
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.

Supported MXNet Modeling Layers	
Activation	Applies an activation function to the input

Supported MXNet Modeling Layers	
BatchNorm	Applies batch normalization
Concat	Joins input arrays along a given axis
_contrib_MultiBoxDetection	Converts a multibox detection prediction
_contrib_MultiBoxPrior	Generates prior boxes from data, sizes, and ratios
Convolution	Applies a convolution layer on input
Deconvolution	Applies a transposed convolution on input
elemwise_add	Applies element-wise addition of arguments
Flatten	Collapses the higher dimensions of an input into an 2-dimensional array
FullyConnected	Applies a linear transformation of $Y = WX + b$ on input x
InputLayer	Specifies the input to a neural network
L2Norm	Applies L2 normalization to the input array
LRN (Local Response Normalization)	Applies local response normalization to the input array
Pooling	Performs pooling on the input
Reshape	Reshapes the input array with a different view without changing the data
ScaleShift	Applies scale and shift operations on input elements
SoftmaxActivation	Applies Softmax activation to the input
SoftmaxOutput	Computes the gradient of cross-entropy loss with respect to Softmax output
transpose	Permutes the dimensions of an array
UpSampling	Performs nearest-neighbor or bilinear upsampling to input
_mul	Performs multiplication
_Plus	Performs an element-wise sum of the input arrays with broadcasting

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 \(p. 41\)](#)

- [Supporting TensorFlow Layers \(p. 41\)](#)

Supported TensorFlow Models

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

Supported TensorFlow Models	
Inception	An image classification model trained on the ImageNet dataset using TensorFlow
MobileNet	An image classification model trained on the ImageNet dataset using TensorFlow
NasNet	An image classification model trained on the ImageNet dataset using TensorFlow
ResNet	An image classification model trained on the ImageNet dataset using TensorFlow
VGG	An image classification model trained on the ImageNet dataset using TensorFlow

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.

Supported TensorFlow Layers	
Add	Computes element-wise addition
AvgPool	Performs average pooling on the input
BatchToSpaceND	Rearranges data from batch into blocks of spatial data
BiasAdd	Adds bias
Const	Creates a constant tensor
Conv2D	Computes a 2-D convolution
Conv2DBackpropInput	Computes the gradients of convolution with respect to the input
Identity	Returns a tensor with the same shape and contents as input
Maximum	Computes element-wise maximization.
MaxPool	Performs the max pooling on the input
Mean	Computes the mean of elements across dimensions of a tensor
Mul	Computes element-wise multiplication
Neg	Computes numerical negative value element-wise

Supported TensorFlow Layers	
Pad	Pads a tensor
Placeholder	Inserts a placeholder for a tensor that will be always fed
Prod	Computes the product of elements across dimensions of a tensor
RandomUniform	Outputs random values from a uniform distribution
Range	Creates a sequence of numbers
Relu	Computes rectified linear activations
Reshape	Reshapes a tensor
Rsqrt	Computes reciprocal of square root
Shape	Returns the shape of a tensor
Softmax	Computes Softmax activations
SpaceToBatchND	Zero-pads and then rearranges blocks of spatial data into batch
Square	Computes element-wise square
Squeeze	Removes dimensions of size 1 from the shape of a tensor
StopGradient	Stops gradient computation
Sub	Computes element-wise subtraction
Sum	Computes the sum of elements across dimensions of a tensor
Tile	Constructs a tensor by tiling a given tensor

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 \(p. 42\)](#)
- [Supporting Caffe Layers \(p. 43\)](#)

Supported Caffe Models

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

Supported Caffe Models	
AlexNet	An image classification model trained on the ImageNet dataset using Caffe
DenseNet	An image classification model from the original Torch model

Supported Caffe Models	
Inception	An image classification model converted from the original Apache MXNet model
MobileNet	An image classification model trained on the ImageNet dataset using Caffe
ResNet	An image classification model trained on the ImageNet dataset using Caffe
SqueezeNet	An image classification model trained on the ImageNet dataset using Caffe
VGG	An image classification model trained on the ImageNet dataset using Caffe

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.

Supported Caffe Layers	
BatchNorm	Normalizes the input to have 0-mean and/or unit variance across the batch
Concat	Concatenates input blobs
Convolution	Convolves the input with a bank of learned filters
Deconvolution	Performs in the opposite sensor of the Convolution layer
Dropout	Performs dropout
Eltwise	Performs element-wise operations, such as product and sum, along multiple input blobs
Flatten	Reshapes the input blob into flat vectors
InnerProduct	Computes an inner product
Input	Provides input data to the model
LRN (Local Response Normalization)	Normalizes the input in a local region across or within feature maps
Permute	Permutates the dimensions of a blob
Pooling	Pools the input image by taking the max, average, etc., within regions
Power	Computes the output as $(\text{shift} + \text{scale} * x)^{\text{power}}$ for each input element x
ReLU	Computes rectified linear activations

Supported Caffe Layers	
Reshape	Changes the dimensions of the input blob, without changing its data
ROIPooling	Applies pooling for each region of interest
Scale	Computes the element-wise product of two input blobs
Slice	Slices an input layer to multiple output layers along a given dimension
Softmax	Computes the Softmax activations
Tile	Copies a blob along specified dimensions

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 Video Stream from AWS DeepLens Device in Browser.”](#) (p. 46). 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

- [View Video Streams from AWS DeepLens 2019 Edition Device in Browser](#) (p. 45)
- [View Video Streams from AWS DeepLens Device in Browser](#) (p. 46)
- [View Video Streams on Your AWS DeepLens Device](#) (p. 48)
- [Creating a Lambda Function for Viewing the Project Stream](#) (p. 50)

View Video Streams from AWS DeepLens 2019 Edition Device in Browser

To view project or live streams from the AWS DeepLens 2019 Edition device in a browser, you must have the device's self-signed streaming certificate downloaded to your computer and then uploaded to a supported browser you'll use. For instructions on how to download the streaming certificate and install it into the browser, see **Step 6** of [View or Update Your AWS DeepLens 2019 Edition Device Settings](#) (p. 22).

To view video streams from AWS DeepLens 2019 Edition device in a browser

1. Connect the device to your computer with the provided USB cable, if it's not already connected.
2. Do one of the following to launch your devices' **AWS DeepLens Stream** viewer:
 - Open the device details page on the AWS DeepLens console and choose **View video stream** under **Video streaming** to open the stream viewer in a new browser tab.
 - Alternatively, open a new browser tab manually, type `https://your-device-ip-address:4000` in the address bar, where *your-device-ip-address* stands for the IP address of your device (e.g., 192.168.14.78) shown under **Device details** in the AWS DeepLens console.
3. Because the streaming certificate is self-signed, the browser will display a warning. To accept the warning, follow browser-specific instructions on screen to accept the certificate for the stream viewer to be launched.

It may take about 30 seconds for the stream viewer to show up.

4. Before any project is deployed, choose **Live stream** to view live videos from your AWS DeepLens device. After a project is deployed, you can also choose **Project stream** to view processed videos output from the device.
5. When done with viewing video streams, disconnect the device from your computer to return the computer to its normal network configuration.

View Video Streams from AWS DeepLens Device in Browser

Note

To view a project's output 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" \(p. 105\)](#).

Note

To view a project's output in Chrome on Mac El Capitan or earlier, you must provide a password to load the streaming certificate for viewing the project output. If you have such an old Mac operating system and want to use Chrome, skip the procedure below and follow the instructions in [How to View Project Output in the Chrome Browser on Mac El Capitan or Earlier? \(p. 118\)](#) to set up your browser to view the project output.

To view project output from your AWS DeepLens device in a supported web browser

1. If you have not already downloaded the streaming certificate when [registering your device \(p. 30\)](#) or if you have lost the downloaded copy, do one of the following to download the streaming certificate. The steps are different depending on whether you use the device setup application or not.

- a. Download the streaming certificate without using the device setup page:

- i. Establish an SSH connection to your AWS DeepLens device from your computer:

```
ssh aws_cam@device_local_ip_address
```

An example of `device_local_ip_address` would be `192.168.0.47`. After entering the correct device password, which you specified during the device registration, you're now logged in to the device.

- ii. Make a local copy (`my_streaming_cert.pfx`) of the streaming certificate (`client.pfx`) on the device:

```
sudo cp /opt/awscam/awsmedia/certs/client.pfx /home/aws_cam/  
my_streaming_cert.pfx
```

When prompted, enter the device password to complete the above commands.

Make sure that the owner of the device-local copy is `aws_cam`.

```
sudo chown aws_cam /home/aws_cam/my_streaming_cert.pfx
```

After this, log out of the device:

```
exit
```

- iii. Transfer the copy of the streaming certificate from the device to your computer:

```
scp aws_cam@device_local_ip_address:/home/aws_cam/my_streaming_cert.pfx ~/Downloads/
```

The example used the `Downloads` folder under the user's home directory (`~/`) to transfer the certificate to. You can choose any other writeable directory as the destination folder.

- b. Follow the following steps to download the streaming certificate for the AWS DeepLens device:
 - i. [Return your device to its setup mode, if necessary, and connect your computer to the device's **ADMC-NNNN** Wi-Fi network \(p. 26\).](#)
 - ii. Start the device setup app at <http://deeplens.config>.
 - iii. Follow the on-screen instructions to download the streaming certificate for viewing a project's output streams.
2. Import into your supported web browser the streaming certificate you downloaded during the [device registration \(p. 30\)](#).
 - a. For FireFox (Windows and macOS 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, if your AWS DeepLens software version is 1.3.23 or higher, type DeepLens in the password input field. If your AWS DeepLens software version is 1.3.22 or lower, leave the password field blank and follow the on-screen instruction to finish importing the streaming certificate.

Note

Depending on the version of the FireFox you use, you may need to follow steps below, instead:

1. From **Preferences**, choose **Advanced**.
 2. Choose the **Certificates** tab.
 3. Choose **View Certificates**.
 4. On **Certificate Manager**, choose the **Your certificates** tab.
 5. Choose **Import**.
 6. Navigate to and open the downloaded streaming certificate.
 7. When prompted for a password, if your AWS DeepLens software version is 1.3.23 or higher, type DeepLens in the password input field. If your AWS DeepLens software version is 1.3.22 or lower, choose **OK** without entering one.
- b. For Chrome (macOS Sierra or higher), follow these steps:
 - i. On your macOS, 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, if your AWS DeepLens software version is 1.3.23 or higher, type DeepLens in the password input field. If your AWS DeepLens software version is 1.3.22 or lower, 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, if your AWS DeepLens software version is 1.3.23 or higher, type `DeepLens` in the password input field. If your AWS DeepLens software version is 1.3.22 or lower, leave the password field blank and follow the on-screen instructions to finish importing the streaming certificate.
3. To view output streams, open a supported browser window and navigate 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.

You can also use the AWS DeepLens console to view the project stream. To do so, follow the steps below:

- a. In the navigation pane, choose **Devices**.
- b. From the **Devices** page, choose your AWS DeepLens device.
- c. Under **Project output** on your device's details page, expand the **View the video output** section.
- d. Choose a supported browser from **Select a browser to import the streaming certificate** drop-down list and, if necessary, follow the instructions.
- e. Choose **View stream**, while your computer is connected to the device's Wi-Fi (AMDC-**NNNN**) network.

View Video Streams on Your AWS DeepLens Device

In addition to [viewing your AWS DeepLens output streams in a browser \(p. 46\)](#), 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

- [View Live Streams on Your AWS DeepLens Device \(p. 48\)](#)
- [View Project Streams on Your AWS DeepLens Device \(p. 49\)](#)

View Live Streams 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:

```
mplayer -demuxer lavf /opt/awscam/out/ch1_out.h264
```

6. To stop viewing the video stream and end your terminal session, press `Ctrl+C`.

View Project Streams 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:

```
mpplayer -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. <https://docs.aws.amazon.com/deeplens/latest/dg/>

```
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'])
_, frame = awscam.getLastFrame()
_, 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
        outMap = ({ 1: 'aeroplane', 2: 'bicycle', 3: 'bird', 4: 'boat',
                    5: 'bottle', 6: 'bus', 7: 'car', 8: 'cat',
                    9: 'chair', 10: 'cow', 11: 'dining table',
                    12: 'dog', 13: 'horse', 14: 'motorbike',
                    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},'.format(outMap[obj['label']], obj['prob'] )
            label_show = "{: {:.2f}%".format(outMap[obj['label']],
obj['prob']*100 )
            cv2.putText(frame, label_show, (xmin,
ymin-15),cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 165, 20), 4)
            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 Video Stream from AWS DeepLens Device in Browser." \(p. 46\)](#) or [View Project Streams on Your AWS DeepLens Device \(p. 49\)](#) to view the processed project stream.

Working with AWS DeepLens Sample Projects

When your AWS DeepLens device is registered and connected to the AWS Cloud, you're ready to create an AWS DeepLens project and deploy it to the device to run computer vision application on. An AWS DeepLens project is made up of an AWS DeepLens model and the associated AWS Lambda function to run inference with. You can use a pre-trained model in your project or you can train a custom model.

AWS DeepLens provides several sample projects for you to deploy and run inference on right out of the box. Here, you'll learn how to use a sample project and deploy it to your AWS DeepLens device. Before we start, let's get an overview of the sample projects.

Topics

- [AWS DeepLens Sample Projects Overview \(p. 53\)](#)
- [Create and Deploy an AWS DeepLens Sample Project in the AWS DeepLens Console \(p. 55\)](#)
- [Relay an AWS DeepLens Project Output through AWS SMS \(p. 57\)](#)
- [Use Amazon SageMaker to Provision a Pre-trained Model for a Sample Project \(p. 63\)](#)

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.

Note

When deploying an Amazon SageMaker-trained SSD model, you must first run `deploy.py` (available from <https://github.com/apache/incubator-mxnet/tree/master/example/ssd/>) to

convert the model artifact into a deployable mode. After cloning or downloading [the MXNet repository](#), run the `git reset --hard 73d88974f8bca1e68441606fb0787a2cd17eb364` command before calling `deploy.py` to convert the model, if the latest version does not work.

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 [Use Amazon SageMaker to Provision a Pre-trained Model for a Sample Project](#) (p. 63)

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](#)

Bird Classification

This project makes prediction of the top 5 bird species from a static bird photo captured by the AWS DeepLens camera.

This project uses the [ResNet-18](#) neural network architecture to train the model with the [CUB-200 dataset](#). The trained model can identify 200 different bird species. Because the number of categories are large, the project outputs only the top 5 most probable inference results. To reduce the background noise for improved precision, a cropped zone located at the middle of the camera image is used for inference. You can view the cropped zone from the project video streaming. By positioning the static bird photo in the box zone, inference results are illustrated on top left of the project view.

- **Project model:** deeplens-bird-detection
- **Project function:** deeplens-bird-detection

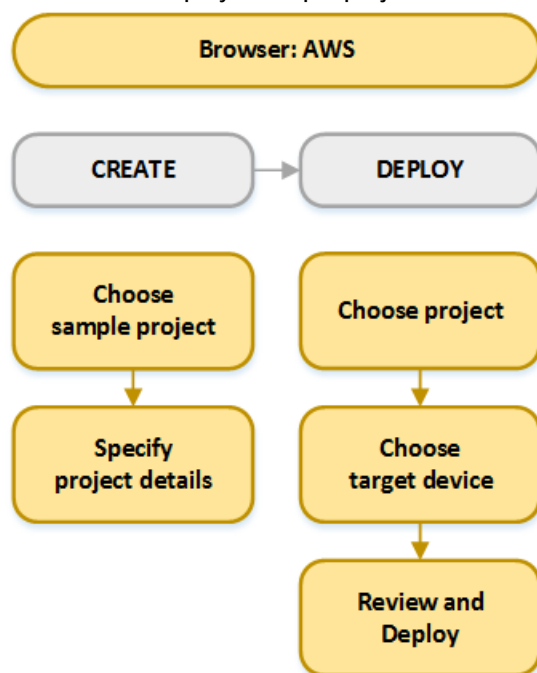
Create and Deploy an AWS DeepLens Sample Project in the AWS DeepLens Console

In this walkthrough, you'll use the AWS DeepLens console to create an AWS DeepLens project from the [Object Detection sample project template \(p. 53\)](#) to create an AWS DeepLens project. The pre-trained

object detection model can analyze images from a video stream captured on your AWS DeepLens device and identify an objects as one of as many as 20 labeled image types. The instructions presented here apply to creating and deploying other AWS DeepLens sample project.

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 inference function. You can find the information for the individual projects in [AWS DeepLens Sample Projects Overview \(p. 53\)](#).

The following diagram presents a high-level overview of the processes to use the AWS DeepLens console to create and deploy a sample project.



Create and Deploy Your Project

Follow the steps below to create and deploy the Object Detection sample project.

To create and deploy an AWS DeepLens sample project using the AWS DeepLens console

1. Open the AWS DeepLens console by navigating to <https://console.aws.amazon.com/deeplens/> from a web browser.
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. Choose **Create**.

This returns you to the **Projects** screen where the project you just created is listed with your other projects.

5. On the **Projects** screen, choose the radio button to the left of your project name or choose the project to open the project details page, then choose **Deploy to device**.
6. 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.
7. 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.

8. 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 \(p. 45\)](#).

Relay an AWS DeepLens Project Output through AWS SMS

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 [Use Amazon SageMaker to Provision a Pre-trained Model for a Sample Project \(p. 63\)](#)) to use the AWS IoT rules engine and an AWS Lambda function.

Topics

- [Create and Configure the Lambda Function \(p. 57\)](#)
- [Disable the AWS IoT Rule \(p. 62\)](#)

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, **<your name>_hotdog_notifier**.
5. For **Permissions**, choose the **Create a new Role from AWS policy templates** under **Execution role**.
6. Type a name for the role; for example, **<your name>_hotdog_notifier**.
7. For **Policy Templates**, choose **SNS Publish policy** and **AWS IoT Button permissions**.

Basic information

Function name

Enter a name that describes the purpose of your function.

mytest-hotdog-notifier

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)

Choose the language to use to write your function.

Node.js 10.x

Permissions [Info](#)


Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and

▼ Choose or create an execution role

Execution role

Choose a role that defines the permissions of your function. To create a custom role, go to the [IAM console](#).

Create a new role from AWS policy templates

 Role creation might take a few minutes. The new role will be scoped to the current function. To u

Role name

Enter a name for your new role.

mytest-hotdog-notifier-role

Use only letters, numbers, hyphens, or underscores with no spaces.

Policy templates [Info](#)

Choose one or more policy templates.

AWS IoT Button permissions 
SNS

Amazon SNS publish policy 
SNS

8. Choose **Create function**.

Add an AWS IoT Rule

After the Lambda function is created successfully, you need to set up an AWS IoT rule to trigger the action you specify in your Lambda function (the next step) when an event occurs in the data source. To set up the rule, follow the steps below to add an AWS IoT trigger to the function.

1. Choose **+ Add trigger**. You may need expand the **Design** section in the Lambda console, if it's not already expanded.
2. Choose **AWS IoT** under **Trigger configuration**.
3. For **IoT type**, choose **Custom IoT rule**.
4. For **Rule**, choose **Create a new rule**.
5. For **Rule name**, type a name (**<your-name>_search_hotdogs**).
6. Optionally, give a description for the rule under **Rule description**.
7. Under **Rule query statement**, type into the box an AWS IoT topic query statement of the following format, replacing the red text with the AWS IoT topic for your AWS DeepLens.

```
Select Hotdog from '/$aws/deeplens/<aws/things/  
deeplens_5e6d406g-2bf4-4444-9d4f-4668f7366855/infer'
```

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

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

To find the AWS IoT topic for your AWS DeepLens, navigate to **Devices** on your AWS DeepLens, choose your device, then scroll to the bottom of the device detail page.

8. Toggle on the **Enable trigger** option.
9. Choose **Add** to finish creating the AWS 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 JavaScript 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.

```
/**  
 * 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:

Notification by	Key	Value
Email	email	Your complete email address. Example: YourAlias@ISP.com
Phone	phone_number	Your phone number with country code. Example: US +1 8005551212

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

1. Navigate to the AWS IoT console at <https://console.aws.amazon.com/iot>.
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 [Create and Deploy an AWS DeepLens Sample Project in the AWS DeepLens Console \(p. 55\)](#).

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**.

Use Amazon SageMaker to Provision a Pre-trained Model for a Sample Project

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 \(p. 64\)](#)
- [Step 2: Create an Amazon SageMaker Notebook Instance \(p. 65\)](#)
- [Step 3: Edit the Model in Amazon SageMaker \(p. 66\)](#)
- [Step 4: Optimize the Model \(p. 67\)](#)
- [Step 5: Import the Model \(p. 68\)](#)
- [Step 6: Create an Inference Lambda Function \(p. 69\)](#)
- [Step 7: Create a New AWS DeepLens Project \(p. 70\)](#)
- [Step 8: Review and Deploy the Project \(p. 71\)](#)
- [Step 9: View Your Model's Output \(p. 72\)](#)

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/>.
2. Choose **Create bucket**.
3. On the **Name and region** tab:
 - a. Name the bucket **deeplens-sagemaker-*<some_string>***, where *<some_string>* can be any string to make the bucket name unique across AWS. An example would be your account name or ID.

Note

The bucket name must begin with **deeplens-sagemaker-** to work with the default access policies. Otherwise, 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 **Configure options** tab, choose **Next**.
 5. On the **Set permissions** tab, choose **Grant Amazon S3 Log Delivery group write access to this bucket** from the **Manage system permissions** drop-down menu, then choose **Next**.
 6. On the **Review** tab, review your settings then choose **Create bucket**.
 7. On the created bucket's page, choose the **Overview** tab and then choose **Create folder**.
 8. Name the folder *test* then choose **Save**.

To use the folder with the example notebook instance to be created next, you must name the folder as *test*, unless you change the default folder name in the notebook.

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 **Notebook instances** from the navigation pane and then choose **Create notebook instance**.
3. On the **Create notebook instance** page, then do the following:
 - a. Under **Notebook instance settings**, type a name for the notebook in the **Notebook instance name** input field; for example, `<your-name>-hotdog`.
 - b. For **Notebook instance type**, choose `ml.t2.medium`.
 - c. For **Permissions and encryption**, under **IAM role**, choose **Create a new role**, if this is the first time you run a notebook, type `deeplens-sagemaker` in the **Specific S3 buckets** input field, and then choose **Create role**.

After you've created the first notebook instance for AWS DeepLens, you can choose an available IAM role from the **Use existing role** list.

If you've already [created the AWSDeepLensSageMaker role \(p. 12\)](#) as part of the setup, 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. You can find the ARN of your Amazon SageMaker role as follows:

- 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. All other settings are optional. Skip them for this exercise.

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. Make sure 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 Jupiter**.
5. Open a new tab in your browser and navigate to <https://github.com/aws-samples/reinvent-2017-deeplens-workshop>.
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 kernel is `conda_mxnet_p36`. If it isn't, change the kernel.
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** 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 [Optimize a Custom Model \(p. 75\)](#).

Step 5: Import the Model

Import the edited model into AWS DeepLens.

1. Open the AWS DeepLens console by navigating to <https://console.aws.amazon.com/deeplens/> from a web browser.
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 [Create and Publish an AWS DeepLens Inference Lambda Function \(p. 76\)](#).

Step 7: Create a New AWS DeepLens Project

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

1. Open the AWS DeepLens console by navigating to <https://console.aws.amazon.com/deeplens/> from a web browser.
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, for example, **deeplens-hotdog-no-hotdog**, then choose **Add function**.
9. Choose **Create**.

Step 8: Review and Deploy the Project

1. Open the AWS DeepLens console by navigating to <https://console.aws.amazon.com/deeplens/> from a web browser.
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 \(p. 45\)](#).

Working with AWS DeepLens Custom Projects

After you've explored one or more sample projects, you may want to create and deploy your own AWS DeepLens projects. It means that you'll need to do all or most of the following.

- Train a custom deep learning model either in Amazon SageMaker or elsewhere.
- Import the model into AWS DeepLens.
- Create and publish an inference function.
- Create a AWS DeepLens project and add the model and the inference function to it.
- Deploy the project to your AWS DeepLens device.

We'll walk you through these tasks in this section.

Topics

- [Import Your Amazon SageMaker Trained Model \(p. 73\)](#)
- [Import an Externally Trained Model \(p. 73\)](#)
- [Optimize a Custom Model \(p. 75\)](#)
- [Create and Publish an AWS DeepLens Inference Lambda Function \(p. 76\)](#)
- [Create and Deploy a Custom AWS DeepLens Project \(p. 83\)](#)
- [Use Amazon SageMaker Neo to Optimize Inference on AWS DeepLens \(p. 84\)](#)

Import Your Amazon SageMaker Trained Model

To use your Amazon SageMaker trained model for your computer vision application, you must import it into AWS DeepLens.

To import your Amazon SageMaker trained model into AWS DeepLens

1. Open the AWS DeepLens console at <https://console.aws.amazon.com/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**.

Import an Externally Trained Model

To use an externally trained model, import it.

To import an externally trained model

1. Sign in to the AWS Management Console for AWS DeepLens at <https://console.aws.amazon.com/deeplens/>.
2. In the navigation pane, choose **Models**, then choose **Import model**.
3. In the **Import source** section, 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 hyphens.
 - c. Optional. For **Description**, type a description of the model.
 - d. Choose **Import model**.

Optimize a Custom Model

AWS DeepLens uses the Computer Library for Deep Neural Networks (cLDNN) to leverage the GPU for performing inferences. If your model artifacts are not compatible with the cLDNN format, you must convert your model artifacts by calling the model optimization (mo) module. The process is known as model optimization. Model artifacts output by MXNet, Caffe or TensorFlow are not optimized.

To optimize your model artifacts, call the [mo.optimize \(p. 131\)](#) method as illustrated in the following code:

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

To load a model artifact for inference in your inference Lambda function, specify the model artifact path on the device. For an unoptimized model artifact, you must use the `model_path` returned from the above method call.

Models are deployed to the `/opt/awscam/artifacts` directory on the device. An optimized model artifact is serialized into an XML file and you can assign the file path directly to the `model_path` variable. For an example, see [the section called "Create and Publish an Inference Lambda Function" \(p. 76\)](#).

For MXNet model artifacts, the model optimizer can convert `.json` and `.params` files. For Caffe model artifacts, the model optimizer can take `.prototxt` or `.caffemodel` files. For TensorFlow model artifacts, the model optimizer expects the frozen graph (`.pb`) file.

For more information about the model optimizer, see [Model Optimization \(mo\) Module \(p. 131\)](#).

Create and Publish an AWS DeepLens Inference Lambda Function

Besides importing your custom model, you must create and publish an inference Lambda function to make inference of image frames from the video streams captured by your AWS DeepLens device, unless an existing and published Lambda function meet your application requirements.

Because the AWS DeepLens device is an [AWS IoT Greengrass core](#) device, the inference function is run on the device in a Lambda runtime as part of the AWS IoT Greengrass core software deployed to your AWS DeepLens device. As such, you create the AWS DeepLens inference function as an AWS Lambda function.

To have all the essential AWS IoT Greengrass dependencies included automatically, you can use the Lambda function blueprint of `greengrass-hello-world` as a starting point to create the AWS DeepLens inference function.

The engine for AWS DeepLens inference is the [awscam module \(p. 125\)](#) of the [AWS DeepLens device library \(p. 125\)](#). Models trained in a supported framework must be optimized to run on your AWS DeepLens device. Unless your model is already optimized, you must use the model optimization module ([mo \(p. 131\)](#)) of the device library to convert framework-specific model artifacts to the AWS DeepLens-compliant model artifacts that are optimized for the device hardware.

In this topic, you learn how to create an inference Lambda function that performs three key functions: preprocessing, inference, and post processing. For the complete function code, see [The Complete Inference Lambda Function \(p. 81\)](#).

To create and publish an inference Lambda function for your AWS DeepLens project

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, for example, `deeplens-my-object-inferer`. The function name must start with `deeplens`.
 - 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, make 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.

Here, we use the Cat and Dog Detection project function as an example.

9. Follow the steps below to add code for your Lambda function to the code editor for the `greengrassHelloWorld.py` file.
 - a. Import dependent modules:

```
from threading import Thread, Event
```



```
import os
import json
import numpy as np
import awscam
import cv2
import greengrasssdk
```

```
from threading import Thread, Event
import os
import json
import numpy as np
import awscam
import cv2
import greengrasssdk
```

where,

- The `os` module allows your Lambda function to access the AWS DeepLens device operating system.
 - The `json` module lets your Lambda function work with JSON data.
 - The `awscam` module allows your Lambda function to use the [AWS DeepLens device library \(p. 125\)](#). For more information, see [Model Object \(p. 126\)](#).
 - The `mo` module allows your Lambda function to access the AWS DeepLens model optimizer. For more information, see [Model Optimization \(mo\) Module \(p. 131\)](#).
 - The `cv2` module lets your Lambda function access the [Open CV](#) library used for image preprocessing, including local display of frames from video feeds originating from the device.
 - The `greengrasssdk` module exposes the AWS IoT Greengrass API for the Lambda function to send messages to the AWS Cloud, including sending operational status and inference results to AWS IoT.
 - The `threading` module allows your Lambda function to access Python's multi-threading library.
- b. Append the following Python code as a helper class for local display of the inference results:

```
class LocalDisplay(Thread):
    """ Class for facilitating the local display of inference results
    (as images). The class is designed to run on its own thread. In
    particular the class dumps the inference results into a FIFO
    located in the tmp directory (which lambda has access to). The
    results can be rendered using mplayer by typing:
    mplayer -demuxer lavf -lavfdopts format=mjpeg:probesize=32 /tmp/
    results.mjpeg
    """
    def __init__(self, resolution):
        """ resolution - Desired resolution of the project stream """
        # Initialize the base class, so that the object can run on its own
        # thread.
        super(LocalDisplay, self).__init__()
        # List of valid resolutions
        RESOLUTION = {'1080p' : (1920, 1080), '720p' : (1280, 720), '480p' : (858,
480)}
        if resolution not in RESOLUTION:
            raise Exception("Invalid resolution")
        self.resolution = RESOLUTION[resolution]
        # Initialize the default image to be a white canvas. Clients
        # will update the image when ready.
        self.frame = cv2.imencode('.jpg', 255*np.ones([640, 480, 3]))[1]
```

```
self.stop_request = Event()

def run(self):
    """ Overridden method that continually dumps images to the desired
        FIFO file.
    """
    # Path to the FIFO file. The lambda only has permissions to the tmp
    # directory. Pointing to a FIFO file in another directory
    # will cause the lambda to crash.
    result_path = '/tmp/results.mjpeg'
    # Create the FIFO file if it doesn't exist.
    if not os.path.exists(result_path):
        os.mkfifo(result_path)
    # This call will block until a consumer is available
    with open(result_path, 'w') as fifo_file:
        while not self.stop_request.isSet():
            try:
                # Write the data to the FIFO file. This call will block
                # meaning the code will come to a halt here until a consumer
                # is available.
                fifo_file.write(self.frame.tobytes())
            except IOError:
                continue

def set_frame_data(self, frame):
    """ Method updates the image data. This currently encodes the
        numpy array to jpg but can be modified to support other encodings.
        frame - Numpy array containing the image data of the next frame
        in the project stream.
    """
    ret, jpeg = cv2.imencode('.jpg', cv2.resize(frame, self.resolution))
    if not ret:
        raise Exception('Failed to set frame data')
    self.frame = jpeg

def join(self):
    self.stop_request.set()
```

The helper class (`LocalDisplay`) is a subclass of `Thread`. It encapsulates the process to stream processed video frames for local display on the device or using a web browser:

1. Exposes a constructor (`__init__(self, resolution)`) to initiate the `LocalDisplay` class with the specified image size (`resolution`).
2. Overrides the `run` method, which will be invoked by the `Thread.start` method, to support continuous writing images to the specified file (`result_path`) on the device. The Lambda function can write to files only in the `/tmp` directory. To view the images in this file (`/tmp/results.mjpeg`), start `mplayer` on a device terminal window as follows:

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

3. Exposes the `set_frame_data` method for the main inference function to call to update image data by encoding a project stream frame to JPG (`set_frame_data`). The example code can be modified to support other encodings.
 4. Exposes the `join` method to turn waiting threads active by setting the `Event` object's internal flag to true.
- c. Append to the code editor the following Python code that initializes looping through the inference logic, frame by frame:

```
def infinite_infer_run():
    """ Entry point of the lambda function"""
    try:
```

```
# This cat-dog model is implemented as binary classifier, since the number
# of labels is small, create a dictionary that converts the machine
# labels to human readable labels.
model_type = 'classification'
output_map = {0: 'dog', 1: 'cat'}

# Create an IoT client for sending to messages to the cloud.
client = greengrasssdk.client('iot-data')
iot_topic = '$aws/things/{}/infer'.format(os.environ['AWS_IOT_THING_NAME'])

# Create a local display instance that will dump the image bytes to a FIFO
# file that the image can be rendered locally.
local_display = LocalDisplay('480p')
local_display.start()

# The sample projects come with optimized artifacts, hence only the
artifact
# path is required.
model_path = '/opt/awscam/artifacts/mxnet_resnet18-
catsvsdogs_FP32_FUSED.xml'

# Load the model onto the GPU.
client.publish(topic=iot_topic, payload='Loading action cat-dog model')
model = awscam.Model(model_path, {'GPU': 1})
client.publish(topic=iot_topic, payload='Cat-Dog model loaded')

# Since this is a binary classifier only retrieve 2 classes.
num_top_k = 2

# The height and width of the training set images
input_height = 224
input_width = 224

# Do inference until the lambda is killed.
while True:
    # inference loop to add. See the next step
    ...

except Exception as ex:
    client.publish(topic=iot_topic, payload='Error in cat-dog lambda:
{}'.format(ex))
```

The inference initialization proceeds as follows:

1. Specifies the model type (`model_type`), model artifact path (`model_path`) to load the model artifact ([awscam.Model \(p. 127\)](#)), specifying whether the model is loaded into the device's GPU (`{ 'GPU' : 1 }`) or CPU (`{ 'CPU' : 0 }`). We don't recommend using the CPU because it is much less efficient. The model artifacts are deployed to the device in the `/opt/awscam/artifacts` directory. For the artifact optimized for DeepLens, it consists of an `.xml` file located in this directory. For the artifact not yet optimized for DeepLens, it consists of a JSON file and a another file with the `.params` extension located in the same directory. For optimized model artifact file path, set the `model_path` variable to a string literal:

```
model_path = "/opt/awscam/artifacts/<model-name>.xml"
```

In this example, `<model-name>` is `mxnet_resnet18-catsvsdogs_FP32_FUSED`.

For unoptimized model artifacts, use the [mo.optimize \(p. 131\)](#) function to optimized the artifacts and to obtain the `model_path` value of a given model name (`<model-name>`):

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

Here, the `model_name` value should be the one you specified when [importing the model \(p. 73\)](#). You can also determine the model name by inspecting the S3 bucket for the model artifacts or the local directory of `/opt/awscam/artifacts` on your device. For example, unoptimized model artifacts output by MXNet consists of a `<model_name>-symbol.json` file and a `<model_name>-0000.params` file. If a model artifact consists of the following two files: `hotdog_or_not_model-symbol.json` and `hotdog_or_not_model-0000.params`, you specify `hotdog_or_not_model` as the input `<model_name>` value when calling `mo.optimize`.

2. Specifies `input_width` and `input_height` as the width and height in pixels of images used in training. To ensure meaningful inference, you must convert input image for inference to the same size.
3. Specifies as part of initialization the model type ([model_type \(p. 130\)](#)) and declares the output map (`output_map`). In this example, the model type is `classification`. Other model types are `ssd` (single shot detector) and `segmentation`. The output map will be used to map an inference result label from a numerical value to a human readable text. For binary classifications, there are only two labels (0 and 1).

The `num_top_k` variable refers to the number of inference result of the highest probability. The value can range from 1 to the maximum number of classifiers. For binary classification, it can be 1 or 2.

4. Instantiates an AWS IoT Greengrass SDK (`greengrasssdk`) to make the inference output available to the AWS Cloud, including sending process info and processed result to an AWS IoT topic (`iot_topic`) that provides another means to view your AWS DeepLens project output, although as JSON data, instead of a video stream.
 5. Starts a thread (`local_display.start`) to feed parsed video frames for local display (`LocalDisplay`), [on device \(p. 49\)](#) or [using a web browser \(p. 46\)](#).
- d. Replace the inference loop placeholder (`...`) above with the following code segment:

```
# Get a frame from the video stream
ret, frame = awscam.getLastFrame()
if not ret:
    raise Exception('Failed to get frame from the stream')
# Resize frame to the same size as the training set.
frame_resize = cv2.resize(frame, (input_height, input_width))
# Run the images through the inference engine and parse the results

using
# the parser API, note it is possible to get the output of doInference
# and do the parsing manually, but since it is a classification model,
# a simple API is provided.
parsed_inference_results = model.parseResult(model_type,

model.doInference(frame_resize))
# Get top k results with highest probabilities
top_k = parsed_inference_results[model_type][0:num_top_k]
# Add the label of the top result to the frame used by local display.
# See https://docs.opencv.org/3.4.1/d6/d6e/group_imgproc_draw.html
# for more information about the cv2.putText method.
# Method signature: image, text, origin, font face, font scale, color,
and thickness
cv2.putText(frame, output_map[top_k[0]['label']], (10, 70),
            cv2.FONT_HERSHEY_SIMPLEX, 3, (255, 165, 20), 8)
# Set the next frame in the local display stream.
local_display.set_frame_data(frame)
# Send the top k results to the IoT console via MQTT
cloud_output = {}
for obj in top_k:
```

```
cloud_output[output_map[obj['label']]] = obj['prob']
client.publish(topic=iot_topic, payload=json.dumps(cloud_output))
```

The frame-by-frame inference logic flows as follows:

1. Captures a frame from the DeepLens device video feed (`awscam.getLastFrame()` (p. 125)).
 2. Processes the captured input frame (`cv2.resize(frame, (input_height, input_width))`) to ensure that its dimensions match the dimensions of the frame that the model was trained on. Depending on the model training, you might need to perform other preprocessing steps, such as image normalization.
 3. Performs inference on the frame based on the specified model: `result = model.doInference(frame_resize)`.
 4. Parses the inference result: `model.parseResult(model_type, result)`.
 5. Sends the frame to the local display stream: `local_display.set_frame_data`. If you want to show the human-readable label of the most likely category in the local display, add the label to the captured frame: `cv2.putText`. If not, ignore the last step.
 6. Sends the inferred result to IoT: `client.publish`.
- e. Choose **Save** to save the code you entered.
 - f. From the **Actions** dropdown menu list, choose **Publish new version**. Publishing the function makes it available in the AWS DeepLens console so that you can add it to your custom project.

For questions or help, see the AWS DeepLens forum at [Forum: AWS DeepLens](#).

The Complete Inference Lambda Function

The following code shows the complete Lambda function that, when deployed to the AWS DeepLens device, infers video frames captured from the device to be a cat or dog, based on the model serialized in the model file of `mxnet_resnet18-catsvsdogs_FP32_FUSED.xml` under the `/opt/awscam/artifacts/` directory.

```
*****
#
# Copyright 2018 Amazon.com, Inc. or its affiliates. *
# All Rights Reserved. *
# *
*****
""" A sample lambda for cat-dog detection"""
from threading import Thread, Event
import os
import json
import numpy as np
import awscam
import cv2
import greengrasssdk

class LocalDisplay(Thread):
    """ Class for facilitating the local display of inference results
    (as images). The class is designed to run on its own thread. In
    particular the class dumps the inference results into a FIFO
    located in the tmp directory (which lambda has access to). The
    results can be rendered using mplayer by typing:
    mplayer -demuxer lavf -lavfdopts format=mjpeg:probesize=32 /tmp/results.mjpeg
    """
    def __init__(self, resolution):
        """ resolution - Desired resolution of the project stream """
        # Initialize the base class, so that the object can run on its own
        # thread.
```

```
super(LocalDisplay, self).__init__()
# List of valid resolutions
RESOLUTION = {'1080p' : (1920, 1080), '720p' : (1280, 720), '480p' : (858, 480)}
if resolution not in RESOLUTION:
    raise Exception("Invalid resolution")
self.resolution = RESOLUTION[resolution]
# Initialize the default image to be a white canvas. Clients
# will update the image when ready.
self.frame = cv2.imencode('.jpg', 255*np.ones([640, 480, 3]))[1]
self.stop_request = Event()

def run(self):
    """ Overridden method that continually dumps images to the desired
        FIFO file.
    """
    # Path to the FIFO file. The lambda only has permissions to the tmp
    # directory. Pointing to a FIFO file in another directory
    # will cause the lambda to crash.
    result_path = '/tmp/results.mjpeg'
    # Create the FIFO file if it doesn't exist.
    if not os.path.exists(result_path):
        os.mkfifo(result_path)
    # This call will block until a consumer is available
    with open(result_path, 'w') as fifo_file:
        while not self.stop_request.isSet():
            try:
                # Write the data to the FIFO file. This call will block
                # meaning the code will come to a halt here until a consumer
                # is available.
                fifo_file.write(self.frame.tobytes())
            except IOError:
                continue

def set_frame_data(self, frame):
    """ Method updates the image data. This currently encodes the
        numpy array to jpg but can be modified to support other encodings.
        frame - Numpy array containing the image data of the next frame
        in the project stream.
    """
    ret, jpeg = cv2.imencode('.jpg', cv2.resize(frame, self.resolution))
    if not ret:
        raise Exception('Failed to set frame data')
    self.frame = jpeg

def join(self):
    self.stop_request.set()

def infinite_infer_run():
    """ Entry point of the lambda function"""
    try:
        # This cat-dog model is implemented as binary classifier, since the number
        # of labels is small, create a dictionary that converts the machine
        # labels to human readable labels.
        model_type = 'classification'
        output_map = {0: 'dog', 1: 'cat'}
        # Create an IoT client for sending to messages to the cloud.
        client = greengrasssdk.client('iot-data')
        iot_topic = '$aws/things/{}/infer'.format(os.environ['AWS_IOT_THING_NAME'])
        # Create a local display instance that will dump the image bytes to a FIFO
        # file that the image can be rendered locally.
        local_display = LocalDisplay('480p')
        local_display.start()
        # The sample projects come with optimized artifacts, hence only the artifact
        # path is required.
        model_path = '/opt/awscam/artifacts/mxnet_resnet18-catsvsdogs_FP32_FUSED.xml'
        # Load the model onto the GPU.
```

```
client.publish(topic=iot_topic, payload='Loading action cat-dog model')
model = awscam.Model(model_path, {'GPU': 1})
client.publish(topic=iot_topic, payload='Cat-Dog model loaded')
# Since this is a binary classifier only retrieve 2 classes.
num_top_k = 2
# The height and width of the training set images
input_height = 224
input_width = 224
# Do inference until the lambda is killed.
while True:
    # Get a frame from the video stream
    ret, frame = awscam.getLastFrame()
    if not ret:
        raise Exception('Failed to get frame from the stream')
    # Resize frame to the same size as the training set.
    frame_resize = cv2.resize(frame, (input_height, input_width))
    # Run the images through the inference engine and parse the results using
    # the parser API, note it is possible to get the output of doInference
    # and do the parsing manually, but since it is a classification model,
    # a simple API is provided.
    parsed_inference_results = model.parseResult(model_type,
                                                model.doInference(frame_resize))

    # Get top k results with highest probabilities
    top_k = parsed_inference_results[model_type][0:num_top_k]
    # Add the label of the top result to the frame used by local display.
    # See https://docs.opencv.org/3.4.1/d6/d6e/group_imgproc__draw.html
    # for more information about the cv2.putText method.
    # Method signature: image, text, origin, font face, font scale, color, and
    thickness
    cv2.putText(frame, output_map[top_k[0]['label']], (10, 70),
                cv2.FONT_HERSHEY_SIMPLEX, 3, (255, 165, 20), 8)
    # Set the next frame in the local display stream.
    local_display.set_frame_data(frame)
    # Send the top k results to the IoT console via MQTT
    cloud_output = {}
    for obj in top_k:
        cloud_output[output_map[obj['label']]] = obj['prob']
    client.publish(topic=iot_topic, payload=json.dumps(cloud_output))
except Exception as ex:
    client.publish(topic=iot_topic, payload='Error in cat-dog lambda: {}'.format(ex))

infinite_infer_run()
```

Create and Deploy a Custom AWS DeepLens Project

Topics

- [Create Your Custom Project Using the AWS DeepLens Console \(p. 83\)](#)

Create Your Custom Project Using the AWS DeepLens Console

The following procedure creates a custom AWS DeepLens project containing a custom-trained model and inference function based on the model. You should have imported your model, created and published a custom inference &LAM; function before proceeding further.

To create a custom AWS DeepLens project using the AWS DeepLens console

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.

1. Open the AWS DeepLens console by navigating to <https://console.aws.amazon.com/deeplens/> from a web browser.

2. Choose **Projects**, then choose **Create new project**.
3. On the **Choose project type** page, choose **Create a new blank project**.
4. On the **Specify project details** page, under **Project information**, do the following:
 - a. In **Project name** type a name, for example, `my-cat-and-dog-detection`, to identify your custom project.
 - b. Optionally, in **Description - Optional** provide a description for your custom project.
5. Still on the **Specify project details** page, under **Project content**, do the following:
 - a. Choose **Add model**, choose the radio button next to the name of your custom model you've imported into AWS DeepLens, and, then, choose **Add model**.
 - b. Choose **Add function**, choose the radio button next to the name of your custom inference function you've created and published, and, then, choose **Add function**.

If you don't see your function displayed, verify that it's published.

 - c. Choose **Create** to start creating your custom project.
6. On the **Projects** page, choose the project you just created.
7. Choose **Deploy to device** and follow the on-screen instructions to deploy the project to your device.

Use Amazon SageMaker Neo to Optimize Inference on AWS DeepLens

A trained model might not be optimized to run inference on your device. Before you deploy a trained model to your AWS DeepLens, you can use [Amazon SageMaker Neo](#) to optimize it to run inference on the AWS DeepLens hardware. The process involves these steps:

1. Compile the model.
2. Import the compiled model into the AWS DeepLens console.
3. Deploy the compiled model to your AWS DeepLens device.
4. Load the compiled model into the Neo runtime of the inference engine.

We recommend that you use Neo-compiled models for improved inference performance with minimal effort on your part.

Topics

- [Compile an AWS DeepLens Model \(p. 84\)](#)
- [Import a Compiled Model into the AWS DeepLens Console \(p. 85\)](#)
- [Load the Compiled Model into the AWS DeepLens Inference Engine \(p. 86\)](#)

Compile an AWS DeepLens Model

To use Neo to compile a model, you can use AWS CLI to create and run a compilation job. First create a job configuration file in JSON to specify the following: The Amazon S3 bucket to load the raw model artifacts from, the training data format, and the machine learning framework, which indicates how the job is to run.

The following example JSON file shows how to specify the job configuration. As an example, the file is named `neo_deeplens.json`.

```
{
```



```
{
  "CompilationJobName": "deeplens_inference",
  "RoleArn": "arn:aws:iam::your-account-id:role/service-role/AmazonSageMaker-ExecutionRole-20190429T140091",
  "InputConfig": {
    "S3Uri": "s3://your-bucket-name/headpose/model/deeplens-tf/train/model.tar.gz",
    "DataInputConfig": "{\"data\": [1,84,84,3]}",
    "Framework": "TENSORFLOW"
  },
  "OutputConfig": {
    "S3OutputLocation": "s3://your-bucket-name/headpose/model/deeplens-tf/compile",
    "TargetDevice": "deeplens"
  },
  "StoppingCondition": {
    "MaxRuntimeInSeconds": 300
  }
}
```

The `InputConfig.S3Uri` property value specifies the S3 path to the trained model artifacts. The `S3OutputLocation` property value specifies the S3 folder to contain the Neo-compiled model artifacts. The input data format specified in the `DataInputConfig` property value must be consistent with the original training data format. For more information about the compilation job configuration property values, see [CreateCompilationJob](#).

To create and run the Neo job, type the following AWS CLI command in a terminal window.

```
aws sagemaker create-compilation-job \
--cli-input-json file://neo_deeplens.json \
--region us-west-2
```

In addition to AWS CLI, you can also use the Amazon SageMaker console or the Amazon SageMaker SDK to create and run the compilation job. To learn more, see [the Amazon SageMaker console](#) and [Amazon SageMaker SDK](#) to compile a trained model.

Import a Compiled Model into the AWS DeepLens Console

This section explains how to import a compiled model. When these steps are complete, you can deploy the model to your AWS DeepLens device and then run inference.

To import a Neo-compiled model into the AWS DeepLens console

1. Download the compiled model from the S3 folder as specified by the `S3OutputLocation` path.
2. On your computer, create a folder (*compiled-model-folder*) to unpack the compiled model into. Note the folder name because you need it later to identify the compiled model.

```
mkdir compiled-model-folder
```

3. If you use a macOS computer, set the following flag.

```
COPYFILE_DISABLE=1
```

4. Unpack the downloaded model into the folder (*compiled-model-folder*).

```
tar xzvf model-deeplens.tar.gz -C compiled-model-folder
```

5. Compress the compiled model folder.

```
tar czvf compiled-model-foler.tar.gz compiled-model-folder
```

6. Upload the compressed compiled model folder to S3. Note the target S3 path. You use it import the model in the next step.
7. Go to the AWS DeepLens console and import the model specifying the S3 path chosen in the previous step. In the model's settings, set the model's framework to `Other`.

The compiled model is now ready for deployment to your AWS DeepLens device. The steps to deploy a compiled model are the same as an uncompiled model.

Load the Compiled Model into the AWS DeepLens Inference Engine

To load a compiled model into your AWS DeepLens inference engine, specify the folder name that contains the compiled model artifacts and the Neo runtime. Do this when you instantiate the `awscam.Model` class in the Lambda function. The following code example demonstrates this.

```
import awscam
model = awscam.Model('compiled-model-folder', {'GPU': 1}, runtime=1)
ret, image = awscam.getlastFrame()
infer_results = model.doInference(image)
```

The function now runs inference with a compiled model in the Neo runtime. To use the compiled model for inference, you must set `runtime=1` and select GPU (`{ 'GPU' : 1 }`) when creating the `model` object.

The output `infer_results` returns the raw inference results as a dictionary object, where `infer_results[i]` holds the result of the i^{th} output layer. For example, if an image classification network (e.g., resnet-50) has only one Softmax output layer, `infer_results[0]` is a Numpy array with size $N \times 1$ that gives the probability for each of the N labeled images.

Building AWS DeepLens Project Tutorials

This section presents end-to-end tutorials to guide your through building, deploying and testing AWS DeepLens projects with deep learning models trained in supported machine learning frameworks.

Topics

- [Build and Run the Head Pose Detection Project with TensorFlow-Trained Model \(p. 86\)](#)

Build and Run the Head Pose Detection Project with TensorFlow-Trained Model

In this tutorial, we will walk you through an end-to-end process to build and run an AWS DeepLens project to detect head poses. The project is based on the Head pose detection sample project. Specifically, you'll learn:

1. How to train a computer vision model using the TensorFlow framework in Amazon SageMaker, including converting the model artifact to the protobuf format for use by AWS DeepLens.
2. How to create an inference Lambda function to infer head poses based on frames captured off the video feed from the AWS DeepLens video camera.
3. How to create an AWS DeepLens project to include the Amazon SageMaker-trained deep learning model and the inference Lambda function.


4. How to deploy the project to your AWS DeepLens device to run the project and verify the results.


Topics


- [Get the AWS Sample Project on GitHub](#) (p. 87)
- [Set Up the Project Data Store in Amazon S3](#) (p. 89)
- [Prepare Input Data for Training In Amazon SageMaker](#) (p. 90)
- [Train a Head Pose Detection Model in Amazon SageMaker](#) (p. 91)
- [Create an Inference Lambda Function to Detect Head Poses](#) (p. 96)
- [Create and Deploy the Head Pose Detection Project](#) (p. 102)


Get the AWS Sample Project on GitHub


This tutorial is based on the [head pose detection sample project](#) (p. 55). The sample project's source code, including the [preprocessing script](#) used to prepare for input data from the original [Head Pose Image Database](#), is available on [GitHub](#).


 **aws-samples / headpose-estimator-apache-mxnet**

 **Code**


 **Issues** 0


 **Pull requests** 0


 **Projects** 0

 **Insights**

Head Pose estimator using Apache MXNet. HeadPose_ResNet50_Tutorial.ipynb helps with the flow of developing a CNN model from the scratch including data augmentation, fine-tuning, model artifacts, validation and inference.


 **60** commits












 **1** branch

 **0** releases

Branch: **master** ▾

New pull request

 **araitats** Add files via upload

 .github	Creating initial file from template
 HeadPose_SageMaker_PythonSDK	Add files via upload
 testIMs	Add files via upload
 CODE_OF_CONDUCT.md	Creating initial file from template
 CONTRIBUTING.md	Creating initial file from template
 HeadPose_ResNet50_Tutorial.ipynb	Update HeadPose_ResNet50_Tutorial.ipynb
 HeadPose_ResNet50_Tutorial_Gluon.ipynb	Update HeadPose_ResNet50_Tutorial_Gluon
 LICENSE	Creating initial file from template
 NOTICE	Creating initial file from template
 README.md	Update README.md
 preprocessingDataset_py2.py	Add files via upload

The HeadPose_ResNet50_Tutorial.ipynb file provides a Notebook template as a starting point to learn and explore training our model. The preprocessingDataset_py2.py can be used as-is to prepare input data for training.

You can download and uncompress the zipped GitHub project file to a local drive. Or you can clone the GitHub project. To clone the project, open a terminal window and type the following Git command in a given working directory:

```
git clone https://github.com/aws-samples/headpose-estimator-apache-mxnet
```

By default, the downloaded or cloned project folder is `headpose-estimator-apache-mxnet-master`. We will use this default folder name for the project folder throughout this tutorial.

Having the GitHub project as a template, we're now ready to start building our project by [setting up the project's data store in S3](#) (p. 89).

Set Up the Project Data Store in Amazon S3

For this project, we will use Amazon SageMaker to train the model. To do so, we need to prepare an S3 bucket and four subfolders to store the input needed for the training job and the output produced by the training job.

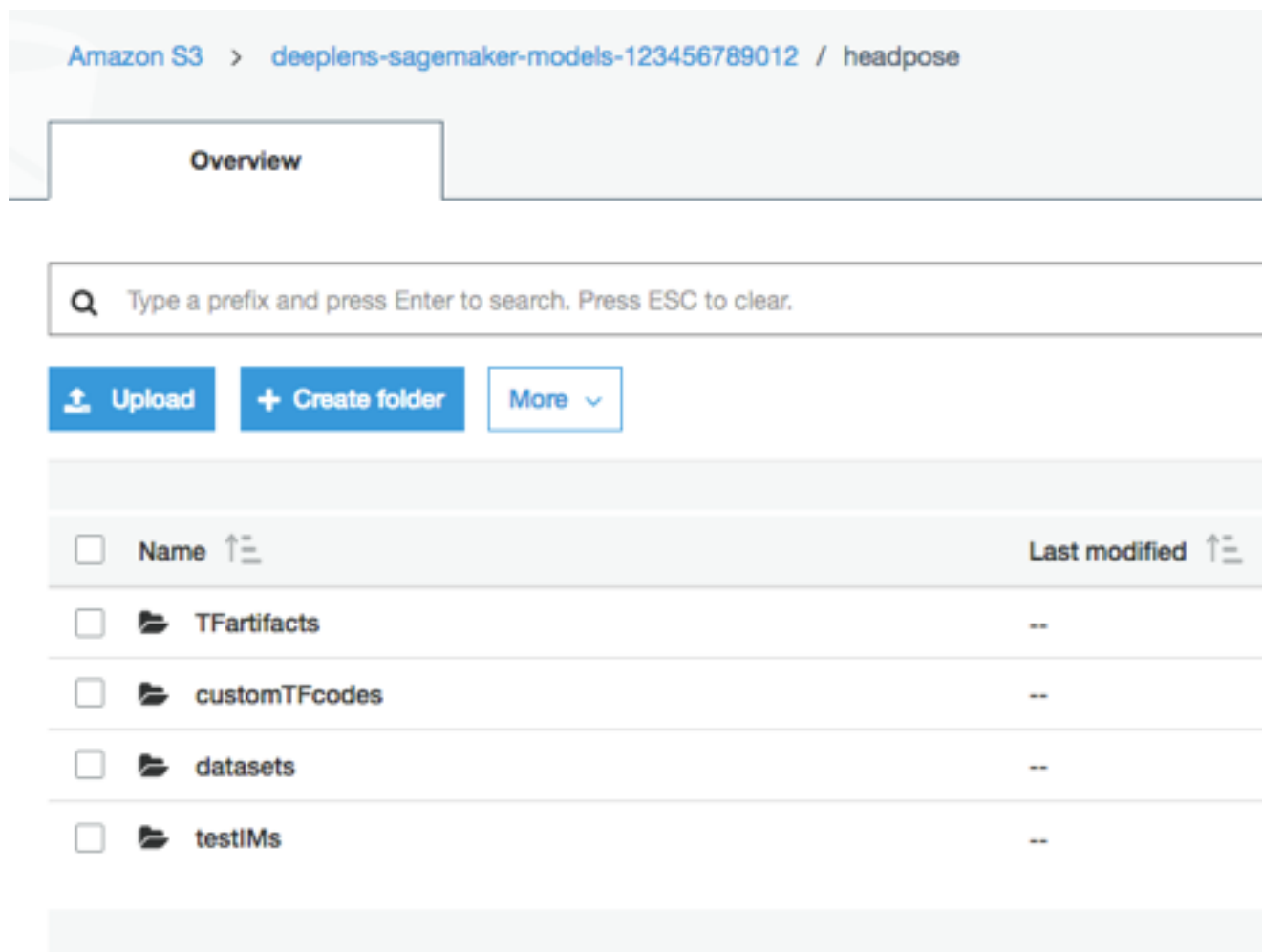
We will create an Amazon S3 bucket and name it `deeplens-sagemaker-models-<my-name>`. Fill in your name or some ID in the *<my-name>* placeholder to make the bucket name unique. In this bucket, we create a `headpose` folder to hold data for training the model specific for head pose detection.

We then create the following four subfolders under the `headpose` folder:

- `deeplens-sagemaker-models-<my-name>/headpose/TFartifacts`: to store training output, i.e., the trained model artifacts.
- `deeplens-sagemaker-models-<my-name>/headpose/customTFcodes`
- `deeplens-sagemaker-models-<my-name>/headpose/datasets`: to store training input, the preprocessed images with known head poses.
- `deeplens-sagemaker-models-<my-name>/headpose/testIMs`

Follow the steps below to create the bucket and folders using the Amazon S3 console:

1. Open the Amazon S3 console at <https://console.aws.amazon.com/s3/home?region=us-east-1>.
2. If the Amazon S3 bucket does not exist yet, choose **+ Create bucket** and then type `deeplens-sagemaker-models-<my-name>` in **Bucket name**, use the default values for other options, and choose **Create**. Otherwise, double click the existing bucket name to choose the bucket.
3. Choose **+ Create folder**, type the `headpose` as the folder name, and then choose **Save**.
4. Choose the `headpose` folder just created, and then choose **+ Create folder** to create the four subfolders (named `TFartifacts`, `customTFcodes`, `datasets`, `testIMs`) under `headpose`, one at time.



Next, we need to [prepare the input data for training in Amazon SageMaker \(p. 90\)](#).

Prepare Input Data for Training In Amazon SageMaker

The original [head pose image database \(HeadPoseImageDatabase.tar.gz file\)](#) consists of a set of images (.jpeg files) and labels (.txt files) of head poses of 15 persons in a series of tilt and pan positions. They must be transformed to a single .pkl file as the input to an Amazon SageMaker training job.

To prepare the training data, you can use the preprocessing script (`preprocessingDataset_py2.py`) provided in the [Head Pose Sample Project on Github \(p. 87\)](#). You can run this script on your computer locally or on the AWS Cloud through a Python Notebook instance. In this section, you'll learn how to prepare the training data locally. You'll learn how to do it in a Notebook instance in [the section called "Train a Head Pose Detection Model in Amazon SageMaker" \(p. 91\)](#).

1. Open a terminal and change directory to the directory where you downloaded the [head pose sample project \(p. 87\)](#).
2. Run the following Python (version 2.7) command

```
python preprocessingDataset_py2.py --num-data-aug 15 --aspect-ratio 1
```

If you get an error stating that the `cv2` module is not found, run the following command to install the missing module:

```
python -m pip install opencv-python
```

Then, rerun the preprocessing script.

The process takes 15-20 hours to complete. The preprocessing script outputs the results as the `HeadPoseData_trn_test_x15_py2.pkl` file in the current working directory.

3. Upload the preprocessed training data (`HeadPoseData_trn_test_x15_py2.pkl`) file into the `deeplens-sagemaker-models-<my-name>/headpose/datasets` folder in Amazon S3. You can do this using the Amazon S3 console or, as shown as follows, using the following [AWS CLI](#) command:

```
aws s3 cp HeadPoseData_trn_test_x15_py2.pkl s3://deeplens-sagemaker-models-<my-name>/headpose/datasets
```

Note

Instead of running locally, you can prepare the input data on the AWS Cloud by a code cell in a Amazon SageMaker notebook instance, where `cv2` and Python 2.7 (`python2`) are readily available, and run the commands in **Step 2** and **Step 3**, respectively, as follows:

```
!python2 preprocessingDataset_py2.py --num-data-aug 15 --aspect-ratio 1
```

and

```
!aws s3 cp HeadPoseData_trn_test_x15_py2.pkl s3://deeplens-sagemaker-models-<my-name>/headpose/datasets
```

Your notebook instance needs to run on a sufficiently powerful EC2 instance of, e.g., the `m1.p2.xlarge` type.

Now that we have the input data prepared and stored in S3, we're ready to [train the model \(p. 91\)](#).

Train a Head Pose Detection Model in Amazon SageMaker

To train our model in Amazon SageMaker, follow the steps below to create a Notebook using the Amazon SageMaker console:

Create an Amazon SageMaker Notebook

1. Sign in to the Amazon SageMaker console at <https://console.aws.amazon.com/sagemaker/home?region=us-east-1>.
2. From the main navigation pane, choose **Notebook instances**.
3. Under **Notebook instance settings** on the **Create notebook instance** page, do the following:
 - a. Under **Notebook instance name**, type `TFHeadpose`.
 - b. Leave the default values for all other options.
 - c. Choose **Create notebook instance**.
4. Next to the **TFHeadpose** notebook instance, choose **Open**, after the notebook instance status becomes **InService**.
5. On the upper-right corner, choose **Upload** to start importing a notebook instance and other needed files.

6. In the file picker, navigate to the **HeadPost_SageMaker_PythonSDK** folder in the previously cloned or downloaded **headpose-estimator-apache-mxnet-master** project folder. Then, select the following files:
 - **resnet_headpose.py**: This is a script that defines the workflow for training with the deep learning network architecture prescribed in the accompanying **resnet_model_headpose.py** file applied to a specified input data.
 - **resnet_model_headpose.py**: This is a Python script that prescribes the deep learning network architecture used to train our model for head pose detection.
 - **tensorflow_resnet_headpost_for_deeplens.ipynb**: This is a Notebook instance to run an Amazon SageMaker job to manage training following the script of **resnet_headpose.py**, including data preparation and transformation.

Then, choose **Open**.

If you intend to run the preprocessing script on the AWS Cloud, navigate to the **headpose-estimator-apache-mxnet-master** folder, select the **preprocessingDataset_py2.py**, and choose **Open**.

7. On the **Files** tab in the **TFHeadpose** notebook, choose **Upload** for each of the newly selected files to finish importing the files into the notebook. You're now ready to run an Amazon SageMaker job to train your model.
8. Choose **tensorflow_resnet_headpose_for_deeplens.ipynb** to open the notebook instance in a separate browser tab. The notebook instance lets you step through the tasks necessary to run an Amazon SageMaker job to train your model and to transform the model artifacts to a format supported by AWS DeepLens.
9. Run an Amazon SageMaker job to train your model in the notebook instance. The implementation is presented in separate code cells that can be run sequentially.
 - a. Under **Set up the environment**, the code cell contains the Python code to configure the data store for the input and output of the Amazon SageMaker model training job.

```
import os
import sagemaker
from sagemaker import get_execution_role

s3_bucket = 'deeplens-sagemaker-models-<my-name>'
headpose_folder = 'headpose'

#Bucket location to save your custom code in tar.gz format.
custom_code_folder = 'customTFcodes'
custom_code_upload_location = 's3://{}/{}/{}'.format(s3_bucket, headpose_folder,
    custom_code_folder)

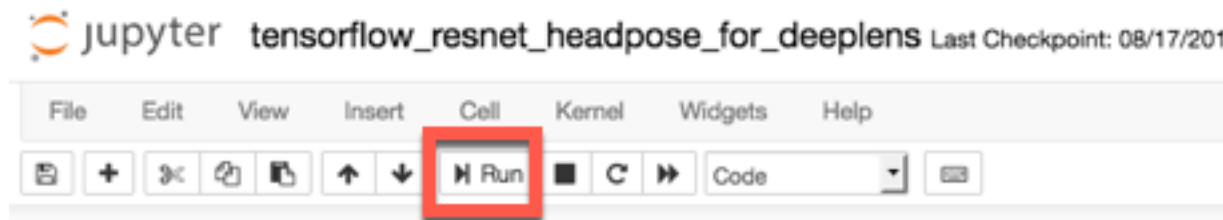
#Bucket location where results of model training are saved.
model_artifacts_folder = 'TFartifacts'
model_artifacts_location = 's3://{}/{}/{}'.format(s3_bucket, headpose_folder,
    model_artifacts_folder)

#IAM execution role that gives SageMaker access to resources in your AWS account.
#We can use the SageMaker Python SDK to get the role from our notebook
environment.

role = get_execution_role()
```

To use the [S3 bucket and folders \(p. 89\)](#) as the data store, assign your S3 bucket name (e.g., `deeplens-sagemaker-models-<my-name>`) to the `s3_bucket` variable. Make sure that the specified folder names all match what you have in the Amazon S3 bucket.

To execute this code block, choose **Run** from the menu bar of the notebook instance.



Set up the environment

```
In [3]: import os
import sagemaker
from sagemaker import get_execution_role

sagemaker_session = sagemaker.Session() # not used, it appears.

s3_bucket = 'deeplens-sagemaker-models-<my-id>'
headpose_folder = 'headpose'

#Bucket location to save your custom code in tar.gz format.
custom_code_folder = 'customTFcodes'
custom_code_upload_location = 's3://{}/{}/{}'.format(s3_bucket, headpose_folder, custom_code_folder)
```

- b. Under **Create a training job using the sagemaker.tensorflow.TensorFlow estimator**, the code cell contains the code snippet that performs the following tasks:
 1. Instantiate a `sagemaker.tensorflow.TensorFlow` estimator class (`estimator`), with a specified training script and process configuration.
 2. Start training the model (`estimator.fit(...)`) with the estimator, with the training data in a specified data store.

The code snippet is shown as follows:

```
from sagemaker.tensorflow import TensorFlow

source_dir = os.path.join(os.getcwd())

# AWS DeepLens currently supports TensorFlow version 1.4 (as of August 24th 2018).
estimator = TensorFlow(entry_point='resnet_headpose.py',
                        framework_version = 1.4,
                        source_dir=source_dir,
                        role=role,
                        training_steps=25000, evaluation_steps=700,
                        train_instance_count=1,
                        base_job_name='deeplens-TF-headpose',
                        output_path=model_artifacts_location,
                        code_location=custom_code_upload_location,
                        train_instance_type='ml.p2.xlarge',
                        train_max_run = 432000,
                        train_volume_size=100)

# Head-pose dataset "HeadPoseData_trn_test_x15_py2.pkl" is in the following S3
folder.
dataset_location = 's3://{}/{}/datasets'.format(s3_bucket, headpose_folder)
```

```
estimator.fit(dataset_location)
```

To create the estimator object, you assign to `entry_point` the name of the Python file containing [the custom TensorFlow model training script](#). For this tutorial, this custom code file is `resnet_headpose.py` that was uploaded to the same directory where the notebook instance is located. For `framework_version`, specify a TensorFlow version supported by AWS DeepLens.

With the `train_instance_type` of `ml.p2.xlarge`, it takes about 6.7 billable compute hours to complete the training job (`estimator.fit(...)`). You can experiment with [other train_instance_type options](#) to speed up the process or to optimize the cost.

The successfully trained model artifact (`model.tar.gz`) is output to your S3 bucket:

```
s3://deeplens-sagemaker-models-<my-name>/headpose/TFartifacts/<training-job-name>/output/model.tar.gz
```

where `<training-job-name>` is of the `<base_job_name>-<timestamp>`. Using the training code above, an example of the `<training-job-name>` would be `deeplens-TF-headpose-2018-08-16-20-10-09-991`. You must transform this model artifact into a frozen protobuf format that is supported by AWS DeepLens.

- c. To transform the trained model artifact (`model.tar.gz`) into a frozen protobuf file (`frozen.model.pb`), do the following:
 - i. Run the following code to use the AWS SDK for Python (`boto3`) in a code cell in the notebook to download the trained model artifact from your S3 bucket to your notebook:

```
import boto3
s3 = boto3.resource('s3')

#key = '{}/{}/{}/output/model.tar.gz'.format(headpose_folder,
#      model_artifacts_folder, estimator.latest_training_job.name)
key = '{}/{}/{}/output/model.tar.gz'.format(headpose_folder,
      model_artifacts_folder, 'deeplens-TF-headpose-2018-08-16-20-10-09-991')

print(key)
s3.Bucket(s3_bucket).download_file(key, 'model.tar.gz')
```

You can verify the downloaded files by running the following shell command in a code cell of the notebook and then examine the output.

```
!ls
```

- ii. To uncompress the trained model artifact (`model.tar.gz`), run the following shell command in a code cell:

```
!tar -xvf model.tar.gz
```

This command will produce the following output:

```
export/
export/Servo/
export/Servo/1534474516/
export/Servo/1534474516/variables/
export/Servo/1534474516/variables/variables.data-00000-of-00001
export/Servo/1534474516/variables/variables.index
```

```
export/Servo/1534474516/saved_model.pb
```

The path to a model directory is of the `export/**` pattern. You must specify the model directory path to make a frozen protobuff file from the model artifact. You'll see how to get this directory path in the next step.

- iii. To get the model directory and cache it in memory, run the following Python code in a code cell of the notebook instance:

```
import glob
model_dir = glob.glob('export/**')
# The model directory looks like 'export/Servo/{Assigned by Amazon SageMaker}'
print(model_dir)
```

The output is `['export/Servo/1534474516']`.

You can proceed to freezing the graph and save it in the frozen protobuff format.

- iv. To freeze the TensorFlow graph and save it in the frozen protobuff format, run the following Python code snippet in a code cell of the notebook instance. The code snippet does the following:
 1. Calls `convert_variables_to_constants` from the `tf.graph_util` module to freeze the graph,
 2. Calls `remove_training_nodes` from the `tf.graph_util` module to remove all unnecessary nodes.
 3. Calls `optimize_for_inference` from the `optimize_for_inference_lib` module to generate the `inference_graph_def`.
 4. Serializes and saves the file as a protobuff.

```
import tensorflow as tf
from tensorflow.python.tools import optimize_for_inference_lib
def freeze_graph(model_dir, input_node_names, output_node_names):
    """Extract the sub graph defined by the output nodes and convert
    all its variables into constant
    Args:
        model_dir: the root folder containing the checkpoint state file,
        input_node_names: a comma-separated string, containing the names of all
        input nodes
        output_node_names: a comma-separated string, containing the names of
        all output nodes,

    """

    # We start a session using a temporary fresh Graph
    with tf.Session(graph=tf.Graph()) as sess:
        # We import the meta graph in the current default Graph
        tf.saved_model.loader.load(sess,
            [tf.saved_model.tag_constants.SERVING], model_dir)

        # We use a built-in TF helper to export variables to constants
        input_graph_def = tf.graph_util.convert_variables_to_constants(
            sess, # The session is used to retrieve the weights
            tf.get_default_graph().as_graph_def(), # The graph_def is used to
            retrieve the nodes
            output_node_names.split(",") # The output node names are used to
            select the usefull nodes
        )

        # We generate the inference graph_def
```

```
input_node_names=['Const_1']    # an array of the input node(s)
output_graph_def =
optimize_for_inference_lib.optimize_for_inference(tf.graph_util.remove_training_nodes(input_node_names.split(","), # an array of input nodes

output_node_names.split(","), # an array of output nodes

tf.float32.as_datatype_enum)
# Finally we serialize and dump the output graph_def to the filesystem
with tf.gfile.GFile('frozen_model.pb', "wb") as f:
    f.write(output_graph_def.SerializeToString())

freeze_graph(model_dir[0], 'Const_1', 'softmax_tensor')
```

As the result, the model artifact is transformed into the frozen protobuf format (frozen_model.pb) and saved to the notebook instance's home directory (model_dir[0]).

In the code above, you must specify the input and output nodes, namely, 'Const_1' and 'softmax_tensor'. For more details, see the [resnet_model_headpose.py](#).

When creating an AWS DeepLens project later, you'll need to add this frozen graph to the project. For this you must upload the protobuf file to an Amazon S3 folder. For this tutorial, you can use your Amazon SageMaker training job's output folder (s3://deeplens-sagemaker-models-*<my-name>*/headpose/Tfartifacts/*<sagemaker-job-name>*/output) in S3. However, the model is considered an externally trained model in AWS DeepLens.

- d. To upload the frozen graph to your Amazon SageMaker training job's output folder, run the following Python code snippet in a code cell of the running notebook instance:

```
data = open('frozen_model.pb', "rb")
#key = '{}/{}/{}/output/frozen_model.pb'.format(headpose_folder,
#model_artifacts_folder, estimator.latest_training_job.name)
key = '{}/{}/{}/output/frozen_model.pb'.format(headpose_folder,
#model_artifacts_folder, 'deeplens-TF-headpose-2018-08-16-20-10-09-991')
s3.Bucket(s3_bucket).put_object(Key=key, Body=data)
```

Once uploaded, the model is ready to be imported into your AWS DeepLens project. Before creating the project, we must [create a Lambda function that performs inference based on this trained model](#) (p. 96).

Create an Inference Lambda Function to Detect Head Poses

Before creating an AWS DeepLens project for deployment to your AWS DeepLens device for head pose detection, you must create and publish a Lambda function to make inference based on the trained model.

To create and publish the inference Lambda function, follow the instruction given in [the section called "Create and Publish an Inference Lambda Function" \(p. 76\)](#), but replace the code in the greengrassHelloWorld.py file with one similar to the following that is used in the [Head Pose Detection Sample project](#) (p. 55).

```
""" This is a lambda function that demonstrates how one can use a deep learning network
(ResNet) to detect the person's head pose. We use a shaded rectangle to indicate
the region that the persons head is pointed towards. We also display a red ball
that moves with the person's head.
```

```
"""
from threading import Thread, Event
import os
import json
import numpy as np
import cv2
import greengrasssdk
import awscam
import mo

class LocalDisplay(Thread):
    """ Class for facilitating the local display of inference results
    (as images). The class is designed to run on its own thread. In
    particular the class dumps the inference results into a FIFO
    located in the tmp directory (which lambda has access to). The
    results can be rendered using mplayer by typing:
    mplayer -demuxer lavf -lavfdopts format=mjpeg:probesize=32 /tmp/results.mjpeg
    """
    def __init__(self, resolution):
        """ resolution - Desired resolution of the project stream"""
        # Initialize the base class, so that the object can run on its own
        # thread.
        super(LocalDisplay, self).__init__()
        # List of valid resolutions
        RESOLUTION = {'1080p' : (1920, 1080), '720p' : (1280, 720), '480p' : (858, 480)}
        if resolution not in RESOLUTION:
            raise Exception("Invalid resolution")
        self.resolution = RESOLUTION[resolution]
        # Initialize the default image to be a white canvas. Clients
        # will update the image when ready.
        self.frame = cv2.imencode('.jpg', 255*np.ones([640, 480, 3]))[1]
        self.stop_request = Event()

    def run(self):
        """ Overridden method that continually dumps images to the desired
        FIFO file.
        """
        # Path to the FIFO file. The lambda only has permissions to the tmp
        # directory. Pointing to a FIFO file in another directory
        # will cause the lambda to crash.
        result_path = '/tmp/results.mjpeg'
        # Create the FIFO file if it doesn't exist.
        if not os.path.exists(result_path):
            os.mkfifo(result_path)
        # This call will block until a consumer is available
        with open(result_path, 'w') as fifo_file:
            while not self.stop_request.isSet():
                try:
                    # Write the data to the FIFO file. This call will block
                    # meaning the code will come to a halt here until a consumer
                    # is available.
                    fifo_file.write(self.frame.tobytes())
                except IOError:
                    continue

    def set_frame_data(self, frame):
        """ Method updates the image data. This currently encodes the
        numpy array to jpg but can be modified to support other encodings.
        frame - Numpy array containing the image data of the next frame
        in the project stream.
        """
        ret, jpeg = cv2.imencode('.jpg', cv2.resize(frame, self.resolution))
        if not ret:
            raise Exception('Failed to set frame data')
        self.frame = jpeg
```

```
def join(self):
    self.stop_request.set()

class HeadDetection():
    """ Custom class that helps us post process the data. In particular it draws
        a ball that moves across the screen depending on the head pose.
        It also draws a rectangle indicating the region that the person's head is pointing
        to. We divide the frame into 9 distinct regions.
    """
    def __init__(self, circ_cent_x, circ_cent_y):
        """ circ_cent_x - The x coordinate for the center of the frame
            circ_cent_y - The y coordinate for the center of the frame
        """
        self.result_thread = LocalDisplay('480p')
        self.result_thread.start()
        self.circ_cent_x = circ_cent_x
        self.circ_cent_y = circ_cent_y
        # Compute the maximum x and y coordinates.
        self.x_max = 2 * circ_cent_x
        self.y_max = 2 * circ_cent_y
        # Number of quadrants to split the image into.
        # This is model dependent.
        self.quadrants = 9

    def update_coords(self, frame, change_x, change_y, label):
        """ Helper method that draws a rectangle in the region the person is looking at.
            It also draws a red ball that changes its speed based on how long a person
            has been looking at a particular direction.
            frame - The frame where the ball and rectangle should be drawn
            change_x - The amount to increment the x axis after drawing the red ball.
            change_y - The amount to increment the y axis after drawing the red ball.
            label - Label corresponding to the region that the person's head is looking at.
        """
        # Set coordinates of the rectangle that will be drawn in the region that the
        # person is looking at.
        rect_margin = 10
        rect_width = frame.shape[1] // 3 - rect_margin * 2
        rect_height = frame.shape[0] // 3 - rect_margin * 2
        # Set the draw options.
        overlay = frame.copy()
        font_color = (20, 165, 255)
        line_type = 8
        font_type = cv2.FONT_HERSHEY_DUPLEX
        # Draw the rectangle with some text to indicate the region.
        if label == 0:
            cv2.putText(frame, "Down Right", (1024, 440 - 15), font_type, 3, font_color,
line_type)
            cv2.rectangle(overlay,
                          (frame.shape[1] // 3 * 2 + rect_margin,
                           frame.shape[0] // 3 * 2 + rect_margin),
                          (frame.shape[1] // 3 * 2 + rect_width,
                           frame.shape[0] // 3 * 2 + rect_height),
                          (0, 255, 0), -1)
        elif label == 1:
            cv2.putText(frame, "Right", (1024, 440 - 15), font_type, 3, font_color,
line_type)
            cv2.rectangle(overlay,
                          (frame.shape[1] // 3 * 2 + rect_margin,
                           frame.shape[0] // 3 * 1 + rect_margin),
                          (frame.shape[1] // 3 * 2 + rect_width,
                           frame.shape[0] // 3 * 1 + rect_height),
                          (0, 255, 0), -1)
        elif label == 2:
            cv2.putText(frame, "Up Right", (1024, 440 - 15), font_type, 3, font_color,
line_type)
            cv2.rectangle(overlay,
```

```
        (frame.shape[1] // 3 * 2 + rect_margin, rect_margin),
        (frame.shape[1] // 3 * 2 + rect_width, rect_height),
        (0, 255, 0), -1)
    elif label == 3:
        cv2.putText(frame, "Down", (1024, 440 - 15), font_type, 3, font_color,
line_type)
        cv2.rectangle(overlay,
            (frame.shape[1] // 3 * 1 + rect_margin,
             frame.shape[0] // 3 * 2 + rect_margin),
            (frame.shape[1] // 3 * 1 + rect_width,
             frame.shape[0] // 3 * 2 + rect_height),
            (0, 255, 0), -1)
    elif label == 4:
        cv2.putText(frame, "Middle", (1024, 440 - 15), font_type, 3, font_color,
line_type)
        cv2.rectangle(overlay,
            (frame.shape[1] // 3 * 1 + rect_margin,
             frame.shape[0] // 3 * 1 + rect_margin),
            (frame.shape[1] // 3 * 1 + rect_width,
             frame.shape[0] // 3 * 1 + rect_height),
            (0, 255, 0), -1)
    elif label == 5:
        cv2.putText(frame, "Up", (1024, 440 - 15), font_type, 3, font_color, line_type)
        cv2.rectangle(overlay,
            (frame.shape[1] // 3 * 1 + rect_margin, rect_margin),
            (frame.shape[1] // 3 * 1 + rect_width, rect_height),
            (0, 255, 0), -1)
    elif label == 6:
        cv2.putText(frame, "Down Left", (1024, 440 - 15), font_type, 3, font_color,
line_type)
        cv2.rectangle(overlay,
            (rect_margin, frame.shape[0] // 3 * 2 + rect_margin),
            (rect_width, frame.shape[0] // 3 * 2 + rect_height),
            (0, 255, 0), -1)
    elif label == 7:
        cv2.putText(frame, "Left", (1024, 440 - 15), font_type, 3, font_color,
line_type)
        cv2.rectangle(overlay,
            (rect_margin, frame.shape[0] // 3 * 1 + rect_margin),
            (rect_width, frame.shape[0] // 3 * 1 + rect_height),
            (0, 255, 0), -1)
    else:
        cv2.putText(frame, "Up Left", (1024, 440 - 15), font_type, 3, font_color,
line_type)
        cv2.rectangle(overlay, (rect_margin, rect_margin),
            (rect_width, rect_height), (0, 255, 0), -1)

    # Set the opacity
    alpha = 0.2
    cv2.addWeighted(overlay, alpha, frame, 1 - alpha, 0, frame)
    # Draw the red ball at the previous x-y position.
    cv2.circle(frame, (self.circ_cent_x, self.circ_cent_y), 50, (0, 0, 255), -1)
    # Update the balls x-y coordinates.
    self.circ_cent_x += int(change_x)
    self.circ_cent_y += int(change_y)
    # Make sure the ball stays inside the image.
    if self.circ_cent_x > self.x_max:
        self.circ_cent_x = self.x_max
    if self.circ_cent_x < 0:
        self.circ_cent_x = 0
    if self.circ_cent_y > self.y_max:
        self.circ_cent_y = self.y_max
    if self.circ_cent_y < 0:
        self.circ_cent_y = 0
    # Send the post processed frame to the FIFO file for display
    self.result_thread.set_frame_data(frame)
```

```
def adjust_x_vel(self, velocity, pct_of_center):
    """ Helper for computing the next x coordinate.
        velocity - x direction velocity.
        pct_of_center - The percentage around the center of the image that
        we should consider a dead zone.
    """
    x_vel = 0
    if self.circ_cent_x < (1.0 - pct_of_center) * self.x_max/2:
        x_vel = velocity
    if self.circ_cent_x > (1.0 - pct_of_center) * self.x_max/2:
        x_vel = -1 * velocity
    return x_vel

def adjust_y_vel(self, velocity, pct_of_center):
    """ Helper for computing the next y coordinate.
        velocity - y direction velocity.
        pct_of_center - The percentage around the center of the image that
        we should consider a dead zone.
    """
    y_vel = 0
    if self.circ_cent_y < (1.0 - pct_of_center) * self.y_max/2:
        y_vel = velocity
    if self.circ_cent_y > (1.0 - pct_of_center) * self.y_max/2:
        y_vel = -1 * velocity
    return y_vel

def send_data(self, frame, parsed_results):
    """ Method that handles all post processing and sending the results to a FIFO file.
        frame - The frame that will be post processed.
        parsed_results - A dictionary containing the inference results.
    """
    label = parsed_results[0]["label"]
    # Use the probability to determine the velocity, scale it by 100 so that the ball
    # moves across the screen at reasonable rate.
    velocity = parsed_results[0]["prob"] * 100
    pct_of_center = 0.05

    if label == 0:
        self.update_coords(frame, velocity, velocity, label)
    elif label == 1:
        self.update_coords(frame, velocity,
                           self.adjust_y_vel(velocity, pct_of_center), label)
    elif label == 2:
        self.update_coords(frame, velocity, -1*velocity, label)
    elif label == 3:
        self.update_coords(frame, self.adjust_x_vel(velocity, pct_of_center),
                           velocity, label)
    elif label == 4:
        self.update_coords(frame, self.adjust_x_vel(velocity, pct_of_center),
                           self.adjust_y_vel(velocity, pct_of_center),
                           label)
    elif label == 5:
        self.update_coords(frame, self.adjust_x_vel(velocity, pct_of_center),
                           -1*velocity, label)
    elif label == 6:
        self.update_coords(frame, -1*velocity, velocity, label)
    elif label == 7:
        self.update_coords(frame, -1*velocity, self.adjust_y_vel(velocity,
pct_of_center),
                           label)
    elif label == 8:
        self.update_coords(frame, -1*velocity, -1*velocity, label)

def get_results(self, parsed_results, output_map):
    """ Method converts the user entered number of top inference
        labels and associated probabilities to json format.
```



```
        parsed_results - A dictionary containing the inference results.
        output_map - A dictionary that maps the numerical labels returned
                     the inference engine to human readable labels.
    """
    if self.quadrants <= 0 or self.quadrants > len(parsed_results):
        return json.dumps({"Error" : "Invalid"})

    top_result = parsed_results[0:self.quadrants]
    cloud_output = {}
    for obj in top_result:
        cloud_output[output_map[obj['label']]] = obj['prob']
    return json.dumps(cloud_output)

def head_detection():
    """ This method serves as the main entry point of our lambda.
    """
    # Creating a client to send messages via IoT MQTT to the cloud
    client = greengrasssdk.client('iot-data')
    # This is the topic where we will publish our messages too
    iot_topic = '$aws/things/{}/infer'.format(os.environ['AWS_IOT_THING_NAME'])
    try:
        # define model prefix and the amount to down sample the image by.
        input_height = 84
        input_width = 84
        model_name="frozen_model"
        # Send message to IoT via MQTT
        client.publish(topic=iot_topic, payload="Optimizing model")
        ret, model_path = mo.optimize(model_name, input_width, input_height, platform='tf')
        # Send message to IoT via MQTT
        client.publish(topic=iot_topic, payload="Model optimization complete")
        if ret is not 0:
            raise Exception("Model optimization failed, error code: {}".format(ret))
        # Send message to IoT via MQTT
        client.publish(topic=iot_topic, payload="Loading model")
        # Load the model into cl-dnn
        model = awscam.Model(model_path, {"GPU": 1})
        # Send message to IoT via MQTT
        client.publish(topic=iot_topic, payload="Model loaded")
        # We need to sample a frame so that we can determine where the center of
        # the image is located. We assume that
        # resolution is constant during the lifetime of the Lambda.
        ret, sample_frame = awscam.getLastFrame()
        if not ret:
            raise Exception("Failed to get frame from the stream")
        # Construct the custom helper object. This object just lets us handle
        postprocessing
        # in a manageable way.
        head_pose_detection = HeadDetection(sample_frame.shape[1]/2,
        sample_frame.shape[0]/2)
        # Dictionary that will allow us to convert the inference labels
        # into a human a readable format.
        output_map = ({0:'Bottom Right', 1:'Middle Right', 2:'Top Right', 3:'Bottom
        Middle',
                        4:'Middle Middle', 5:'Top Middle', 6:'Bottom Left', 7:'Middle Left',
                        8:'Top Left'})
        # This model is a ResNet classifier, so our output will be classification.
        model_type = "classification"
        # Define a rectangular region where we expect the persons head to be.
        crop_upper_left_y = 440
        crop_height = 640
        crop_upper_left_x = 1024
        crop_width = 640
        while True:
            # Grab the latest frame off the mjpeg stream.
            ret, frame = awscam.getLastFrame()
            if not ret:
```

```
        raise Exception("Failed to get frame from the stream")
    # Mirror Image
    frame = cv2.flip(frame, 1)
    # Draw the rectangle around the area that we expect the persons head to be.
    cv2.rectangle(frame, (crop_upper_left_x, crop_upper_left_y),
                  (crop_upper_left_x + crop_width, crop_upper_left_y +
crop_height),
                  (255, 40, 0), 4)
    # Crop the the frame so that we can do inference on the area of the frame where
    # expect the persons head to be.
    frame_crop = frame[crop_upper_left_y:crop_upper_left_y + crop_height,
                      crop_upper_left_x:crop_upper_left_x + crop_width]
    # Down sample the image.
    frame_resize = cv2.resize(frame_crop, (input_width, input_height))
    # Renormalize the image so that the color magnitudes are between 0 and 1
    frame_resize = frame_resize.astype(np.float32)/255.0
    # Run the down sampled image through the inference engine.
    infer_output = model.doInference(frame_resize)
    # Parse the results so that we get back a more manageable data structure.
    parsed_results = model.parseResult(model_type, infer_output)[model_type]
    # Post process the image and send it to the FIFO file
    head_pose_detection.send_data(frame, parsed_results)
    # Send the results to MQTT in JSON format.
    client.publish(topic=iot_topic,
                  payload=head_pose_detection.get_results(parsed_results,
output_map))

    except Exception as ex:
        msg = "Lambda has failure, error msg {}: ".format(ex)
        client.publish(topic=iot_topic, payload=msg)
# Entry point of the lambda function.
head_detection()
```

The `LocalDisplay` class defined in the Lambda function above serializes processed images, in a dedicated thread, to a specified FIFO file that serves as the source to replay the inference results as the project stream.

The `HeadDetection` class handles post-process of parsed data, including calling out a detected head position and pose in the project stream's local display.

The `head_detection` function coordinates the whole process to infer head poses of capture frames from AWS DeepLens video feeds, including loading the model artifact deployed to your AWS DeepLens device. To load the model, you must specify the path to the model artifact. To get this path you can call the `mo.optimize` method and specify `"frozen_model"` as the `model_name` input value. This model name corresponds to the file name without the `.pb` extension of the model artifact uploaded to Amazon S3.

Having trained your model and uploaded it to S3 and created the Lambda function in your account, you're now ready to [create and deploy the head pose detection project \(p. 102\)](#).

Create and Deploy the Head Pose Detection Project

Before creating the AWS DeepLens project, you must import the model into AWS DeepLens. Because the original Amazon SageMaker-trained model artifact is converted to a protobuf file, you must treat the transformed model artifact as externally trained.

To import the customized Amazon SageMaker-trained model for head pose detection

1. Go to the AWS DeepLens console.
2. Choose **Models** from the main navigation pane.
3. Choose **Import model**.

4. On the **Import model to AWS DeepLens** page, do the following:
 - a. Under **Import source**, choose **Externally trained model**.
 - b. Under **Model settings**, do the following:
 - i. For **Model artifact path**, type the model's S3 URL, e.g., `s3://deeplens-sagemaker-models-<my-name>/headpose/Tfartifacts/<sagemaker-job-name>/output/frozen_model.pb`.
 - ii. For **Model name**, type a name for your model, e.g., `my-headpose-detection-model`.
 - iii. For **Model framework**, choose `TensorFlow`.
 - iv. For **Description - Optional**, type a description about the model, if choose to do so.

After importing the model, you can now create an AWS DeepLens project to add the imported model and the published inference Lambda function.

To create a custom AWS DeepLens project for head pose detection

1. In the AWS DeepLens console, choose **Projects** from the main navigation pane.
2. In the **Projects** page, choose **Create new project**.
3. On the **Choose project type** page, choose **Create a new blank project**. Then, choose **Next**.
4. On the **Specify project details** page, do the following:
 - Under **Project information**, type a name for your project in the **Project name** input field and, optionally, give a project description in the **Description - Optional** input field.
5. Under **Project content**, do the following:
 - a. Choose **Add model**. Then, select the radio button next to your head pose detection model imported earlier and choose **Add model**.
 - b. Choose **Add function**. Then, select the radio button next to the published Lambda function for head pose detection and choose **Add function**.

The function must be published in AWS Lambda and named with the **deeplens-** prefix to make it visible here. If you've published multiple versions of the function, make sure to choose the function of the correction version.
6. Choose **Create**.

With the project created, you're ready to deploy it to your registered AWS DeepLens device. Make sure to remove any active project from the device before proceeding further.

1. In the **Projects** page of the DAWS DeepLensL console, choose the newly created project for head pose detection.
2. In the selected project details page, choose **Deploy to device**.
3. On the **Target device** page, select the radio button next to your registered device and then choose **Review**.
4. On the **Review and deploy** page, choose **Deploy** after verifying the project details.
5. On the device details page, examine the deployment status by inspecting **Device status** and **Registration status**. If they're **Online** and **Registered**, respectively, the deployment succeeded. Otherwise, verify that the imported model and the published Lambda function are valid.
6. After the project is successfully deployed, you can view the project output in one of the following ways:
 - a. View the JSON-formatted output published by the inference Lambda function in the AWS IoT Core console. For instructions, see [the section called "View Project Output" \(p. 35\)](#).

- b. View streaming video in a supported web browser. For instructions, [the section called "View Video Stream from AWS DeepLens Device in Browser."](#) (p. 46).

This completes this tutorial to build and deploy your head pose detection project.

Managing Your AWS DeepLens Device

The following topics explain how to manage your AWS DeepLens device and access device resource from a AWS IoT Greengrass Lambda function:

Topics

- [Securely Boot Your AWS DeepLens Device \(p. 105\)](#)
- [Update Your AWS DeepLens Device \(p. 105\)](#)
- [Deregistering Your AWS DeepLens Device \(p. 106\)](#)
- [Deleting AWS DeepLens Resources \(p. 107\)](#)

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 \(p. 30\)](#)). 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.

Note

If your updates gets stuck in an endless loop, try to check and turn on the **Unsupported updates** option under **Install updates from:** on the **Updates** tab in **Software & Updates**, an Ubuntu system utility on the device.

To manually update your AWS DeepLens on the device

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 an SSH terminal

1. Go to the AWS DeepLens console, and do the following:

- a. Choose **Devices** from the main navigation pane.
- b. From the device list, choose your AWS DeepLens device to open the device information page.
- c. From the **Device details** section, copy a local IP address of the selected AWS DeepLens device.



2. Start a terminal and type the following SSH command, using the above example's IP address (192.168.1.36):

```
ssh aws_cam@192.168.1.36
```

3. Run each of the following commands:

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

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

1. Open the AWS DeepLens console at <https://console.aws.amazon.com/deeplens/home?region=us-east-1#firstrun/>.
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 \(p. 12\)](#)
- [Connect to Your AWS DeepLens Device's Wi-Fi Network \(p. 26\)](#)
- [Set Up Your AWS DeepLens Device \(p. 30\)](#)

- [Verify Your AWS DeepLens Device Registration Status \(p. 34\)](#)

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 \(p. 106\)](#).

Topics

- [Delete AWS DeepLens Resources Using the AWS Console \(p. 107\)](#)

Delete AWS DeepLens Resources Using the AWS Console

Topics

- [Delete an AWS DeepLens Project \(p. 107\)](#)
- [Delete an AWS DeepLens Model \(p. 107\)](#)
- [Delete an AWS DeepLens Device Profile \(p. 107\)](#)
- [Delete AWS DeepLens Logs from CloudWatch Logs \(p. 108\)](#)

Delete an AWS DeepLens Project

1. Open the AWS DeepLens console at <https://console.aws.amazon.com/deeplens/home?region=us-east-1#firstrun/>.
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.

Delete an AWS DeepLens Model

1. Open the AWS DeepLens console at <https://console.aws.amazon.com/deeplens/home?region=us-east-1#firstrun/>.
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.

Delete an AWS DeepLens Device Profile

To deleting an unused device profile, you deregister the device. For more information, see [Deregistering Your AWS DeepLens Device \(p. 106\)](#).

Delete AWS DeepLens Logs from CloudWatch Logs

If you exposed personally identifiable information in the AWS IoT 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 IoT 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 \(p. 109\)](#)
- [Troubleshooting AWS DeepLens \(p. 111\)](#)

AWS DeepLens Project Logs

When you create a new project, AWS DeepLens automatically configures [AWS IoT Greengrass Logs](#). AWS IoT 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 IoT 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 \(p. 109\)](#)
- [File System Logs for AWS DeepLens \(p. 110\)](#)

CloudWatch Logs Log Groups for AWS DeepLens

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

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

Log Group	Contents
aws/greengrass/ GreengrassSystem/spectre	Diagnostic logs related to the local shadow of AWS IoT
aws/greengrass/ GreengrassSystem/syncmanager	Diagnostic logs for the device's AWS IoT topic subscription
aws/greengrass/ GreengrassSystem/tes	Diagnostic logs related to provisioning IAM credentials for calling a Lambda function
aws/greengrass/ Lambda/ <i>lambda_function_name</i>	Diagnostic logs reported by the specified Lambda inference function of your AWS DeepLens project

For more information about CloudWatch Logs, including access role and permissions requirements and limitations, see [CloudWatch Logs](#) in the *AWS IoT Greengrass Developer Guide*.

File System Logs for AWS DeepLens

AWS IoT 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.

Log Directory	Description
/opt/awscam/greengrass/ ggc/var/log	The top-level directory for the AWS IoT Greengrass logs for AWS DeepLens. The crash logs are in the crash.log file in this directory.
/opt/awscam/greengrass/ ggc/var/log/system	The subdirectory that contains the AWS IoT Greengrass system component logs.
/opt/awscam/greengrass/ ggc/var/log/user	The subdirectory that contains logs for user-defined Lambda inference functions.

For more information about AWS IoT Greengrass file system logs for AWS DeepLens, including access requirements, see [File System Logs](#) in the *AWS IoT Greengrass Developer Guide*.

Troubleshooting AWS DeepLens

Use the following guidelines to troubleshoot issues with AWS DeepLens.

Topics

- [Troubleshooting AWS DeepLens Software Issues](#) (p. 111)
- [Troubleshooting Device Registration Issues](#) (p. 115)
- [Troubleshooting Issues with Model Deployments to the AWS DeepLens Device](#) (p. 121)

Troubleshooting AWS DeepLens Software Issues

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

Topics

- [How to Determine Which Version of AWS DeepLens Software You Have?](#) (p. 111)
- [How to Check the Latest Version of the awscam Software Package?](#) (p. 112)
- [How to Reactivate Wi-Fi Hotspots After Restarting the Device?](#) (p. 112)
- [How to Ensure That a Successfully Deployed Lambda Inference Function Can Be Called?](#) (p. 113)
- [How to Update the awscam Software Package to the Latest Version?](#) (p. 113)
- [How to Upgrade the awscam Software Package Dependencies?](#) (p. 114)
- [How to Install Security Updates on the Device?](#) (p. 115)
- [How to Handle Unsupported Architecture During a Device Update?](#) (p. 115)
- [How to Reset the Device Software to the Factory Settings?](#) (p. 115)

How to 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.
4. Make note of the **Device version**.

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
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                Version             Architecture        Description
+++-=====
ii  awscam                1.2.3              amd64              awscam
```

How to Check the Latest Version of the awscam Software Package?

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
```

How to 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.

How to 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 [How to Determine Which Version of AWS DeepLens Software You Have? \(p. 111\)](#) to roll back the Linux kernel to Ubuntu 4.10.17+.

How to Update the awscam Software Package 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-conf/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 install awscam
```

Allow several minutes for the update to finish. If AWS DeepLens reports an error during the upgrade process, see [How to Upgrade the awscam Software Package Dependencies?](#) (p. 114).

How to Upgrade the awscam Software Package 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:

```
sudo apt-get upgrade
```

How to 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:

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

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

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

If you get `unmet dependencies` error messages, follow the instruction in [How to Upgrade the awscam Software Package Dependencies?](#) (p. 114) to resolve the errors.

How to Handle Unsupported Architecture During a Device Update?

If you run the `sudo apt-get update` command and get the `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.

How to 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

- [Why Does My Attempt to Use the AWS DeepLens Console to Register a Device Go into an Apparently Endless Loop and Fail to Complete?](#) (p. 116)
- [How to Ensure My AWS DeepLens 2019 Edition Device Is Detectable During Registration?](#) (p. 116)
- [How to Turn your Device Back to its Setup Mode to Complete Device Registration or Update Device Settings?](#) (p. 117)

- [How to Connect Your Device to Your Home or Office Wi-Fi Network When the Wi-Fi SSID or Password Contains Special Characters?](#) (p. 117)
- [What IAM Roles to Choose When None Is Available?](#) (p. 118)
- [How to View Project Output in the Chrome Browser on Mac El Capitan or Earlier?](#) (p. 118)
- [How to Fix an Incomplete Certificate \(.zip\) File?](#) (p. 119)
- [How to Fix an Incorrectly Uploaded Certificate File?](#) (p. 119)
- [How to Resolve the Maximum User Limits Exceeded Restriction for Devices, Projects, or Models?](#) (p. 119)
- [How to Fix Failed Deregistration of the AWS DeepLens Device?](#) (p. 120)
- [How to Resolve Failed Registration of the AWS DeepLens Device?](#) (p. 120)
- [How to Open the Device Setup Page When the Device's Local IP Address \(192.168.0.1\) Is Not Accessible?](#) (p. 120)
- [How to Make the Device's Wi-Fi Network Visible on Your Computer When the Device Wi-Fi Indicator Is Blinking or After Your Computer Has Connected to the Device?](#) (p. 121)
- [How to Fix an Unsuccessful Device Wi-Fi Reset?](#) (p. 121)

Why Does My Attempt to Use the AWS DeepLens Console to Register a Device Go into an Apparently Endless Loop and Fail to Complete?

When you use the AWS DeepLens console to register a device, if the registration fails to complete after you choose **Register device**, you may have used an unsupported browser. Make sure that you do not use the following browser:

Unsupported Browser for the AWS DeepLens Console

- Internet Explorer 11 on Windows 7

How to Ensure My AWS DeepLens 2019 Edition Device Is Detectable During Registration?

When registering your AWS DeepLens 2019 Edition device, your computer may not detect the device after you've connected your device to your computer with a USB cable. This would prevent you from completing the registration.

To ensure that your device can be detected, make sure the following conditions are met:

- Your device is powered on.
- Your computer is connected to the device through the REGISTRATION USB port at the bottom.
- Your computer is not connected to Ethernet and a VPN network.
- Your firewall is not blocking the DeepLens network connection.
- You're using Chrome, Firefox or Safari and have the latest OS update installed.

If the above conditions are met and your device still remains undetected, disconnect and reconnect the USB cable to the REGISTRATION USB port on the device, then wait for a minute or two.

The network interface service order on your computer may play a role to prevent your device from being detected during registration. For example, if the USB-C interface has a higher preference over the standard USB interface on your computer, your computer could default to USB-C to scan attached USB devices. Your AWS DeepLens device would become invisible. In this case, you need to reorder the

preference of network types. To do so, you can follow, for example, [this tip on a Mac computer](#) or [this tip on a Windows computer](#).

How to Turn your Device Back to its Setup Mode to Complete Device Registration or Update Device Settings?

If you don't see your device's Wi-Fi network SSID among the available Wi-Fi networks on your computer and the device's Wi-Fi indicator (the middle LED light) does not blink, you cannot connect to the device to set up the device and to complete the registration.

Your device is in its setup mode when the Wi-Fi indicator (the middle LED light) on the front of the device blinks. When you start registering your device, the device is booted into the setup mode after you power it on. Only in the setup mode is the device's Wi-Fi (AMDC-**NNNN**) network visible in the list of available Wi-Fi networks on your computer. After 30 minutes, the device exits the setup mode, and the Wi-Fi indicator stays on and stops blinking,

Your device must be in the setup mode for you to connect your computer to the device, to configure the device to complete the registration or to update the device settings, or to view the output of a deployed project. If you don't finish the setup within the allotted time, you must reset the device back to its setup mode and then reconnect your computer to the device's Wi-Fi network.

After connecting your computer to the device's Wi-Fi network, navigate to `http://deeplens.config` from a web browser to open the device configuration page and to complete setting up your device. For an initial registration, you may want type `http://deeplens.amazon.net` in the browser's address bar to open the configuration page.

For detailed instructions to connect to your device's Wi-Fi network, see [the section called "Connect to Device" \(p. 26\)](#).

How to Connect Your 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, back slashes, or white spaces), you might not be able to connect your AWS DeepLens device to the home or office Wi-Fi network when setting up your device on the device configuration page (`http://deeplens.amazon.net` or `http://deeplens.config`).

To work around this, follow the steps below to use SSH commands to log into the device and connect it to your home or office Wi-Fi network.

To connect your DeepLens device to your home or office Wi-Fi network when the Wi-Fi SSID and password contains special characters

1. Connect to your device's Wi-Fi network.
 - a. Power on your DeepLens and press the power button on the front to turn on the device. Skip this step if the device is already turned on.
 - b. Wait until the device enters into the setup mode when the Wi-Fi indicator on the front of the device starts to flash. If you skipped the previous step, you can turn the device into its setup mode by pressing a paper clip into the reset pinhole on the back of the device and, after feeling a click, wait for about 20 seconds for the Wi-Fi indicator to blink.
 - c. Connect your computer to the device's Wi-Fi network by selecting the device Wi-Fi network SSID (of the AMDC-**NNNN** format) from the list of available networks and type the network password. The SSID and password are printed on the bottom of your device. On a Windows computer, choose **Connecting using a security key instead** when prompted for **Enter the PIN from the router label (usually 8 digits)**.

2. After the Wi-Fi connection is established, open a web browser on your 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**.

Important

After choosing **Save** do not choose **Finish**. Otherwise, you'll be disconnected from the device and can't proceed in the following step.

If you've chosen **Finish**, follow **Step 1-3** above to get back to the device's Wi-Fi network before continuing to the next step.

4. While your computer is connected to the device (AMDC-**NNNN**) network Wi-Fi connection, open a terminal on your 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 to execute the SSH command.

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 wlan0
```

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. Go to `http://deeplens.amazon.net` or `http://deeplens.config` on your browser to continue setting up the device, as prescribed by [Step 3.b in Set Up Your Device \(p. 30\)](#).

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 repeat this workaround.

What IAM Roles to Choose When None Is Available?

If [the required IAM roles \(p. 9\)](#) are not available in your account when you register your device using the AWS DeepLens console for the first, you will be prompted to create the roles with a single click of button. If you wish to create the required IAM roles without using the AWS DeepLens console, follow the instructions in [the section called "Create IAM Roles for Your Project" \(p. 10\)](#).

How to View Project Output in the Chrome Browser on Mac El Capitan or Earlier?

On a Mac computer (El Capitan and earlier), you must provide a password to upload the streaming certificate into the key chain while using Chrome. To use Chrome to view project output streamed from your device, you must update the device software to associate the password of DeepLens with the

certificate first and then use the password to add the streaming certificate to the key chain. The steps are detailed in the following procedure.

To set up the DeepLens password on the device to enable viewing the project output in Chrome on Mac El Capitan or earlier

1. Update the device software to set up the DeepLens password for the streaming certificate:
 - a. Connect to your AWS DeepLens device using an SSH or uHDMI connection.
 - b. On the device terminal, type `$wget https://s3.amazonaws.com/cert-fix/cert_fix.zip`.
 - c. Type `$unzip cert_fix.zip` to unzip the file.
 - d. Type `$cd cert_fix` to step into the unzipped directory.
 - e. Type `$sudo bash install.s` to install the updates.
 - f. Restart your device.
2. After the device is started, upload the streaming certificate to your Mac (El Capitan or earlier) for viewing project output:
 - a. [Connect \(p. 26\)](#) your computer to the device's Wi-Fi (AMDC-~~NNNN~~) network
 - b. Open the device setup page in your browser using the `deeplens.config` URL
 - c. Choose **Enable streaming certificates**. Ignore the browser instructions.
 - d. Choose the **Download streaming certificates** button.
 - e. After the certificates are downloaded, rename the downloaded certificates from `download` to `my_device.p12` or `some_other_unique_string.p12`.
 - f. Double click on your certificate to upload it into the key chain.
 - g. The key chain will ask you for a certificate password, enter DeepLens.
 - h. To view the project output streamed from your device, open Chrome on your Mac computer and navigate to your device's video server web page at `https://your_device_ip:4000` or use the AWS DeepLens console to open the page.

You'll be asked to select a streaming certificate from the key chain. However, the key chain refers to any streaming certificate uploaded in **Step 2.f** above as `client cert`. To differentiate any multiple like-named key-chain entries, use the expiration date and validation date when choosing the streaming certificate.
 - i. Add the security exception. When prompted, enter your computer system's username and password, and choose to have the browser remember it so you don't have to keep entering it.

How to Fix an Incomplete Certificate (.zip) File?

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

How to Fix an Incorrectly Uploaded Certificate File?

If you accidentally uploaded an incorrect certificate (.zip) file, upload the correct certificate (.zip) file to overwrite the incorrect one.

How to Resolve the Maximum User Limits Exceeded Restriction for Devices, Projects, or Models?

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

How to Fix Failed Deregistration of the AWS DeepLens Device?

Stop deregistration and reregister the device.

To stop deregistering and then reregister an AWS DeepLens device

1. Turn the device back to its setup 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.

How to 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 IoT 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 \(p. 12\)](#) and check with the AWS DeepLens console tips for each role.

How to Open the Device Setup Page When the Device's Local IP Address (192.168.0.1) Is Not Accessible?

If your AWS DeepLens device is running software (`awscam`) version 1.3.6 or later, you can open the device setup page by navigating to `http://deeplens.amazon.net` (the first time you register the device) or `http://deeplens.config` in a browser window on your computer. Your computer must be [connected to the device's \(AMDC-nnnn\) Wi-Fi network \(p. 26\)](#).

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, you can navigate to `http://192.168.0.1` to launch the device setup page if your AWS DeepLens device software package is older than 1.2.4 or you can navigate to `http://10.105.168.217` if the device software package is 1.2.4 or newer.

If your device software is older than 1.2.4 and 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 to, for example, `192.168.1.0`. For instructions on resetting your router's IP address, see the documentation for your router. For example, to change a D-Link router 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.

How to Make the Device's Wi-Fi Network Visible on Your Computer When the Device Wi-Fi Indicator Is Blinking or After Your Computer Has Connected to the Device?

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

It takes up to 2 minutes to reboot a device and for the device Wi-Fi network to be ready to accept connection requests.

How to Fix an Unsuccessful Device Wi-Fi Reset?

If the device's Wi-Fi indicator (middle LED light) isn't blinking after you reset the device by pressing a pin into the reset pinhole on the back of it, connect the device to the internet with an Ethernet cable, connect the device to a monitor, a keyboard and a mouse, log in to the device, open a terminal window, 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  
sudo reboot
```

The successful reboot sets up the device's Wi-Fi network, and the middle LED light on the device will start to blink.

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

- [How to Resolve an Access Denied Error Encountered While Downloading a Model After Device Registration Went through without Errors? \(p. 122\)](#)
- [How to Resolve the ModelDownloadFailed Error? \(p. 122\)](#)
- [Resolve Model Optimization Failure Reported As std::bad_alloc\(\) \(p. 122\)](#)
- [How to Resolve Model Optimization Failure Caused by a Missing stride Attribute? \(p. 122\)](#)
- [How to Resolve Model Optimization Failure When the List Object Is Missing the shape Attribute? \(p. 123\)](#)
- [How to Ensure Reasonable Inference for a Model? \(p. 124\)](#)
- [How to Determine Why AWS DeepLens Classifies Data Incorrectly When the Model Performs Well on the Validation Set? \(p. 124\)](#)

How to 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 IoT 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.

How to Resolve the ModelDownloadFailed Error?

When using the AWS DeepLens console, you must provide two IAM roles for AWS IoT 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.

How to 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:

```
{
  "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](#).

How to 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:

```
{
  "op": "Convolution",
  "name": "deep_dog_conv0_fwd",
  "attr": {
    "dilata": "(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".

In contrast, the following is an example of a null operator:

```
{
  "op": "null",
  "name": "deep_dog_conv0_weight",
  "attr": {
    "__dtype__": "0",
    "__lr_mult__": "1.0",
    "__shape__": "(64, 0, 3, 3)",
    "__wd_mult__": "1.0"
  }
},
```

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 \(p. 38\)](#).

How to 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.

How to 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 \(p. 125\)](#)
- [Model Optimization \(mo\) Module \(p. 131\)](#)
- [DeepLens_Kinesis_Video Module for Amazon Kinesis Video Streams Integration \(p. 140\)](#)

awscam Module for Inference

The `awscam` module lets you grab video frames from your AWS DeepLens device and load a deep learning model into the inference engine of the device. Use the model to run inference on captured image frames.

Topics

- [awscam.getLastFrame Function \(p. 125\)](#)
- [Model Object \(p. 126\)](#)

awscam.getLastFrame Function

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

Request Syntax

```
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

class awscam.Model

Represents an AWS DeepLens machine learning model.

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

Methods

- [Model Class Constructor \(p. 127\)](#)
- [model.doInference Method \(p. 128\)](#)
- [model.parseResult Method \(p. 130\)](#)

Model Class Constructor

Creates an `awscam.Model` object to run inference in a specific type of processor and a particular inference runtime. For models that are not Neo-compiled, the `Model` object exposes parsing the raw inference output in order to return model-specific inference results.

Request Syntax

```
import awscam
model = awscam.Model(model_topology_file, loading_config, runtime=0)
```

Parameters

- `model_topology_file`—Required. When `runtime=0`, this parameter value is the path to an optimized model (.xml) file output by the [mo.optimize](#) (p. 127) method. When `runtime=1`, this parameter value is the name of the folder containing the compiled model artifacts, which include a .json file, a .params file, and a .so file.

Neo-compiled models support only the Classification model type. Other models support Classification, Segmentation, and Single Shot MultiBox Detector (SSD).

Note

When deploying an Amazon SageMaker-trained SSD model, you must first run `deploy.py` (available from <https://github.com/apache/incubator-mxnet/tree/master/example/ssd/>) to convert the model artifact into a deployable mode. After cloning or downloading [the MXNet repository](#), run the `git reset --hard 73d88974f8bca1e68441606fb0787a2cd17eb364` command before calling `deploy.py` to convert the model, if the latest version does not work.

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

Valid values:

- `{"GPU": 1}`—Loads the model into the GPU. To run inference in the Neo runtime (`runtime=1`), you must select GPU (`{ 'GPU' : 1 }`).
- `{"GPU": 0}`—Loads the model into the CPU.
- `runtime` int—Optional. Specifies the runtime to run inference in.

Valid values:

- 0—Intel DLDT as the default inference runtime.
- 1—The Neo runtime to run inference on a compiled model.

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

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

Parameters

- `video_frame`—Required. A video frame from the AWS DeepLens video feeds.

Return Type

- `dict` — The inference result, as a dictionary of output layers based on the loaded model.

Returns

Returns a `dict` object with entries of key-value pairs. The key of an entry is the identifier of an output layer of the model. For the output of inference running in the default Intel DLDT runtime, the output layer identifier is the output layer name as specified in the model. For example, with MXNet, the output layer names are defined in the model's `.json` file. For the output of inference running in the Neo runtime, the output layer identifier is the ordinal number of the output layer as defined in the model. The value of an entry is a `list` object that holds the probabilities of the input `video_frame` over the labeled images used to train the model.

Example

Sample output:

Running inference in the Intel DLDT runtime on a model with an output layer named `SoftMax_67` of six labels, the inference outcome has the following format.

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

If the Neo runtime (`runtime=1`) is used to run the inference, the raw inference result is a `dict` object, where `results[i]` holds the i^{th} output layer. For example, for an image classification network such as Resnet-50 with only one Softmax output layer, `results[0]` is a Numpy array with size $N \times 1$ that gives the probability for each of the N labeled image types.

The following example output shows the raw inference result of the same model (above) running in the Neo runtime.

```
{  
  0 : array(  
    [  
      2.41881448e-08,  
      3.57339691e-09,  
      1.00263861e-07,  
      5.40415579e-09,  
      4.37702547e-04,  
      6.16787545e-08  
    ],  
    dtype=float32)  
}
```

model.parseResult Method

Parses the raw inference results of some commonly used models, such as classification, SSD, and segmentation models, which are not Neo-compiled. For customized models, you write your own parsing functions.

Request Syntax

```
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`

Note

When deploying an Amazon SageMaker-trained SSD model, you must first run `deploy.py` (available from <https://github.com/apache/incubator-mxnet/tree/master/example/ssd/>) to convert the model artifact into a deployable mode. After cloning or downloading the [MXNet repository](#), run the `git reset --hard 73d88974f8bca1e68441606fb0787a2cd17eb364` command before calling `deploy.py` to convert the model, if the latest version does not work.

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

Return Type

- `dict`

Returns

Returns a `dict` with a single entry of key-value pair. The key is the `model_type` value. The value is a list of `dict` objects. Each of the returned `dict` objects contains the probability calculated by the model for a specified label. For some model types, such as `ssd`, the returned `dict` object also contains other information, such as the location and size of the bounding box the inferred image object is located in.

Example

Sample output:

For the raw inference performed in the Intel DLDT runtime, the parsed inference result on a `classification` model type has the following format:

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

The corresponding parsed inference result on an `ssd` model type contains bounding box information and has the following format:

```
{
  "ssd": [
    {"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}
  ]
}
```

For inferences performed in the Neo runtime, the parsed inference result is the same as the raw inference result.

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 \(p. 38\)](#) 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 Method \(p. 131\)](#)
- [Troubleshooting the Model Optimizer \(p. 134\)](#)

mo.optimize Method

Converts AWS DeepLens model artifacts from a Caffe (`.prototxt` or `.caffemodel`), MXNet (`.json` and `.params`), or TensorFlow (`.pb`) representation to an AWS DeepLens representation and performs necessary optimization.

Syntax

```
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:

Value	Description
Caffe or caffe	The optimization converts Caffe model artifacts (of the <code>.prototxt</code> or <code>.caffemodel</code> files) to AWS DeepLens model artifacts.
MXNet, mxNet or mx	The optimization converts Apache MXNet model artifacts (of the <code>.json</code> and <code>.params</code> files) to AWS DeepLens model artifacts. This is the default option.
TensorFlow or tensorflow or tf	The optimization converts TensorFlow model artifact (of the frozen graph <code>.pb</code> files) 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

Item Name	Applicable Platforms	Description
<code>--img-format</code>	All	Image format. The default value is BGR.
<code>--img-channels</code>	All	Number of image channels. The default value is 3.
<code>--precision</code>	All	Image data type. The default value is FP16.
<code>--fuse</code>	All	A switch to turn on (ON) or off (OFF) fusing of linear operations to convolution. The default value is ON.
<code>--models-dir</code>	All	Model directory. The default directory is <code>/opt/awscam/artifacts</code> .
<code>--output-dir</code>	All	Output directory. The default directory is <code>/opt/awscam/artifacts</code> .
<code>--input_proto</code>	Caffe	The <code>prototxt</code> file path. The default value is an empty string (<code>" "</code>).

Item Name	Applicable Platforms	Description
<code>--epoch</code>	MXNet	Epoch number. The default value is 0.
<code>--input_model_is_text</code>	TensorFlow	A Boolean flag that indicates whether the input model file is in text protobuf format (<code>True</code>) or not (<code>False</code>). The default value is <code>False</code> .

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. For possible cause of failures and corrective actions when the method call fails, see the status table below.

Type: integer

status	Cause	Action
0	Model optimization succeeded.	No action needed.
1	Model optimization failed because the requested platform is not supported.	<ul style="list-style-type: none"> Choose a supported platform. Make sure that the platform name is spelled correctly.
2	Model optimization failed because you are using inconsistent platform versions.	<ul style="list-style-type: none"> 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>. Make sure that there are no unsupported layers in the model for the target platform. Make sure that your awscam software is up-to-date. See Troubleshooting the Model Optimizer (p. 134) for recommended actions for error messages reported in the CloudWatch Logs for AWS DeepLens and on your AWS DeepLens device.

To load the optimized model for inference, call the [awscam.Model \(p. 127\)](#) 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? \(p. 135\)](#)
- [How to create a bare Caffe model with only prototxt? \(p. 135\)](#)
- [How to handle the "Unable to create ports for node with id." error? \(p. 135\)](#)
- [How to Handle "Invalid proto file: there is neither 'layer' nor 'layers' top-level messages." error? \(p. 135\)](#)
- [How to handle the "Old-style inputs \(via 'input_dims'\) are not supported. Please specify inputs via 'input_shape'." error? \(p. 135\)](#)
- [How to handle the "Invalid prototxt file: value error." error? \(p. 137\)](#)
- [How to handle the "Error happened while constructing caffe.Net in the Caffe fallback function." error? \(p. 137\)](#)
- [How to handle the "Cannot infer shapes due to exception in Caffe." error? \(p. 137\)](#)
- [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? \(p. 137\)](#)
- [How to handle the "Input shape is required to convert MXNet model. Please provide it with --input_width and --input_height." error? \(p. 137\)](#)
- [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? \(p. 138\)](#)
- [How to handle the "Specified input model does not exist." error? \(p. 138\)](#)
- [How to handle the "Failed to create directory ... Permission denied!" error? \(p. 138\)](#)
- [How to handle the "Discovered data node without inputs and value." error? \(p. 138\)](#)
- [How to handle the "Placeholder node doesn't have input port, but input port was provided." error? \(p. 138\)](#)
- [How to handle the "No or multiple placeholders in the model, but only one shape is provided, cannot set it." error? \(p. 138\)](#)
- [How to handle the "Cannot write an event file for the tensorboard to directory." error? \(p. 138\)](#)
- [How to handle the "Stopped shape/value propagation at node." error? \(p. 138\)](#)
- [How to handle the "Module tensorflow was not found. Please install tensorflow 1.2 or higher." error? \(p. 139\)](#)
- [How to handle the "Cannot read the model file: it is incorrect TensorFlow model file or missing." error? \(p. 139\)](#)
- [How to handle the "Cannot pre-process TensorFlow graph after reading from model file. File is corrupt or has unsupported format." error? \(p. 139\)](#)
- [How to handle the "Input model name is not in an expected format, cannot extract iteration number." error? \(p. 139\)](#)
- [How to handle the "Cannot convert type of placeholder because not all of its outputs are 'Cast' to float operations." error? \(p. 139\)](#)
- [How to handle the "Data type is unsupported." error? \(p. 139\)](#)
- [How to handle the "No node with name ..." error? \(p. 139\)](#)
- [How to handle the "Module mxnet was not found. Please install mxnet 1.0.0" error? \(p. 139\)](#)
- [How to handle the "The following error happened while loading mxnet model ..." error? \(p. 139\)](#)

- [How to handle the "... elements of ... were clipped to infinity while converting a blob for node \[...\] to ..." error?](#) (p. 140)
- [How to handle the "... elements of ... were clipped to zero while converting a blob for node \[...\] to ..." error?](#) (p. 140)
- [How to handle the " topology contains no 'input' layers." error?](#) (p. 140)

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:

```
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:

```
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:

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

2. Input layer format 2:

```
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:

```
layer
{
  name: "data"
  type: "Input"
  top: "data"
  input_param {shape: {dim: 1 dim: 3 dim: 600 dim: 1000}}
}

layer
{
  name: "im_info"
  type: "Input"
```

```
top: "im_info"
input_param {shape: {dim: 1 dim: 3}}
```

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. 135) 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. 138).

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. 135).

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 " 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 use the `DeepLens_Kinesis_Video` module to stream five-hour video feeds from the caller's AWS DeepLens device to Kinesis Video Streams.

```
import time
import os
import DeepLens_Kinesis_Video as dkv
from botocore.session import Session
import greengrasssdk

def greengrass_hello_world_run():
    # Create the green grass client so that we can send messages to IoT console
    client = greengrasssdk.client('iot-data')
```



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

# Stream configuration, name and retention
# Note that the name will appear as deeplens-myStream
stream_name = 'myStream'
retention = 2 #hours

# Amount of time to stream
wait_time = 60 * 60 * 5 #seconds

# Use the boto session API to grab credentials
session = Session()
creds = session.get_credentials()

# Create producer and stream.
producer = dkv.createProducer(creds.access_key, creds.secret_key, creds.token, "us-east-1")
client.publish(topic=iot_topic, payload="Producer created")
kvs_stream = producer.createStream(stream_name, retention)
client.publish(topic=iot_topic, payload="Stream {} created".format(stream_name))

# Start putting data into the KVS stream
kvs_stream.start()
client.publish(topic=iot_topic, payload="Stream started")
time.sleep(wait_time)
# Stop putting data into the KVS stream
kvs_stream.stop()
client.publish(topic=iot_topic, payload="Stream stopped")

# Execute the function above
greengrass_hello_world_run()
```

In this example, we call `dkv.createProducer` to instantiate the Kinesis Video Streams Producer SDK client. We then call the `producer.createStream` to set up streaming from the AWS DeepLens device. We control the length of video feeds by calling `my_stream.start`, `time.sleep` and `my_stream.stop`. The stored video is retained in Kinesis Video Streams for two hours because we set the retention parameter to 2.

Topics

- [DeepLens_Kinesis_Video.createProducer Function \(p. 141\)](#)
- [Producer Object \(p. 142\)](#)
- [Stream Object \(p. 143\)](#)

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

```
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. 142)

Example

```
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. 142)

Producer.createStream Method

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

Syntax

```
stream = producer (p. 142).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 \(p. 143\)](#) 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.

Topics

- [Stream.isActive Property \(p. 143\)](#)
- [Stream.start Method \(p. 143\)](#)
- [Stream.stop Method \(p. 144\)](#)

Stream.isActive Property

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

Syntax

```
stream (p. 143).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. 143).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. 143).stop()
```

Parameters

- None

Returns

- None

AWS DeepLens Security

Topics

- [Data Protection \(p. 145\)](#)
- [Authentication and Access Control \(p. 145\)](#)
- [Incident Response \(p. 145\)](#)
- [Update Management \(p. 145\)](#)

Data Protection

AWS DeepLens does not collect data from the device. Video frames from the camera and inference results are confined to the device locally. As a user, you control how and where to publish inference results to other AWS services in the project's Lambda function. Your account remains the sole ownership of the affected data regardless which service it is published to.

The AWS DeepLens supports hardware-level full-disk encryption for the data storage on device. The data encryption does not require any user intervention or configuration.

For the purpose of device registration and maintenance, communications between your AWS DeepLens device and your computer requires HTTPS connection, except for connecting to the device softAP endpoint in the device setup mode that requires authentication with password. The default password is printed at the bottom of the AWS DeepLens device.

For on-device disk encryption, keys are stored in a hardware component known as Trusted Platform Module (TPM). When the disk is removed from the device, data becomes inaccessible.

Authentication and Access Control

See [Set Up Required Permissions \(p. 9\)](#).

Incident Response

To detect device software issues, check the [logs on the device \(p. 110\)](#) or the [CloudWatch Logs for AWS DeepLens \(p. 109\)](#).

Update Management

AWS DeepLens makes available device software updates. As a user, you decide whether or not to install the updates to the device. You can do so in the AWS DeepLens console. For a security patch, you'll be forced to install the patch.

AWS DeepLens provides device software updates, periodically, as software patches or new features. You can choose to re-flash the image.

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.

update-history-change	update-history-description	update-history-date
The release of AWS DeepLens 2019 Edition (p. 146)	Improves the device registration experience and adds support of Amazon SageMaker Neo for automatic inference optimization. For more information, see Register AWS DeepLens 2019 Edition Device and Use Amazon SageMaker Neo to Optimize Inference on AWS DeepLens .	July 10, 2019
The Bird Classification Sample Project (p. 146)	Adds Bird Classification to the AWS DeepLens Sample Projects. For more information see The Bird Classification Sample Project .	April 8, 2019
Support for models created with TensorFlow and Caffe (p. 146)	Adds TensorFlow and Caffe as supported modeling frameworks. For more information see Supported TensorFlow Models and Layers and Supported Caffe Models and Layers .	June 14, 2018
Simplified IAM roles creation (p. 146)	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 Register Your Device .	June 14, 2018
Secure boot (p. 146)	Adds secure booting. The AWS DeepLens device securely boots to prevent the installation of unauthorized operating systems. For more information see Securely Boot Your Device .	June 14, 2018
Integration with Amazon Kinesis Video Streams (p. 146)	Uses the Kinesis Video Streams for AWS DeepLens Video library to send video feeds from an AWS DeepLens device to Kinesis	June 14, 2018

	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 AWS DeepLens Kinesis Video Integration Library Module .	
Ability to view device output streams in a browser (p. 146)	Adds support for using a browser to view unprocessed device streams or processed project streams. For more information, see View Your AWS DeepLens Device Output in a Browser .	June 14, 2018
Named device setup URL (p. 146)	Makes the AWS DeepLens device setup app accessible through the URL of <code>http://deeplens.config</code> . For more information see Set up Your Device .	June 5, 2018
Troubleshooting Guide (p. 146)	Provides a list of commonly asked questions and answers about troubleshooting common AWS DeepLens issues. For more information see Troubleshooting Guide .	May 3, 2018
Gluon support (p. 146)	Adds support for Gluon models. For more information, see Supported MXNet Models Exposed by the Gluon API .	March 13, 2018
Importing from Amazon SageMaker (p. 146)	Simplifies the process for importing a model trained with Amazon SageMaker. For more information, see Importing Your Amazon SageMaker-Trained Model .	February 22, 2018
Model optimization (p. 146)	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 .	January 31, 2018

[AWS DeepLens Developer Guide \(p. 146\)](#)

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.

November 29, 2017

AWS Glossary

For the latest AWS terminology, see the [AWS Glossary](#) in the *AWS General Reference*.