# Table of Contents

What Is AWS DeepLens? ................................................................. 1
Hardware ......................................................................................... 1
Supported Frameworks ................................................................. 2
For First-time Users ....................................................................... 2
More Info ....................................................................................... 2
How AWS DeepLens Works .......................................................... 3
Getting Started ............................................................................. 4
Prerequisites ................................................................................... 4
  Create an AWS Account ............................................................... 4
  Create an IAM User ..................................................................... 5
Register AWS DeepLens ............................................................... 5
Connect Your AWS DeepLens ...................................................... 8
Set Up Your AWS DeepLens ....................................................... 9
Verify Your AWS DeepLens Is Connected .................................... 9
Working with Projects ................................................................. 10
Supported Frameworks ............................................................... 10
  Apache MXNet .......................................................................... 10
Deploying a Project ...................................................................... 11
Viewing Project Output .............................................................. 12
  Viewing a Device Stream .......................................................... 12
  Viewing a Project Stream ......................................................... 12
Creating a Lambda Function for Viewing the Project Stream ....... 13
Working with Sample Projects .................................................... 16
  Sample Projects Overview ...................................................... 16
  Creating and Deploying a Sample Project ................................. 17
  Extending Functionality ........................................................... 19
  Editing an Existing Model with Amazon SageMaker ................. 23
Working with Custom Projects .................................................... 34
  Optimizing a Custom Model .................................................... 35
  Creating an Inference Lambda .................................................. 36
Managing Your Device ............................................................... 42
  Updating Your AWS DeepLens Device ..................................... 42
  Deregister Your AWS DeepLens .............................................. 42
Model Optimizer API ................................................................. 44
  Method mo.optimize() ............................................................ 44
Device Library ............................................................................. 46
  Model ...................................................................................... 46
  Constructor ............................................................................. 47
  model.doInference(video_frame) ................................................. 48
  model.parseResult(model_type, raw_infer_result) ..................... 49
  Method awscam.getLastFrame() ................................................. 50
Appendix ...................................................................................... 51
Troubleshooting ......................................................................... 52
  Create IAM Roles .................................................................... 53
    Create an AWS DeepLens Service Role ................................. 54
    Create a Greengrass Service Role ......................................... 55
    Create a Greengrass Group Role ........................................... 56
    Create an Amazon SageMaker Role .................................... 57
    Create an AWS Lambda Role ............................................... 57
Document History ........................................................................ 58
AWS Glossary ............................................................................. 59
What Is AWS DeepLens?

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

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

AWS DeepLens works with the following AWS services:

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

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

Topics

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

AWS DeepLens Hardware

The AWS DeepLens camera includes the following:

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

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

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

Currently, AWS DeepLens supports only the Apache MXNet framework. For more information, see Apache MXNet (p. 10).

Are You an AWS DeepLens First-time User?

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

1. **Read How AWS DeepLens Works (p. 3)**—Explains AWS DeepLens and how to use it to develop computer vision applications.
2. **Explore Amazon SageMaker**—Explains some of the basic functionality of Amazon SageMaker. AWS DeepLens uses Amazon SageMaker to build and train CNN models. You use Amazon SageMaker to create your own AWS DeepLens models and projects.
3. **Learn about the AWS DeepLens Device Library (p. 46)**—The device library describes the classes and methods that you can use in your Lambda functions.
4. **Perform the tasks in Getting Started with AWS DeepLens (p. 4)**—Set up your AWS account, create the AWS Identity and Access Management (IAM) permissions and roles that you need to run AWS DeepLens, and register and set up your AWS DeepLens device.

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

a. **Creating and Deploying an AWS DeepLens Sample Project (p. 17)**—Walks you through creating sample AWS DeepLens project, which is included with your device.

b. **Editing an Existing Model with Amazon SageMaker (p. 23)**—Walks you through creating and training a model using Amazon SageMaker.

c. **Extending any Project’s Functionality (p. 19)**—Walks you through taking output from your AWS DeepLens and using it to trigger an action.

More Info

- AWS DeepLens Forum
How AWS DeepLens Works

The following diagram illustrates how AWS DeepLens works.

1. When turned on, the AWS DeepLens captures a video stream.
2. Your AWS DeepLens produces two output streams:
   - **Device stream**—The video stream passed through without processing
   - **Project stream**—The results of the model's processing video frames

   1. The Inference Lambda function receives unprocessed video frames.
   2. The Inference Lambda function passes the unprocessed frames to the project's deep learning model, where they are processed.
   3. The Inference Lambda function receives the processed frames from the model and passes the processed frames on in the project stream.

<table>
<thead>
<tr>
<th>Label</th>
<th>Prob</th>
</tr>
</thead>
<tbody>
<tr>
<td>794</td>
<td>0.6810352202623</td>
</tr>
<tr>
<td>669</td>
<td>0.0477042198101</td>
</tr>
<tr>
<td>905</td>
<td>0.0444286055863</td>
</tr>
<tr>
<td>564</td>
<td>0.0426652309893</td>
</tr>
<tr>
<td>706</td>
<td>0.0312749966979</td>
</tr>
</tbody>
</table>

For more information, see Viewing AWS DeepLens Project Output (p. 12).
Getting Started with AWS DeepLens

Before using AWS DeepLens, register your device, connect it, set it up, and verify that it’s connected. The following graphic shows where you perform each step.

Topics
• Prerequisites (p. 4)
• Register Your AWS DeepLens Device (p. 5)
• Connect AWS DeepLens to the Network (p. 8)
• Set Up Your AWS DeepLens Device (p. 9)
• Verify That Your AWS DeepLens Is Connected (p. 9)

Prerequisites

Before you can begin using AWS DeepLens, you need an AWS account and an IAM user.

Create an AWS Account

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

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

To create an AWS account

2. Choose Create a Free Account.
3. Follow the instructions on the page.

Part of the sign-up process involves receiving a phone call and entering a PIN using the phone keypad.
Create an IAM User

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

Important
Record your Access Key and Secret Key. You need them to make calls with the AWS CLI.

To create an IAM user

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

Now that you have an AWS account and IAM user, continue to Register Your AWS DeepLens Device (p. 5).

Register Your AWS DeepLens Device

Use your computer’s browser to register your device and download a certificate.

Before you continue, complete the Prerequisites (p. 4).

To register AWS DeepLens

2. Choose Register device.
3. For Device name, type a name for your AWS DeepLens, then choose Next.

Use only alphanumeric characters and dashes (-).

4. If this is your first time registering an AWS DeepLens device, create the following AWS Identity and Access Management (IAM) roles. They give AWS DeepLens the permissions it needs to perform tasks on your behalf.

If you have already created these roles, skip to step 5.

a. IAM role for AWS DeepLens

From the list, choose AWSDeepLensServiceRole. If AWSDeepLensServiceRole isn't listed, choose Create role in IAM and follow these steps in the IAM console.

i. Accept the DeepLens service and DeepLens use case by choosing Next: Permissions.
ii. Accept the AWSDeepLensServiceRolePolicy policy by choosing Next: Review.
iii. Accept the role name `AWSDeepLensServiceRole` and the provided description by choosing *Create role*. Do not change the role name.

iv. Close the IAM window.

b. **IAM role for AWS Greengrass service**

From the list, choose `AWSDeepLensGreengrassRole`. If `AWSDeepLensGreengrassRole` isn't listed, choose *Create role in IAM* and follow these steps in the IAM console.

i. Accept the *Greengrass* service and *Greengrass* use case by choosing *Next: Permissions*.

ii. Accept the `AWSGreengrassResourceAccessRolePolicy` policy by choosing *Next: Review*.

iii. Accept the role name `AWSDeepLensGreengrassRole` and the provided description by choosing *Create role*. Do not change the role name.

iv. Close the IAM window.

c. **IAM role for AWS Greengrass device groups.**

From the list, choose `AWSDeepLensGreengrassGroupRole`. If `AWSDeepLensGreengrassGroupRole` isn't listed, choose *Create role in IAM* and follow these steps in the IAM console.

i. Accept the *DeepLens* service and the *DeepLens - Greengrass Lambda* use case by choosing *Next: Permissions*.

ii. Accept the `AWSDeepLensLambdaFunctionAccessPolicy` policy by choosing *Next: Review*.

iii. Accept the role name `AWSDeepLensGreengrassGroupRole` and the provided description by choosing *Create role*. Do not change the role name.

iv. Close the IAM window.

d. **IAM role for Amazon SageMaker**

From the list, choose `AWSDeepLensSagemakerRole`. If `AWSDeepLensSagemakerRole` isn't listed, choose *Create role in IAM* and follow these steps in the IAM console.

i. Accept the *SageMaker* service and the *SageMaker - Execution* use case by choosing *Next: Permissions*.

ii. Accept the `AmazonSageMakerFullAccess` policy by choosing *Next: Review*.

iii. Accept the role name `AWSDeepLensSagemakerRole` and the provided description by choosing *Create role*. Do not change the role name.

iv. Close the IAM window.

e. **IAM role for AWS Lambda**

From the list, choose `AWSDeepLensLambdaRole`. If `AWSDeepLensLambdaRole` isn't listed, choose *Create role in IAM* and follow these steps i the IAM console.

i. Accept the *Lambda* service and the *Lambda* use case by choosing *Next: Permissions*.

ii. Accept the `AWSLambdaFullAccess` policy by choosing *Next: Review*.

iii. Accept the role name `AWSDeepLensLambdaRole` and the provided description by choosing *Create role*. Do not change the role name.

iv. Close the IAM window.

5. In AWS DeepLens, on the *Set permissions* page, choose *Refresh IAM roles*, then do the following:

- For *IAM role for AWS DeepLens*, choose `AWSDeepLensServiceRole`.
- For *IAM role for AWS Greengrass service*, choose `AWSDeepLensGreengrassRole`.
- For *IAM role for AWS Greengrass device groups*, choose `AWSDeepLensGreengrassGroupRole`.
- For *IAM role for Amazon SageMaker*, choose `AWSDeepLensSagemakerRole`.
- For *IAM role for AWS Lambda*, choose `AWSDeepLensLambdaRole`. 
Important

- Attach the roles exactly as described. Otherwise, you might have trouble deploying models to AWS DeepLens.
- If any of the lists do not have the specified role, find that role in step 4, follow the directions to create the role, choose Refresh IAM roles, and return to where you were in step 5.

6. Choose Next.
7. On the Download certificate page, choose Download certificate, then choose Save File. Note where you save the certificate file because you need it later.

After the certificate has been downloaded, choose Register.

Important
The certificate is a .zip file. You attach it to AWS DeepLens in .zip format, so don’t unzip it. Certificates aren’t reusable. You need to generate a new certificate every time you register your device.
Connect AWS DeepLens to the Network

Before you can use AWS DeepLens, you have to connect it to the network.

To connect to your AWS DeepLens

1. Start your AWS DeepLens device by plugging the power cord into an outlet and the other end into the back of your device. Turn on the AWS DeepLens by pressing the On/Off button on the front of the device.

2. On your computer, choose the SSID for your AWS DeepLens from the list of available networks. The SSID and password are on the bottom of your device.

When prompted, type the AWS DeepLens password.
Set Up Your AWS DeepLens Device

Use your computer to set up your AWS DeepLens device.

To set up your device

1. In a browser, open a new tab and navigate to http://192.168.0.1.
2. On the Device page:
   a. Connect to the network.
      Choose your local network, type the password, then choose Next. If you are using Ethernet to connect to AWS DeepLens, choose the Ethernet option.
   b. Upload the certificate.
      Locate and choose the certificate that you downloaded from the AWS DeepLens console, then choose Upload certificate.
      The certificate is saved as a .zip file in your Downloads directory. Don’t unzip the file. You attach the certificate as a .zip file.
   c. Configure device access.
      i. Create a password for the device—You need this password to access and update your AWS DeepLens.
      ii. SSH server—We recommend disabling SSH. SSH allows you to log in without using the AWS DeepLens console.
      iii. Automatic updates—We recommend enabling this option. Enabling automatic updates keeps your device’s software up-to-date.
   d. Review the settings and finish setting up the device.
      To modify settings, choose Edit for the setting that you want to change.
      Choose Finish.

Verify That Your AWS DeepLens Is Connected

After you set up your device, your computer automatically connects to the internet. This can take a few seconds. When your device is connected, you see the following message:

After the connection is established, you can return to the AWS DeepLens console. You are now ready to deploy an AWS DeepLens project. For more information, see Creating and Deploying an AWS DeepLens Sample Project (p. 17).

If you fail to establish a connection, return to Connect AWS DeepLens to the Network (p. 8) and repeat the steps for setting up your device and connecting it to the network.
Working with AWS DeepLens Projects

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

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

Topics
- AWS DeepLens Supported Frameworks (p. 10)
- Deploying an AWS DeepLens Project (p. 11)
- Viewing AWS DeepLens Project Output (p. 12)
- Working with AWS DeepLens Sample Projects (p. 16)
- Working with AWS DeepLens Custom Projects (p. 34)

AWS DeepLens Supported Frameworks

Currently, AWS DeepLens supports only the Apache MXNet framework and the following MXNet layers.

Apache MXNet

Supported MXNet Layers

<table>
<thead>
<tr>
<th>Activation</th>
<th>BatchNorm</th>
<th>Concat</th>
<th>Convolution</th>
<th>elemwise_add</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pooling</td>
<td>Flatten</td>
<td></td>
<td>FullyConnected</td>
<td>InputLayer</td>
</tr>
<tr>
<td>Reshape</td>
<td>ScaleShift</td>
<td></td>
<td>SoftmaxActivation</td>
<td>Smoothing</td>
</tr>
</tbody>
</table>

_contrib_MultiBoxPrior contrib_MultiBoxDetection _Plus                Deconvolution _mul
Deploying an AWS DeepLens Project

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

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

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

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

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

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

For instructions on viewing your project’s output, see Viewing AWS DeepLens Project Output (p. 12).
AWS DeepLens Developer Guide
Viewing Project Output

Viewing AWS DeepLens Project Output

AWS DeepLens 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 done by the model on the video frames.

**Topics**
- Viewing a Device Stream (p. 12)
- Viewing a Project Stream (p. 12)
- Creating a Lambda Function for Viewing the Project Stream (p. 13)

**Viewing a Device Stream**

**To view the unprocessed device stream**
1. Plug your AWS DeepLens 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.

**Viewing a Project Stream**

**To view a project stream**
1. Plug your AWS DeepLens 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:

```
mplayer --demuxer lavf -lavfdopts format=mjpeg:probesize=32 /tmp/results.mjpeg
```
6. To stop viewing the video stream and end your terminal session, press Ctrl+C.
Creating a Lambda Function for Viewing the Project Stream

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

Create a Lambda function for your custom projects

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

```python
# Copyright Amazon AWS DeepLens, 2017
import os
import greengrasssdk
from threading import Timer
import time
import awscam
import cv2
from threading import Thread

# Creating a greengrass core sdk client
client = greengrasssdk.client('iot-data')

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

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

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

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

def greengrass_infinite_infer_run():
    try:
        modelPath = '/opt/awscam/artifacts/mxnet_deploy_ssd_resnet50_300_FP16_FUSED.xml'
        modelType = 'ssd'
        input_width = 300
        input_height = 300
        max_threshold = 0.25
```
results_thread = FIFO_Thread()
results_thread.start()

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

# Load model to GPU (use {"GPU": 0} for CPU)
mcfg = {"GPU": 1}
model = awscam.Model(modelPath, mcfg)
client.publish(topic=iotTopic, payload="Model loaded")

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 ret == False:
        raise Exception("Failed to get frame from the stream")
    # Resize frame to fit model input requirement
    frameResize = cv2.resize(frame, (input_width, input_height))
    # Run model inference on the resized frame
    inferOutput = model.doInference(frameResize)
    # Output inference result 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 += '}

    label += '"null": 0.0'
    label += '}
    cv2.putText(frame, label, (xmin, ymin-15), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 165, 20), 4)
    cv2.putText(frame, "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 above
greengrass_infinite_infer_run()

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

After you've created and deployed the Lambda function, see Viewing a Project Stream (p. 12).
Working with AWS DeepLens Sample Projects

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

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

Topics
- AWS DeepLens Sample Projects (p. 16)
- Creating and Deploying an AWS DeepLens Sample Project (p. 17)
- Extending any Project's Functionality (p. 19)
- Editing an Existing Model with Amazon SageMaker (p. 23)

AWS DeepLens Sample Projects

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.

Object Recognition

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

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

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

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.

**Hot Dog Recognition**

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

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

You can edit this model by creating Lambda functions that are triggered by the model's output. For example, if the model detects a hot dog, a Lambda function could send you an SMS message. To learn how to create this Lambda function, see Editing an Existing Model with Amazon SageMaker (p. 23)

**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.

**Activity 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, juggligng, 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.

**Creating and Deploying an AWS DeepLens Sample Project**

To help you get started with AWS DeepLens, we provide a number of sample AWS DeepLens project templates that you can use to create projects and get you up and going quickly. For more information, see AWS DeepLens Sample Projects (p. 16).

In this walkthrough, you create the Object Detection project. The Object Detection project analyzes images within a video stream on your AWS DeepLens device to identify objects. It can recognize as many as 20 types of objects.
Your web browser is the interface between you and your AWS DeepLens device. You perform all of the following activities on your browser after logging on to AWS:

Step 1: Create Your Project

The following procedure creates the Object Detection sample project.

To create an AWS DeepLens project using a sample project

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

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

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

### Step 2: Deploy Your Project

In this walkthrough, you deploy the Object Detection project.

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

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

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

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

**Important**

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

For instructions on viewing your project's output, see **Viewing AWS DeepLens Project Output** (p. 12).

### Extending any Project's Functionality

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

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

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

In this exercise, you modify the project you previously created and edited (see **Editing an Existing Model with Amazon SageMaker** (p. 23)) to use the AWS IoT rules engine and an AWS Lambda function.

**Contents**

- **Create and Configure the Lambda Function** (p. 20)
  - **Create a Lambda Function** (p. 20)
  - **Add an AWS IoT Rule** (p. 20)
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. Place your cursor in the Blueprints box, then choose Blueprint name. When Blueprint name: appears, type and choose iot-button-email.

4. Choose Author from scratch.

5. Type a name for the Lambda function, for example, <your name>_hotdog_notifier.

6. For Role, keep Create a new Role from template(s).

7. Type a name for the role; for example, <your name>_hotdog_notifier.

8. For Policy Templates, choose SNS Publish policy.

9. Choose Create function.

Add an AWS IoT Rule

1. Scroll down to aws-iot.

2. For IoT type, choose Custom IoT rule.

3. For Rule, choose Create new rule.

4. Type a name (<your-name>_search_hotdogs) and a description for the rule.

5. Paste the following into the Rule query statement box. Replace the red text with the AWS IoT topic for your AWS DeepLens. To find the AWS IoT topic, navigate to Devices on your AWS DeepLens, choose your device, then scroll to the bottom of the device detail page.

   Select Hotdog from '/$aws/deeplens/KJHFD-DKJO87-LJLKD/inference'

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

   { "Hotdog" : "0.5438" }

6. Choose Enable trigger.

7. Scroll to the bottom of the page and choose Create function.

Configure the Lambda Function

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

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

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

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

   ```javascript
   /*
   * This is a sample Lambda function that sends an SMS notification when your
   * AWS DeepLens device detects a hot dog.
   * Follow these steps to complete the configuration of your function:
   * Update the phone number environment variable with your phone number.
   */
   const AWS = require('aws-sdk');

   /**
   * Replace the next line of code with one of the lines of code from the list following
   * this code block.
   */
   const var=process.env.var;
   const SNS = new AWS.SNS({ apiVersion: '2010-03-31' });

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

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

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

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

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

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

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

**To test your configuration**

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

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

Test Using the Hot Dog Project

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

1. Navigate to https://console.aws.amazon.com/deeplens/home?region=us-east-1#firstrun/ and choose Projects/Create a project template/Hotdog or Not Hotdog.
2. Deploy the project to your device.

   For more information, see Creating and Deploying an AWS DeepLens Sample Project (p. 17).
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.

2. Choose Act, then choose the rule that you created for this exercise, `<your-name>_search_hotdogs`.
3. In the upper-right corner, choose Actions, then choose Disable.
Editing an Existing Model with Amazon SageMaker

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

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

Contents

- Step 1: Create an Amazon S3 Bucket (p. 24)
- Step 2: Create an Amazon SageMaker Notebook Instance (p. 25)
- Step 3: Edit the Model in Amazon SageMaker (p. 27)
- Step 4: Optimize the Model (p. 28)
- Step 5: Import the Model (p. 29)
- Step 6: Create an Inference Lambda Function (p. 30)
- Step 7: Create a New AWS DeepLens Project (p. 31)
- Step 8: Review and Deploy the Project (p. 32)
- Step 9: View Your Model’s Output (p. 33)
Step 1: Create an Amazon S3 Bucket

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

1. Sign in to the AWS Management Console and open the Amazon S3 console at https://console.aws.amazon.com/s3/.

   Make sure you are in the US East (N. Virginia) region.

2. Choose Create bucket.

3. On the Name and region screen:
   a. Name the bucket `deeplens-sagemaker-your-full-name`. It is important that the bucket name begins with `deeplens-sagemaker-` or the services will not be able to access it.
   b. Verify that you are in US East (N. Virginia).
   c. Choose Next.

4. On the Set properties screen choose Next.

5. On the Set permissions screen, verify that both Objects and Object permissions have both the Read and Write permissions set, then choose Next.

6. On the Review screen, review your settings then choose Create bucket.

   You return to the Amazon S3 screen.

7. On the Amazon S3 screen, locate and choose your bucket's name.

8. On your bucket's screen, choose Permissions, then under Public access choose Everyone.

9. On the Everyone popup, under Access to objects enable List objects and Write objects. Under Access to this bucket's ACL enable Read bucket permissions and Write bucket permissions, then choose Save.

10. After you return to your bucket's page, choose Overview then choose Create folder.

11. Name the folder test then choose Save.
Step 2: Create an Amazon SageMaker Notebook Instance

Create an Amazon SageMaker notebook instance.

1. Open the Amazon SageMaker console at https://console.aws.amazon.com/sagemaker/.
   Make sure that you have chosen the us-east-1 — US East (N. Virginia) Region.
2. Choose Create notebook instance.
3. On the Create notebook instance page, do the following:
   a. For Notebook instance name, type a name; for example, `<your-name>-hotdog`.
   b. For Instance type, choose ml.t2.medium.
   c. For IAM role choose Enter a custom IAM role ARN, paste the Amazon Resource Name (ARN) of your Amazon SageMaker role in the Custom IAM role ARN box.
      To find the ARN of your Amazon SageMaker role:
      i. Open the IAM console at https://console.aws.amazon.com/iam/.
      ii. In the navigation pane, choose Roles.
      iii. Find the AWSDeepLensSagemakerRole and choose its name. This takes you to the role's Summary page.
      iv. On the Summary page, locate and copy the Role ARN. The ARN will look something like this:

```
arn:aws:iam::<account id>:role/AWSDeepLensSagemakerRole
```
   d. Both VPC and Encryption key are optional. Skip them.
      
      **Note**
      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 notebook instance.
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.

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

   ```bash
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>` (macOS `Cnd+_<Enter>`) or choose the run icon (†).

   **Important**

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

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

   - Hotdog_or_not_model-0000.params
   - Hotdog_or_not_model-symbol.json

27
Step 4: Optimize the Model

Now that you have a trained mxNet model there is one final step that is required before you run the model on the AWS DeepLens’s GPU. The trained mxNet model does not come in a computationally optimized format. If we deploy the model in the original format it will run on the CPU via mxNet at sub optimal speed. In order to run the model at optimal speed on the GPU we need to perform model optimization. For instructions on how to optimize your MXNet model, see Optimizing a Custom Model (p. 35).
Step 5: Import the Model

Import the edited model into AWS DeepLens.

2. Choose Models, then choose Import model.
3. For Import model to AWS DeepLens, choose Externally trained model.
4. For Model settings, do the following:
   a. For Model artifact, type the path to the artifacts that you uploaded to the Amazon S3 bucket in the previous step. The path begins with s3://, for example 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.

You can now use the AWS Lambda console to create a Lambda function that uses your model, create a project, and deploy it to your AWS DeepLens device.
Step 6: Create an Inference Lambda Function

In this section you will use the AWS Lambda console to create a Lambda function that uses your model. For specific instructions with sample code, see Creating an AWS DeepLens Inference Lambda Function (p. 36).
Step 7: Create a New AWS DeepLens Project

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

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

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

   **Important**

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

To view your model's output, follow the instructions at Viewing AWS DeepLens Project Output (p. 12).
When you are ready, you can create and deploy your own models. The topics in this section cover the tasks you need to perform to create, optimize, and deploy your personal projects.

Topics

- Optimizing a Custom Model (p. 35)
- Creating an AWS DeepLens Inference Lambda Function (p. 36)
Optimizing a Custom Model

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

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

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

For information on how to create an inference Lambda function that includes the model optimizer, see Creating an AWS DeepLens Inference Lambda Function (p. 36).

For more information about the model optimizer, see AWS DeepLens Model Optimizer API (p. 44).
Creating an AWS DeepLens Inference Lambda Function

In this topic, you create an inference Lambda function that performs three key functions: preprocessing, inference, and postprocessing. Each step is accompanied by the associated code. For the complete function code, see The Completed Lambda Function (p. 40).

To create an AWS DeepLens inference Lambda function

1. Sign in to the AWS Management Console and open the AWS Lambda console at https://console.aws.amazon.com/lambda/.
2. Choose Create function, then choose Blueprints.
3. Choose the Blueprints box then from the dropdown, choose Blueprint name and type greengrass-hello-world.
4. When the greengrass-hello-world blueprint appears, choose it, then choose Configure.
5. In the Basic information section:
   a. Type a name for your Lambda function.
   b. From the Role list, choose Choose an existing role.
   c. Choose AWSDeepLensLambdaRole, which you created when you registered your device.
6. Scroll to the bottom of the page and choose Create function.
7. In the function code box, makes sure that the handler is greengrassHelloWorld.function_handler. The name of the function handler must match the name of the Python script that you want to run. In this case, you are running the greengrassHelloWorld.py script, so the name of the function handler is greengrassHelloWorld.function_handler.
8. Delete all of the code in the GreengrassHelloWorld.py box and replace it with the code that you generate in the rest of this procedure.
9. Add the following code to your Lambda function.
   a. Import these required packages:
      
      ```python
      import os
      import awscam
      import mo
      import cv2
      from threading import Thread
      ```
   b. Create a Greengrass SDK client. You will use this client to send messages to the cloud.
      ```python
      client = greengrasssdk.client('iot-data')
      ```
c. Create an AWS IoT topic for your Lambda function's messages. You can access this topic in the AWS IoT console.

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

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

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

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

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

f. Define an inference class.

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

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

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

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

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

iii. Initialize the model optimizer. The model optimizer converts the deployed model to cLDNN format, which is accessible to the AWS DeepLens GPU. The model optimizer returns the path to the post-optimized artifacts.

```python
```
iv. Load the model into the inference engine. To use the CPU, instead of the GPU, specify "GPU":0. The CPU is much less efficient, so we don't recommend using it.

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

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

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

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

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

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

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

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

```python
topk = 5
```

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

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

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

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

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

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

xi. Perform inference on the resized frame.

infer_output = model.doInference(frame_resize)

def doInference(frame)

xii. Parse the results.

parsed_results = model.parseResult(model_type, infer_output)

def parseResult(model_type, result)

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

top_k = parsed_results[model_type][0:topk]

def top_k

xiv. Send the results to the cloud.

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

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

Then send it to the cloud.

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

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

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

def putText(frame, label, (0,22), FONT_HERSHEY_SIMPLEX, 1, (255, 165, 20), 4)

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

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

dxvii. Run the function and view the results.

```
greengrass_infinite_infer_run()
```

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

The Completed Lambda Function

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

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

client = greengrasssdk.client('iot-data')
iot_topic = '$aws/things/{}/infer'.format(os.environ['AWS_IOT_THING_NAME'])
jpeg = None
Write_To_FIFO = True

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

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

def greengrass_infinite_infer_run():
    input_width  = 224
    input_height = 224
    model_name = 'squeezenet_v1'
    error, model_path = mo.optimize(model_name, input_width, input_height)
    model = awscam.Model(model_path, {'GPU':1})
    client.publish(topic=iot_topic, payload="Model loaded.")

    def __init__(self):
        Thread.__init__(self)

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

def greengrass_infinite_infer_run():
    input_width  = 224
    input_height = 224
    model_name = 'squeezenet_v1'
    error, model_path = mo.optimize(model_name, input_width, input_height)
    model = awscam.Model(model_path, {'GPU':1})
    client.publish(topic=iot_topic, payload="Model loaded.")
```

# define the type of model
# possibilities are:
# segmentation - for neural style transfers
# ssd - (single shot detector) for object localization
# classification - for image classification
model_type = "classification"

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

# define the number of classifiers to see
topk = 5

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

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

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

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

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

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

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

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

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

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

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

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

Topics
- Updating Your AWS DeepLens Device (p. 42)
- Deregister Your AWS DeepLens (p. 42)

Updating Your AWS DeepLens Device

When you set up your device, you had the option to enable automatic updates (see Step 3.iii in Set Up Your AWS DeepLens Device (p. 9)). If you enabled automatic updates, you don't need to do anything more to update the software on your device. If you didn't enable automatic updates, you need to manually update your device periodically.

To manually update your AWS DeepLens using your password

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

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

To manually update your AWS DeepLens using your IP address

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

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

3. Run each of the following commands:

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

Deregister Your AWS DeepLens

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

2. Remove all projects associated with the device.
   a. Choose Devices, then choose the radio button of the device that you want to deregister.
   b. In Projects, choose Remove projects.

   **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.

3. Deregister the device.
   a. Choose Devices.
   b. Choose the name of the device you want to deregister, then choose Deregister.
   c. When the warning appears, choose Deregister.

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

- Register Your AWS DeepLens Device (p. 5)
- Connect AWS DeepLens to the Network (p. 8)
- Set Up Your AWS DeepLens Device (p. 9)
- Verify That Your AWS DeepLens Is Connected (p. 9)
AWS DeepLens Model Optimizer API

Custom models run inefficiently on the CPU. To run a custom model efficiently on the GPU, use the AWS DeepLens model optimizer API. The model optimizer class has a single method, `mo.optimize`, which optimizes your model to CI-DNN format so it can run on the GPU.

```python
class mo
```

Represents an AWS DeepLens model optimizer.

Methods

- `Method mo.optimize(model_name, input_width, input_height, [platform], [aux_inputs]) (p. 44)`

**Method mo.optimize(model_name, input_width, input_height, [platform], [aux_inputs])**

Optimizes the AWS DeepLens model specified by `model_name` so that it runs efficiently on the AWS DeepLens GPU, instead of the CPU.

**Request Syntax**

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

**Parameters**

- `model_name`—Required. The name of the model to optimize. Type: string.
- `input_width`—Required. The width of the input image in pixels. The value must be a non-negative integer less than or equal to 1024. Type: integer.
- `input_height`—Required. The height of the input image in pixels. The value must be a non-negative integer less than or equal to 1024. Type: integer.
- `platform`—Optional. The platform for this optimization. The default value is `mxNet`, which is currently the only supported platform. Type: string.
- `aux_inputs`—Optional. A Python dictionary that contains auxiliary inputs that are not common to all platform. Type: Dict.

**mxNet Platform aux_inputs**

<table>
<thead>
<tr>
<th>aux_input name</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--img-format</td>
<td>BGR</td>
</tr>
<tr>
<td>--img-channels</td>
<td>3</td>
</tr>
<tr>
<td>--precision</td>
<td>FP16</td>
</tr>
<tr>
<td>--fuse</td>
<td>ON</td>
</tr>
<tr>
<td>--models-dir</td>
<td>/opt/awscam/artifacts</td>
</tr>
</tbody>
</table>
**Method mo.optimize()**

<table>
<thead>
<tr>
<th>aux_input name</th>
<th>Default value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--output-dir</td>
<td>/opt/awscam/artifacts</td>
</tr>
</tbody>
</table>

**Response**

- **error**—The error code. For error codes, their cause, and corrective actions, see the following table. Type: integer.
- **model_path**—For convenience, this method returns the path and the artifact name so that you don't need to know the name or the path of the artifact to load the model. To load the model, call the model optimizer API and specify the model_path. Type: string.

**Error Codes**

<table>
<thead>
<tr>
<th>Error Code</th>
<th>Description and Action</th>
</tr>
</thead>
</table>
| 0          | **Cause:** Model optimizer ran successfully.  
**Action:** None. |
| 1          | **Cause:** The requested platform is not supported.  
Currently, mxNet is the only supported platform.  
**Action:**  
- Make sure that the platform name is spelled correctly.  
- Choose a supported platform. |
| 2          | **Cause:** Model optimizer failed.  
**Action:**  
- Make sure that you are running the latest version of the platform. To check your version, at a command prompt, run `pip install mxnet`.  
- Make sure that there are no unsupported layers in the model for the target platform.  
- Make sure that the model optimizer is up to date. |
AWS DeepLens Device Library

The AWS DeepLens Python device library provides classes and methods for the following:

- Running your inference code on your AWS DeepLens device. You can use these classes and methods in Lambda functions in your inference Lambda code.
- Uploading the AWS DeepLens video stream to the Kinesis Video Streams stream in the AWS Cloud.

AWS DeepLens Device Library

- **Model** (p. 46)
  - Constructor (p. 47)
  - model.doInference(video_frame) (p. 48)
  - model.parseResult(model_type, raw_infer_result) (p. 49)
- Method awscam.getLastFrame() (p. 50)

Model

class awscam.Model

Represents an AWS DeepLens machine learning model.

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

Methods

- Constructor (p. 47)
- model.doInference(video_frame) (p. 48)
- model.parseResult(model_type, raw_infer_result) (p. 49)
Constructor

The constructor for a awscam.Model

Request Syntax

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

Parameters

- **model_topology_file**—Required. A neural network topology file (.xml) from the Intel model optimizer.

  Models supported: Classification and Single Shot MultiBox Detector (SSD)

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

  **Permitted values:**

  - `{"GPU":1}`—Loads the model into the GPU
  - `{"GPU":0}`—Loads the model into the CPU
model.doInference(video_frame)

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

Request Syntax

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

Parameters

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

Return Type

- **dict list

Returns

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

Example

Sample output:

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

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

Request Syntax

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

Parameters

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

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

Return Type

- `dict`

Returns

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

Example

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

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

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

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

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

**Request Syntax**

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

**Parameters**

- None

**Return Type**

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

The following topics cover common issues with AWS DeepLens and how to resolve them. As we learn of more issues we will add to this section. For troubleshooting help, see AWS DeepLens Troubleshooting Guide.
Create IAM Roles for AWS DeepLens

You use AWS Identity and Access Management (IAM) to define policies and roles that are needed to access resources used by AWS DeepLens.

Important
You should not changes the policies in any of these roles. If you need additional or more restrictive policies, create a new role with a different name.
Create an AWS DeepLens Service Role

Create a service policy that allows access to AWS DeepLens resources, then create an IAM role and attach the AWS DeepLens policy to it.

**To create the AWSDeepLensServiceRole**

1. Open the IAM console at https://console.aws.amazon.com/iam/.
2. From the navigation pane, choose Roles, then choose Create role.
3. From the list of AWS service roles, choose DeepLens.
4. In the Select your use case section, choose *DeepLens Allows DeepLens to create required resources and make calls on your behalf*, then choose Next: Permissions.
5. The AWSDeepLensServiceRolePolicy policy is already listed under Policy name, so choose Next: Review.
6. For Role name, type AWSDeepLensServiceRole.
7. Choose Create role.

Your new role is now listed among the roles on the Roles screen.
Create a Greengrass Service Role

Create an AWS Greengrass service role that allows AWS DeepLens to access AWS Greengrass to deploy the model and Lambda functions to the device.

To create the AWS Greengrass service role

1. Choose Roles, then choose Create role.
2. For the trusted entity, choose AWS service, choose AWS Greengrass, then choose Next: Permissions.
3. On Attach permissions policies, search for Greengrass and choose the box to the left of the AWSGreengrassResourceAccessRolePolicy policy name. Choose Next: Review.
4. On the Review screen:
   a. Name the role AWSDeepLensGreengrassRole. The role name must begin with deeplens.
   b. Review your Trusted entities and Policies settings.
   c. To return to the previous page to make changes, choose Previous. To create the role, choose Create role.
5. After you have created the role, it's included in the list of roles. Find the AWSDeepLensGreengrassRole and choose it. On the Summary page, find and record the Role ARN. You need it later.
Create a Greengrass Group Role

Create an AWS Greengrass group role that allows your AWS DeepLens to be an AWS Greengrass core device. This defines how messages are passed between devices, the AWS Greengrass core, and Lambda functions.

To create the AWSGreengrassGroupRole

1. From the navigation pane, choose Roles, then choose Create role.
2. From the list of AWS service roles, choose DeepLens.
3. In the Select your use case section, choose DeepLens - GreenGrass Lambda Allows DeepLens to access administrative Lambda functions that run on a DeepLens device on your behalf., then choose Next: Permissions.
4. The AWSDeepLensLambdaFunctionAccessPolicy policy is already listed under Policy name, so choose Next: Review.
5. For Role name, type AWSDeepLensGreengrassGroupRole.
6. Choose Create role.

Your new role is now listed among the roles on the Roles screen.
Create an Amazon SageMaker Role

Create an Amazon SageMaker role that allows AWS DeepLens to access Amazon SageMaker for model optimization and training.

To create the AWSDeepLensSagemakerRole

1. From the navigation pane, choose Roles, then choose Create role.
2. From the list of AWS service roles, scroll down and choose Sagemaker.
3. In the Select your use case section, choose Sagemaker - Execution Allows SageMaker notebook instances, training jobs, and models to access S3, ECR, and CloudWatch on your behalf., then choose Next: Permissions.
4. The AmazonSageMakerFullAccess policy is already listed under Policy name, so choose Next: Review.
5. For Role name, type AWSDeepLensSagemakerRole.
6. Choose Create role.

Your new role is now listed among the roles on the Roles screen.

Create an AWS Lambda Role

To create the AWSDeepLensLambdaRole

1. From the navigation pane, choose Roles, then choose Create role.
2. From the list of AWS service roles, choose Lambda.
3. In the Select your use case section, choose Lambda Allows Lambda functions to call AWS services on your behalf., then choose Next: Permissions.
4. Choose Next: Review.

You do not need to attach a policy for the Lambda role.
5. For Role name, type AWSDeepLensLambdaRole.
6. Choose Create role.

Your new role is now listed among the roles on the Roles screen.
Document History for AWS DeepLens

The following table describes the documentation history for AWS DeepLens.

- **API version:** 1.2.0
- **Latest documentation update:** January 31, 2018

<table>
<thead>
<tr>
<th>Change</th>
<th>Description</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Optimization</td>
<td>AWS DeepLens added support for optimizing your custom model so it runs on the GPU rather than the CPU. For more information, see:</td>
<td>January 31, 2018</td>
</tr>
<tr>
<td></td>
<td>• Optimizing a Custom Model (p. 35)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• AWS DeepLens Model Optimzer API (p. 44)</td>
<td></td>
</tr>
<tr>
<td>New guide</td>
<td>This is the first release of the AWS DeepLens Developer Guide.</td>
<td>November 29, 2017</td>
</tr>
</tbody>
</table>
For the latest AWS terminology, see the AWS Glossary in the AWS General Reference.