AWS Machine Learning Blog

Building a Pictionary-style game with AWS DeepLens and Amazon Alexa

by Amit Choudhary and Phu Nguyen | on 05 AUG 2020 | in Artificial Intelligence, AWS DeepLens | Permalink | Comments | Share

Are you bored of the same old board games? Tired of going through the motions with charades week after week? In need of a fun and exciting way to mix up game night? Well we have a solution for you!

From the makers of AWS DeepLens, Guess My Drawing with DeepLens is a do-it-yourself recipe for building your very own Machine Learning (ML)-enabled Pictionary-style game! In this post, you learn how to harness the power of AWS DeepLens, the AWS programmable video camera for developers to learn ML, and Amazon Alexa, the Amazon cloud-based voice service.

You start by learning to deploy a trained model to AWS DeepLens that can recognize sketches drawn on a whiteboard and pair it with an Alexa skill that serves as the official scorekeeper.

When your recipe is complete, the fun begins!

Solution overview

Guess My Drawing with AWS DeepLens uses Alexa to host a multi-player drawing challenge game. To get started, gather your game supplies mentioned in the **Prerequisites** section.

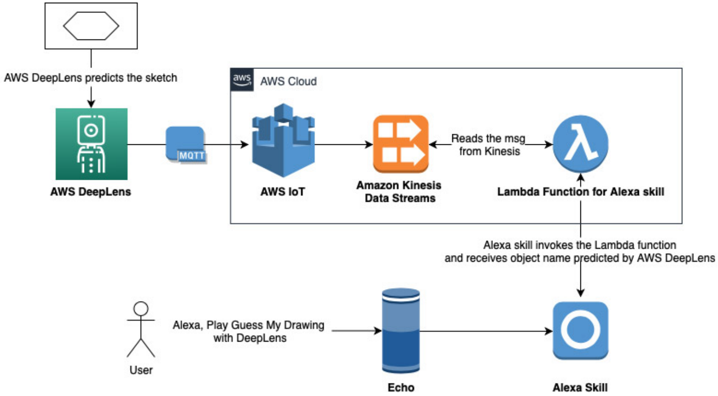
To initiate gameplay, simply say, "Alexa, play Guess My Drawing with DeepLens." Alexa explains the game rules and asks how many players are playing the game. The players decide the turn order.

Alexa provides each player with a common word. For example, Alexa may say, "Your object to draw is bowtie." The player has 12 seconds to draw it on a whiteboard without writing letters or words.

When time runs out, the player stops drawing and asks Alexa to share the results. The ML model running on AWS DeepLens predicts the object that you drew. If the object matches with what Alexa asks, Alexa awards 10 points. If DeepLens can't correctly guess the drawing or the player takes more than 12 seconds to draw, no points are earned.

Alexa prompts the next participant with their word, repeating until all participants have taken a turn. After each round, Alexa provides a score update. The game ends after five rounds, and whoever has the highest score wins the game!

The following diagram shows the architecture of our solution.



This tutorial includes the following steps:

1. Create an AWS DeepLens inference AWS Lambda function to isolate the drawing area and feed each camera frame into the model to generate predictions on the sketches.
2. Deploy a pre-trained trained model included in this post to AWS DeepLens to perform image classification.
3. Create an AWS IoT Core rule to send the results to Amazon Kinesis Data Streams.
4. Create a custom Alexa skill with a different Lambda function to retrieve the detected objects from the Kinesis data stream and have Alexa verbalize the result to you.

Prerequisites

Before you begin this tutorial, make sure you have the following prerequisites:

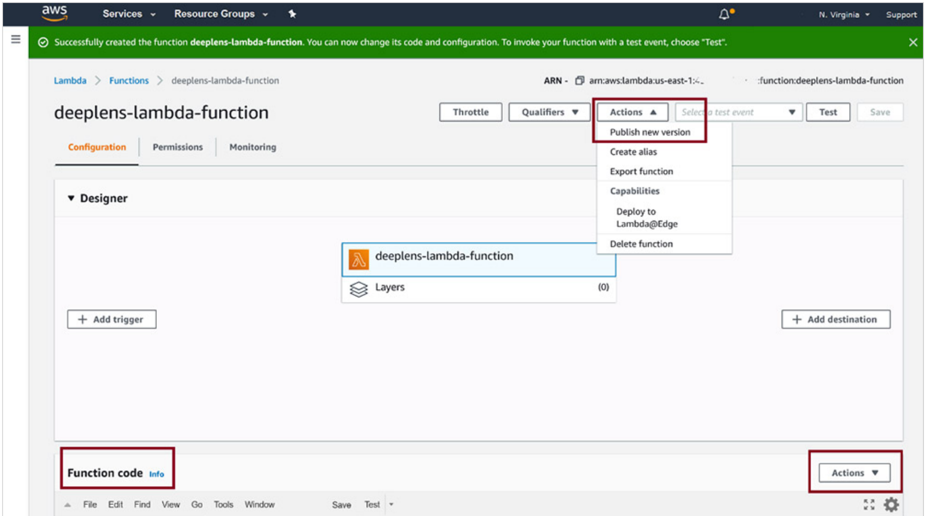
- An AWS account
- An AWS DeepLens device, available from the following:
  - Amazon.com (US)
  - Amazon.ca (Canada)
  - Amazon.co.jp (Japan)
  - Amazon.de (Germany)
  - Amazon.co.uk (UK)
  - Amazon.fr (France)
  - Amazon.es (Spain)
  - Amazon.it (Italy)
- An Amazon Alexa device
- A whiteboard or paper pad (unlined)
- A marker or writing utensil

Creating an AWS DeepLens inference Lambda function

In this section, you create an inference function that you deploy to AWS DeepLens. The inference function isolates the drawing area, optimizes the model to run on AWS DeepLens, and feeds each camera frame into the model to generate predictions.

To create your function, complete the following steps:

1. Download aws-deeplens-pictionary-lambda.zip.
2. On the Lambda console, choose **Create function**.
3. Choose **Author from scratch** and choose the following options:
    2. For **Runtime**, choose **Python 2.7**.
    3. For **Choose or create an execution role**, choose **Use an existing role**.
    4. For **Existing role**, enter `service-role/AWSDeepLensLambdaRole` .
4. After you create the function, go to the **Function code**
5. From the **Actions** drop-down menu in **Function code**, choose **Upload a .zip file**.
6. Upload the `aws-deeplens-pictionary-lambda.zip` file you downloaded earlier.
7. Choose **Save**.
8. From the **Actions** drop-down menu, choose **Publish new version**.
9. Enter a version number and choose **Publish**.



Publishing the function makes it available on the AWS DeepLens console so you can add it to your custom project.

Understanding the Lambda function

You should pay attention to the following two files:

- **labels.txt** – This file is for the inference function to translate the numerical result from the model into human readable labels used in our game. It contains a list of 36 objects on which the model has been trained to recognize sketches.
- **lambda_function.py** – This file contains the preprocessing algorithm and the function being called to generate predictions on drawings and send back results.

Because the model was trained on digital sketches with clean, white backgrounds, we have a preprocessing algorithm that helps isolate the drawing and remove any clutter in the background. You can find the algorithm to do this in the `isolate_image()` function inside the `lambda_function.py` file. In this section, we walk you through some important parts of the preprocessing algorithm.

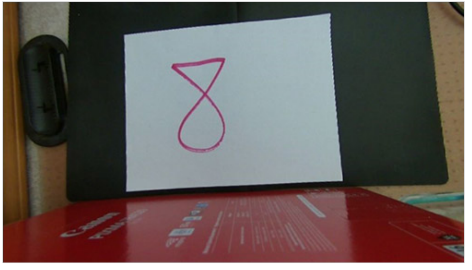
Fisheye calibration

AWS DeepLens uses a wide-angle lens to get as much information as possible in the frame. As a result, any input frame is distorted, especially for the rectangular shape of a whiteboard. Therefore, you need to perform fisheye calibration to straighten the edges. As part of this post, we provide the calibration code to undistort your AWS DeepLens images. The following code straightens the edges and eliminates the distortion:

```
def undistort(frame):
    frame_height, frame_width, _ = frame.shape
 K=np.array([[511.98828907136766, 0.0, 426.48016197546474],
                [0.0, 513.8644747557715, 236.89875770956868],
                [0.0, 0.0, 1.0]])
    D=np.array([[-0.10969105781526832], [0.03463562293251206],
                [-0.2341226037892333], [0.34335682066685935]])
    DIM = (int(frame_width/3), int(frame_height/3))
    frame_resize = cv2.resize(frame, DIM)
    map1, map2 = cv2.fisheye.initUndistortRectifyMap(K, D, np.eye(3), \
                                                 K, DIM, cv2.CV_16SC2)
    undistorted_img = cv2.remap(frame_resize, map1, map2, \
                                interpolation=cv2.INTER_LINEAR, \
                                borderMode=cv2.BORDER_CONSTANT)
    return undistorted_img
```

The following screenshots shows the raw image captured by AWS DeepLens.

The following screenshot shows the results of the undistort function with fisheye calibration.



The next code section enhances the images to eliminate the effects caused by different lighting conditions:

```
enh_con = ImageEnhance.Contrast(img_colored)
contrast = 5.01
img_contrasted = enh_con.enhance(contrast)
image = img_contrasted
image = np.array(image)
```

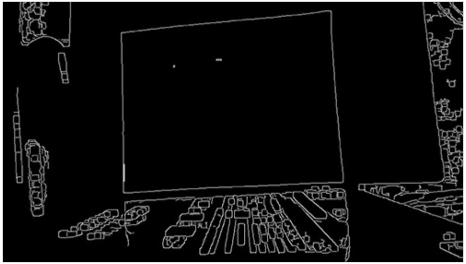
The following screenshot shows the results of the contrast enhancement.

Canny Edge Detection

The next part of the preprocessing algorithm uses OpenCV's Canny Edge Detection technique to find the edges in the image. See the following code:

```
# these constants are carefully picked
    MORPH = 9
    CANNY = 84
    HOUGH = 25
    img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    cv2.GaussianBlur(img, (3,3), 0, img)
    # this is to recognize white on white
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT,(MORPH,MORPH))
    dilated = cv2.dilate(img, kernel)
    edges = cv2.Canny(dilated, 0, CANNY, apertureSize=3)
    lines = cv2.HoughLinesP(edges, 1,  3.14/180, HOUGH)
    for line in lines[0]:
        cv2.line(edges, (line[0], line[1]), (line[2], line[3]), (255,0,0), 2, 8)
```

The following screenshot shows the results from applying the Canny Edge Detector.



For more information about how Canny Edge Detection works, see the Canny Edge Detection tutorial on the OpenCV website.

Finding contours

After the edges are found, you can use OpenCV's `findContours()` function to extract the polygon contours from the image. This function returns a list of polygon shapes that are closed and ignores any open edges or lines. See the following code:

```
contours, _ = cv2.findContours(edges.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
contours = filter(lambda cont: cv2.arcLength(cont, False) > 100, contours)
contours = filter(lambda cont: cv2.contourArea(cont) > 10000, contours)
 result = None
for idx, c in enumerate(contours):
      if len(c) < Config.min_contours:
          continue
      epsilon = Config.epsilon_start
      while True:
            approx = cv2.approxPolyDP(c, epsilon, True)
            approx = approx.reshape((len(approx), 2))
            new_approx = []
            for i in range(len(approx)):
                if 80 < approx[i][0] < 750:
                    new_approx.append(approx[i])
            approx = np.array(new_approx)
            if (len(approx) < 4):
                break
            if math.fabs(cv2.contourArea(approx)) > Config.min_area:
```

For more information, see Contours: Getting Started.

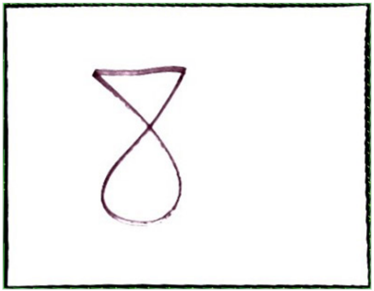
Perspective transformation

Finally, the preprocessing algorithm does perspective transformation to correct for any skew. The following code helps achieve perspective transformation and crop a rectangular area:

```
M = cv2.getPerspectiveTransform(src_rect, dst_rect)
warped = cv2.warpPerspective(image, M, (w, h))
```

The following image is the input of the preprocessing algorithm.

The following image is the final result.



Performing inference

In this section, you learn how to perform inference with an ML model and send back results from AWS DeepLens.

AWS DeepLens uses the Intel OpenVino model optimizer to optimize the ML model to run on DeepLens hardware. The following code optimizes a model to run locally:

```
error, model_path = mo.optimize(model_name, INPUT_WIDTH, INPUT_HEIGHT)
```

The following code loads the model:

```
model = awscam.Model(model_path, {'GPU': 1})
```

The following code helps run the model frame-per-frame over the images from the camera:

```
ret, frame = awscam.getLastFrame()
```

Viewing the text results in the cloud is a convenient way to make sure the model is working correctly. Each AWS DeepLens device has a dedicated `iot_topic` automatically created to receive the inference results. The following code sends the messages from AWS DeepLens to the IoT Core console:

```
# 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))
```

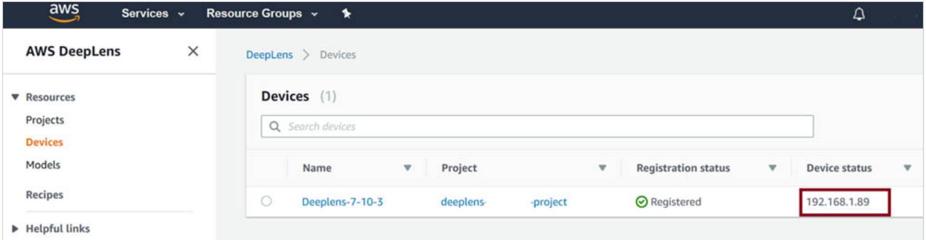
Deploying the model to AWS DeepLens

In this section, you set up your AWS DeepLens device, import a pre-trained model, and deploy the model to AWS DeepLens.

Setting up your AWS DeepLens device

You first need to register your AWS DeepLens device, if you haven't already.

After you register your device, you need to install the latest OpenCV (version 4.x) packages and Pillow libraries to enable the preprocessing algorithm in the DeepLens inference Lambda function. To do so, you need the IP address of AWS DeepLens on the local network, which is listed in the **Device details** section. You also need to ensure that Secure Shell (SSH) is enabled for your device. For more information about enabling SSH on your device, see View or Update Your AWS DeepLens 2019 Edition Device Settings.



Open a terminal application on your computer. SSH into your DeepLens by entering the following code into your terminal application:

```
ssh aws_cam@<YOUR_DEEPLENS_IP>
```

Then enter the following commands in the SSH terminal:

```
sudo su
pip install --upgrade pip
pip install opencv-python
pip install pillow
```

Importing the model to AWS DeepLens

For this post, you use a pre-trained model. We trained the model for 36 objects on The Quick Draw Dataset made available by Google, Inc., under the CC BY 4.0 license. For each object, we took 1,600 images for training and 400 images for testing the model from the dataset. Holding back 400 images for testing allows us to measure the accuracy of our model against images that it has never seen.

For instructions on training a model using Amazon SageMaker as the development environment, see AWS DeepLens Recipes and Amazon SageMaker: Build an Object Detection Model Using Images Labeled with Ground Truth.

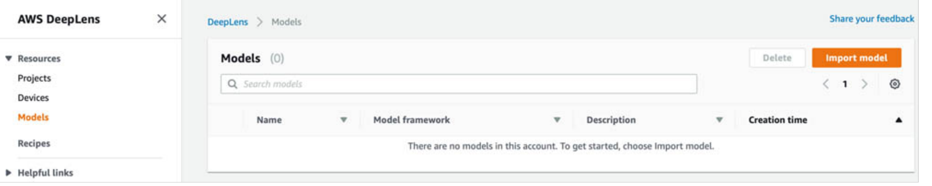
To import your model, complete the following steps:

1. Download the model aws-deeplens-pictionary-game.tar.gz.
2. Create an Amazon Simple Storage Service (Amazon S3) bucket to store this model. For instructions, see How do I create an S3 Bucket?

The S3 bucket name must contain the term `deeplens`. The AWS DeepLens default role has permission only to access the bucket with the name containing `deeplens`.

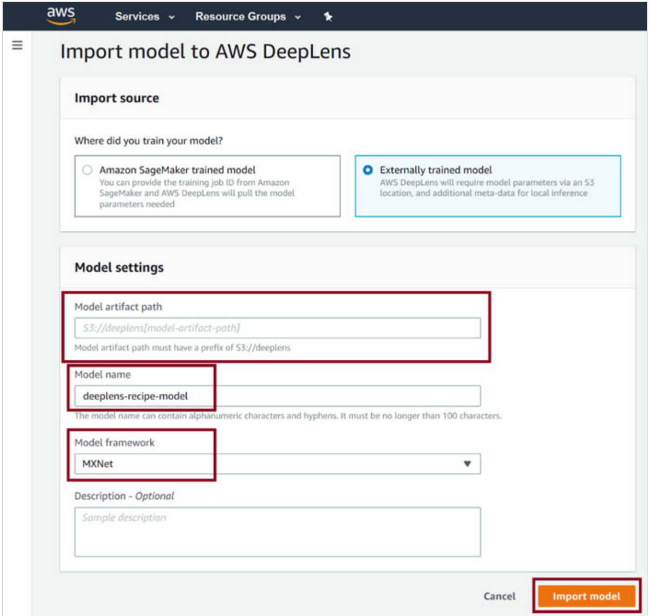
3. After the bucket is created, upload `aws-deeplens-pictionary-game.tar.gz` to the bucket and copy the model artifact path.
4. On the AWS DeepLens console, under **Resources**, choose **Models**.

5. Choose **Import model**.



6. On the **Import model to AWS DeepLens** page, choose **Externally trained model**.
7. For **Model artifact path**, enter the Amazon S3 location for the model you uploaded earlier.
8. For **Model name**, enter a name.
9. For **Model framework**, choose **MXNet**.

10. Choose **Import model**.

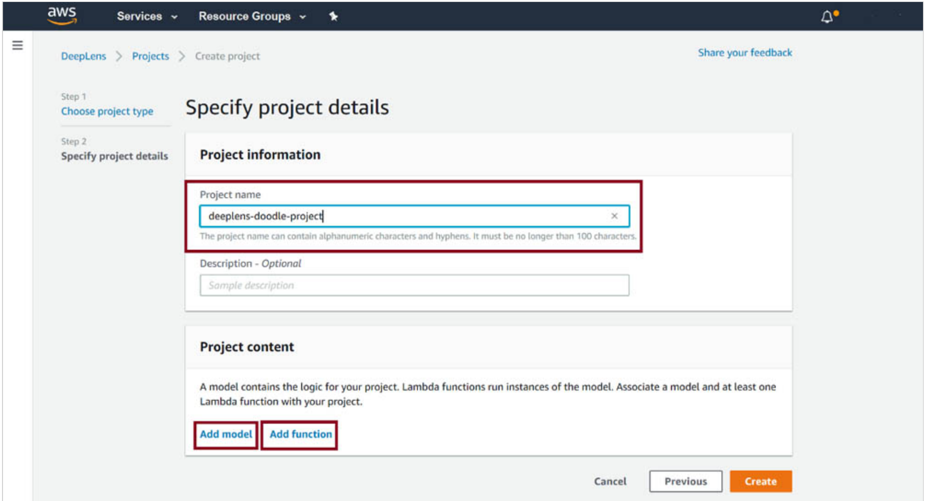


Deploying the model to your AWS DeepLens device

To deploy your model, complete the following steps:

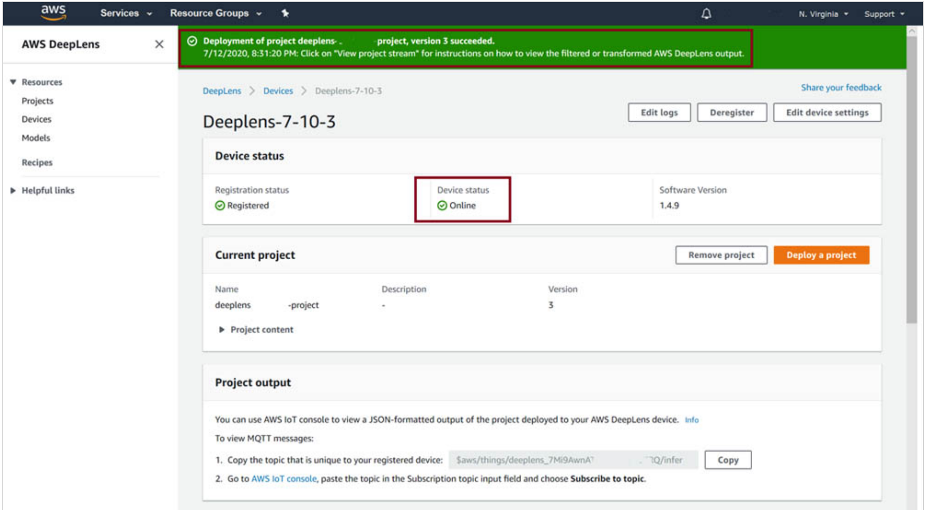
1. On the AWS DeepLens console, under **Resources**, choose **Projects**.
2. Choose **Create new project**.
3. Choose **Create a new blank project**.
4. For **Project name**, enter a name.
5. Choose **Add model** and choose the model you imported earlier.
6. Choose **Add function** and choose the Lambda function you created earlier.

7. Choose **Create**.



8. Select your newly created project and choose **Deploy to device**.
9. On the **Target device** page, select your device from the list.
10. On the **Review and deploy** page, choose **Deploy**.

The deployment can take up to 5 minutes to complete, depending on the speed of the network your AWS DeepLens is connected to. When the deployment is complete, you should see a green banner message that the deployment succeeded.



To verify that the project was deployed successfully, you can check the text prediction results sent to the cloud via AWS IoT Greengrass. For instructions, see Using the AWS IoT Greengrass Console to View the Output of Your Custom Trained Model (Text Output).

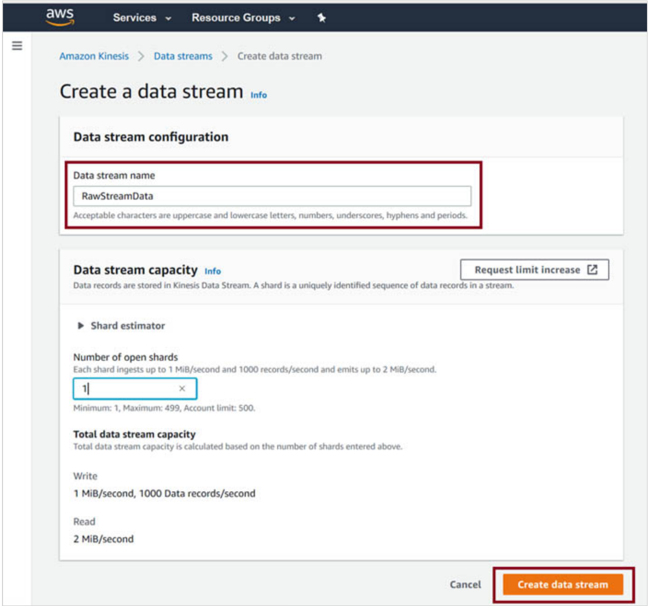
In addition to the text results, you can view the pose detection results overlaid on top of your AWS DeepLens live video stream. For instructions, see Viewing AWS DeepLens Output Streams.

Sending results from AWS DeepLens to a data stream

In this section, you learn how to send messages from AWS DeepLens to a Kinesis data stream by configuring an AWS IoT rule.

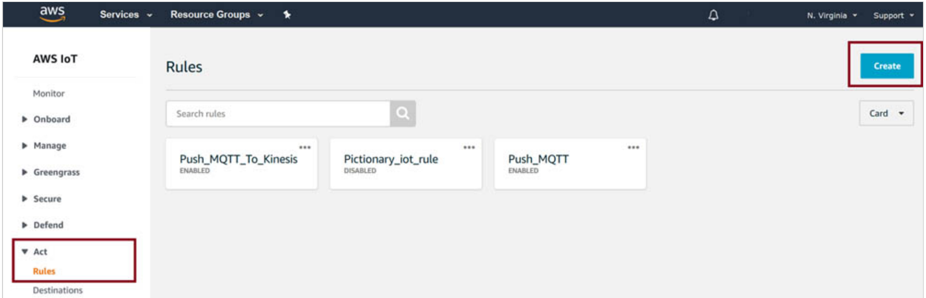
1. On the Kinesis console, create a new data stream.
2. For **Data stream name**, enter a name.
3. For **Number of shards**, choose 1.

4. Choose **Create data stream**.



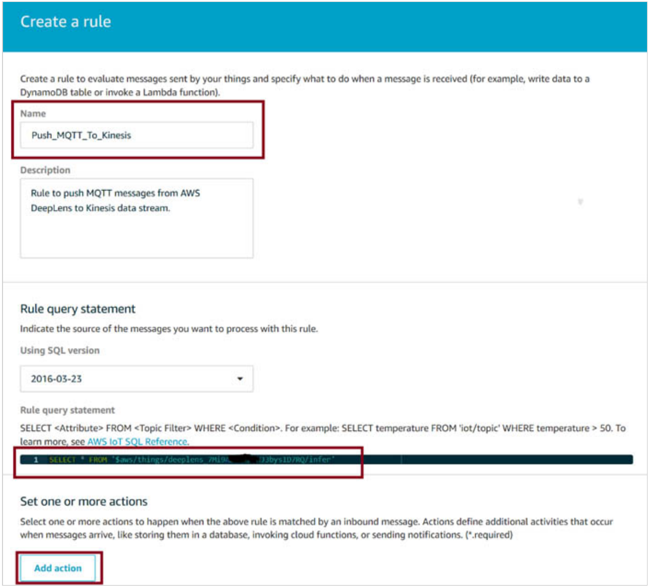
5. On the AWS IoT console, under **Act**, choose **Rules**.

6. Choose **Create** to set up a rule to push MQTT messages from AWS DeepLens to the newly created data stream.



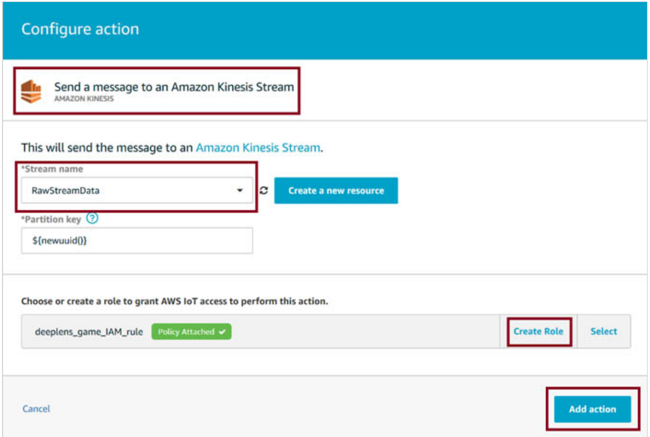
7. On the **Create a rule** page, enter a name for your rule.
8. For **Rule query statement**, enter the DeepLens device MQTT topic.

9. Choose **Add action**.



10. Choose **Send a message to an Amazon Kinesis Stream**.
11. Choose **Configuration**.
12. Choose the data stream you created earlier.
13. For **Partition key**, enter `${newuuid()}`.
14. Choose **Create a new role** or **Update role**.
15. Choose **Add action**.

16. Choose **Create rule** to finish the setup.



Now that the rule is set up, MQTT messages are loaded into the data stream.

Amazon Kinesis Data Streams is not currently available in the AWS Free Tier, which offers a free trial for a group of AWS services. For more information about the pricing of Amazon Kinesis Data Streams, see link.

We recommend that you delete the data stream after completing the tutorial because charges occur on an active data stream even when you aren't sending and receiving messages.

Creating an Alexa skill

In this section, you first create a Lambda function that queries a data stream and returns the sketches detected by AWS DeepLens to Alexa. Then, you create a custom Alexa skill to start playing the game.

Creating a custom skill with Lambda

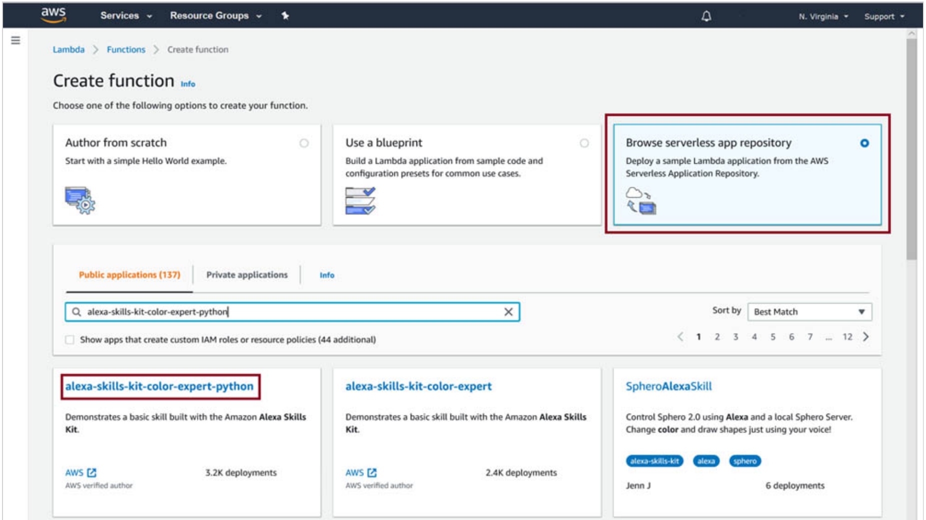
To create your custom skill in Lambda, complete the following steps:

1. On the Lambda console, create a new function.

The easiest way to create an Alexa skill is to create the function from the existing blueprints or serverless app repository provided by Lambda and overwrite the code with your own.

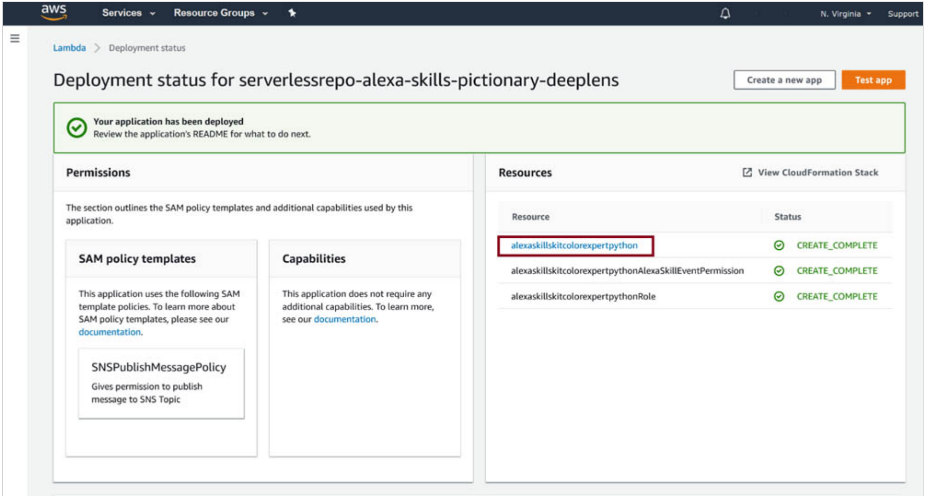
2. For **Create function**, choose **Browse serverless app repository**.

3. For **Public repositories**, search for and choose **alexa-skills-kit-color-expert-python**.



4. Under **Application settings**, enter an application name and `TopicNameParameter`.
5. Choose **Deploy**.

6. When the application has been deployed, open the Python file.



7. Download the alexa-lambda-function.py file onto your computer.
8. Copy the Python code from the file and replace the sample code in the `lambda_function.py` file in the **Function code** section.

This function includes the entire game logic, reads data from the data stream, and returns the result to Alexa. Be sure to change the Region from the default (`us-east-1`) if you're in a different Region. See the following code:

```
kinesis = boto3.client('kinesis', region_name='us-east-1'
```

9. Set the **Timeout** value to 20 seconds.

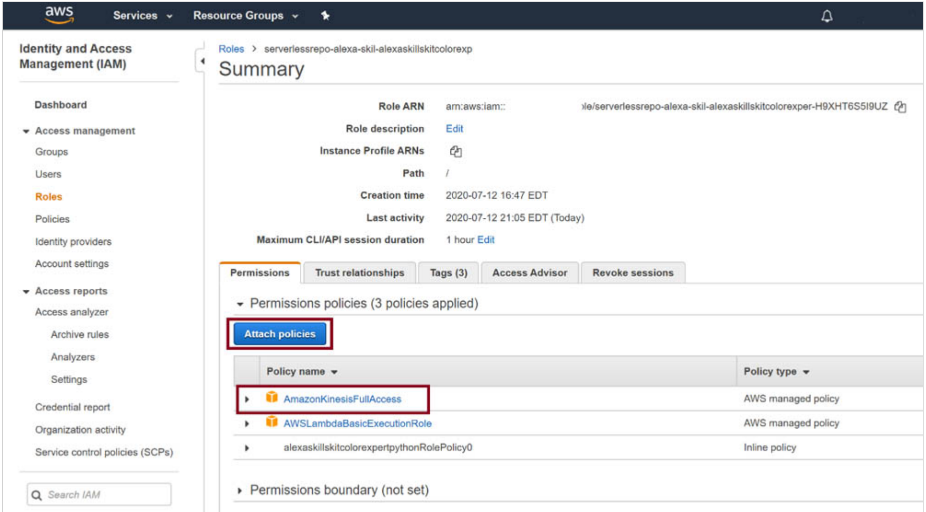
You now need to give your Lambda function IAM permissions to read data from the data stream.

10. In your Lambda function editor, choose **Permissions**.
11. Choose the **Role name** under the **Execution role.**

You're directed to the IAM role editor.

12. In the editor, choose **Attach policies**.
13. Enter `Kinesis` and choose **AmazonKinesisFullAccess**.

14. Choose **Attach policy**.

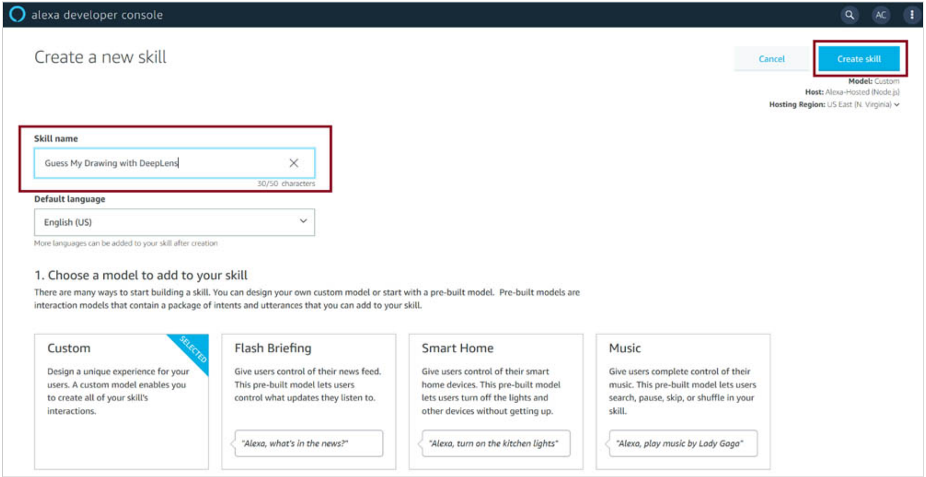


Creating a custom skill to play the game

To create your second custom skill to start playing the game, complete the following steps:

1. Log in to the Alexa Developer Console.
2. Create a new custom Alexa skill.
3. On the Create a new skill page, for **Skill name**, enter a skill name
4. For **Choose a model to add to your skill**, choose **Custom**.
5. For **Choose a method to host your skill's backend resources,** choose **Provision your own.**

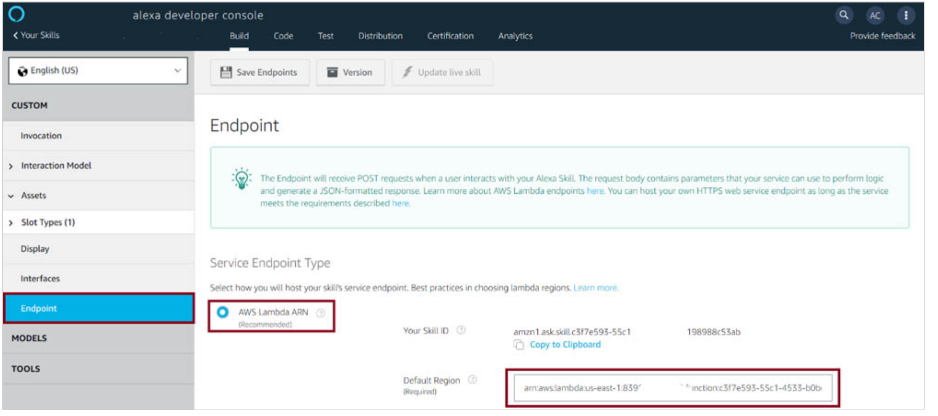
6. Choose **Create skill**.



7. On the next page, choose the default template to add your skill.
8. Choose **Continue with template**.

After about 1–2 minutes, your skill appears on the console.

9. In the **Endpoint** section, enter the Amazon Resource Name (ARN) of the Lambda function created for the Alexa skill in the previous step.

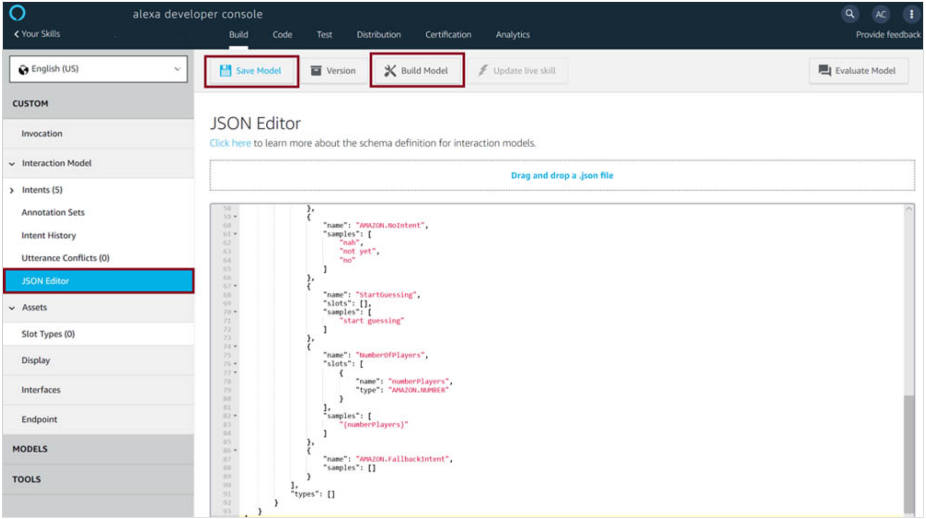


10. Download [alexa-skill-json-code.txt]() onto your computer.
11. Copy the code from the file and paste in the Alexa skill JSON editor to automatically configure intents and sample utterances for the custom skill.

In the Alexa architecture, intents can be thought of as distinct functions that a skill can perform. Intents can take arguments that are known here as *slots*.

12. Choose **Save Model** to apply the changes.

13. Choose **Build Model**.

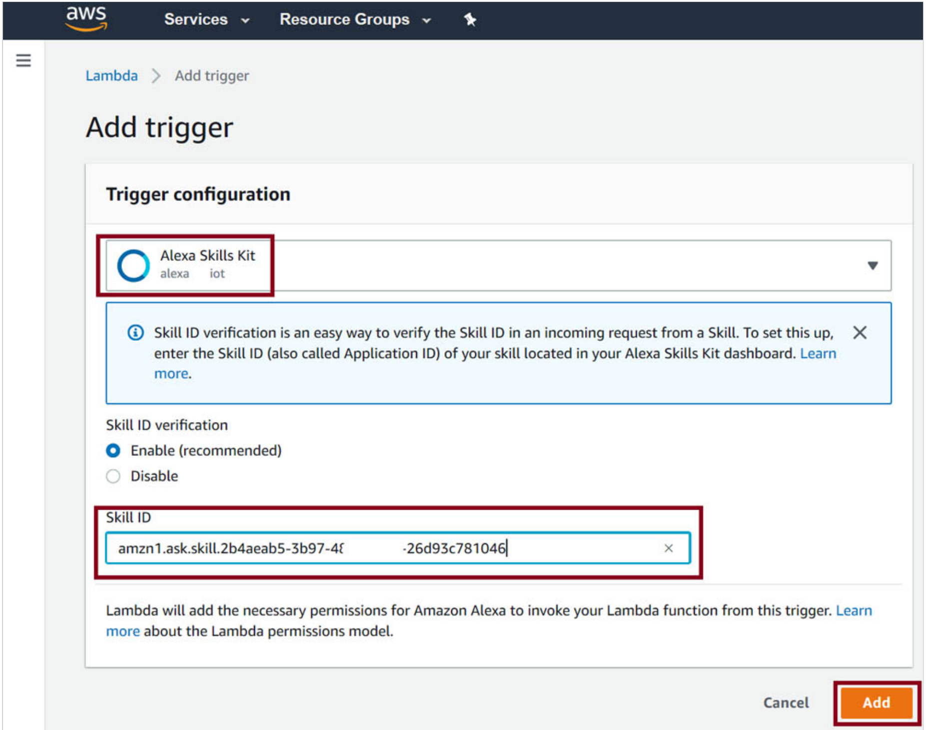


14. On the Lambda console, open the Lambda function for the Alexa skill you created earlier.

You need to enable the skill by adding a trigger to the Lambda function.

15. Choose **Add trigger**.
16. Choose **Alexa Skills Kit**.
17. For **Skill ID**, enter the ID for the Alexa skill you created.

18. Choose **Add**.



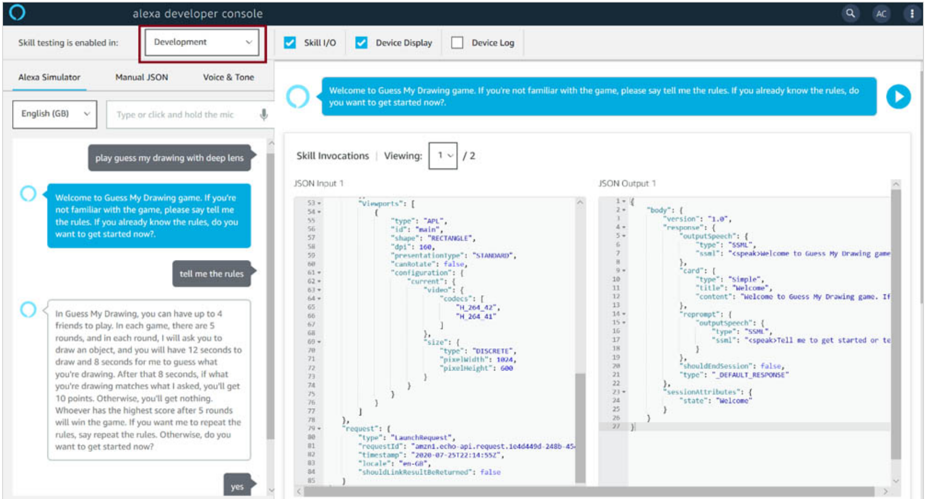
Testing the skill

Your Alexa skill is now ready to tell you the drawings detected by AWS DeepLens. To test with an Alexa-enabled device (such as an Amazon Echo), register the device with the same email address you used to sign up for your developer account on the Amazon Developer Portal. You can invoke your skill with the wake word and your invocation name: “Alexa, Play Guess My Drawing with DeepLens.”

The language in your Alexa companion app should match with the language chosen in your developer account. Alexa considers English US and English UK to be separate languages.

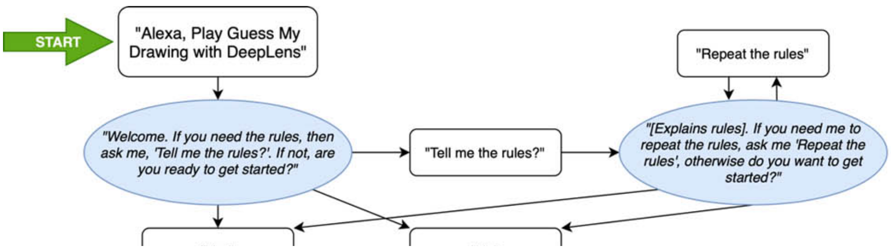
Alternatively, the **Test** page includes a simulator that lets you test your skill without a device. For **Skill testing is enabled in**, choose **Development**. You can test your skill with the phrase, “Alexa, Play Guess My Drawing with DeepLens.”

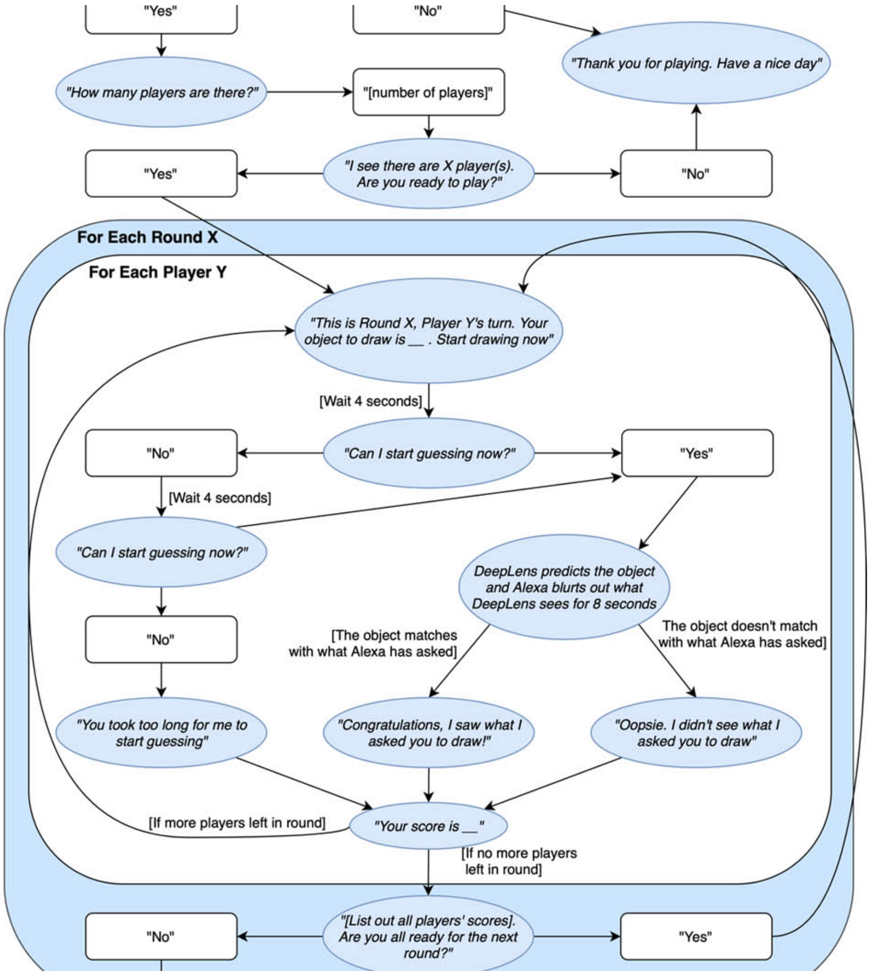
Windows 10 users can download the free Alexa app from the Microsoft Store and interact with it from their PC.



For more information on testing your Alexa skill, see Test Your Skill. For information on viewing the logs, check Amazon CloudWatch logs for AWS Lambda.

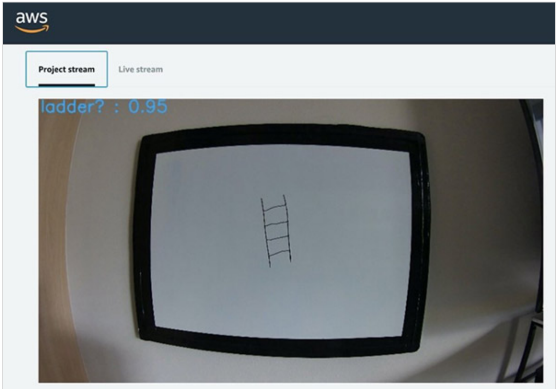
The following diagram shows the user interaction flow of our game.





"Yes"
"No"
"Thank you for playing. Have a nice day"
"How many players are there?"
"[number of players]"
"Yes"
"I see there are X player(s). Are you ready to play?"
"No"
For Each Round X
For Each Player Y
"This is Round X, Player Y's turn. Your object to draw is __ . Start drawing now"
[Wait 4 seconds]
"No"
"Can I start guessing now?"
"Yes"
[Wait 4 seconds]
"Can I start guessing now?"
DeepLens predicts the object and Alexa blurts out what DeepLens sees for 8 seconds
"No"
[The object matches with what Alexa has asked]
The object doesn't match with what Alexa has asked]
"You took too long for me to start guessing"
"Congratulations, I saw what I asked you to draw!"
"Oopsie. I didn't see what I asked you to draw"
[If more players left in round]
"Your score is __"
[If no more players left in round]
"No"
"[List out all players' scores]. Are you all ready for the next round?"
"Yes"

The following images show the prediction outputs of our model with the name of an object and its probability. You need to have your AWS DeepLens located in front of a rectangular-shaped whiteboard or a piece of white paper to ensure that the edges are visible in the frame.



Conclusion

In this post, you learned about the preprocessing algorithm to isolate a drawing area and how to deploy a pre-trained model onto AWS DeepLens to recognize sketches. Next, you learned how to send results from AWS IoT to Kinesis Data Streams. Finally, you learned how to create a custom Alexa skill with Lambda to retrieve the detected objects in the data stream and return the results to players via Alexa.

For other tutorials, samples, and project ideas with AWS DeepLens, see AWS DeepLens Recipes.

About the Authors

Amit Choudhary is a Senior Product Manager Technical Intern. He loves to build products that help developers learn about various machine learning techniques in a fun and hands-on manner.

Phu Nguyen is a Product Manager for AWS DeepLens. He builds products that give developers of any skill level an easy, hands-on introduction to machine learning.

Brian Nguyen is an undergraduate senior in majoring Electrical Engineering with a concentration of Digital Signal & Image Processing at University of Washington Seattle.

Hanwen Guo is an undergraduate senior majoring in Electrical Engineering with a concentration of Digital Signal & Image Processing at University of Washington Seattle.

Jack Ma is an undergraduate senior majoring in Electrical Engineering with a concentration of Embedded Systems at University of Washington Seattle.

Sairam Tabibu is pursuing a master's degree in Electrical Engineering with an interest in machine learning/deep learning at University of Washington Seattle

Aaron Liang is pursuing a master's degree in Electrical Engineering with an interest in software engineering at University of Washington Seattle

Resources

Getting Started
What's New
Top Posts
Official AWS Podcast
AWS Case Studies

Follow

Twitter
Facebook
LinkedIn
Twitch
RSS Feed
Email Updates

Related Posts

Announcing the express testing capability in Amazon Lex

Sales forecasting in SAP with Amazon forecast

Learn from the first place winner of the first AWS DeepComposer Chartbusters Bach to the Future Challenge

Learn why AWS is the best cloud to run Microsoft Windows Server and SQL Server workloads

How TalkingData uses AWS open source Deep Java Library with Apache Spark for machine learning inference at scale

How artificial intelligence and Amazon Alexa are teaching students to write

Explore new machine learning insights on episode 6 of Twitch AWS Power Hour

Launching the AI and ML Rapid Adoption Assistance Initiative for AWS Public Sector Partners