



Proof of concept for a Scalable Image Classifier with a Retraining Pipeline on Amazon Web Services (AWS)

Module Assignment for

CS5024 - Theory and Practice of Advanced AI Ecosystems

Student Name : **Siddharth Prince**

Student ID : **23052058**

Revision Timestamp: 03/05/2024 23:22:02

1. Abstract

The main idea for this assignment is to create an ML ecosystem architecture where a deployed model gets retrained over time with new human labelled data collected from users. The motivation for this was the topic of my master's dissertation where I will be attempting to train a Deep Learning network to score breast cancer pathology slide images for the intensity of staining corresponding to the intensity of HER2 gene expression. When it comes to such medical use cases, we lean towards deep learning/machine vision tools to aid doctors and pathologists and speed up the process. We would not however just rely on the machine's prediction fully. Hence it is natural to expect that we can keep getting an influx of human labelled data. So, the idea is to not only deploy an ML model using SageMaker but to also demonstrate that the AWS ecosystem with all its different services and integrations can be leveraged to make the best of the above specified use case to retrain the model so that it can automatically improve with time and more data.

Contents

1. Abstract	2
2. Introduction	3
3. AI Ecosystem Architecture Used	3
3.1. SageMaker	4
3.2. Client-side Application	4
3.3. API Gateway + Lambdas	6
3.3.1. Lambda Function: sagemakerEndpointInvoker	9
3.3.2. Lambda function: retrainPipeline	9
4. Model Description	10
5. Scalability Considerations	11
6. References	11
Figure 1: Architecture of the proof of concept auto retraining ML ecosystem	3
Figure 2: Code snippet of classifyImage POST API endpoint being called to send the image to sagemakerEndpointInvoker Lambda function	5
Figure 3: Webapp UI	5
Figure 4: Successful deployment of webapp via AWS Amplify	6
Figure 5: API Gateway endpoints for the getPrediction API deployed to the 'dev' stage	7
Figure 6: /classifyImage POST request triggers sagemakerEndpointInvoker Lambda function	7
Figure 7: /humanLabel POST request triggers retrainingPipeline Lambda function	8
Figure 8: CORS error when trying to hit the POST API via the webapp	8
Figure 9: sagemakerEndpointInvoker Lambda function's code	9
Figure 10: Prediction results of deployed pretrained Resnet18 model	10

2. Introduction

It is exceedingly difficult to get access to a lot of quality medical data especially for data that is specific to specialised machine learning/deep learning models because of a couple of factors. One is the inherent privacy concerns associated with it (as it should). The other is just the lack of availability of data (usually images) for specific diagnoses like the diverse types of cancer for instance. Having a steady flow of new data for specialised models to improve over time can help with this provided the proposed system (product) is also built in a way that it is compliant with all necessary privacy and ethics regulations. Also, to note for such medical use cases is that they need to have exceedingly high accuracy scores to be reliable especially if they are to be used to inform decisions that can save lives. False positives/negatives become a real issue for such cases. The basic idea for the implementation is to have an API gateway instance that connects to Lambda functions which trigger actions like getting the model prediction, saving the newly user labelled/verified data to an S3 bucket and starting the training job when set thresholds are hit.

3. AI Ecosystem Architecture Used

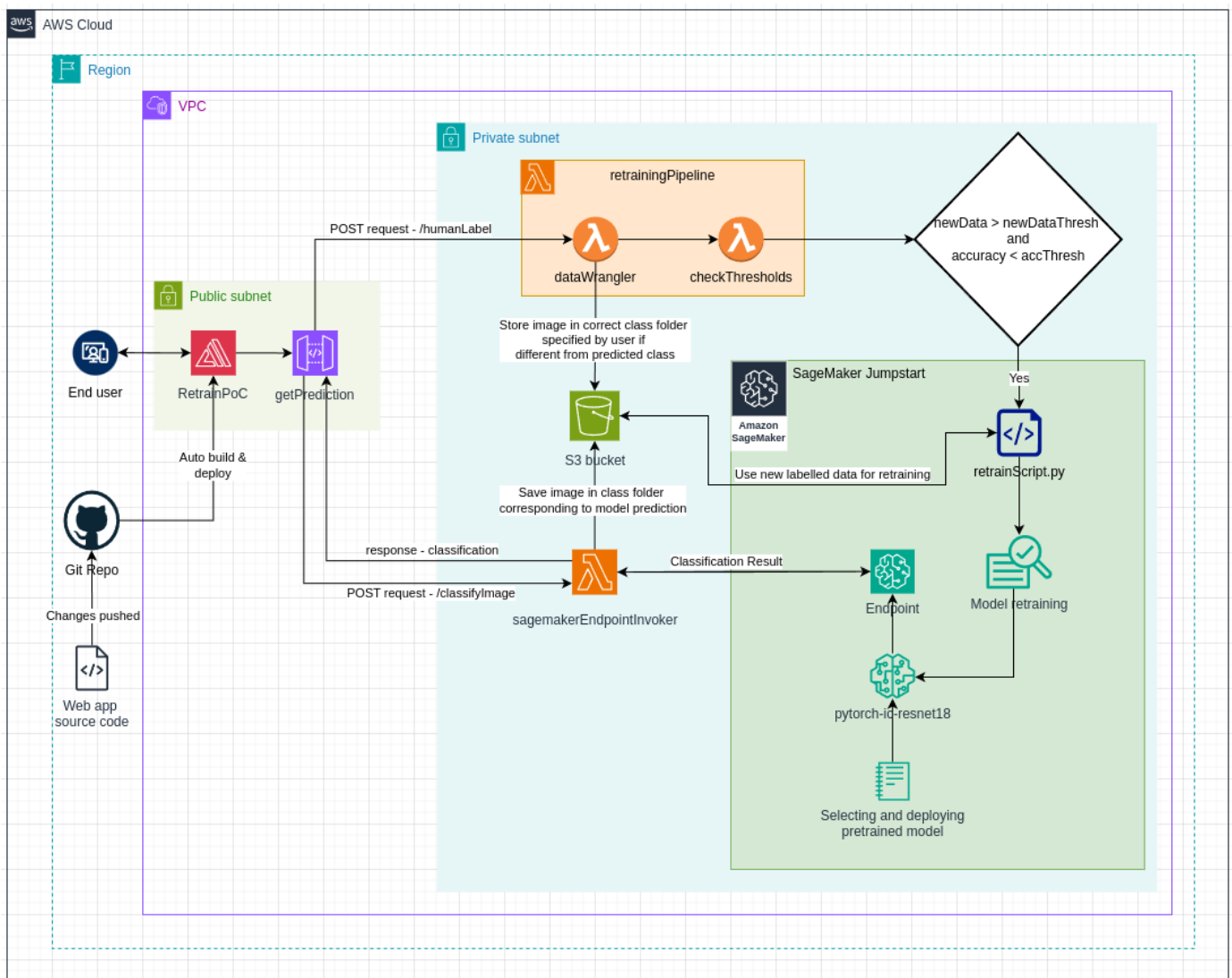


Figure 1: Architecture of the proof of concept auto retraining ML ecosystem

Proof of concept for a Scalable Image Classifier with a Retraining Pipeline on Amazon Web Services (AWS)

The architecture diagram for the proposed proof of concept, 1 can be broken down into three main working parts or modules. They are,

1. SageMaker
2. Client-side application
3. API Gateway + Lambdas

3.1. SageMaker

This encompasses all the machine learning aspects of the architecture. How this is implemented can vary based on what the requirement is for the application. Usually, one would want to train their own model from scratch with a custom data set considering the target use case of digital pathology. However, for demonstration purposes, the approach used here was to take advantage of the SageMaker Jumpstart (Amazon AWS (a)) models which are pre-trained and can be easily deployed for many general classification purposes depending on the pretrained model of choice. The code to retrieve the pretrained models and deploy to a SageMaker instance was adapted from a SageMaker Jumpstart notebook from the AWS GitHub repository (Amazon AWS, 2024). As the user is expected to send in one image at a time for classification (Patrick Denny (a), 2024), a SageMaker Endpoint for the pretrained Jumpstart model was setup to provide the classification. The pretrained model I chose for this project was Resnet18 (Microsoft Research - Kaiming He, et. al, 2015). The reason being that it is a fairly small model without need for much compute and fits the requirement of the PoC which is to just have any capable image classifier.

The other half of the SageMaker puzzle is the retraining part. Code from the same AWS GitHub Jumpstart example (Amazon AWS, 2024) was clear enough and was used for running a training job on the model with new images and corresponding labels. However, an issue I ran into here was with having a way to redeploy these models into SageMaker Endpoints such that the same Endpoint instance is updated instead of a new one being spun up. This is an issue because the AWS Lambda functions whose implementation has been discussed in 3.3. API Gateway + Lambdas needs the SageMaker Endpoint name for it to integrate and pass on the data to it. If it keeps changing every time retraining is done, this won't work and is something I'm yet to solve.

3.2. Client-side Application

For the client-side application, I created a single page static web page which takes in an image and passes this to the AWS Lambda service, "sagemakerEndpointInvoker" via a POST API request whose endpoint is exposed via an AWS API Gateway instance, "getPrediction". The UI is a simple form as seen in Figure 3. Once the image is submitted the file is sent via the getPrediction's /classifyImage endpoint as a POST request as shown in the code snippet from Figure 2.

Proof of concept for a Scalable Image Classifier with a Retraining Pipeline on Amazon Web Services (AWS)

```
console.log('Image input:', imageInput.files[0]);
formData.append('image', imageInput.files[0]);
console.log('Form data:', formData);

// Make API POST request to AWS Lambda function
const lambdaResponse = await fetch('https://t8pqzch4ol.execute-api.us-east-1.amazonaws.com/dev/classifyImage', {
  method: 'POST',
  headers: {
    'Access-Control-Allow-Origin': '*'
  },
  body: formData
});
```

Figure 2: Code snippet of classifyImage POST API endpoint being called to send the image to sagemakerEndpointInvoker Lambda function

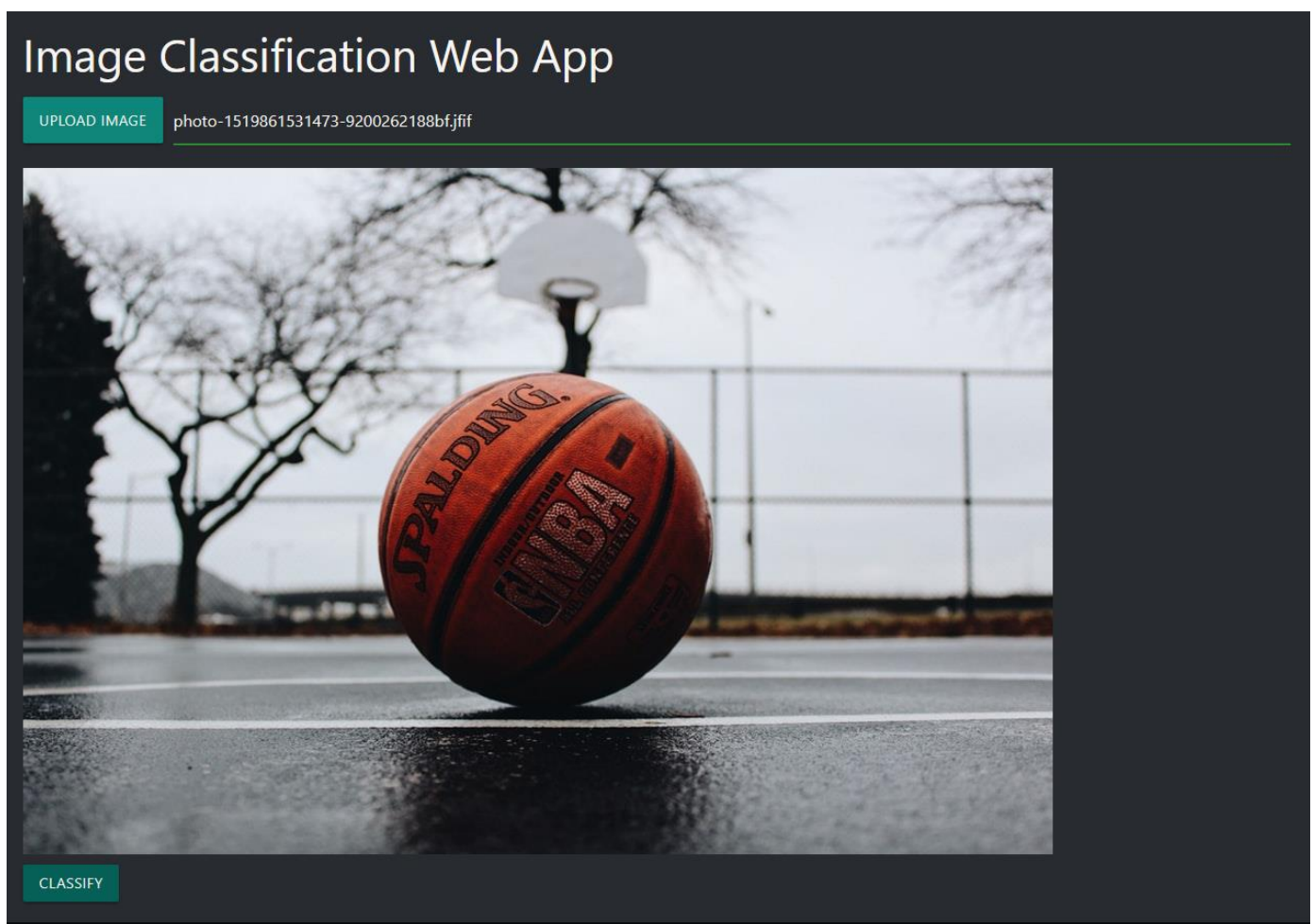


Figure 3: Webapp UI

The response is the top 5 predicted class labels from the deployed SageMaker Endpoint of which the first one i.e., the one with most confidence is displayed as the predicted class which then exposes a feedback form with two radio buttons. If the user can select either “correct” or “incorrect” based on their judgement. If they select “incorrect”, an additional drop-down menu will appear displaying a list of all the top 5 predicted classification labels and a “custom” option. The user can choose one of the other classes or enter their own custom label in an input box if they choose “custom”. The “Submit Feedback” button triggers the second API POST request made via the exposed endpoint,

“humanLabel”. The JavaScript code is available in the submitted index.html source file under the webapp/ directory.

In terms of the AWS ecosystem and deploying this webapp, I used AWS Amplify (Amazon AWS (b)) to quickly host the single page webapp. YouTube videos and tutorials were followed to learn how to connect this with the ecosystem at large via the use of API calls (Tutorials, 2022). VCS integration and auto deployment was also easy to set up and I did the same with my code being pushed to a new repository (Prince, 2024) I’d created in my GitHub account for this project. Figure 4 shows the successful deployment of the webapp after the latest commit push to the git repository.

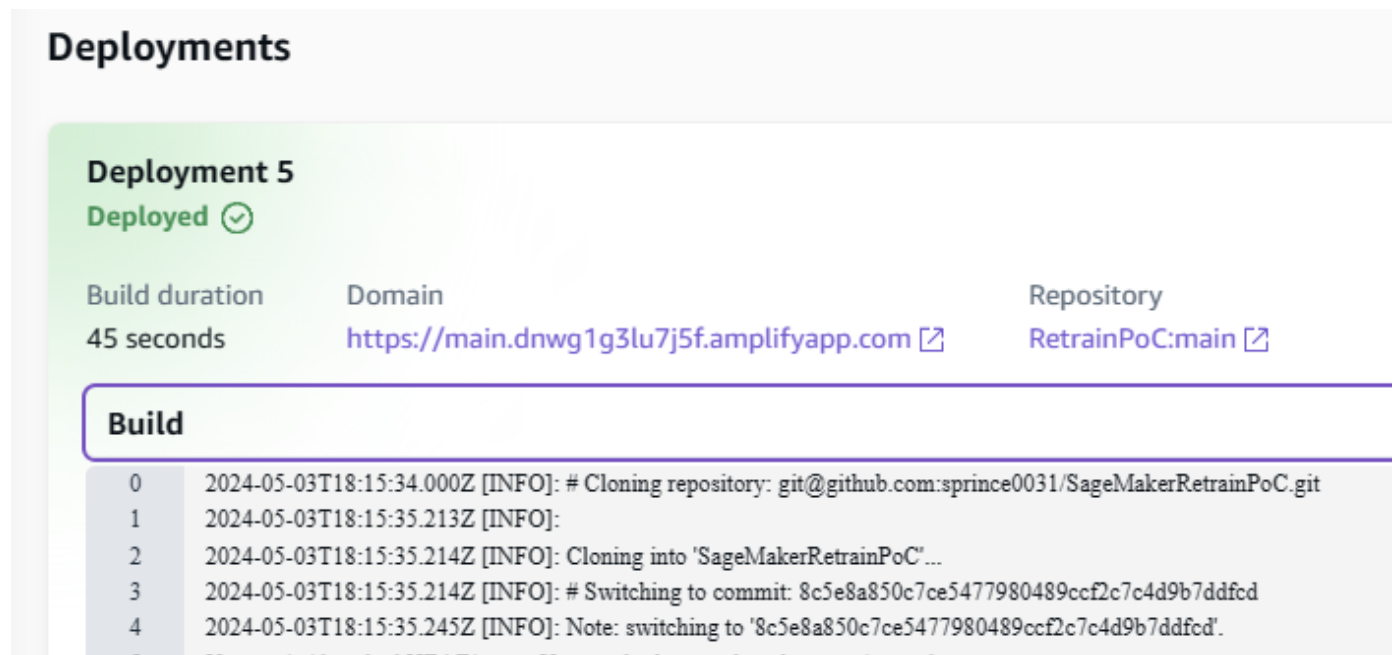


Figure 4: Successful deployment of webapp via AWS Amplify

3.3. API Gateway + Lambdas

Lambda functions really make a lot of the tasks simple without having to think much about provisioning dedicated compute in terms of EC2 instances (serverless compute - (Patrick Denny (c), 2024)) which made it perfect for this architecture and use case. They are the special sauce that in theory should tie all this together to create a great AI ecosystem where it automatically retrains to improve itself with time. The 2 Lambda functions and the API Gateway instance used for this project were introduced in 3.2. Client-side Application. These will be elaborated on below.

The API Gateway has two endpoints exposed which are /classifyImage and /humanLabel as can be seen in Figure 5.

Proof of concept for a Scalable Image Classifier with a Retraining Pipeline on Amazon Web Services (AWS)

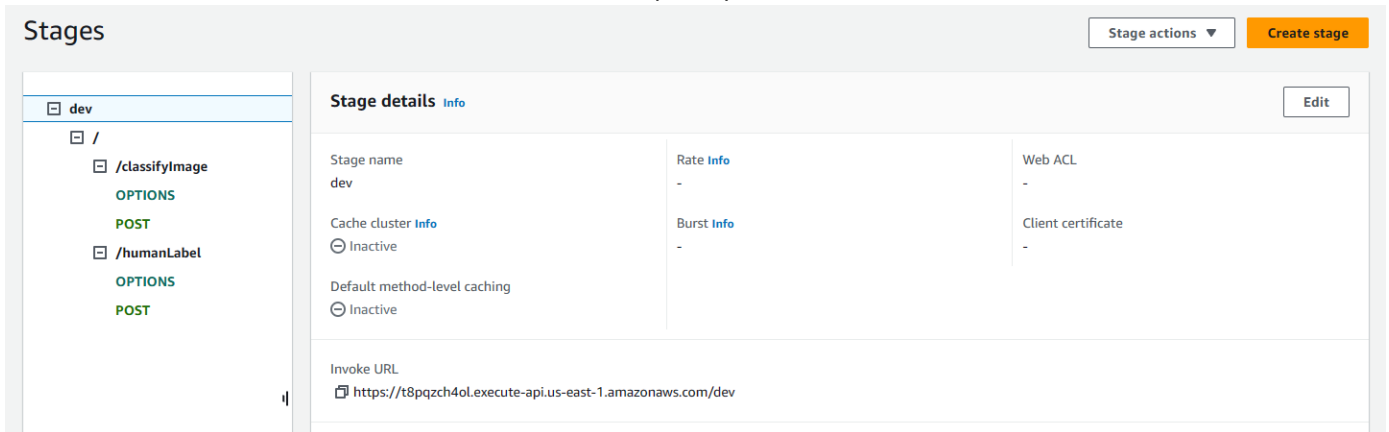


Figure 5: API Gateway endpoints for the `getPrediction` API deployed to the 'dev' stage

The `/classifyImage` and `/humanLabel` POST resources are connected to the `sagemakerEndpointInvoker` and `retrainPipeline` Lambda functions respectively as can be seen from Figure 6 and Figure 7.

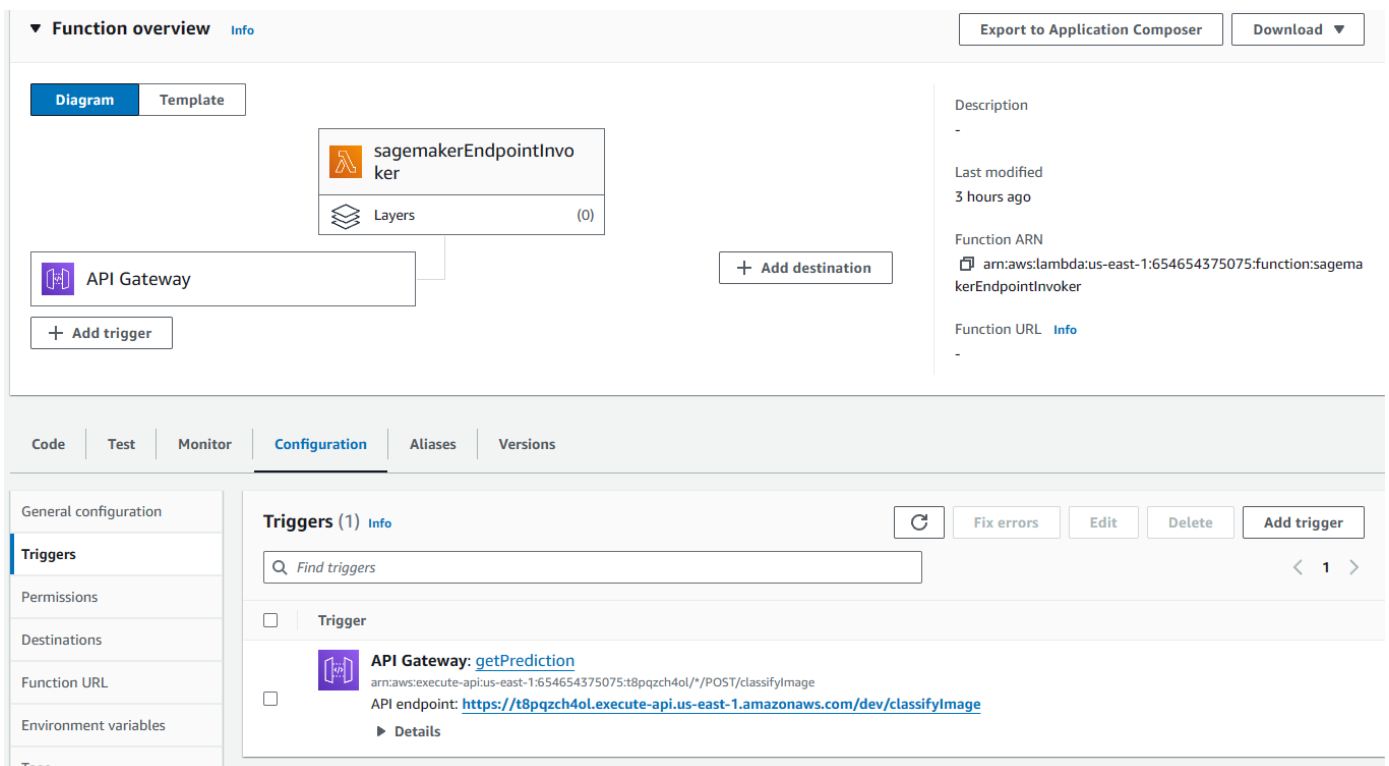


Figure 6: `/classifyImage` POST request triggers `sagemakerEndpointInvoker` Lambda function

Proof of concept for a Scalable Image Classifier with a Retraining Pipeline on Amazon Web Services (AWS)

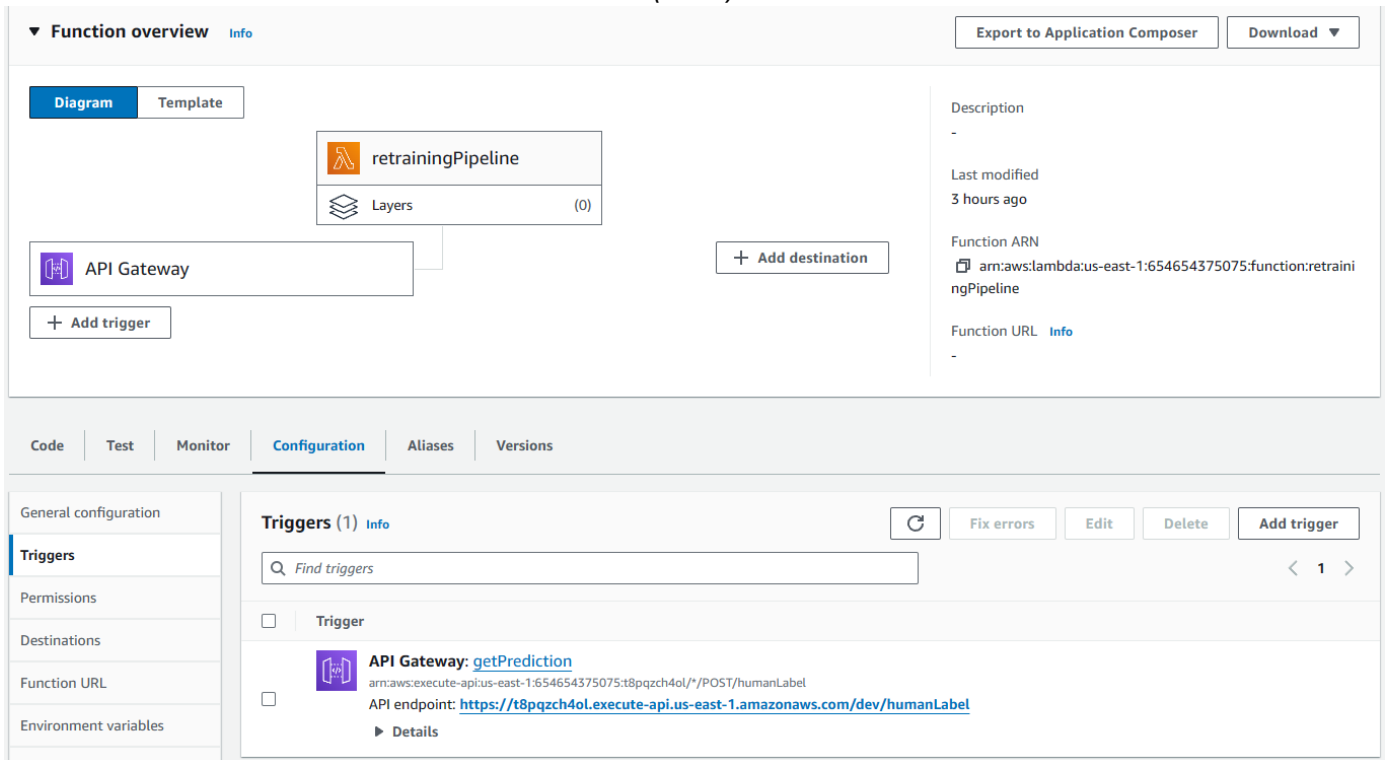


Figure 7: /humanLabel POST request triggers retrainingPipeline Lambda function

The required access policies to allow the Lambdas to access SageMaker resources were attached. However, despite having CORS enabled in the API Gateway configuration for both API endpoints, I could not overcome the issue of the CORS error when trying to hit the APIs from my webapp. Figure 8 shows the error from the browser console. This breaks the application in its current form at the time of submission.

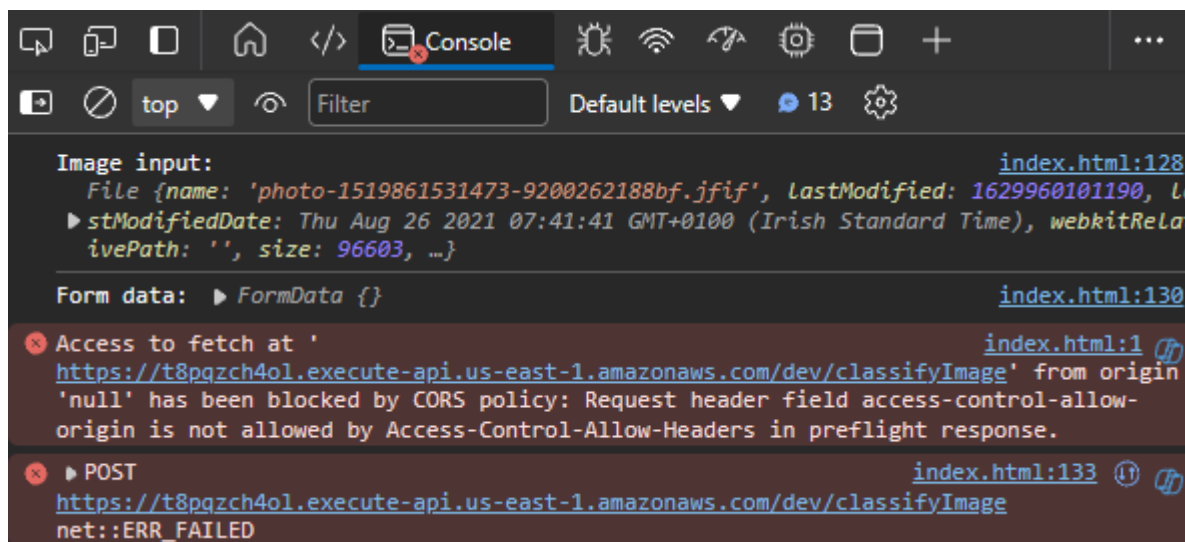
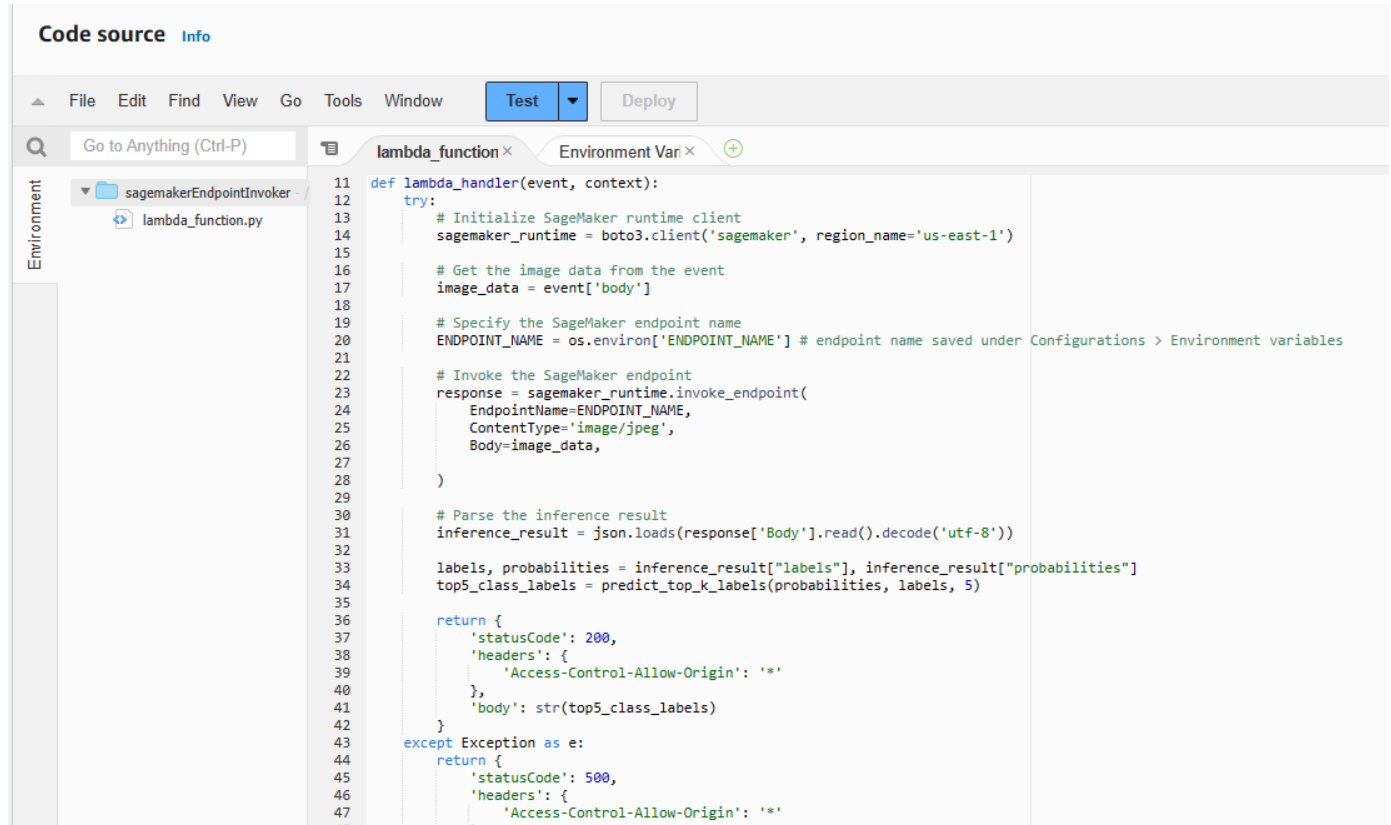


Figure 8: CORS error when trying to hit the POST API via the webapp

Coming to the functionality of the Lambda functions, we'll dive deeper into what each of them aim to achieve in the below sections.

3.3.1. Lambda Function: sagemakerEndpointInvoker

The sole function for this is to pass the image input from the client side, get the predicted classification from the SageMaker Endpoint and pass it back to the client as the response to the /classifyImage POST request. This is shown in the code snippet in Figure 9.



```
11 def lambda_handler(event, context):
12     try:
13         # Initialize SageMaker runtime client
14         sagemaker_runtime = boto3.client('sagemaker', region_name='us-east-1')
15
16         # Get the image data from the event
17         image_data = event['body']
18
19         # Specify the SageMaker endpoint name
20         ENDPOINT_NAME = os.environ['ENDPOINT_NAME'] # endpoint name saved under Configurations > Environment variables
21
22         # Invoke the SageMaker endpoint
23         response = sagemaker_runtime.invoke_endpoint(
24             EndpointName=ENDPOINT_NAME,
25             ContentType='image/jpeg',
26             Body=image_data,
27         )
28
29         # Parse the inference result
30         inference_result = json.loads(response['Body'].read().decode('utf-8'))
31
32         labels, probabilities = inference_result["labels"], inference_result["probabilities"]
33         top5_class_labels = predict_top_k_labels(probabilities, labels, 5)
34
35         return {
36             'statusCode': 200,
37             'headers': {
38                 'Access-Control-Allow-Origin': '*'
39             },
40             'body': str(top5_class_labels)
41         }
42     except Exception as e:
43         return {
44             'statusCode': 500,
45             'headers': {
46                 'Access-Control-Allow-Origin': '*'
47             },
48             'body': str(e)
49         }
```

Figure 9: sagemakerEndpointInvoker Lambda function's code

Ideally, this code should also include functionality to save the image file to the S3 bucket that was automatically provisioned when the SageMaker policies were created for the first time. How it would be saved would be in a folder with the predicted class name in this step. This is also illustrated in the architecture diagram (Figure 1) where there is a flow to the S3 bucket from this Lambda function.

3.3.2. Lambda function: retrainPipeline

This Lambda function has two distinct parts to play. One is that it takes in the human feedback after the classification result was passed and determines if the human feedback agrees with the original classification or if there is a mismatch. Based on this information it will either leave the image stored in the S3 bucket under the same labelled folder or it will organise it accordingly if the human label is different. It also updates a metadata JSON file in the S3 bucket which holds data such as number of new data points (latest count which keeps getting incremented by one for every new request cycle), total mismatches (to calculate the current accuracy score) and a record of how many mismatches for each class observed (the intention is mostly for manual data analysis later). The next part of the function is to check for two threshold values. The first is a minimum number of new data points that need to be available to consider a retrain cycle. The second is a minimum accuracy threshold the model should uphold. This is also illustrated in the architecture diagram within the decision box

Proof of concept for a Scalable Image Classifier with a Retraining Pipeline on Amazon Web Services (AWS)

(Figure 1). When the conditions are satisfied i.e., the model's accuracy score has fallen below its minimum acceptable accuracy and the number of new data points is higher than the set batch count, the retrainPipeline Lambda function invokes the retrain logic for the model whose code would be similar to the example code from section 4 in the AWS Jumpstart notebook reference (Amazon AWS, 2024). I have also not been able to implement this at the time of submission because I ran into the issue of being able to retrain the model that is currently deployed instead of deleting the instance and pulling a fresh Jumpstart model from scratch. My intuition is that there should be a way to save the weights of the deployed instance which can then be used to initialise the Jumpstart model. This issue might have a more straightforward fix if working with a model trained from scratch. Having the same SageMaker Endpoint be updated is also required because the Lambda function is dependent on the endpoint name for it to pass input. If a new instance is spun up every time a retraining job occurs, this is something that will need to be handled additionally which is extra overhead.

Once training is finished, the model can automatically be deployed (again ideally to the same endpoint) which would all be handled from the retrain script.

4. Model Description

The primary goal of this project was to gain more of an understanding of deploying an AI/ML model in the AWS ecosystem and building it to a degree that it can be a blueprint for a real-world product/service, which is also to my understanding, the primary focus of this module. As described in 3.1. SageMaker, the model used is a Pytorch Resnet18 pretrained model provided by the SageMaker Jumpstart offering. The code used for the project to select the model based on the model id and deploying it was taken from the sample notebook provided by the AWS GitHub repository (Amazon AWS, 2024). The working code with outputs is uploaded in the form of an ipynb notebook called sagemakerModel.ipynb under the \sagemakerCode directory in the source code zip.

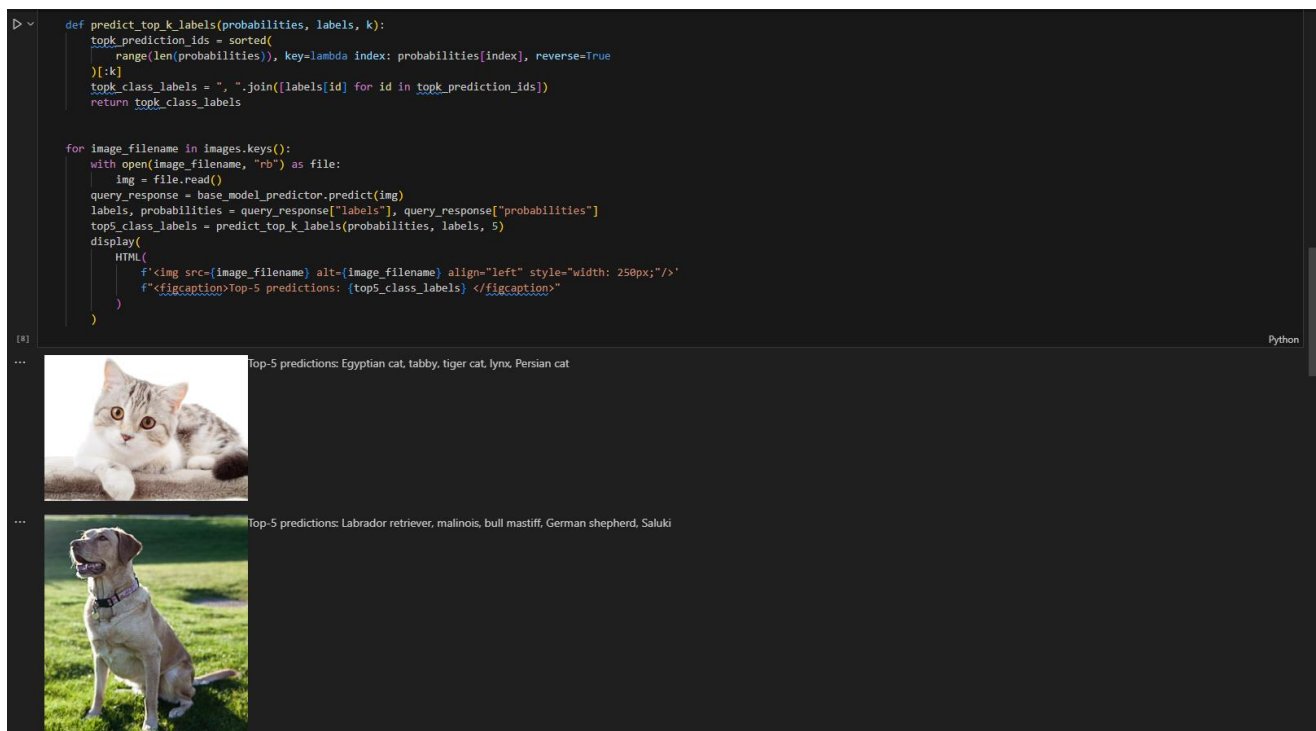


Figure 10: Prediction results of deployed pretrained Resnet18 model

The deployed model was also tested, and it did it accurately for as was expected. This can be seen from Figure 10. The model just doesn't predict for the genus of animal but also the species with a top 5 (number 'k' specified in the code) prediction result from the model instance.

5. Scalability Considerations

In terms of scalability, the design and workflow of the architecture tries to consider efficiency of resources as much as possible. For instance, instead of running retraining batch jobs at set intervals, they are run only when a required threshold is met as discussed in 3.3.2. Lambda function: `retrainPipeline`. With the use case also requiring single inferences for each user, we don't need to have a separate scheduler taking up extra resources periodically checking for the current accuracy either. This is checked only when the model is being actively used for inference triggered on a need basis via API calls.

However, should the need arise for a high concurrency in requests for inference to the SageMaker Endpoint, we have options to use AWS' Autoscaling feature (Patrick Denny (b), 2024) for the SageMaker instances (Amazon AWS (c)). We can also fine-tune the scaling with manual configurations of load balancing (Patrick Denny (b), 2024).

We could consider shutting down SageMaker instances for periods of no activity for a new one to be spun up based on the saved models when required. But considering the target use case of digital pathology/medical imaging where the situation might require high availability, this would not be ideal.

6. References

Amazon AWS (a). (n.d.). *SageMaker JumpStart - Amazon SageMaker*. Retrieved from AWS Documentation - Developer Guide:

<https://docs.aws.amazon.com/sagemaker/latest/dg/studio-jumpstart.html>

Amazon AWS (b). (n.d.). *AWS Amplify - user Guide*. Retrieved from AWS Documentation:

<https://docs.aws.amazon.com/amplify/latest/userguide/welcome.html>

Amazon AWS (c). (n.d.). *SageMaker - Auto scaling overview*. Retrieved from AWS Documentation - Developer Guide: <https://docs.aws.amazon.com/sagemaker/latest/dg/endpoint-auto-scaling-prerequisites.html>

Amazon AWS. (2024, February 2). *Introduction to JumpStart - Image Classification*. Retrieved from GitHub | AWS - Amazon SageMaker Examples: https://github.com/aws/amazon-sagemaker-examples/blob/main/introduction_to_amazon_algorithms/jumpstart_image_classification/Amazon_JumpStart_Image_Classification.ipynb

Microsoft Research - Kaiming He, et. al. (2015). *Deep Residual Learning for Image Recognition*. Retrieved from arXiv | Computer Science - Computer Vision and Pattern Recognition: <https://arxiv.org/pdf/1512.03385>

Proof of concept for a Scalable Image Classifier with a Retraining Pipeline on Amazon Web Services (AWS)

Patrick Denny (a). (2024, April 12). *AWS SageMaker - Section 6: Hosting and using the model.*

Retrieved from CS5024 - Theory and Practice of Advanced AI Ecosystems:

<https://learn.ul.ie/d2l/le/lessons/17937/topics/654914>

Patrick Denny (b). (2024, March 19). *AWS Automatic Scaling and Monitoring - Tuesday Week 8*

Material. Retrieved from CS5024 - Theory and Practice of Advanced AI Ecosystems:

<https://learn.ul.ie/d2l/le/lessons/17937/topics/650590>

Patrick Denny (c). (2024, March 8). *AWS Cloud - Compute (Section 5) - Friday Week 6 Lecture.*

Retrieved from CS5024 - Theory and Practice of Advanced AI Ecosystems:

<https://learn.ul.ie/d2l/le/lessons/17937/topics/646049>

Prince, S. (2024, May). *SageMakerRetrainPoC - sprince0031.* Retrieved from GitHub:

<https://github.com/sprince0031/SageMakerRetrainPoC/tree/main/webapp>

Tutorials, T. T. (2022, June 27). *AWS Project: Architect and Build an End-to-End AWS Web Application from Scratch, Step by Step.* Retrieved from YouTube:

https://youtu.be/7m_q1ldzw0U?si=SvMyCgLFGbZx295l