COMP4651 Project Report

ResNet Model Deployment in AWS Lambda

Introduction

This project presents two deployment approaches of the ResNet machine learning model for image classification. The first implementation is serverless, in which the ResNet model is deployed as a lambda function in AWS Lambda, a CPU-based FaaS framework. This approach is 'serverless' in the sense that no maintenance or monitoring of servers are needed. The second implementation is a server-based local approach, where a Python deployment script is written, using the ResNet model to perform inference in the local machine. The performance metrics of these two approaches would be monitored, displayed, and analyzed. The aim of this project is to compare the attributes of these two approaches, and conclude which approach is better in different scenario.

Architectural Overview

Our serverless approach is implemented based on AWS Lambda, the FaaS service provided by AWS that enables users to run a function. The schematics of our serverless architecture could be found in Figure 1.

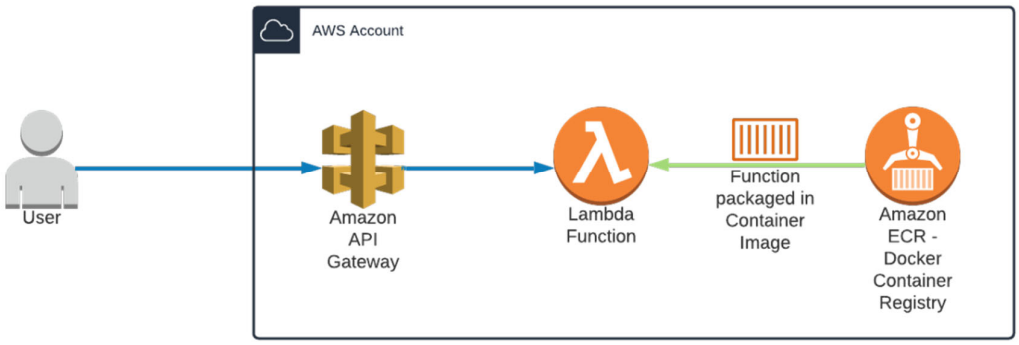


Figure 1: AWS Lambda Architecture Overview

Our entire serverless program is set up in the following steps:

1. **File preparation:** We first write the lambda function, implementing the actual model inference as well as the pre/post processing steps. A Dockerfile is constructed to package the lambda function, the ImageNet class names, and the dependencies of the lambda function script into a single docker image. These files are packaged together since the lambda function need these files to run smoothly.

2. **Setting up Lambda:** We then build this docker image, and push it into Amazon ECR – a registry storage service that is capable of storing Docker Images.We create a function in AWS Lambda from the Container image URL stored in Amazon ECR. This lambda function would be up and running after this step

3. **API creation:** We then create an HTTP API to call this function for image classification. An API would provide ease of access to various users. Users could access and run the lambda function simply by fetching the API.

Inference Rundown

This is how we perform inference in our serverless approach. This is the logic behind our lambda function and could be verified from the code submitted.

1. **User request**: A user uploads an image and clicks the "Classify" button. The frontend checks the image. It must be JPG, JPEG or PNG. The image is then encoded to base64 format. The frontend would then send the POST request to /classify endpoint. The backend script would call the Lambda API to run the lambda function on the base64-encoded input image.

2. **Pre-processing:** The lambda function would receive a base64-encoded image as input. The first step is naturally to decode it back to an image in the RGB format. After resizing, standard ImageNet transforms follow to make the image a suitable input to the ResNet model. This includes: Resizing to 256px, center-cropping to 224px, converting the image to tensor, and normalizing the image.

3. **Model inference**: A pre-trained ResNet-50 model is loaded from torchvision library into the module level so that it could be re-used for multiple invocations. This model then takes in the pre-processed image and carries out classification.

4. **Post-processing:** We apply softmax function to the model output to obtain the class probabilities. The output is then prepared by parsing class names and IDs from the iamgenet_class text file. A JSON response consisting of: Top 5 predictions (with probabilities), Image size information, performance metrics, and CORS headers (for web access).
5. **Response to frontend:** The JSON response is then set to the caller (backend), which then displays the information to the frontend for user to view.

Performance Metric Collection

Performance metrics are collected in two ways: local computation inside lambda function, and Amazon X-Ray. Metrics are collected from X-ray since its utilizes powerful monitoring

infrastructure in AWS, and metrics are almost immediately available after the lambda function completes. In case of any delays, however, X-Ray metrics would be updated once every 5 seconds for 10 times after the classification result completed to ensure we get the most recent X-Ray metrics, these two approaches provided metrics every time the Lambda is called.

1. **Metrics calculated in lambda function:**
   Memory Usage, Memory Utilization
   Original and processed image size, time needed to parse JSON input

2. **Metrics collected from Amazon X-Ray:**
   Execution time: Inference time, Initialization time, invocation time, overhead time
   Function and trace details: HTTP status, request ID, trace ID

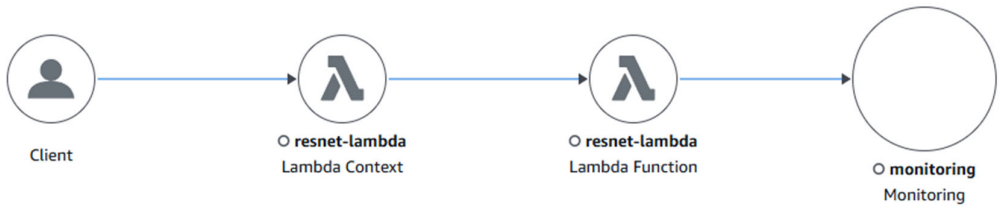
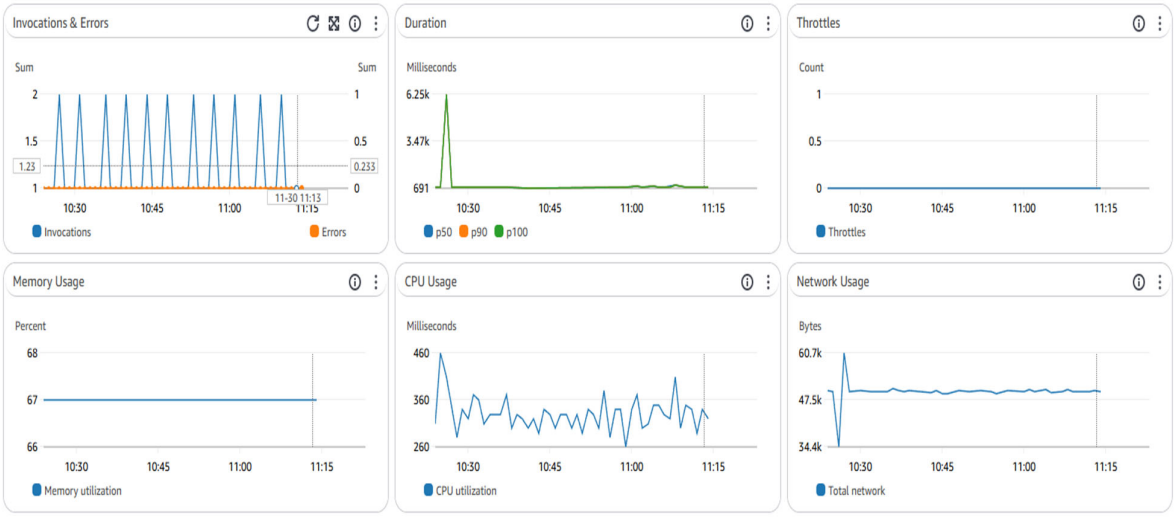


Figure 2: Amazon X-Ray Trace Details

Additionally, the capture the trend and long-term performance of the metrics, AWS CloudWatch is used to display metrics over a period of time. A CloudWatch dashboard (see figure) is created for monitoring the performance of a 30 minutues testing, where lambda is called every 30 seconds continuously.



Server Approach

We implemented the second server-based approach in a separate python script, this script load the pre-trained ResNet-50 model and perform classification locally. Performance metrics are then measured.

Performance Analysis

In terms of inference time, the server approach is more efficient that the serverless Lambda approach. The inference time for one image in local server approach takes around 12 – 16 ms running on a 13th Gen Intel(R) Core(TM) i7-13700K CPU with 3.40 GHz, while a warm start in the serverless Lambda approach takes around 700 - 800 ms on average. The serverless approach also has a cold start problem, meaning that a lambda function needs extra time for initialization when it has not been run for some time. Usually, this initialization takes around 5000 – 5500 ms. In general, the server approach is faster than the serverless. We hypothesized that this is due to the following factors:

1. **No overhead**. The lambda approach requires base64 encoding/decoding of images, while the local script works on images directly, no JSON serialization is needed.
2. **No network latency**. Lambda requires API Gateway request time, invocation time, etc. These would slow down the classification progress
3. **No cold start**. The inference time is significantly higher when there is a cold start, while local script does not experience any cold start.
4. **Difference in hardware**. The CPU that runs the local script might be stronger. We don't exactly know the specifications of the CPU running the lambda function, since lambda provide and manage all the CPU and memory allocation behind the scenes. Therefore, the difference in performance might be due to the difference of CPU specs.

In terms of memory, the max memory used in the lambda function is 670 – 690 MB according to AWS X-Ray. The memory usage of our local script is around … . The serverless approach generally use more memory, we suspect that this is due to the base64 encoding, which generally increase payload size by 33%. Note that API Gateway has a 10MB size limit, so our image size has to remain small, preferably under 3 MB. The memory usage of the lambda function is set to 1024MB.

Serverless vs Server

Based on our performance analysis, we summarize the pros and cons for both approaches:

Aspect	Serverless (AWS Lambda)	Traditional Server-Based
Performance	Efficient scaling but may have cold starts and resource limits.	No network latency and overhead, consistent performance, customizable resources.

Cost	Pay per execution, a pay-as-you-go, potentially cheaper for low-traffic apps. Free tier available with 1M request / month. No upfront cost	Fixed costs, more predictable for services with a consistently high demand, as more control over hardware is given.
Scalability	Auto-scaling and parallelism are built-in; scales well for parallel processing.	Full control over scaling but requires more setup and may involve higher operational costs for concurrent processing.

The server approach might be preferable when performance is critical, and working with consistently high volumes of requests. Other motivations to use the server approach might be data privacy, offline workflow, etc.

On the other hand, the serverless approach would be advantages when sporadic workloads are expected, or when minimum infrastructure requirement and maintenance is preferable. AWS also provides good monitoring service and integration to other AWS service.

Conclusion

This project explores a server deployment of ResNet-50 via a local python script, and a serverless deployment in AWS Lambda. We tested the classifier and collect the performance metrics for both approaches for comparison. Performance analysis is then conducted using the collected statistics. We discovered that both approaches have their strengths and weaknesses, and users should choose carefully between these two approaches when they are deploying computer programs.