# OPERATIONALIZING AN AWS ML PROJECT

Udacity AWS Machine Learning Engineer Nanodegree

## 1. SAGEMAKER SETUP

### 1.1. Sagemaker instance selection

Sagemaker's region is `us-east-1` (N.Virginia). There are many machine learning instances available in that region. The instance was chosen based on cost and capacity. Since the size of the data and training model is not large, it doesn't require to have high CPU and memory. Therefore, the most cost efficient instance that can process the data for the project is `ml.t3.medium` with 2 vCPU and 4GiB of memory.

Below is the snippet of available instances in US East region.

Region: US East (N. Virginia)

| Standard Instances | vCPU | Memory | Price per Hour |
|---|---|---|---|
| ml.t3.medium | 2 | 4 GiB | $0.05 |
| ml.t3.large | 2 | 8 GiB | $0.10 |
| ml.t3.xlarge | 4 | 16 GiB | $0.20 |
| ml.t3.2xlarge | 8 | 32 GiB | $0.399 |
| ml.m5.large | 2 | 8 GiB | $0.115 |

### 1.2. Data loading and uploading

The data was downloaded from `https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip` to personal `s3://project4-us-files1/`.

The data contains `train`, `test` and `valid` folders with subfolders indicating dog breed.

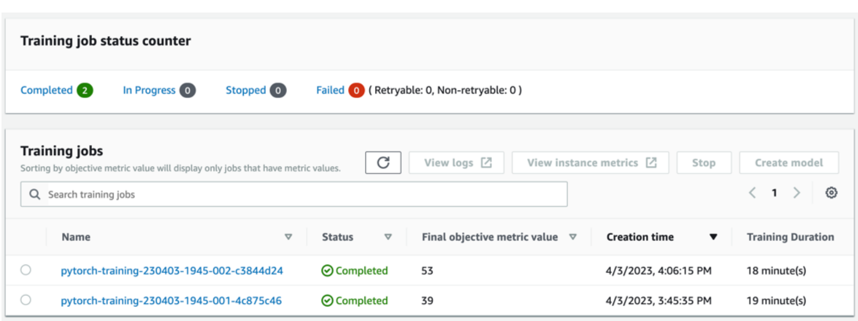
## 2. MODEL TRAINING

### 2.1. Hyperparameter tuning

The following hyperparameters were tuned.

` hyperparameter_ranges = { "learning_rate": ContinuousParameter(0.001, 0.1),
"batch_size": CategoricalParameter([32, 64, 128, 256, 512]) } `

The tuning job was successfully.



Training job status counter

Completed 2 In Progress 0 Stopped 0 Failed 0 ( Retryable: 0, Non-retryable: 0 )

Training jobs

Sorting by objective metric value will display only jobs that have metric values.

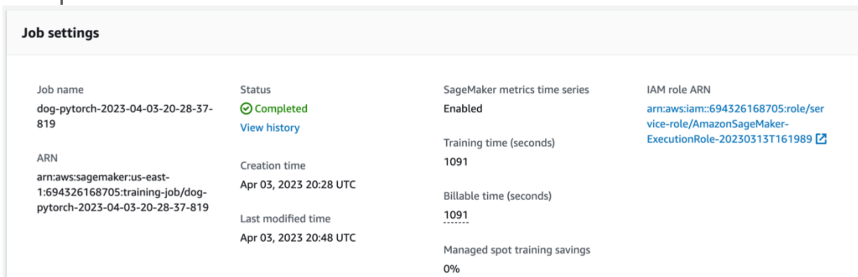
| Name | Status | Final objective metric value | Creation time | Training Duration |
|---|---|---|---|---|
| pytorch-training-230403-1945-002-c3844d24 | Completed | 53 | 4/3/2023, 4:06:15 PM | 18 minute(s) |
| pytorch-training-230403-1945-001-4c875c46 | Completed | 39 | 4/3/2023, 3:45:35 PM | 19 minute(s) |

### 2.2. Best model training

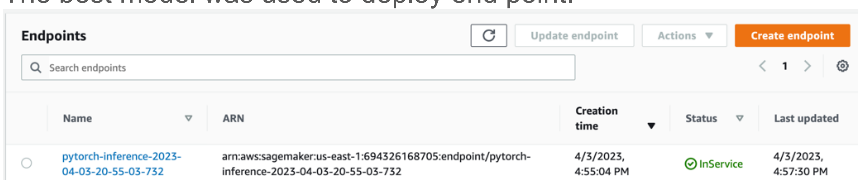
The best hyperparameters from tuning job are below:

` {"batch_size": "32", "learning_rate": "0.0018776517232838084"}`

The parameters were used to train the model.

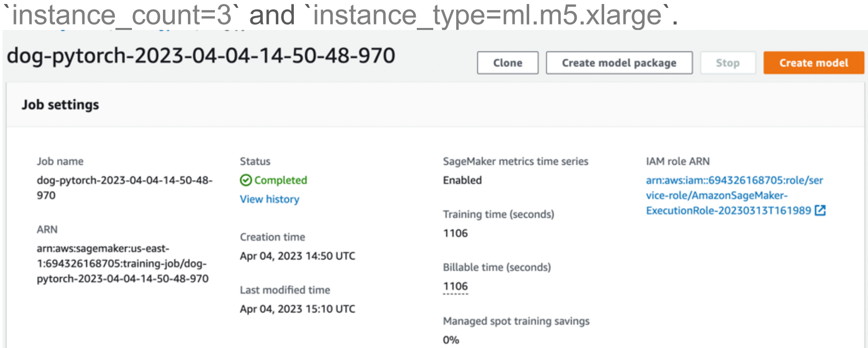



The best model was used to deploy end point.

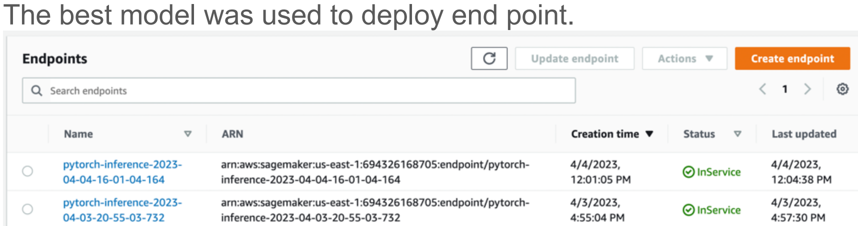



## 2.3. Multi-instance training

The same model as best estimator was used for multi-instance training with `instance_count=3` and `instance_type=ml.m5.xlarge`.




The best model was used to deploy end point.




# 3. EC2 TRAINING

## 3.1. EC2 instance setup

The data is small and the model doesn't require large resource. `Deep Learning AMI GPU PyTorch 1.13.1 (Ubuntu 20.04) 20230326` AMI was chosen, however, `t3.small` instance type was not supported by AMI. Although `t3.small` is the most cost efficient, it is a general-purpose instance and not suitable for accelerated computing such as GPU. The suitable instance types were P and G instance types. From the [instance types](instance types), G3 had the lowest GPUs and reasonable CPUs which would be suitable for our application. Therefore, `g3s.xlarge` was chosen.

## 3.2. Model training

1. Download and unzip the data

'wget https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip`
`unzip dogImages.zop`

2. Make directory
   `mkdir TrainedModels`

3. Created `solution.py` model training script from `ec2train1.py`

4. Activated pytorch environment
   `conda activate pytorch`

5. The model training
   `python solution.py`

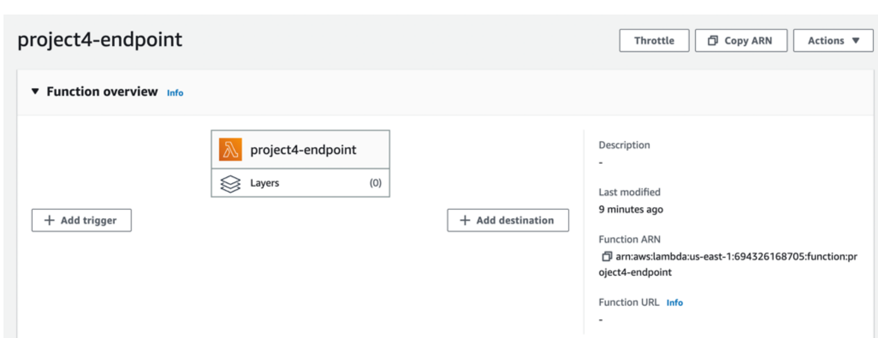

### 3.3. Code difference

The sagemaker code requires explicitly passing arguments such as train and test data locations, learning rate and epochs when fitting the model. On the other hand, EC2 training (`ec2train1.py`) had the arguments specified in the script.

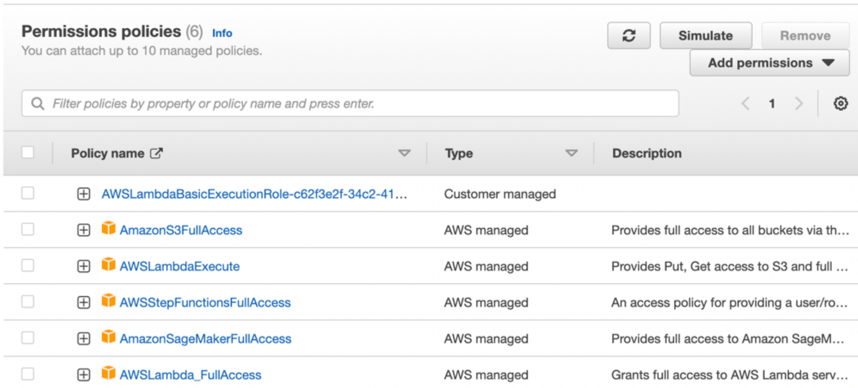
The dataset was accessed directly from `dogImages` folder when training on EC2 and the model is saved to `TrainedModels` folder. Working on EC2, we need to specify where the input is and where to save the output. Also, the data needs to be on the EC2 instance.

## 4. LAMBDA – ACCESS AND CONCURRENCE

`lambdafunction.py` script was used to invoke the endpoint. The endpoint name was changed to `pytorch-inference-2023-04-04-16-01-04-164` which is the endpoint for multi-instance trained model. In the script, `sagemaker-runtime` session was initiated through boto3 and the `lambda_handler` function uses the runtime to call `invoke_endpoint`. The response of the endpoint invocation is stored to `result` variable and the function returns `statusCode` 200 if successful and response of the invocation.



When the lambda function was created, following accesses were added to the role to perform invocation. If the full accesses were not granted, the lambda function can’t invoke endpoint. Therefore, managing these policies are important first step to secure who can access the endpoint.

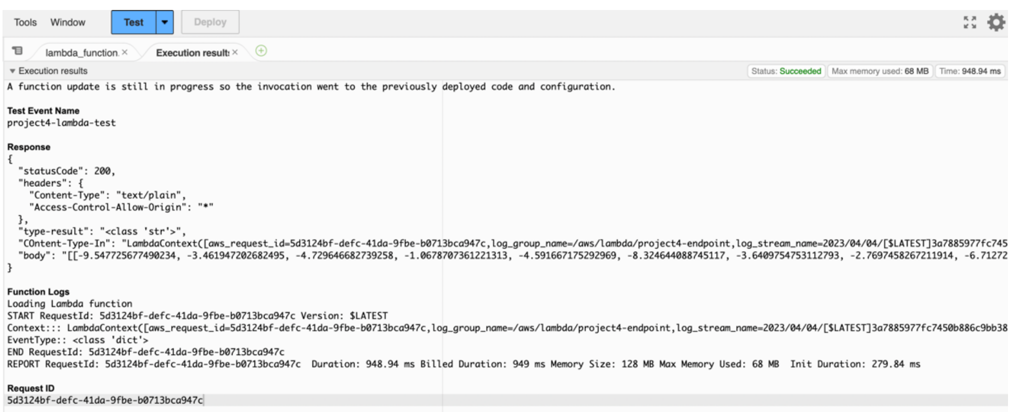


Permissions policies (6) Info
You can attach up to 10 managed policies.

| Policy name | Type | Description |
| --- | --- | --- |
| AWSLambdaBasicExecutionRole-c62f3e2f-34c2-41... | Customer managed | |
| AmazonS3FullAccess | AWS managed | Provides full access to all buckets via th... |
| AWSLambdaExecute | AWS managed | Provides Put, Get access to S3 and full ... |
| AWSStepFunctionsFullAccess | AWS managed | An access policy for providing a user/ro... |
| AmazonSageMakerFullAccess | AWS managed | Provides full access to Amazon SageM... |
| AWSLambda_FullAccess | AWS managed | Grants full access to AWS Lambda serv... |

The lambda function was tested by passing following image to the endpoint and it ran successfully.

` { "url": "https://s3.amazonaws.com/cdn-origin-etr.akc.org/wp-content/uploads/2017/11/20113314/Carolina-Dog-standing-outdoors.jpg" } `



```
Execution results
A function update is still in progress so the invocation went to the previously deployed code and configuration.

Test Event Name
project4-lambda-test

Response
{
  "statusCode": 200,
  "headers": {
    "Content-Type": "text/plain",
    "Access-Control-Allow-Origin": "*"
  },
  "type-result": "<class 'str'>",
  "COntent-Type-In": "LambdaContext([aws_request_id=5d3124bf-defc-41da-9fbe-b0713bca947c,log_group_name=/aws/lambda/project4-endpoint,log_stream_name=2023/04/04/[$LATEST]3a7885977fc745
  "body": "[[-9.547725677490234, -3.461947202682495, -4.729646682739258, -1.0678707361221313, -4.591667175292969, -8.324644088745117, -3.6409754753112793, -2.7697458267211914, -6.71272
}

Function Logs
Loading Lambda function
START RequestId: 5d3124bf-defc-41da-9fbe-b0713bca947c Version: $LATEST
Context::: LambdaContext([aws_request_id=5d3124bf-defc-41da-9fbe-b0713bca947c,log_group_name=/aws/lambda/project4-endpoint,log_stream_name=2023/04/04/[$LATEST]3a7885977fc7450b886c9bb38
EventType:: <class 'dict'>
END RequestId: 5d3124bf-defc-41da-9fbe-b0713bca947c
REPORT RequestId: 5d3124bf-defc-41da-9fbe-b0713bca947c  Duration: 948.94 ms Billed Duration: 949 ms Memory Size: 128 MB Max Memory Used: 68 MB  Init Duration: 279.84 ms

Request ID
5d3124bf-defc-41da-9fbe-b0713bca947c
```

By setting up concurrency and autoscaling, we can handle high volume traffic without any performance issue.

We can setup reserved or provisioned concurrencies. Reserved concurrency guarantees a maximum number of instances at lower cost, but the traffic exceeds the maximum number of instances, then it can create latency. On the other hand, provisioned concurrency is always on regardless of the traffic but costly. Combination of concurrency and auto-scaling, we can handle sudden spikes in traffic without affecting the performance. The number of concurrency depends on the application and traffic. It’s important to test before choosing any autoscaling or concurrency type to decide which is best for the project.

For this project, the traffic is unknown. Therefore, for the lambda function, 5 reserved concurrency instances and 3 provisioned concurrency instances were set.