# Operationalizing ML workflow on Sagemaker

# (Writeup)

## Training in a Sagemaker Instance

1. Chose ml.t2.medium as the instance for the jupyter environment. Choosing this because this is familiar to me as used in previous projects and chapters and this is all around good instance with 2x vCPUs and 4GB of RAM to run some jupyter notebooks.

2. Have done both single instance training and multi instance training. In my sagemaker, they both took about the same time to train. But the models are different with inference scores and the multi instance training seems to be less loss and better accuracy.

The images for this sections are in the Sagemaker Training Folder.

## Training in an EC2 instance

1. Chose t2.medium as my ec2 instance as I think it is a fair one, micro ones are too small and I think can't get the jobs done. 2 vCPUs with 4GB of RAM in t2.medium might be the sweet spot between performance and cost.

2. It's a pain that VIM's indentation is not good for long text paste, I had to manually input the codes to ***solution.py*** and run the training. The training took about only 30 minutes on this instance and it was succeed, and saved the model in the */TrainedModels* path as ***model.pth***.

3. Plot of CPU Utilizing during the training time is also in the corresponding image folder.

The images for this sections are in the Sagemaker EC2 Training Folder.

## Differences between Sagemaker and EC2

1. The difference of codes between the Step 1 and 2 are pretty simple. The sagemaker notebook's code use a different script as entry point for the model training which mean the model is trained in the separate container, that's why we need to save ***os.environ*** channel variables specified for sagemaker containers like "SM_CHANNEL_TRAINING, SM_OUTPUT_DIR" etc. and call these in the function files to recognize the paths of the data inputs and outputs. Can be found out more in here["https://github.com/aws/sagemaker-containers#id20"]. The EC2 training can be said as local training as both the code and the training run in the same machine, so its codes don't need to call the ***os.environ*** variables with channel names and the path can be specify by only using ***os.path*** function.

2. For that we don’t use sagemaker here in EC2 instance, the function calls for the sagemaker don’t need to import and use in the EC2 training. The drawback is that we can’t see the algorithm metrics easily in cloudwatch console as using sagemaker and can only see if the model is training and complete successful by viewing the CPU utilization of the instance.

3. As there is no Estimator or Tuner function to call the script and implement, the call for train the model and save it, is in the code already. Model training, creating a fully connected network with pretrained model and creating dataloaders are the same in both.

4. Next difference thing is that in EC2 training, all variables like hyperparameters and output locations, etc. are already declare in the script so no ***argparse*** is needed.

These are the differences I’ve found out after doing the 2 different training jobs in Sagemaker and EC2.

## Lambda Function

1. The lambda function acts as a trigger for invoking the deployed endpoint. It takes the input as JSON format and pass it to the endpoint to make the prediction and then pass out the result.
2. In the lambda code, the main functions are reading the input data, take the data, predicting the data and pass it, after that we can output the result to the Lambda function’s output body.
3. The given Lambda function output the prediction as a list of 133 numbers which represent the each score for the breed of dogs. I have altered it a little to be convenient for the user. I use numpy’s ***argmax*** function to take the maximum value of the array and output its position which is the index of the dog so we can easily check which dog is in the input image.
4. AWS Lambda functions are designed to be lightweight and are serverless, so we can’t just import numpy module from nowhere, this is why I added a layer to the lambda function which makes it possible to import the numpy module. This is a simple process and special thanks to KLayers team where we can use the public and globally available lambda with just adding the ARN(Amazon Resource Name). The AWS Lambda layers GitHub repo is here. [“https://github.com/keithrozario/Klayers/”]

The images for this sections are in the Lambda Function Folder.

## Security

1. Run the previous section’s lambda function AS IT will throw an error which is validation error because it doesn’t have access to the sagemaker endpoint. So I have added security policy to my lambda’s role with **AmazonSageMakerFullAccess** and it can now access the sagemaker endpoint.

2. The original result is here, an array of 133 values. [[-3.560807228088379, -2.3610072135925293, -0.8625258207321167, -0.17722102999687195, -2.1440281867980957, -3.120462417602539, -1.0959322452545166, 0.2171446532011032, -4.681222438812256, -1.3835368156433105, -1.2933207750320435, -3.779334545135498, -1.4543734788894653, 1.898128628730774, -1.9487924575805664, 0.7895327210426331, -4.596983909606934, -0.9622823596000671, -1.54801344871521, 1.5244377851486206, -1.968691110610962, -0.9859333038330078, -4.072300910949707, -2.7488625049591064, -2.8064982891082764, -6.242099285125732, -1.2242759466171265, -1.0663299560546875, -3.906348705291748, -1.7638208866119385, -1.4837985038757324, -1.3667646646499634, -4.613922595977783, -0.2851935923099518, -5.034514427185059, -5.533158302307129, -2.6331684589385986, -3.4135677814483643, 0.35371682047843933, -1.9390536546707153, -1.8252032995224, -2.119920253753662, 0.5255250930786133, -2.07735538482666, 0.09988265484571457, -3.689462423324585, -1.0837600231170654, 0.06851010024547577, -1.049939751625061, -0.15046218037605286, -2.7590506076812744, -1.9658691883087158, -4.304177284240723, -2.7950918674468994, -2.4758119583129883, 0.4787639379501343, -3.595694065093994, -4.044513702392578, -0.568463921546936, -0.52855384349823, -4.169152736663818, -2.5703585147857666, -3.3848400115966797, -3.6305768489837646, -1.5054829120635986, -5.913352012634277, 0.5828441381454468, -4.9383015632629395, -0.5694289207458496, -0.5188249349594116, 1.416540503501892, -3.3817920684814453, -3.5423991680145264, -4.119844913482666, -4.029443264007568, -1.7315510511398315, -4.739086151123047, -1.2344329357147217, -3.5023646354675293, -2.7102773189544678, 0.5984091758728027, -3.4548792839050293, -0.8053879737854004, -2.3994767665863037, -3.080897092819214, -2.2898976802825928, -0.1788490116596222, -4.654054641723633, -1.0352041721343994, 0.911021888256073, -3.3662216663360596, -3.694307327270508, -2.9581687450408936, -4.153717041015625, -0.9488220810890198, -0.9079939126968384, -2.3487606048583984, -2.724519968032837, -3.251596212387085, -3.505178213119507, -3.890536069869995, 0.22483614087104797, -3.0940070152282715, -3.426342248916626, -4.497158050537109, -5.338107109069824, -2.0497617721557617, -0.4637422263622284, -0.5943436026573181, -0.7306941747665405, -2.1099131107330322, -1.3877867460250854, -5.7542877197265625, -3.0296146869659424, -3.380420207977295, -2.3554749488830566, -3.7495195865631104, 0.0043024942278862, -3.033569097518921, 1.1251094341278076, -1.6520241498947144, -2.788029670715332, -2.43349027633667, -1.4584729671478271, -3.5486342906951904, -3.5071656703948975, -1.9741278886795044, 0.5189701914787292, -3.5623958110809326, -3.3902249336242676, -5.3868727684021, -1.3458232879638672, -3.5965476036071777]] And the best result is index 13 with a value of **1.898128628730774**.

3. There are nothing much in my IAM dashboard as there is just a few resources, and all of the permissions are set according to the needs of it's usage.
4. But the **FullAccess** of my Lambda function may create a security vulnerability as it is too much for just using an endpoint yet I can't find the proper permission for just using an sagemaker endpoint. The **AmazonMachineLearningManageRealTime-EndpointOnlyAccess** policy is out of the line as it is use to create or delete an endpoint and not using them.

The images for this sections are in the Security Folder.

## Concurrency and Auto Scaling

1. We have all the resources and functions ready and now is the time to set the concurrency and auto scaling to the inference model.
2. For the concurrency of lambda function, I have set up the reserved concurrency as 5 which mean the function can handle up to 5 request at the same time. I have also set the always-ready provisioned concurrency as 2 to reduce the latency.
3. For the endpoint, I have set up maximum 2 endpoints and set the scale in cold down to 300 and scale out cold down to 30 to minimize the cost while auto scaling during traffic.

The images for this sections are in the Concurrency and Auto Scaling Folder.

## Thank you so much for your time!

*~EOF~*

*Writeup*
*~ By ~*
*Htin Aung Lu*