**Write-Up**

**Notebook instance:**

The notebook instance which I’ve used for AWS ML projects is ‘*ml.t3.medium*’. This is the least tier instance in the standard instances that could be created in the sagemaker. This instance comes with **2 CPU** cores, **4 GB** memory and costs about **0.05 dollars** per hour. Considering the performance and the memory needed for our project, this instance would be enough to run all the kernel at reasonable speed. Also, we have a constraint of **25 dollars** in our sagemaker account. So, I guess that would do fine.

**EC2 instance:**

The instance which I have used is of *‘t2.micro’* type. This is also the basic instance that would be suitable for our use case. Here, we will only use this instance to train the model. So, the micro level instance would be sufficient. This instance comes with a **single core** of CPU. The AMI we used is *‘Deep Learning AMI GPU PyTorch’*, this AMI comes with a pre-installed pytorch environment, where we shall use it for training the model.

**Difference in *EC2train1.py*:**

In the sagemake instance, we have used *hpo.py*, to run the training to find the optimal hyperparameters. So, to vary the hyperparameters for each time we train, we used arguments. But in *EC2train1.py*, we aren’t finding the optimum hyperparameters. We just mentioned the values of the hyperparameters, at which the model should be trained.

**Note on *lambdafunction.py*:**

This code is specifically for deploying a lambda function to predict the images. Just giving out the path of the file in *JSON* format is enough for the lambda function to predict it. This is a great tool in automation. For the lambda function, we are using the endpoint of the multi instance trained model. The endpoint is connected through the *.invoke\_endpoint()* method, which would pipe the lambda function to the endpoint.

**Workspace Security:**

I think my AWS workspace is secure, because the resources in my account have no permission outside of the workspace, only the admin can access it. When it comes to lambda function, ***AWSSagemakerReadOnlyAccess*** policy gives us access denied error. So, the only way to use the end-point is the ***AWSSagemakerFullAccess*** policy, which only would allow us to use the endpoint. This may be a vulnerability, as we ourselves when checking out lambda function may accidentally augment it or even in worst case delete it. So, this must be a serious thing to consider.

**Concurrency & Auto-scaling:**

The function concurrency for the lambda function is set to *“Use reserved concurrency”*. We have set it to **3 reserved concurrency**, that means the lambda function can handle up to three concurrent requests. In the real-case scenario, 3 would not suffice, we just set it up to 3 because of learning purposes and the constraint we have in our account. We have also updated the endpoint runtime settings, so it can scale up to three instances in the state of receiving maximum traffic. This also would not suit the real-case scenario, we did it for just learning and keeping the account constraint in our mind.