**Project 4: AWS - Operationalizing a ML Workflow - Dog breed Image Classification using AWS SageMaker and PyTorch**

This project uses a number of tools from AWS to setup and prepare a ML model for deployment in production. Supplied as part of the project is the following code:

* Train\_and\_deploy.ipynb (notebook that coordinates training and deployment of image classification)
* Hpo.py (python code that carries out hyperparameter tuning (and training of model)
* Inference2.py (entry point for model endpoint deployment and for inference)
* Lambdafunction.py (code to run lambda function)

The following steps need to be carried out as part of the project:

Step 1: Training and Deployment on SageMaker

Step 2: EC2 Training

Step 3: Lambda Function Setup

Step 4: Lambda Security and Testing

Step 5: Concurrency and auto-scaling

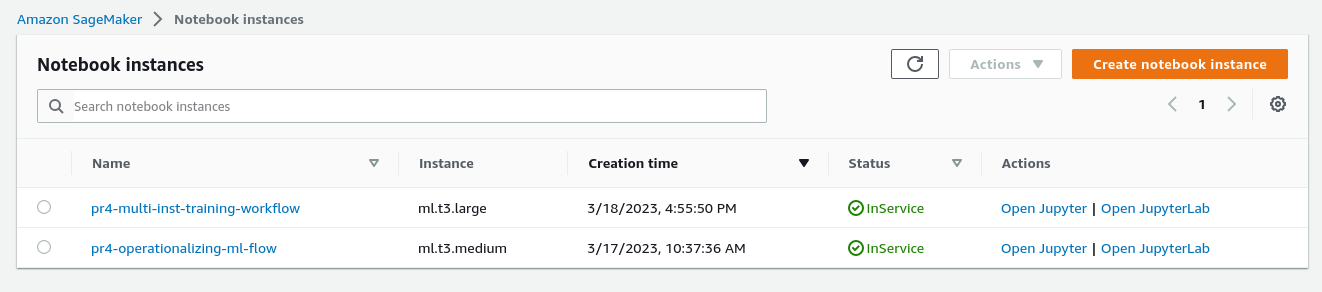
**Step 1: Training and Deployment on SageMaker**

A S3 bucket to store dog images image files as input data was set up. A screenshot is here:

Graphical user interface

Description automatically generated with medium confidence

A small instance ml.t3.medium was set up being one of the cheaper options. Single instance training was performed. Memory issues when re-running code for multi-instance training lead to choosing a larger instance for this case – ml.t3.large (may have not occurred if original instanced had been restarted instead?). A screenshot of the running instances is here:



Tuning was performed using instance type ml.g4dn.xlarge. This instance type, suggested as a default, was tried first and kept since it worked. Training jobs were run ml.m5.xlarge, also a suggested default, and it worked. Note that the number of epochs was limited and if one wanted to reduce tuning and training times and increase number of epochs then larger instances would need to be chosen.

**Single instance training details:**

hyperparameter tuning: pytorch-training-230318-1429

best values: batch\_size: 128, learning\_rate: 0.0034956970896215297'

training: dog-pytorch-2023-03-18-15-21-59-756

endpoint: pytorch-inference-2023-03-18-15-49-10-447

**Multi-instance training details:**

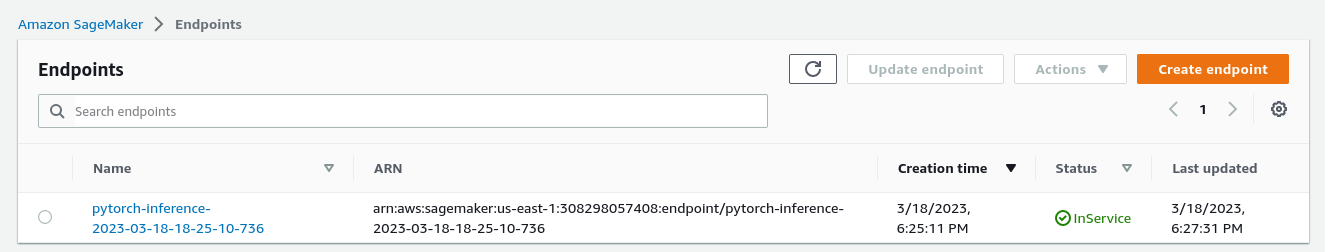
hyperparameter tuning: pytorch-training-230318-1709

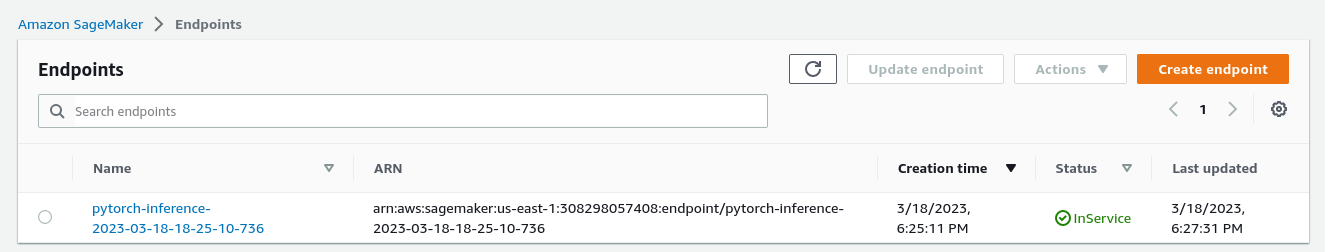
best values: batch\_size: 64, learning\_rate: 0.030884864336889317

training: dog-pytorch-2023-03-18-18-01-30-589

endpoint: pytorch-inference-2023-03-18-18-25-10-736

**Screenshots:** details of tuning and training jobs are in “screenshots”, here are screenshots of deployed endpoints:



****

**Step 2: EC2 Training**

Instead of using a SageMaker instance for model training the model can also be trained using an EC2 instance. A script called “ec2train1.py” was supplied, which was adapted from the notebook train\_and\_deploy.ipynb to work on EC2. It was not possible to select ml.g4dn.xlarge as an instance due to resource limits on GPUs, and ml.m5.xlarge (tbc) was selected.

The number of epochs was reduced from 5 to 3 in the training process to further conserve resources since the main purpose was to illustrate the use of EC2 for model training.

A conda environment also needed to be set up on a Linux AMI (as the advice from mentors was that subscription to AWS Marketplace was not required for this project): details to be added …:

Try and use: “Deep Learning AMI GPU PyTorch 1.13.1 (Amazon Linux 2)”

Here are screenshots of the EC2 instance setup and of the model output:

EC2 instance (running): to be added

EC2 model output: to be added

**Step 3: Lambda Function Setup**

Having a model available to make inferences, we used a lambda function, using the script lambdafunction.py provided, and adapted for our model.

Lambda functions are serverless compute service which can be adapted and scaled to match demand. In the case used here the lambda function links inference input data with a deployed model and performs inferences via deployed endpoints.

The setup of the lambda function code is as follows:

It imports packages, sets logging and then also used sagemaker runtime session using boto3 client. The endpoint to be used for inference is specified next. Then, the lambda\_handler function is defined as the main function, which takes two input arguments (event, context). The handler starts up sagemaker at runtime, and invokes the endpoint to make an inference for the function input arguments. The result of the inference is then returned as part of a json file by the handler.

Screenshots: to be added

S**tep 4: Lambda Security and Testing**

Details: to be added

**Step 5: Concurrency and Auto-scaling**

Details: to be added