OPERATIONALIZING MACHINE LEARNING ON SAGEMAKER – WRITEUP

- **BY RAPHAEL BLANKSON**

1. **TRAINING AND DEPLOYMENT ON SAGEMAKER**
   I selected **ml.t2.medium** for my notebook instance. This instance type has **2 vCPU**s and **4GB** of memory which is very good to run most jupyter notebooks that does not require a GPU. I also chose this instance because it is very cheaper than the rest costing around $0.05 per hour.

   For the training and hyperparameter tuning, I selected **ml.m5.xlarge** instance type because it has **4 vCPU**s and **16 GB** of memory which is more powerful that can speed up the training and tuning job times. It will also ensure that there are no memory issues to prevent the jobs from completing successfully.
   Imagesrc: [screenshots/Training and deployment]

2. **EC2 TRAINING**
   For my EC2 training, I chose the **t2.medium** instance. One of the main things to consider when selecting instances is to make a trade-off between cost and performance. This instance is very cheap costing only $0.0464, with **2 vCPU**s and **4GB** of memory. It also has low to moderate network performance. This good enough to run this project without any issues and also ticks the boxes for affordability and good performance.
   imagesrc: [screenshots/EC2 Training]

   DIFFERENCES BETWEEN SAGEMAKER AND EC2
   The first noticeable difference between the EC2 and sagemaker code is that for EC2, all the training codes are in the same script whilst for the sagmemaker, the training script uses an external file called **hpo.py** as entry point.
   In short, the EC2 can be considered as having local training because all the training code run in the same machine.

Secondly, the EC2 code does not have a main function and since all the hyperparameters and other model variables exist in the same location, there is no need for argparse to be used whilst the sagemaker training uses argparse.

3. **LAMBDA FUNCTIONS**

Lambda functions acts as an intermediary service that allows external users and applications to interact with the ML models in the project.
The lambda function in this project accepts input in JSON format and pass it to the endpoint to make predictions and then the results sent out to the calling application.
After the lambda function accepts the input data, it returns a **JSON** object with the status code, the http header and the results of the prediction in the results body which is a list of length 133 where each number represents the score for each class or dog breed.
imagsrc: [screenshots/Lambda functions]

4. **SECURITY AND TESTING**

The lambda function needed the right security policy to be able to access sagemaker endpoints so I selected the **AmazonSageMakerFullAccess.**
The full access role provided to lambda function may pose some security risk as any user or application that have access to the lambda function may be able to do more things with sagemaker than just the function of the lambda function.
To have a more secured system, the lambda function should only have a policy that allows to do just specific task it is supposed to do.
imagesrc: [**screenshots/Security and Testing**]

Below is the result returned by the lambda function: [[1.0660507678985596,

```
-3.8498575687408447, -2.6550872325897217, -0.7075858116149902, -0.3214341104030609,
-2.1423215866088867, -0.7189298272132874, 0.12265263497829437, -4.225289821624756,
0.6748189926147461, -0.7906899452209473, -3.558218479156494, -2.8959827423095703,
0.9804308414459229, -3.107088327407837, -0.5493598580360413, -3.625596761703491,
0.964867889881134, -4.338134765625, 2.2020795345306396, -3.1580307483673096,
-0.4333062171936035, -3.6774401664733887, -3.4904446601867676, -1.615373969078064,
 -4.561880588531494, 0.6368071436882019, -2.426971197128296, -3.4118194580078125,
-1.1816596984863281, -1.1891441345214844, -3.496084451675415, -3.164612054824829,
 -0.5320661067962646, -4.0865631103515625, -2.039386749267578, -4.621379375457764,
 0.3311939537525177, 0.2068183422088623, -2.008843183517456, -1.2211636304855347,
 -0.7703704237937927, 1.9603004455566406, -0.607782781124115, -0.2088540941476822,
 -4.541134357452393, 0.06538074463605881, -0.6243757009506226, -0.2188795655965805,
 -1.8391863107681274, -3.0454049110412598, -5.70417594909668, -4.922774791717529,
 -3.0724141597747803, -3.8730175495147705, 0.4144132435321808, -3.016660690307617,
 -3.59576153755188, -0.31666263937950134, -1.1928244829177856, -3.7776553630828857,
 -5.078307628631592, -6.228950023651123, -2.8935773372650146, -4.558743000030518,
 -7.263036727905273, 0.6917160153388977, -3.4851784706115723, -0.8205812573432922,
 0.4460907578468323, 1.42412531375885, -3.248924970626831, -3.1711792945861816,
 -0.40685877203941345, -2.454446792602539, -1.6705632209777832, -1.170825481414795,
 -2.0716733932495117, -6.310996055603027, -1.9354995489120483, 0.834751307964325,
 -3.883026361465454, 0.45794442296028137, -0.5367678999900818, -5.659112453460693,
 -4.516526699066162, -0.05782671272754669, -4.112605094909668, -3.4510393142700195,
 0.7245814800262451, -5.311254978179932, -4.092605113983154, -2.299273729324341,
 -5.984002113342285, -1.647143840789795, -0.9740868210792542, -2.669355630874634,
 -2.016641616821289, -1.40985906124115, -2.4403882026672363, -2.513491630554199,
 0.07008165866136551, -1.7633951902389526, -3.215583562850952, -2.8343799114227295,
 -2.315598487854004, -2.038970947265625, -0.38404643535614014, -0.023447182029485703,
 0.5537558794021606, 0.09925755858421326, -1.369574785232544, -5.95085334777832,
 -3.0631139278411865, -4.041447162628174, -1.0685389041900635, -2.420799732208252,
 -0.6028756499290466, -6.156184196472168, 1.861029028892517, 0.6303690671920776,
 -1.1918858289718628, -4.186331272125244, -3.304373264312744, -4.0787858963012695,
 -3.596785306930542, 0.6725872755050659, -0.741853654384613, -1.4244126081466675,
 -5.972278594970703, -5.071226596832275, -1.043948769569397, -0.6688947677612305]]
```

5. CONCURRENCY AND AUTO-SCALING

Setting up the concurrency for my lambda function I needed to make a choice between reserved concurrency and provisioned concurrency. At first, I chose reserved concurrency to ensure that no other function can use the concurrency and also to have the benefit of no charge for configuring reserved concurrency.

I set reserved concurrency to 5 which means my lambda function can have 5 instances running simultaneously. However, since this is a hard maximum, there could be latency issues if request exceeds 5 (high traffic).

To avoid latency issues and have the benefit of both worlds (reserved and provisional), I also setup provisioned concurrency. This gives the benefit that my instances are prepared and ready to respond

immediately to the lambda function invocations, however provisioned concurrency comes with a cost so I set it 2 and this can reduce the latency of my instances.
For auto-scaling, I decided to set my maximum instance count to 3, I set **scale-in cool down** to **300** and **scale-out cool down** to **30** to minimize and also to avoid high latency issues during high traffic periods.
Imgsrc: [**screenshots/concurrency and autoscale**]