# AWS Deep Learning Project Summary

## 1. SageMaker Instance

For this project, I selected the ml.g4dn.xlarge instance as the SageMaker notebook instance. This instance includes 1 NVIDIA T4 GPU, 4 vCPUs, and 16 GiB of memory, which makes it suitable not only for writing and executing training scripts but also for running GPU-accelerated tasks directly from the notebook environment. Using a GPU-backed notebook instance enabled me to perform fast model prototyping, real-time visualization, and lightweight model training and inference without the need to launch separate training jobs.  
  
I also used the same instance type (ml.g4dn.xlarge) for the SageMaker training job. This choice was driven by the model complexity and dataset size in the dog breed classification task, which uses a convolutional neural network (ResNet50) in PyTorch. The GPU acceleration significantly reduced training time compared to CPU-based alternatives. Furthermore, this instance type is more cost-efficient than higher-end options like p3.2xlarge and offers an ideal balance of computing power, memory, and affordability for medium-scale deep learning tasks.

## 2. EC2 Instance Type

For running my training and experimentation outside of SageMaker, I also selected the g4dn.xlarge EC2 instance type. This instance mirrors the configuration of the SageMaker training instance, providing consistent performance and ensuring compatibility across platforms. With its NVIDIA T4 GPU, 4 vCPUs, and 16 GiB of RAM, the g4dn.xlarge is optimized for deep learning workloads and supports the full PyTorch stack with CUDA acceleration.  
  
Using this EC2 setup allowed me to execute Python scripts such as solution.py directly from the terminal in a fully customizable environment. It also provided the flexibility to install system-level dependencies, manage files locally, and debug in a more hands-on fashion compared to the managed SageMaker workflow. This instance type offered reliable performance for training and inference, while also being cost-effective for GPU workloads in EC2.

## 3. EC2 Code

SageMaker workflows are typically executed within Jupyter notebooks and rely on managed training jobs, which abstract infrastructure management. In contrast, my EC2 codebase consists of standard Python scripts that are executed via the command line using python3 solution.py. This approach provided complete control over the training pipeline, including model architecture, optimizer configuration, checkpointing, and file system management. Running training code on EC2 in this way is particularly useful for debugging, rapid iteration, and deploying custom ML pipelines outside the SageMaker ecosystem.

## 4. Lambda Function

In this project, I created an AWS Lambda function to serve the trained model using the SageMaker endpoint. The Lambda function invokes the deployed model using the boto3 SageMaker runtime client. The input and output are handled in JSON format, which ensures compatibility with API Gateway or other calling services. The function deserializes the incoming JSON payload, sends it to the endpoint, and returns the JSON-formatted prediction results to the user.

Results:

"body": "[[0.3931392729282379, 0.19850893318653107, 0.0755343809723854, 0.31221336126327515, 0.44697004556655884, 0.26206398010253906, -0.042854003608226776, 0.46322450041770935, -0.2522737681865692, 0.1290469467639923, 0.3967383801937103, 0.28378981351852417, -0.06825724244117737, 0.2908441126346588, 0.3331942558288574, 0.08570437878370285, 0.2042372226715088, 0.19647397100925446, 0.0952320322394371, 0.264789342880249, 0.18108974397182465, 0.09629255533218384, 0.3082562983036041, 0.26658788323402405, -0.3263208866119385, -0.178895503282547, 0.23071156442165375, -0.17511871457099915, 0.4918203055858612, 0.032269436866045, 0.15762461721897125, 0.34385719895362854, -0.11563944816589355, 0.29311323165893555, 0.03807554394006729, -0.06923995912075043, 0.09912572801113129, 0.23234403133392334, 0.31298550963401794, 0.16855695843696594, 0.3166598975658417, 0.0738036185503006, 0.005443993955850601, 0.35179591178894043, 0.12376400828361511, 0.3837113380432129, 0.07305746525526047, 0.02807636372745037, -0.057758867740631104, -0.0861886516213417, 0.12237401306629181, 0.0044277808628976345, -0.04717976972460747, 0.16363784670829773, 0.03171534091234207, 0.24962501227855682, 0.4702213704586029, 0.07654374837875366, -0.043430306017398834, 0.3008309602737427, 0.30920159816741943, 0.21857842803001404, 0.1497730314731598, -0.14306092262268066, -0.012081118300557137, -0.21612945199012756, -0.3311162292957306, 0.2650277316570282, 0.007646874990314245, 0.11499867588281631, 0.1467774212360382, -0.10772932320833206, -0.15654711425304413, -0.26545876264572144, -0.033667147159576416, 0.35536259412765503, -0.10537370294332504, -0.12271938472986221, 0.09925157576799393, -0.10404932498931885, 0.006626220885664225, 0.2572058439254761, 0.0445546954870224, 0.09840675443410873, -0.19769303500652313, 0.16181354224681854, 0.17421837151050568, -0.031439125537872314, 0.07291010022163391, 0.06051432713866234, 0.14661701023578644, 0.012182756327092648, -0.22974136471748352, -0.06764556467533112, 0.01849948801100254, 0.009895458817481995, 0.09155856817960739, -0.14729028940200806, 0.01223663054406643, -0.3773151636123657, 0.058184102177619934, -0.5459120273590088, 0.31319841742515564, -0.177310049533844, -0.4620038866996765, 0.15241654217243195, 0.005287701264023781, -0.28894445300102234, -0.09919345378875732, -0.35982462763786316, -0.0641218051314354, 0.17663879692554474, -0.2680213451385498, -0.18527017533779144, 0.35935044288635254, -0.4255167543888092, 0.10161100327968597, 0.17783242464065552, -0.41503816843032837, -0.3057018220424652, -0.38359200954437256, -0.29793912172317505, -0.0211506187915802, 0.05588788166642189, -0.28628113865852356, -0.555384635925293, -0.07121771574020386, -0.3526950478553772, -0.004961078986525536, -0.10039050877094269, -0.3092162311077118, -0.4613852798938751, -0.5112559795379639]]"

}

## 5. Lambda Function IAM Role and Security

To ensure secure access, the Lambda function was assigned an IAM role that includes permissions to invoke the SageMaker endpoint. This IAM role strictly allows only the necessary actions, such as 'sagemaker:InvokeEndpoint', preventing over-privileged access. By adhering to the principle of least privilege, the setup minimizes potential attack surfaces and limits the scope of access to only what is required for the Lambda function to perform inference. AWS-managed policies and service roles ensure that interactions remain secure and auditable, aligning with best practices for security in machine learning workflows.

## 6. Concurrency & Auto-scaling

For concurrency, I used provisioned concurrency since we have a requirement receiving a lot of traffic, so we need an automatic way for handling that. So I set provisioned concurrency to 3. For auto-scaling, due to the strength of the instance type I chose(ml.m5.xlarge), I set maximum\_instance\_count to 3. I also set the target value at the configuration of auto-scaling to 25 because I think it’s a suitable threshold to trigger the auto-scaling mechanism. If we choose a small number, then the auto-scaling is over triggered even if it’s not needed. If we choose a high number, then the auto-scaling may never be called. Scale-in and scale-out cool down values were both set to 30 seconds due to a balance of resource management, cost management and time acceptability.