Handwritten Digit Recognition

Initial Setup

- I entered AWS through the gateway in the course and open SageMaker Studio. Downloaded and made the dataset available.
- Started by installing all the requirements in the jupyter notebook " mnist-handwritten-digits-classification-experiment (1).ipynb."
- I used the basic"Python 3 (Data science - 01)"Kernel in this project. Also essential PyTorch libraries have been installed.

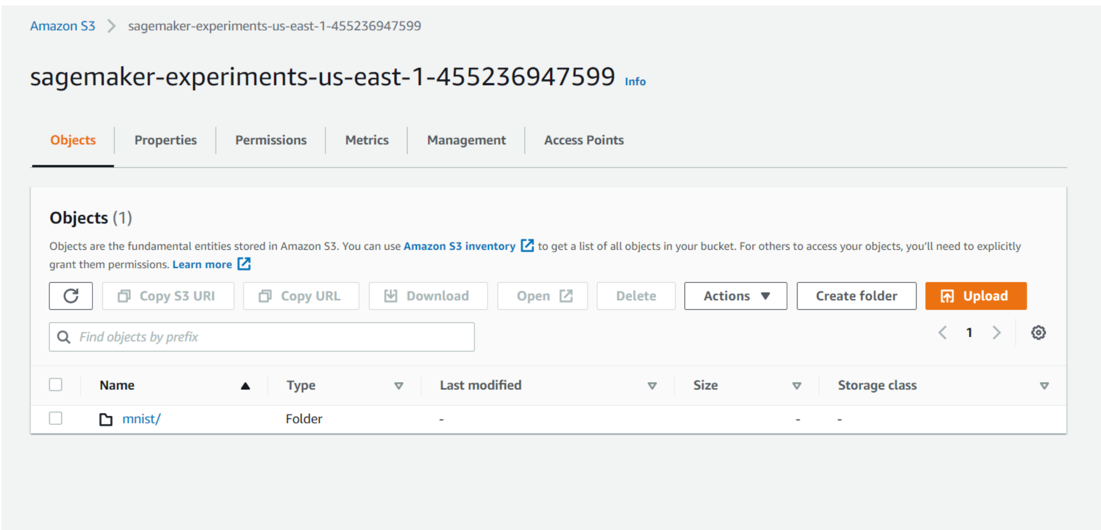

Figure 1: S3 Bucket Setup

Submitted by
Mansi Chilwant

- The project run on SageMaker, such as Autopilot jobs and training jobs, will be logged automatically. You may also keep track of artefacts for other phases in an ML workflow that occur before or after model training, such as data pre-processing or model assessment.
- I trained a Convolutional Neural Network (CNN) model for this project. Adjust the number of hidden channels in the model by adjusting the hyperparameter. Using SageMaker Experiments, make track of parameter settings and model accuracy.
- I focused on various settings for the number of hidden channels in the CNN model while training it on SageMaker. We'll set up a Trial to keep track of each training task. I also made a Trial Component out of the tracker we made before and include it in the Trial. This will add the parameters we collected during the data pre-processing step to the Trial.
- I used 5 hidden channels i.e 2, 5, 10, 20, 32 to train the model
- Each Training job defines the individual hidden channel for training.

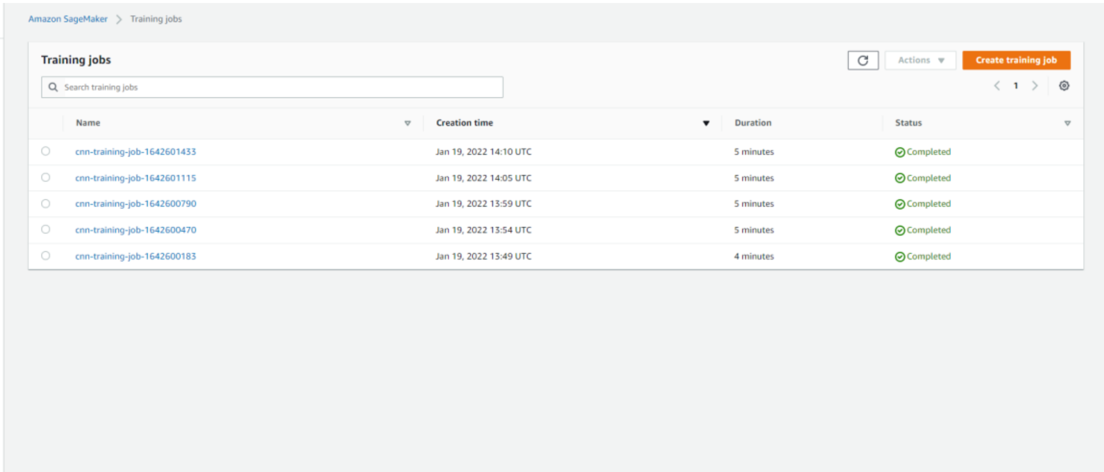

Figure 2: Training jobs

Training

- Training using python script is performed. Script mode for PyTorch is a training script format that allows you to run any PyTorch training script in SageMaker with few changes. Transferring your script to a SageMaker training instance is handled by the SageMaker Python SDK. SageMaker's native PyTorch support on the training instance sets up training-related environment variables and runs your training script. I utilised the SageMaker Python SDK to initiate a training job and deploy the trained model in this tutorial.

```
# Based on https://github.com/pytorch/examples/blob/master/mnist/main.py
lass Net(nn.Module):
   def __init__(self, hidden_channels, kernel_size, drop_out):
       super(Net, self).__init__()
       self.conv1 = nn.Conv2d(1, hidden_channels, kernel_size=kernel_size)
       self.conv2 = nn.Conv2d(hidden_channels, 20, kernel_size=kernel_size)
       self.conv2_drop = nn.Dropout2d(p=drop_out)
       self.fc1 = nn.Linear(320, 50)
       self.fc2 = nn.Linear(50, 10)

   def forward(self, x):
       x = F.relu(F.max_pool2d(self.conv1(x), 2))
       x = F.relu(F.max_pool2d(self.conv2_drop(self.conv2(x)), 2))
       x = x.view(-1, 320)
       x = F.relu(self.fc1(x))
       x = F.dropout(x, training=self.training)
       x = self.fc2(x)
       return F.log_softmax(x, dim=1)


ef _get_train_data_loader(batch_size, training_dir, is_distributed, **kwargs):
   logger.info("Get train data loader")
   dataset = datasets.MNIST(
       training_dir,
       train=True,
       transform=transforms.Compose(
           [transforms.ToTensor(), transforms.Normalize((0.1307,), (0.3081,))]
       ),
       download=False,
   )
   train_sampler = (
       torch.utils.data.distributed.DistributedSampler(dataset) if is_distributed else None
   )
   return torch.utils.data.DataLoader(
       dataset,
       batch_size=batch_size,
```

Figure 3: Python script

- The mnist.py script contains all of the code required to train and host a SageMaker model (the model fn function is used to load a model). The training script is quite similar to a training script that you could run outside of SageMaker, however you may access important training environment attributes through several environment variables, such as:

SM_MODEL_DIR: A string representing the path to the directory to write model artifacts to. These artifacts are uploaded to S3 for mode hosting.

SM_CURRENT_HOST: The name of the current container on the container network.

SM_HOSTS: JSON encoded list containing all the hosts.

SM_CHANNEL_TRAINING: A string representing the path to the directory containing data in the 'training' channel.

- The PyTorch class enables us to run our training function on SageMaker infrastructure as a training task. I needed training script, an IAM role, the number of training instances, the kind of training instance, and hyperparameters to set it up. I executed training task on ml.m5.xlarge instance in this scenario.

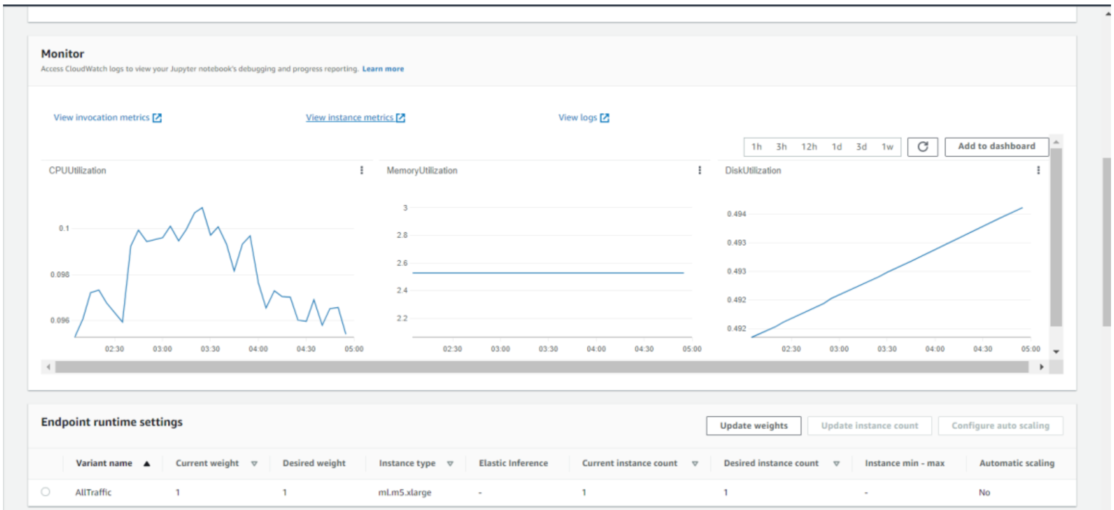

Figure 4: Endpoint monitor on ml.m5.xlarge instance

- The deployed endpoint for the training job used for prediction.

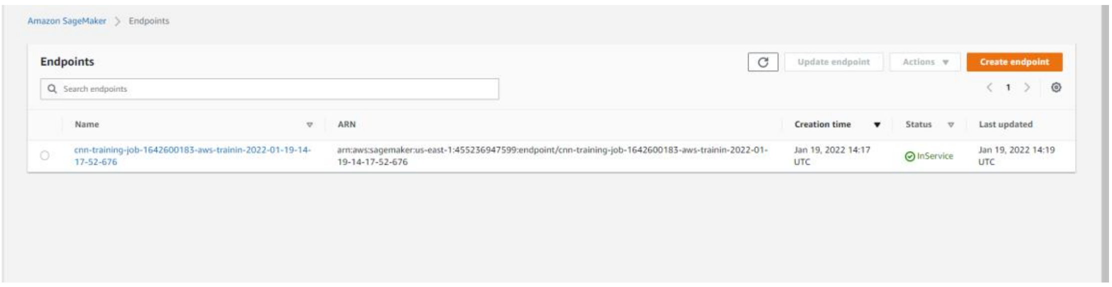

Figure 5: Endpoint

Results:

- The summary of Training and endpoint along side with CPU usage is provided in profiler report.
- Using several hidden channels, I was able to achieve ~ 97 percent accuracy for handwritten digit identification for two epochs.
- The aim of the project was to deliver the highest accuracy for handwritten digit recognition system suing Pytorch model. The tabular data confirms the highest accuracy using Pytorch model.

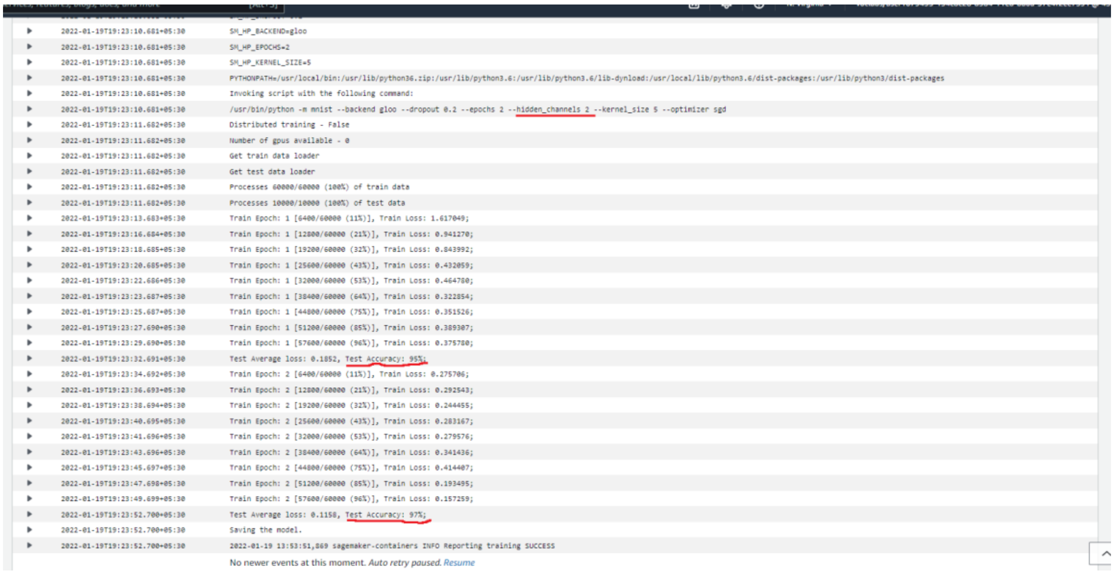

Figure 6: Training Job 1: Hidden Channel: 2, Test Accuracy of min.95% max 97%

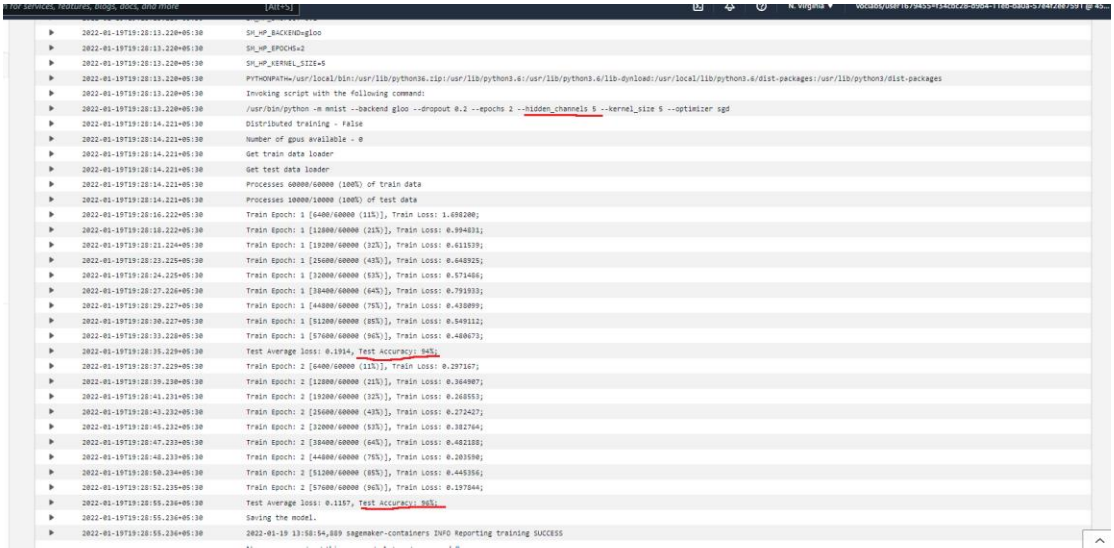

Figure 7: Training Job 2: Hidden Channel: 5, Test Accuracy of min.94% max 96%

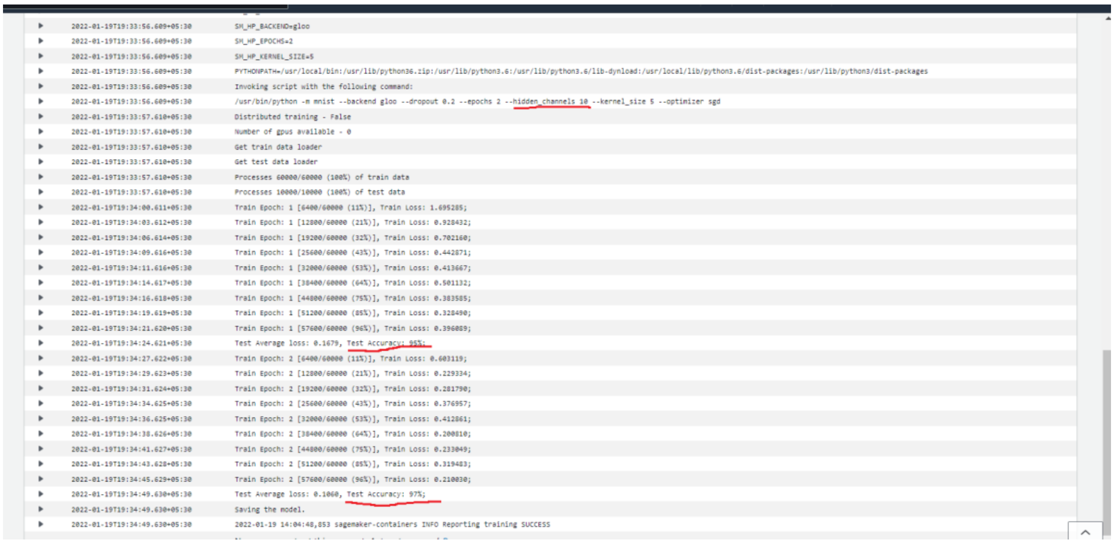

Figure 8: Training Job 3: Hidden Channel: 10, Test Accuracy of min.95% max 97%

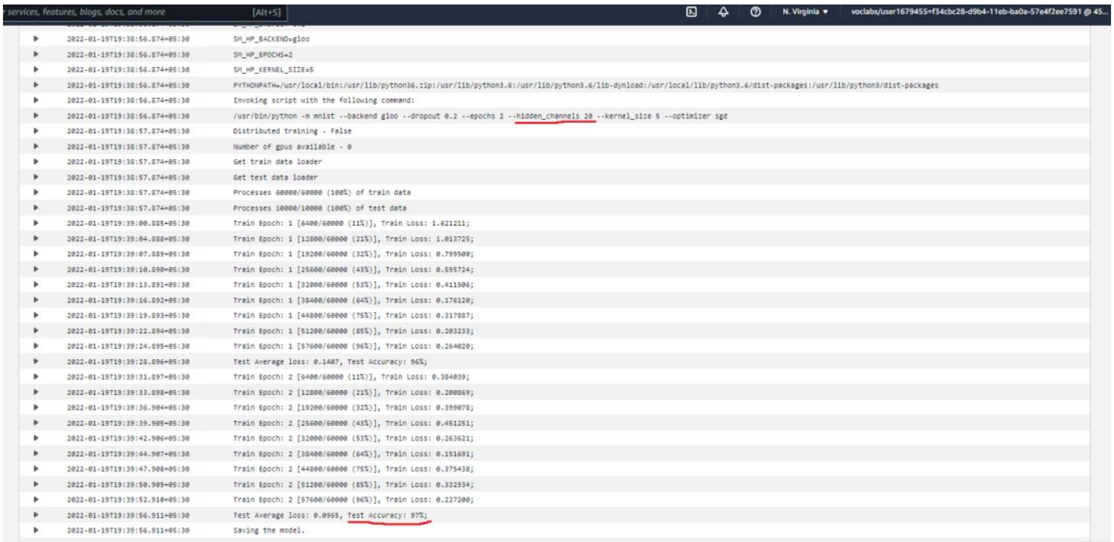

Figure 9: Training Job 4: Hidden Channel: 20, Test Accuracy of min.96% max 97%

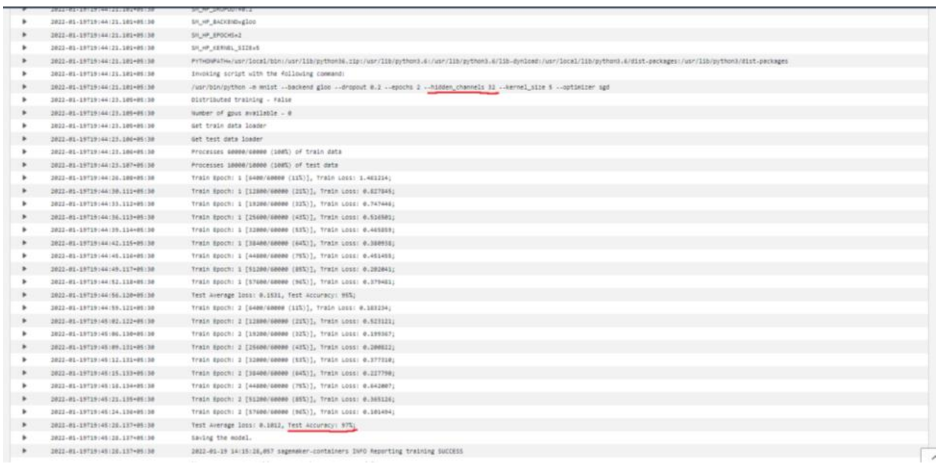

Figure 10: Training Job 5: Hidden Channel: 32, Test Accuracy of min.95% max 97%

Submitted by

Mansi Chilwant