Handwritten Digit Classification

Project Domain background

This project is based on Handwritten Digit Classification. Difference between handwritten and computerised digits can be predicted using this project. For character recognition, neural networks are commonly utilised [1, 2, 3, 4, 5, 6, 7, as well as 8]. We utilised the MNIST database to train and test a neural network classifier. Learning is a key stage in recognition; thus, I employed the gradient descent approach. The synaptic weight of the connections between the neurons is increased during the training phase modified. Attached binary neurons with no modifiable connections make up the initial layer.

A multi-layered convolutional neural network comprising one input layer, hidden layers, and one output layer is developed and shown to detect handwritten digits. The identification of handwritten digits has piqued academics' curiosity. This issue has resulted in a great number of papers and articles being published in recent years. Deep Learning methods such as multilayer CNN that use Pytorch for higher accuracy have been demonstrated in study.

The purpose of this project is to see how accurate CNN is in classifying handwritten digits using different numbers of hidden layers and epochs. The Modified National Institute of Standards and Technology (MNIST) dataset was used in this experiment to evaluate the performance of CNN.[1]

Submitted By:

Mansi Chilwant

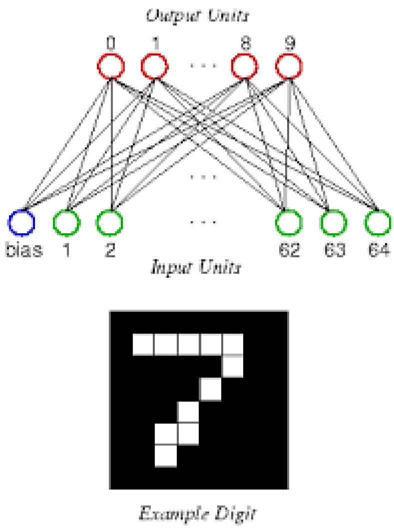

Figure 1: CNN for digit recognition

Problem Statement

Handwritten digit identification is becoming increasingly important in a variety of new applications. Computer vision and machine learning researchers utilise it extensively for practical applications like scanning computerised bank check numbers. However, putting a computerised system in place to do certain tasks is a difficult and time-consuming task. It's difficult to distinguish one person's number handwriting from another since everyone writes differently. The number of characteristics and the classifiers used have a big influence in getting the greatest possible classification accuracy.

The operation of a machine to train itself or recognise digits from various sources such as emails, bank checks, papers, images, and so on, and in various real-world scenarios such as online handwriting recognition on computer tablets or systems, recognise number plates of vehicles, process bank cheque amounts, numeric entries in forms filled out by hand (such as tax forms), and so on.

Because handwritten digits varied in size, breadth, orientation, and justification to margins from person to person, the basic difficulty would be identifying the digits owing to the similarities between digits such as 1 and 7, 5 and 6, 3 and 8, 2 and 5, 2 and 7, and so on. This issue arises more frequently when a single digit is written by a large number of people in a diversity of handwriting styles. Finally, the construction and look of the digits are influenced by the individual's handwriting's originality and variation. Deep learning and machine learning ideas and methods are now introduced.

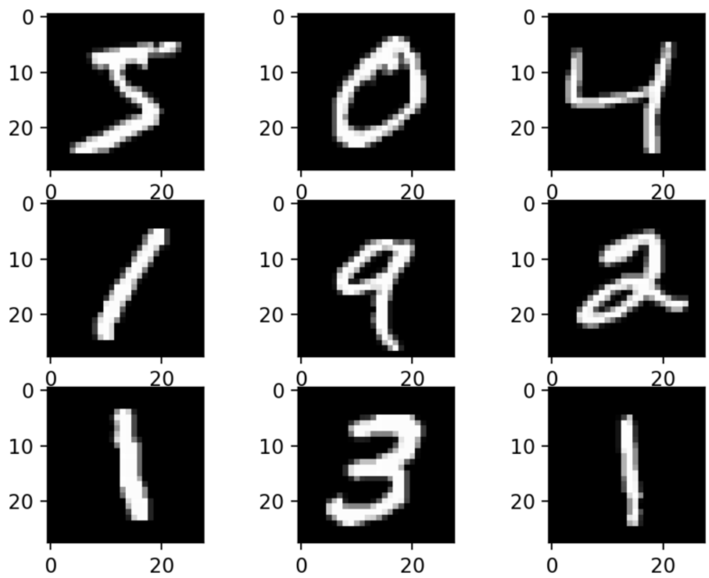

Figure 2: Example Handwritten Digit detection using machine learning

Submitted By:

Mansi Chilwant

Datasets and Inputs

MNIST is a widely used dataset for handwritten digit classification. It consists of 70,000 labeled 28x28 pixel grayscale images of hand-written digits. The dataset is split into 60,000 training images and 10,000 test images. There are 10 classes (one for each of the 10 digits). This tutorial will show how to train and test an MNIST model on SageMaker using PyTorch.

I entered AWS through the gateway in the course and open SageMaker Studio. Downloaded and made the dataset available.

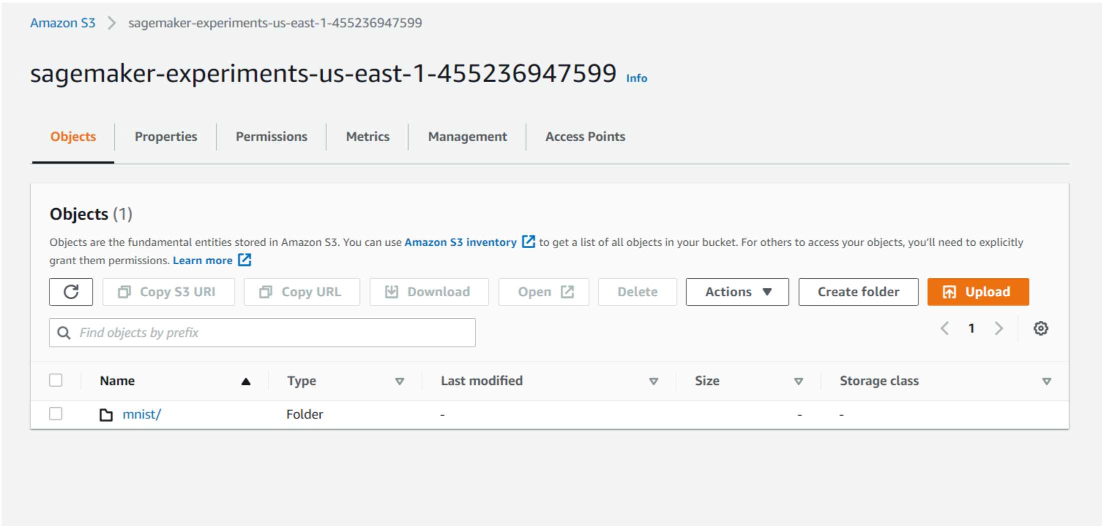

Figure 3: S3 Bucket Setup

Data exploration and visualisation:

After the data is uploaded to the s3 bucket, I normalise the data into the processable format.

Figure 4: Data exploration

Solution Statement

We will be using sagemaker "Script mode". Script mode for PyTorch is a training script format that allows you to run any PyTorch training script in SageMaker. This example can be ran on one or multiple, cpu or gpu instances. The hyperparameters parameter is a dict of values that will be passed to the training script -- you can see how to access these values in the mnist.py script above.

After we've constructed my PyTorch object, we can fit it using the data we uploaded to S3. SageMaker makes sure my data is available in the local filesystem of each worker, so my training script can simply read the data from disk.

Submitted By:

Mansi Chilwant

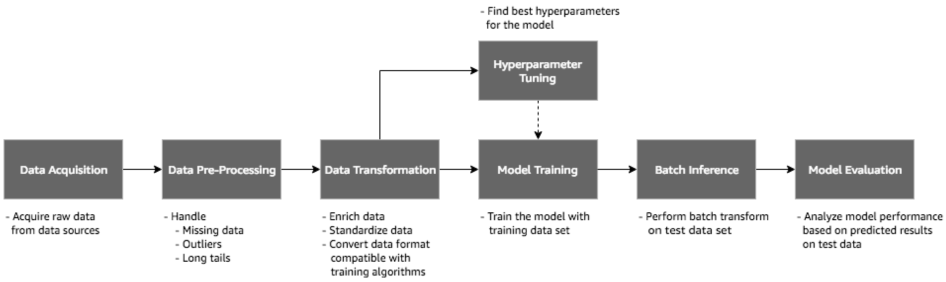

Figure 5: Amazon sagemaker machine learning workflow

Benchmark Model:

The handwritten digit recognition following article. The author came up with some good analysis for using MNIST dataset for detection of handwritten digits.

1. Chen, Feiyang, et al. "Assessing four neural networks on handwritten digit recognition dataset (MNIST)." *arXiv preprint arXiv:1811.08278* (2018).

This project is a try to determine Handwritten Digit Images using the knowledge of the article.

Evaluation Metrics

Since this is detection/classification problem, the accuracy of the analysis and detection of handwritten images will be considered for evaluation purpose.

Submitted By:

Mansi Chilwant

Project Design

The project's concept is simple: initially, we'll develop a Sagemaker notebook instance that performs quite well. Download the dataset to the instance's environment, then upload to an S3 bucket for training purposes. Because the dataset is clean and includes manifest files. After that, we'll tune the model using the best available hyperparameters using Sagemaker tuner, and after that's done, we'll train the model on a GPU enabled instance using Sagemaker estimator. The model will then be deployed to a Sagemaker instance, and lambdas functions with proper permissions will be created to conveniently use the model.

These are steps will be followed.

- Create the SageMaker Session
- Training Data
- Train with SageMaker PyTorch Estimator
- Perform Batch Predictions

Pre-processing:

- 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.

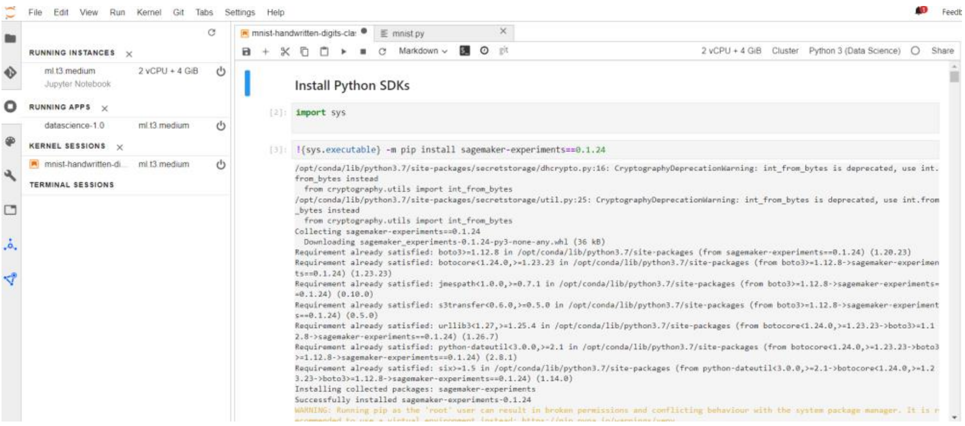

Figure 6: Python 3 (Data science - 01) Kernel

- 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.

Submitted By:

Mansi Chilwant

Training code:

- 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.
- The downloaded dataset is transformed and normalized with the values

```
{ "normalization_mean": 0.1307, "normalization_std": 0.3081, }
```

```
In [11]:
with Tracker.create(display_name="Preprocessing", sagemaker_boto_client=sm) as tracker:
    tracker.log_parameters(
        {
            "normalization_mean": 0.1307,
            "normalization_std": 0.3081,
        }
    )
    # we can log the s3 uri to the dataset we just uploaded
    tracker.log_input(name="mnist-dataset", media_type="s3/uri", value=inputs)
```

Figure 7: Normalization parameters

- After normalizing the dataset is ready to be trained

```
    )
    # we can log the s3 uri to the dataset we just uploaded
    tracker.log_input(name="mnist-dataset", media_type="s3/uri", value=inputs)
```

Step 1 - Set up

Create training

```
In [13]:
mnist_file = Experiment.create(
    experiment_name=f"mnist-hand-written-digits-classification-{int(time.time())}",
    description="Classification of mnist hand-written digits",
    sagemaker_boto_client=sm,
)
print(mnist_file)

Experiment(sagemaker_boto_client=<botocore.client.SageMaker object at 0x7f64028aa210>,experiment_name='mnist-hand-written-digits-classification-1642600126',description='Classification of mnist hand-written digits',tags=None,experiment_arn='arn:aws:sagemaker:us-east-1:455236947599:experiment/mnist-hand-written-digits-classification-1642600126',response_metadata={'RequestId': '12474e06-18fa-4f08-ab5d-1fe976620e6d', 'HTTPStatusCode': 200, 'HTTPHeaders': {'x-amzn-requestid': '12474e06-18fa-4f08-ab5d-1fe976620e6d', 'content-type': 'application/x-amz-json-1.1', 'content-length': '123', 'date': 'Wed, 19 Jan 2022 13:48:46 GMT'}, 'RetryAttempts': 0})
```

Figure 8: Dataset is ready to be trained

- Transferring the 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 the training script.
- The images are trained against these hyperparameters and hidden channel values:
  I utilized 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
class 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)


def _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 9: 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:

```
In [18]:
for i, num_hidden_channel in enumerate([2, 5, 10, 20, 32]):
    # create trial
    trial_name = f"cnn-training-job-{num_hidden_channel}-hidden-channels-{int(time.time())}"
    cnn_trial = Trial.create(
        trial_name=trial_name,
        experiment_name=mnist_file.experiment_name,
        sagemaker_boto_client=sm,
    )
    hidden_channel_trial_name_map[num_hidden_channel] = trial_name


    cnn_trial.add_trial_component(preprocessing_trial_component)


    estimator = PyTorch(
        py_version="py3",
        entry_point="./mnist.py",
        role=role,
        sagemaker_session=sagemaker.Session(sagemaker_client=sm),
        framework_version="1.1.0",
        instance_count=1,
        instance_type="ml.c4.xlarge",
        hyperparameters={
            "epochs": 2,
            "backend": "gloo",
            "hidden_channels": num_hidden_channel,
            "dropout": 0.2,
            "kernel_size": 5,
            "optimizer": "sgd",
        },
        metric_definitions=[
            {"Name": "train:loss", "Regex": "Train Loss: (.*?);"},
            {"Name": "test:loss", "Regex": "Test Average loss: (.*?),"},
            {"Name": "test:accuracy", "Regex": "Test Accuracy: (.*?)%;"},
        ],
        enable_sagemaker_metrics=True,
    )

    cnn_training_job_name = "cnn-training-job-{}".format(int(time.time()))


    estimator.fit(
        inputs={"training": inputs},
        job_name=cnn_training_job_name,
        experiment_config={
            "TrialName": cnn_trial.trial_name,
            "TrialComponentDisplayName": "Training",
        },
        wait=True,
    )


    time.sleep(2)
```

Figure 10: Code that show hyperparameters and hidden channel values

- 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_HOSTS: JSON encoded list containing all the hosts.

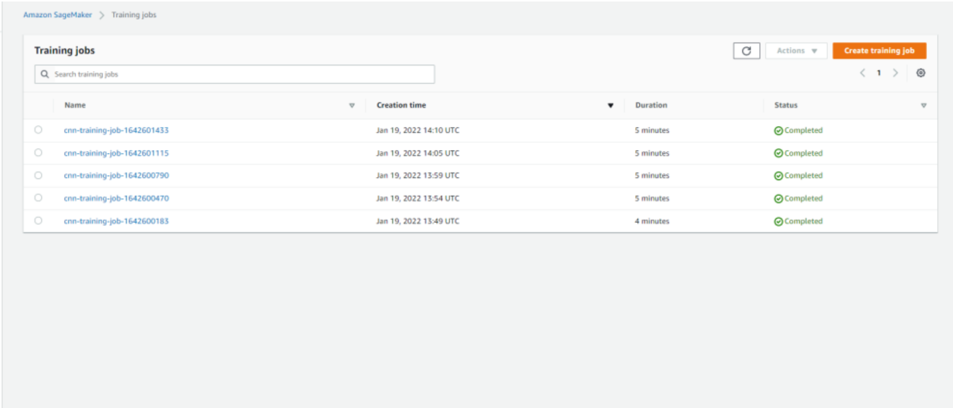

Figure 11: Training Jobs

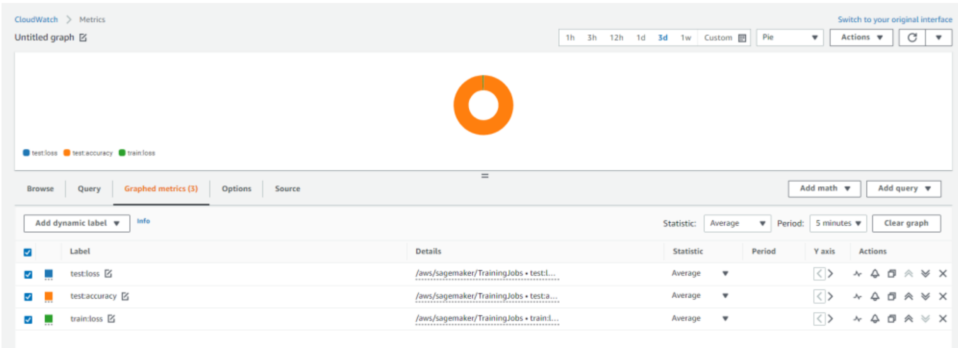

Figure 12: Training Jobs CloudWatch validation metrics

Submitted By:

Mansi Chilwant

Deploying endpoints:

- The PyTorch class enables us to run 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.

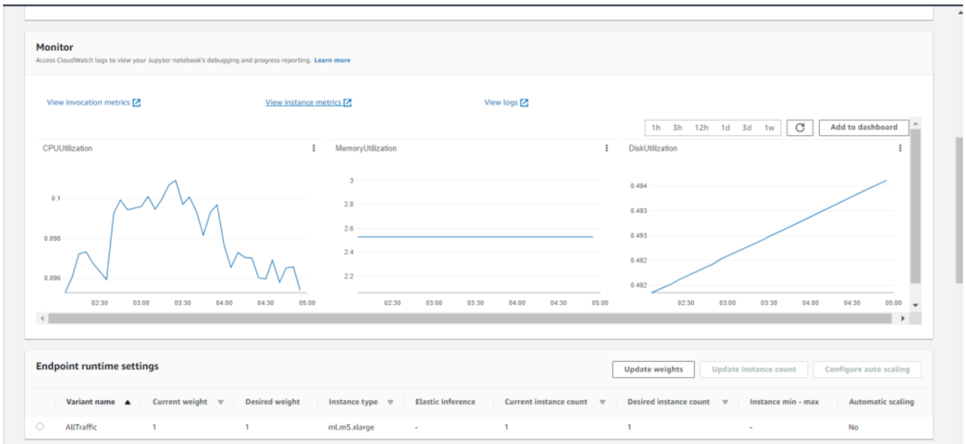

Figure 13: Deployed Endpoint CloudWatch validation metrics

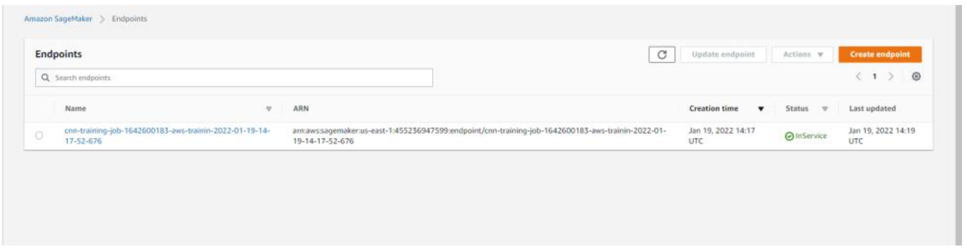

Figure 14: Endpoint

Submitted By:

Mansi Chilwant

Evaluations:

- The summary of Training and endpoint alongside 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.

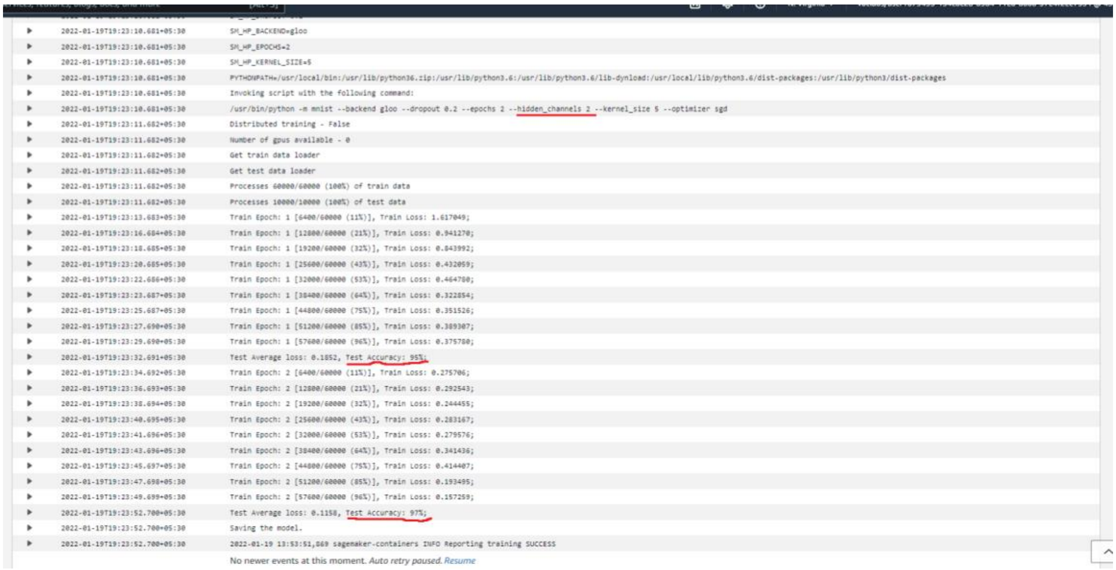

Figure 15 Training Job 1: Hidden Channel: 2, Test Accuracy of min.95% max 97%

Submitted By:

Mansi Chilwant

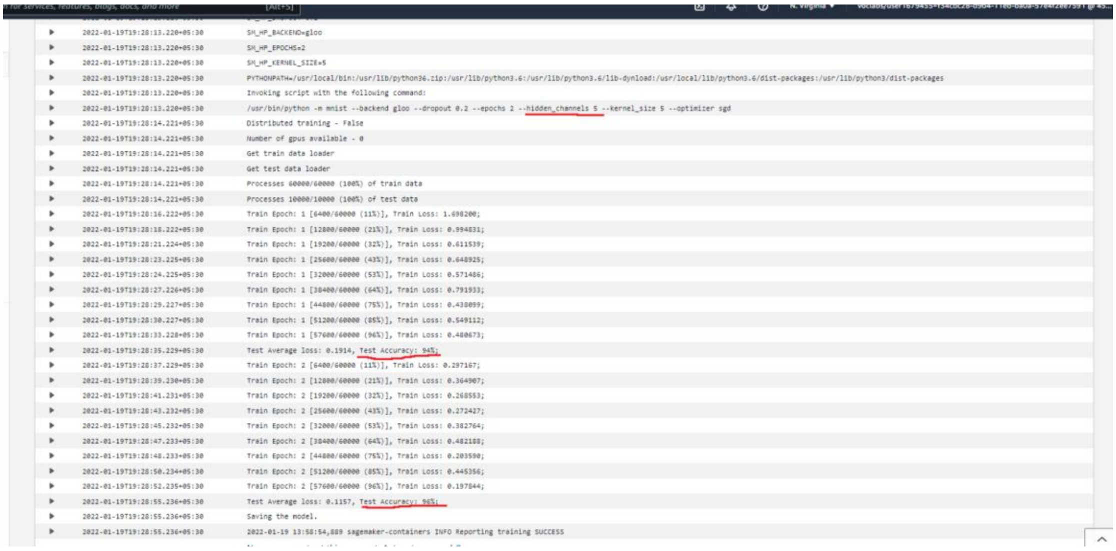

Figure 16: Training Job 2: Hidden Channel: 5, Test Accuracy of min.94% max 96%

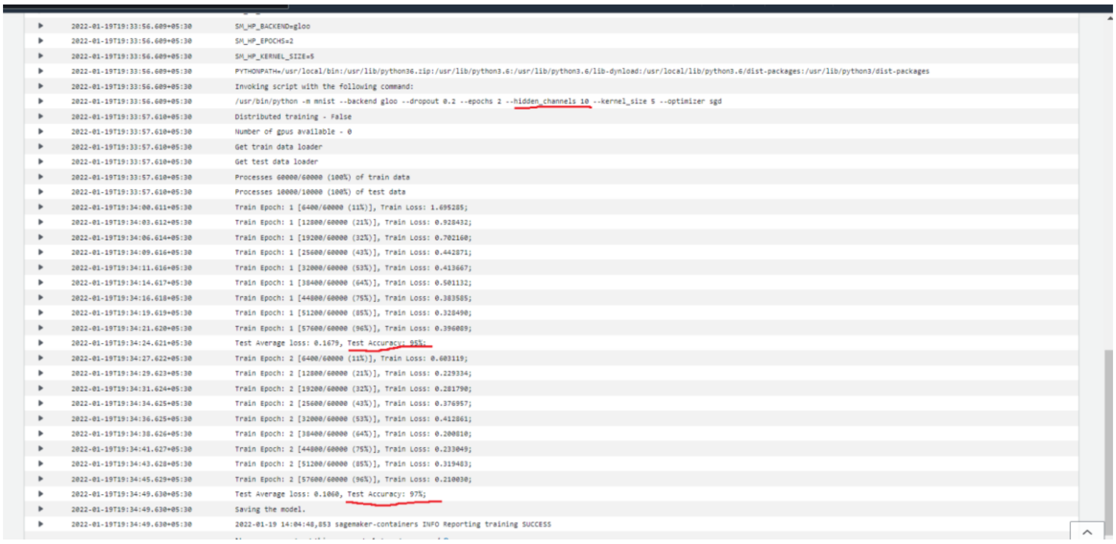

Figure 17: Training Job 3: Hidden Channel: 10, Test Accuracy of min.95% max 97%

Submitted By:

Mansi Chilwant

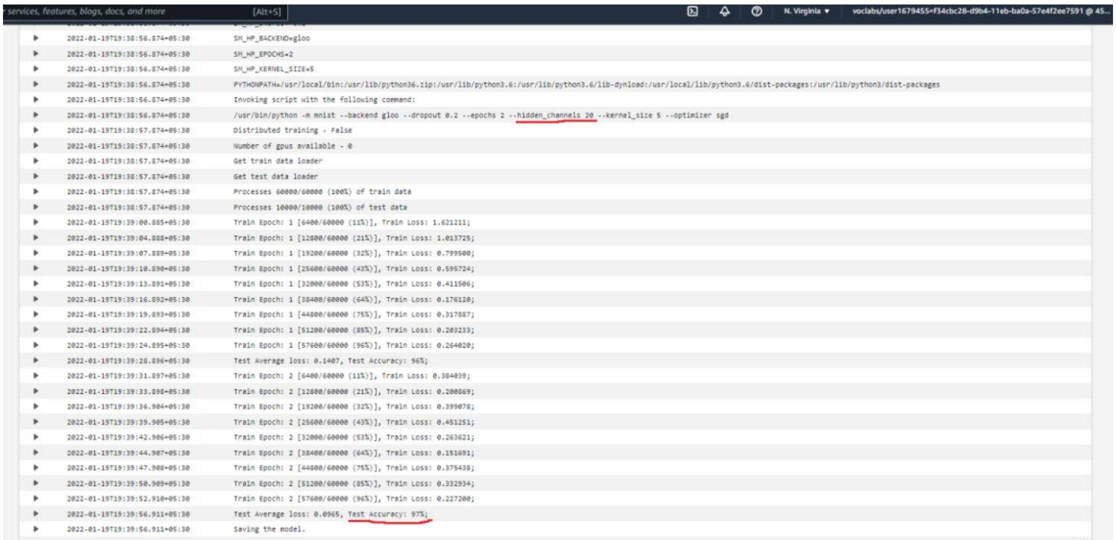

Figure 18: Training Job 4: Hidden Channel: 20, Test Accuracy of min.96% max 97%

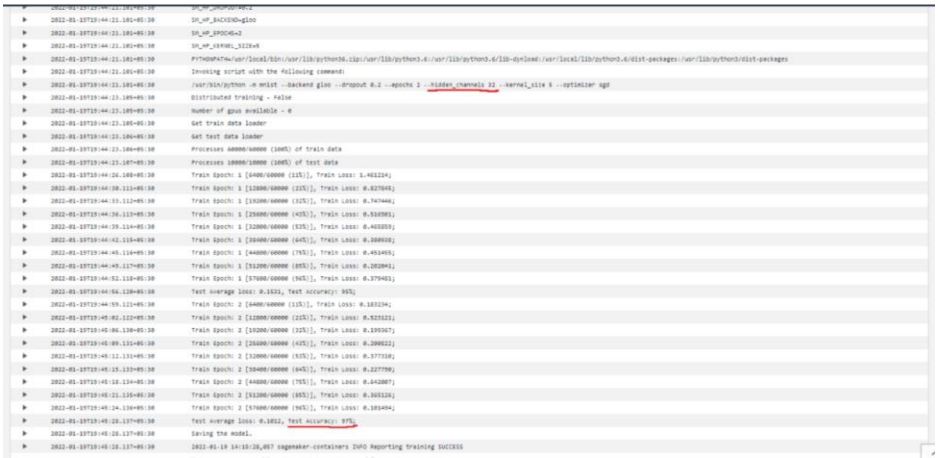

Figure 19: Training Job 5: Hidden Channel: 32, Test Accuracy of min.95% max 97%

Submitted By:

Mansi Chilwant

Justification:

- The aim of the project was to acquire high accuracy in handwritten digit recognition using Convolution neural network (CNN) at AWS platform.
- The project is justified against the benchmark model i.e., ***Chen, Feiyang, et al. "Assessing four neural networks on handwritten digit recognition dataset (MNIST)." arXiv preprint arXiv:1811.08278 (2018).***
- The purpose of this benchmark study was to see which models performed better across MNIST datasets that were partitioned. On the MNIST dataset, we examined four models with different divisions and found that CapsNet performed the best across all datasets.The MNIST dataset was split into four groups: 25%, 50%, 75%, and 100% by the author at random. Table 1 displays the findings of the four models over all split datasets

Accuracy(%) MNIST / Models	25%	50%	75%	100%
CNN	80.73	86.73	91.23	98.32
ResNet	79.46	90.55	93.78	99.16
DenseNet	82.57	89.24	94.20	99.37
CapsNet	87.68	97.12	98.79	99.75

Table1: Benchmark Results display

- The performed project described crating and implementation of Pytorch model in AWS platform. This mode used the python script **mnist.py** to perform the training operation on processed the MNIST dataset. I carried out 5 training jobs with different hidden channel. The following table describe the accuracy for the training job performed.

Submitted By:

Mansi Chilwant

	TrialComponentName	DisplayName	SourceArn	dropout	epochs	hidden_channels	optimizer	test:accuracy - Min	test:accuracy - Max	test:accuracy - Avg	...	test:accuracy - Last	te
0	cnn-training-job-1642600183-aws-training-job	Training	arn:aws:sagemaker:us-east-1:455236947599:train...	0.2	2.0	2.0	"sgd"	95.0	97.0	96.0	...	97.0	
1	cnn-training-job-1642600790-aws-training-job	Training	arn:aws:sagemaker:us-east-1:455236947599:train...	0.2	2.0	10.0	"sgd"	95.0	97.0	96.0	...	97.0	
2	cnn-training-job-1642601115-aws-training-job	Training	arn:aws:sagemaker:us-east-1:455236947599:train...	0.2	2.0	20.0	"sgd"	96.0	97.0	96.5	...	97.0	
3	cnn-training-job-1642601433-aws-training-job	Training	arn:aws:sagemaker:us-east-1:455236947599:train...	0.2	2.0	32.0	"sgd"	95.0	97.0	96.0	...	97.0	
4	cnn-training-job-1642600470-aws-training-job	Training	arn:aws:sagemaker:us-east-1:455236947599:train...	0.2	2.0	5.0	"sgd"	94.0	96.0	95.0	...	96.0	

Table 2: Project Results display

Reference:

1. Recognition of Handwritten Digit using Convolutional Neural Network in Python with Tensorflow and Comparison of Performance for Various Hidden Layers: Fathma Siddique1# , Shadman Sakib2* , Md. Abu Bakr Siddique2$ 1 Department of CSE, International University of Business Agriculture and Technology, Dhaka 1230, Bangladesh 2 Department of EEE, International University of Business Agriculture and Technology, Dhaka 1230, Bangladesh siddiquefathma@gmail.com# , sakibshadman15@gmail.com* , absiddique@iubat.edu

Submitted By:

Mansi Chilwant