Homework 2: Convolutional Neural Network

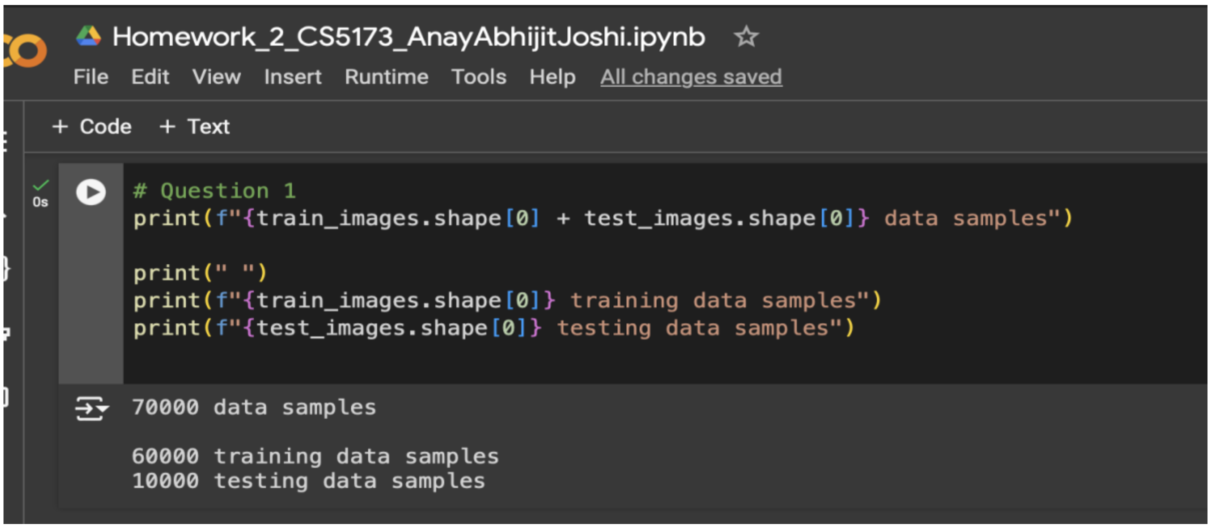
Deadline: 9-30-2023

Name: Anay Abhijit Joshi

Step 1: Data

1) How many data samples are included in the dataset?

The MNIST dataset contains 70,000 images of handwritten digits. This includes 60,000 images for training and 10,000 for testing, with each image sized at 28x28 pixels.


Homework_2_CS5173_AnayAbhijitJoshi.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text

```
# Question 1
print(f"{train_images.shape[0] + test_images.shape[0]} data samples")

print(" ")
print(f"{train_images.shape[0]} training data samples")
print(f"{test_images.shape[0]} testing data samples")
```

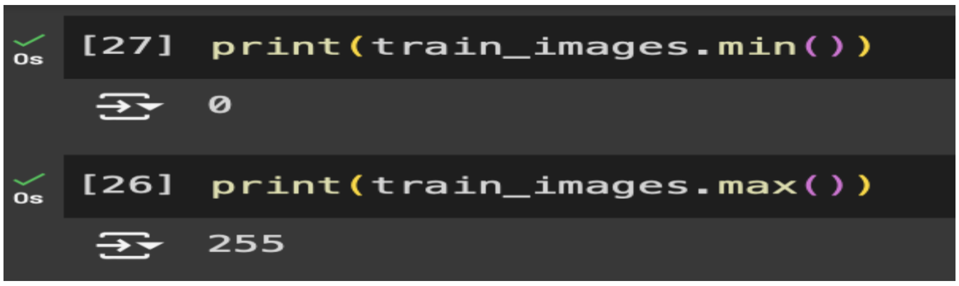
```
70000 data samples

60000 training data samples
10000 testing data samples
```

2) Which problem will this dataset try to address?

The MNIST dataset addresses the classification problem of recognizing handwritten digits from 0 to 9. In my opinion, the main objective of this dataset is to assign the correct label to each image that represents a particular/specific digit, ranging from 0 to 9.

3) What is the minimum value and the maximum value in the dataset?

The pixel values in the given, original dataset range from 0 to 255, where 0 represents black and 255 represents white. However, in my implementation, I normalized these values to fall between 0 and 1 by dividing by 255.0, by normalizing this dataset.

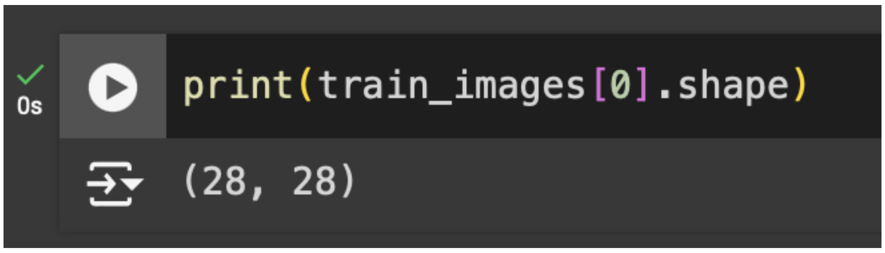


```
[27] print(train_images.min())
0
```

```
[26] print(train_images.max())
255
```

4) What is the dimension of each data sample?

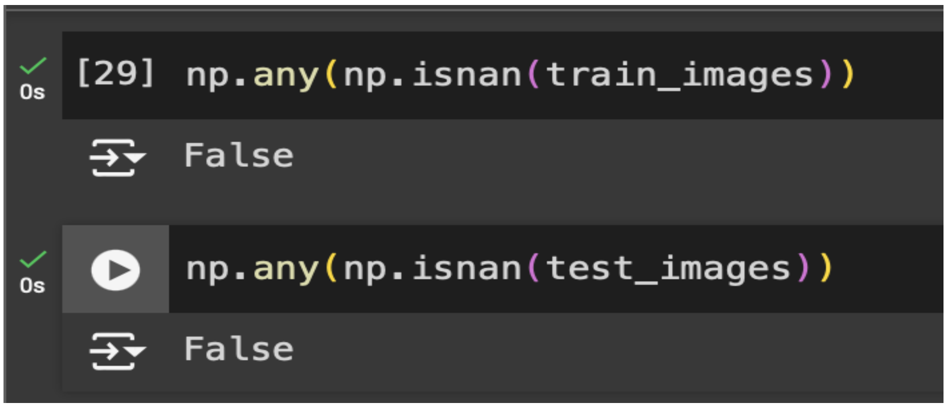
Each sample is a grayscale image with a size of 28x28 pixels. Therefore, this makes the dimension of each data sample to be 28x28.



```
print(train_images[0].shape)
(28, 28)
```

5) Does the dataset have any missing information? E.g., missing features.

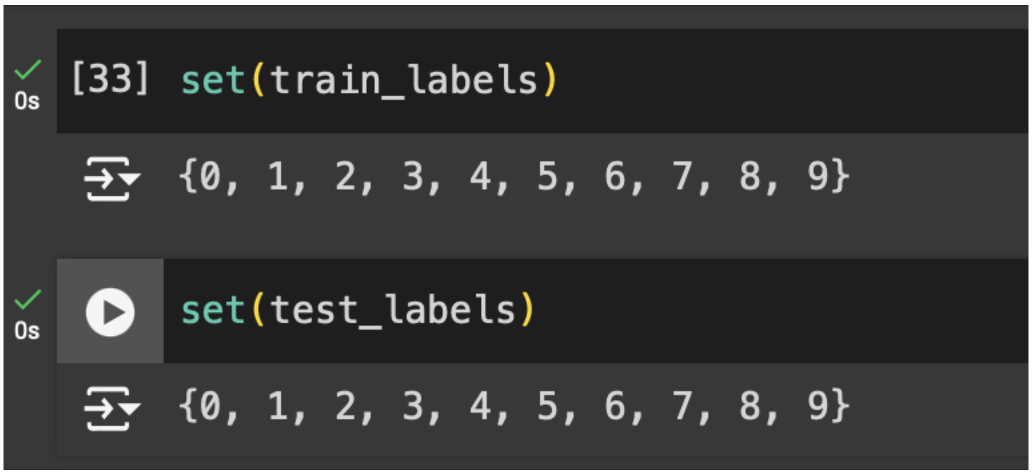
The MNIST dataset is well-prepared and clean, which means it contains no missing values or features. This makes it almost complete.



```
[29] np.any(np.isnan(train_images))
False
np.any(np.isnan(test_images))
False
```

6) What is the label of this dataset?

The label(s) for each image in the dataset are integers (or digital numbers) ranging from 0 to 9, representing the digit shown in the image. In my code, labels were one-hot encoded into vectors of length 10 (i.e., digital numbers ranging from 0 to 9) for training purposes, where 1 indicates the correct digit's position and all other entries are 0.



```
[33] set(train_labels)
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
set(test_labels)
{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
```

7) How many percent of data will you use for training, validation, and testing?

The MNIST dataset is split as follows –

Training set:

The first 60,000 images are used for training machine learning models.

Testing set:

The next 10,000 images are reserved for testing the trained model's performance.

So, after doing calculations, the percent of data for training, validation, and testing would be around 68.57%, 17.14%, and 14.29%, respectively.

8) What kind of data pre-processing will you use for your training dataset?

The data pre-processing steps which I used are:

Normalization:

The pixel values are scaled to be between 0 and 1 by dividing by 255.0

One-hot encoding:

The labels are converted into vectors of length 10, with 1 at the position corresponding to the digit and 0s elsewhere.

Splitting the dataset:

The training data is split into smaller training and validation sets to monitor and evaluate the model's performance during training.

Step 2: Model

Model	**Accuracy**
DNN	0.9764999747276306
ConvNet	0.9911999702453613
ResNet	0.9894999861717224

Step 3: Objective

Cross-entropy is the loss function which I used to train my models.

Step 4: Optimization

Adam Optimizer

For my models, including the DNN, ConvNet, and ResNet, I opted for the Adam Optimizer because it offers a good balance of speed and stability during training. Adam optimizer's adaptive learning rates and use of momentum help handle complex optimization challenges efficiently, without extensive parameter tuning. This is particularly beneficial for deep networks like ResNet, where the architecture involves multiple residual connections, and the risk of vanishing gradients is high. In my opinion, by fine-tuning learning rates individually for each parameter, Adam optimizer ensures faster convergence and helps models (used in Homework 2 like DNN, ConvNet, and ResNet) leverage their architectures effectively, leading to improved accuracy and robust training outcomes. Additionally, Adam optimizer 's ease of use reduces the manual intervention required, making it ideal for this Homework 2, where balancing performance and development efficiency is the main key...

Step 5: Model selection (Accuracy)

Model	LR: 0.1	LR: 0.01	LR: 0.001	LR: 0.0001
DNN	0.955299973487854	0.97079998254776	0.9771000146865845	0.972000002861023
ConvNet	0.9848999977111816	0.9879999756813049	0.9900000095367432	0.9872000217437744
ResNet	0.9815999865531921	0.9865999817848206	0.9753000140190125	0.9620000123977661

Have you experienced different learning rates? Any other learning rate you would like to try?

Yes! The other learning rates which I tried were **0.001** and **0.0001**.

Have your tried other learning rate technique? How do you avoid overfit your model and underfit your model?

For this Homework 2, I used the Adam Optimizer with manually adjusted learning rates, given in the Homework 2 as well some extra learning rates based on Homework 1, to optimize all the models' training process. So, yes, I did apply the other learning rate technique(s) by training all the models with the learning rates of 0.1, 0.01, 0.001 and 0.0001.

To prevent overfitting, I implemented Early Stopping, which stops training when the validation loss plateaus, preventing the model from learning noise in the data. I also used Dropout layers to randomly ignore certain neurons during training, making the model less likely to over-rely on specific paths. Additionally, Batch Normalization was applied to stabilize learning by normalizing the activations between layers.

To avoid underfitting, I ensured sufficient model complexity by using deeper architectures like ResNet, which was already there in Homework 2's requirements, and carefully tuning the learning rate and model parameters to allow effective learning from the dataset...

Step 6: Model performance

Model selection (**F1**)

Model	LR: 0.1	LR: 0.01	LR: 0.001	LR: 0.0001
DNN	0.0167505749224311	0.9493259760634908	0.9771843718331129	0.9701691970424812
ConvNet	0.019307904278462656	0.9825019356415203	0.9906026713630974	0.9850997169258364
ResNet	0.017561901293024953	0.9780083887827903	0.9727081886321793	0.9914990542178667

Model selection (**AUC**)

Model	LR: 0.1	LR: 0.01	LR: 0.001	LR: 0.0001
DNN	0.5	0.9963753735755153	0.9995948984174378	0.9992988676669533
ConvNet	0.5	0.9997565117928378	0.9999461671117373	0.9998268812563488
ResNet	0.5	0.9992653940781467	0.9993149882703417	0.9998393521931469

All the ***Model Training Plots and AUC-ROC Curves*** are attached in my **GitHub Repository**

Please access them here – **Model(s) Training Plots**

Based on my training experience in Homework 2, and the score tables of F1, AUC, and Accuracy, the **Best Model** is "**ConvNet** with **Learning Rate** of **0.001**". However, if we just consider the models and learning rates (0.1 and 0.01 only) given in Homework 2, then the **Best Model** is "**ConvNet** with **Learning Rate** of **0.01**".

This model performs the best amongst all other models with respect to the accuracy, F1, and AUC, and their respective plots.

Finally, I also observed from the above tables, that all models are performing the best for a Learning Rate of 0.001.