MIDAS Task 2

Saloni Parekh

This document contains the methodology, results and conclusions for this project.

A. Problem Statement

1. Use this dataset (https://www.dropbox.com/s/pan6mutc5xj5kj0/trainPart1.zip) to train a CNN. Use no other data source or pretrained networks, and explain your design choices during preprocessing, model building and training. Also, cite the sources you used to borrow techniques. A test set will be provided later to judge the performance of your classifier. Please save your model checkpoints.
2. Next, select only 0-9 training images from the above dataset, and use the pretrained network to train on MNIST dataset. Use the standard MNIST train and test splits (http://yann.lecun.com/exdb/mnist/). How does this pretrained network perform in comparison to a randomly initialized network in terms of convergence time, final accuracy and other possible training quality metrics? Do a thorough analysis. Please save your model checkpoints.
3. Finally, take the following dataset (https://www.dropbox.com/s/otc12z2w7f7xm8z/mnistTask3.zip), train on this dataset and provide test accuracy on the MNIST test set, using the same test split from part 2. Train using scratch random initialization and using the pretrained network part 1. Do the same analysis as 2 and report what happens this time. Try and do qualitative analysis of what's different in this dataset. Please save your model checkpoints.

Please refer to the following pages

B. Methodology

This section will explain the architecture used in this project. The idea is to chain a convolutional neural network (CNN), which extracts the local information from the image, with a Transformer encoder architecture, which reasons about the image as a whole.
Traditionally, classification problems in the computer vision domain are solved using CNNs and fully connected layers(FCs). While this methodology has proven to be quite effective, these problems could use “attention” or context within the image to predict effectively.
An example of context,

In this image, the two curves identified by the red lines are essential to understand that it is the number 3. Although the working is different, this example was given just to explain the intuition.
Therefore, the CNN is followed by a 3-layer Transformer encoder block. The embeddings received from the encoder block are directly used to predict the probabilities of the classes via a FC.

Once the features are extracted by the CNN, it along with the positional embeddings is passed to the encoder. Now although the features can be directly used for classification, passing them to the encoder allows the model to develop much better features. It is so because the encoder is able to understand the relevant features for that particular image because of the concept explained above. The results in the below section support these statements.

Therefore, the architecture is as follows:

```
1. Conv(1, 32) -> ReLU -> Dropout -> Maxpool
2. Conv(32, 64) -> ReLU -> Dropout -> Maxpool
3. Conv(64, 128) -> ReLU -> Dropout -> Maxpool
4. Conv(128, 256) -> ReLU -> Dropout -> Maxpool
5. Conv(256, 512) -> ReLU -> Dropout -> Maxpool
6. Transformer Encoder Layer            7. Transformer Encoder Layer
8. Transformer Encoder Layer            9. Fully Connected Layer (512, 10)
```

C. Training Details and Implementation

The entire project has been implemented in PyTorch on Google Colab with the GPU facility.

All models have been trained using the CrossEntropyLoss and Adam optimizer, with a learning rate of 0.0001. Details for the individual points is given below:

1. Image Classification problem with 62 classes

Use this dataset (https://www.dropbox.com/s/pan6mutc5xj5kj0/trainPart1.zip) to train a CNN. Use no other data source or pretrained networks, and explain your design choices during preprocessing, model building and training. Also, cite the sources you used to borrow techniques. A test set will be provided later to judge the performance of your classifier. Please save your model checkpoints.

Data: 2480 samples Training Data: 1860 samples Validation Data: 620 samples

Data Preprocessing:
The images are resized to 200 * 200 and normalized. As for augmentations, rotations are applied.

Model Architecture:
The explanation for choosing this particular architecture is given above.
Here, the model A is the same as described above except that the last FC outputs the probability for 62 classes.

Training:
After multiple trials, the learning rate was chosen to be 10^-4. The training makes use of CrossEntropyLoss and Adam optimizer.

2. Image Classification problem with 10 classes using MNIST.

Next, select only 0-9 training images from the above dataset, and use the pretrained network to train on MNIST dataset. Use the standard MNIST train and test splits (http://yann.lecun.com/exdb/mnist/). How does this pretrained network perform in comparison to a randomly initialized network in terms of convergence time, final accuracy and other possible training quality metrics? Do a thorough analysis. Please save your model checkpoints.

0-9 Labels

Data: 360 samples	Training Data: 270 samples	Validation Data: 90 samples

MNIST

Data: 60000 samp.	Training Data: 50000 samp.	Validation Data: 10000 samp

This point was divided into 2 stages:

a. Model B was pretrained on images labelled 0-9. These images were taken from the previous point's folder. This model's last fc was reset and the model was trained again on the MNIST dataset.
b. Model C was trained from scratch on the MNIST dataset.

3. Image Classification problem with 10 classes

Finally, take the following dataset (https://www.dropbox.com/s/otc12z2w7f7xm8z/mnistTask3.zip), train on this dataset and provide test accuracy on the MNIST test set, using the same test split from part 2. Train using scratch random initialization and using the pretrained network part 1. Do the same analysis as 2 and report what happens this time. Try and do qualitative analysis of what's different in this dataset. Please save your model checkpoints.

The labels for the images in this point were created by using Model C. Model C predicted the labels for the images from https://www.dropbox.com/s/otc12z2w7f7xm8z/mnistTask3.zip?dl=0. This model was used because of its high accuracy on the test data.

Data: 60000 samp.	Training Data: 45000 samp.	Validation Data: 15000 samp

This point was divided into 2 stages:

a. Here Model A’s last fc was reset and the model was trained again on the images specified for this folder (Model D). The model was tested on MNIST test set.
b. Model E was trained from scratch on the images specified for this folder. The model was tested on MNIST test set.

- Image Classification problem of 62 classes

To compare why the Transformer encoder is essential, experiments were also performed on a Model F without the encoder layers. Therefore, the model architecture was:

1. Conv(1, 32) -> ReLU -> Dropout -> Maxpool
2. Conv(32, 64) -> ReLU -> Dropout -> Maxpool
3. Conv(64, 128) -> ReLU -> Dropout -> Maxpool
4. AveragePooling
5. Fully Connected Layer (128, 62)

Data: 2480 samples   Training Data: 1860 samples   Validation Data: 620 samples

This model was used for comparison sakes only.

D. Results

1. Overall Results

	Data Split (Train/Valid)	# of Epochs	Convergence Time	Train Acc (%)	Valid Acc (%)	Test Acc (%)
Model F	1860/620	100	-	88.65	75.64	-
Model A	1860/620	30	16.96 min	93.99	77.90	
Model B	50000/10000	20	1hr 43 min	99.85	99.15	99.15
Model C	50000/10000	20	1hr 40 min	99.64	98.71	99.00
Model D	45000/15000	20	34 min	99.06	99.07	99.00
Model E	45000/15000	20	42 min	99.39	99.23	99.00

Conclusion:

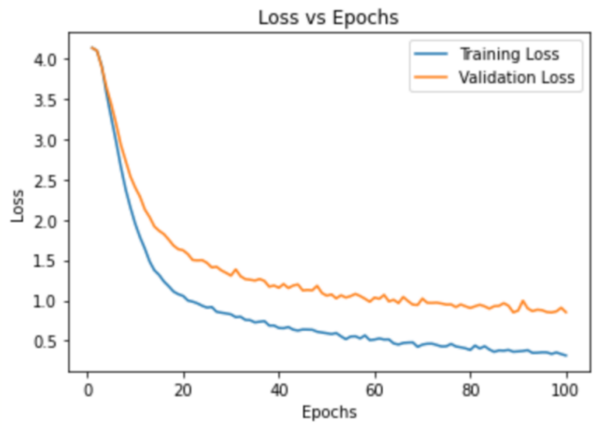
Model F, which is the model without the encoders takes a longer time to achieve a decent accuracy of 88.65 % on the training data and 75.64% on the validation data. Whereas, all the other models which make use of encoders, converge much faster (20 epochs only).
This therefore intensifies the intuition behind using encoders.

Not only did the models converge much faster but they all gave higher accuracies.

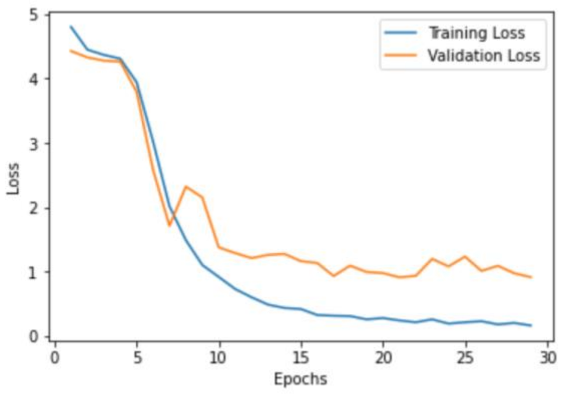
2. Without Encoders vs. With Encoder
   (Model F vs Model A)

	# of Epochs	Train Acc (%)	Valid Acc(%)
Model F	100	88.65	75.64
Model A	20	93.99	77.90

Although the values of training accuracy and validation accuracy are somewhat comparable, the major difference is the number of epochs. It has converged in much lesser epochs than without encoders. This is the major reason that supports the use of encoders in this project.



Without Encoders



With Encoders

3. Retraining vs Training from Scratch

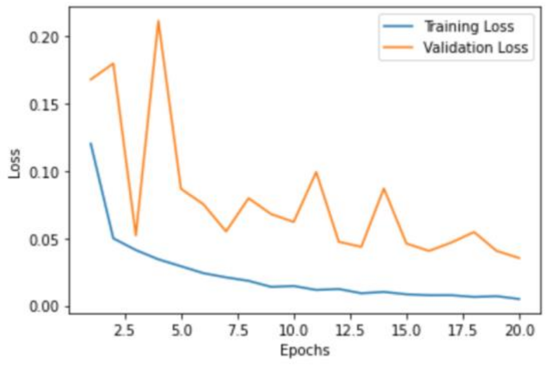
a. For point 2: (Model B vs Model C)

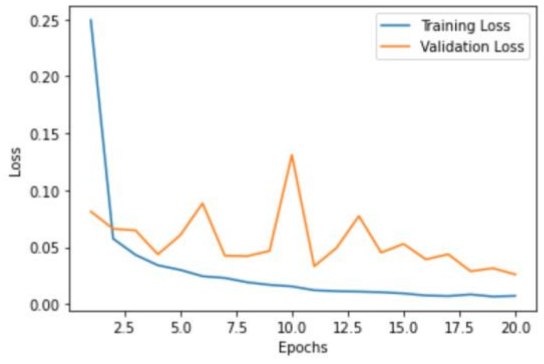
Next, select only 0-9 training images from the above dataset, and use the pretrained network to train on MNIST dataset. Use the standard MNIST train and test splits (http://yann.lecun.com/exdb/mnist/). How does this pretrained network perform in comparison to a randomly initialized network in terms of convergence time, final accuracy and other possible training quality metrics? Do a thorough analysis.

	Data Split (Train/Valid)	# of Epochs	Convergence Time	Train Acc (%)	Valid Acc (%)	Test Acc (%)
Retraining(B)	50000/10000	20	1hr 43 min	99.85	99.15	99.15
Scratch(C)	50000/10000	20	1hr 40 min	99.64	98.71	99.00

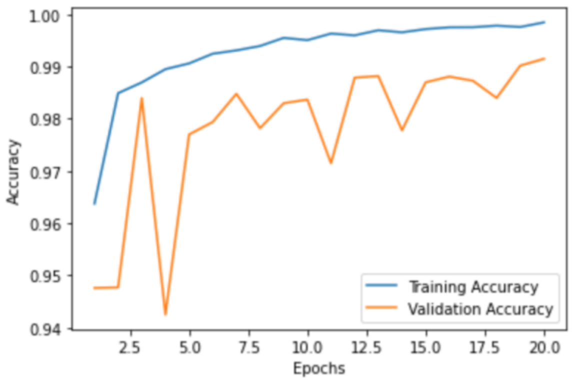
Trained from Scratch

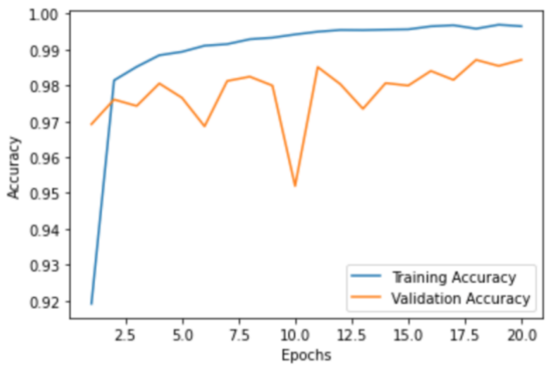
Retraining





Loss vs Epochs:





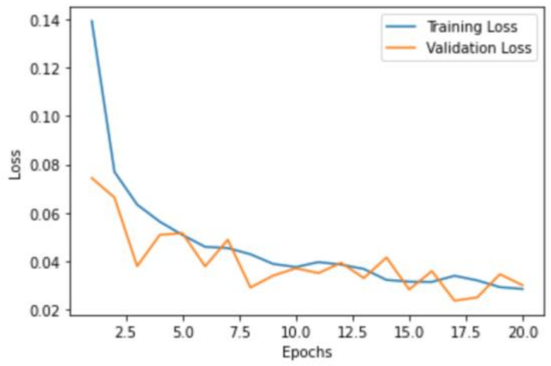
Accuracy vs Epochs

b. For point 3, (Model D vs Model E)

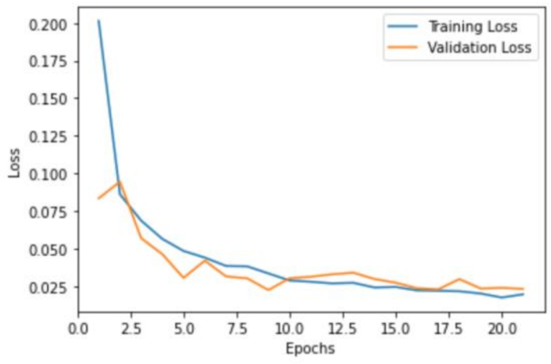
Finally, take the following dataset (https://www.dropbox.com/s/otc12z2w7f7xm8z/mnistTask3.zip), train on this dataset and provide test accuracy on the MNIST test set, using the same test split from part 2. Train using scratch random initialization and using the pretrained network part 1. Do the same analysis as 2 and report what happens this time. Try and do qualitative analysis of what's different in this dataset. Please save your model checkpoints.

	Data Split (Train/Valid)	# of Epochs	Convergence Time	Train Acc (%)	Valid Acc (%)	Test Acc (%)
Retraining(D)	45000/15000	20	34 min	99.06	99.07	99.00
Scratch(E)	45000/15000	20	42 min	99.39	99.23	99.00

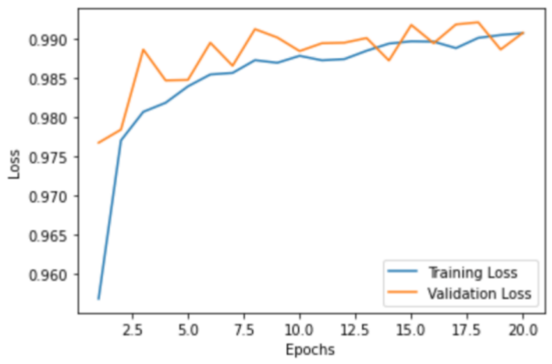
Retraining

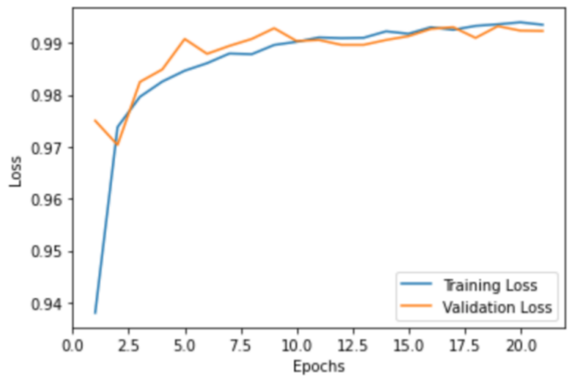


Scratch



Loss vs Epochs

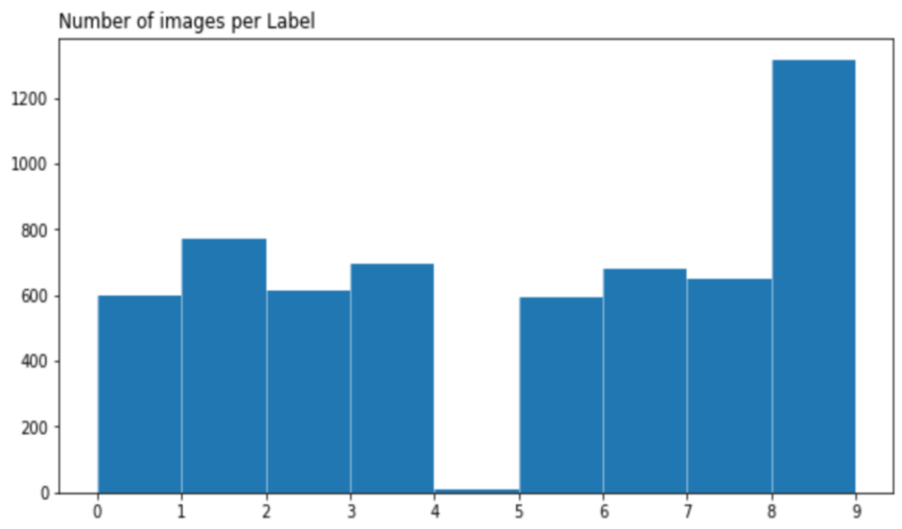




Accuracy vs Epochs

4. Data Analysis for Point 3

For folder 4, this is what the number of images per label look like.



```
1     770
3     693
6     680
9     672
7     647
8     643
2     614
0     596
5     591
4       8
```

Therefore, in folder 4 the images with label 4 are close to minimal in comparison to the other labels.

E. References

1. End-to-End Object Detection with Transformers by FacebookAI
   https://arxiv.org/pdf/2005.12872.pdf