Faculty of Engineering – Cairo University

Electronics And Electrical Communications Department

Fourth year - Mainstream

ELC4028 Neural Networks – Assignment 2

Submitted by:

ID	Sec	Name
9213327	3	محمد أيمن فاروق سيد عبد الغفار
9211027	3	محمد علاء الدين عطفت مصطفى محمد رستم
9211039	4	محمد مجدي مبروك ندا خير
9213468	4	يوسف تامر صلاح الدين السيد
9211418	4	يوسف عصام أبوبكر محمد

Submitted to:

Prof. Mohsen Rashwan

Contents

Problem 1: Multilayer Perceptron (MLP)

- DCT (1 Layer)

```
 Total params: 90,656 (354.13 KB)
 Trainable params: 30,218 (118.04 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 60,438 (236.09 KB)
None
Training time: 4.167790889739999 seconds
Testing time: 0.22723889350891113 seconds
Test accuracy: 95.20000219345093
```

Figure 1: MLP Model using DCT Feature Extraction (1 Layer)

- DCT (3 Layers)

```
 Total params: 189,728 (741.13 KB)
 Trainable params: 63,242 (247.04 KB)
 Non-trainable params: 0 (0.00 B)
 Optimizer params: 126,486 (494.09 KB)
None
Training time: 5.4935669898986816 seconds
Testing time: 0.3879272937774658 seconds
Test accuracy: 95.30000090599060
```

Figure 2: MLP Model using DCT Feature Extraction (3 Layers)

- DCT (5 Layers)

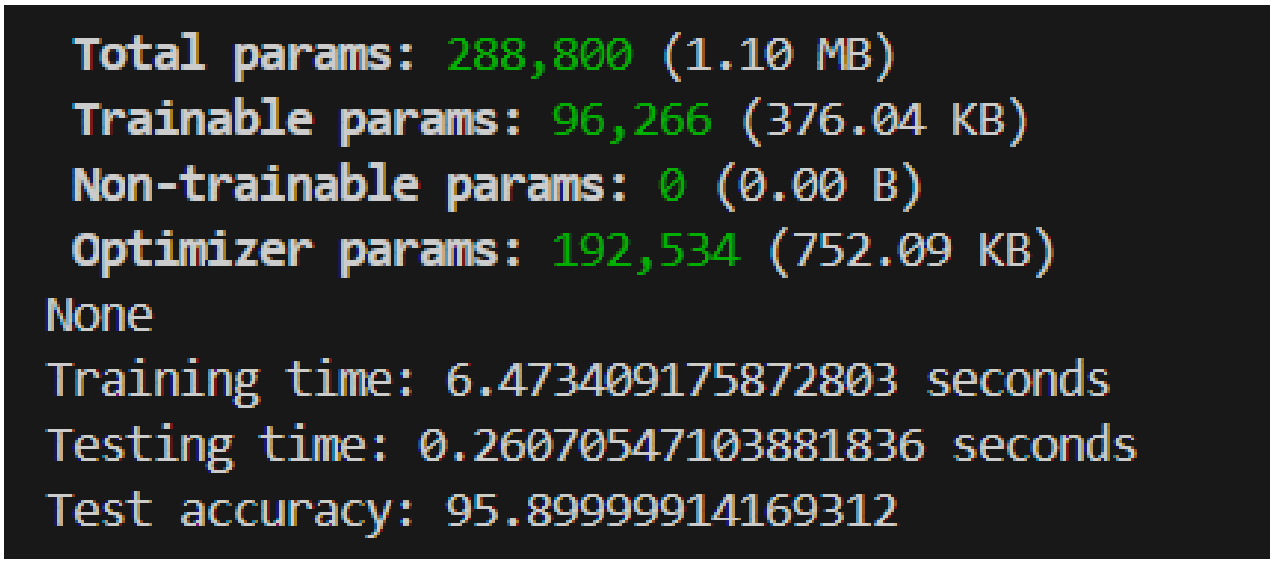


Figure 3: MLP Model using DCT Feature Extraction (5 Layers)

- PCA (1 Layer)

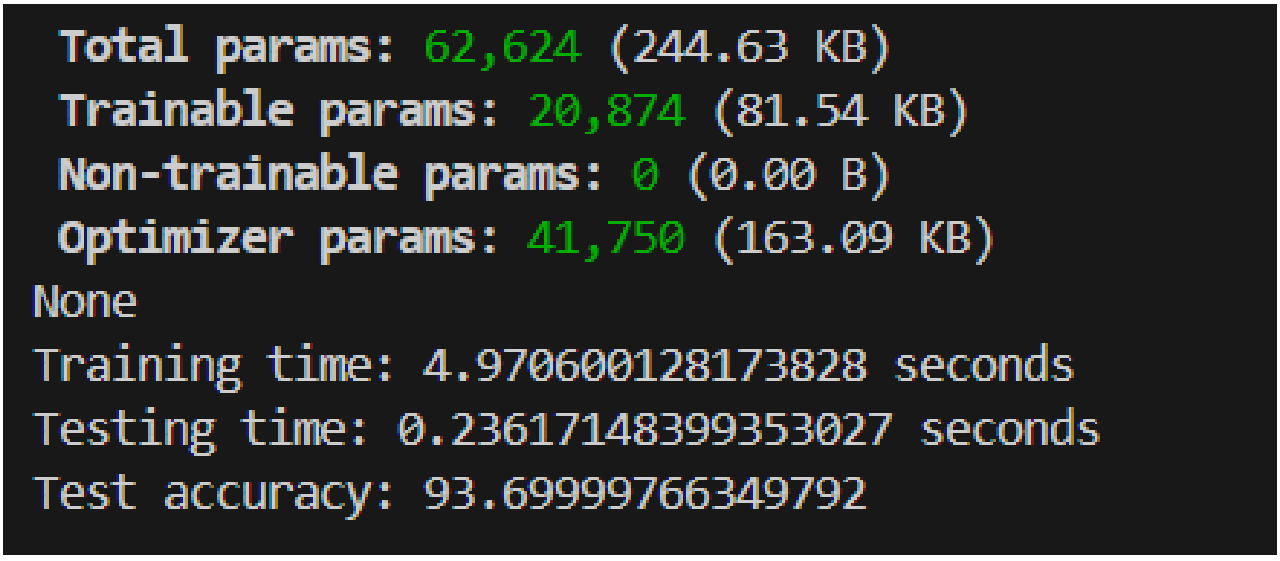


Figure 4: MLP Model using PCA Feature Extraction (1 Layer)

- PCA (3 Layers)

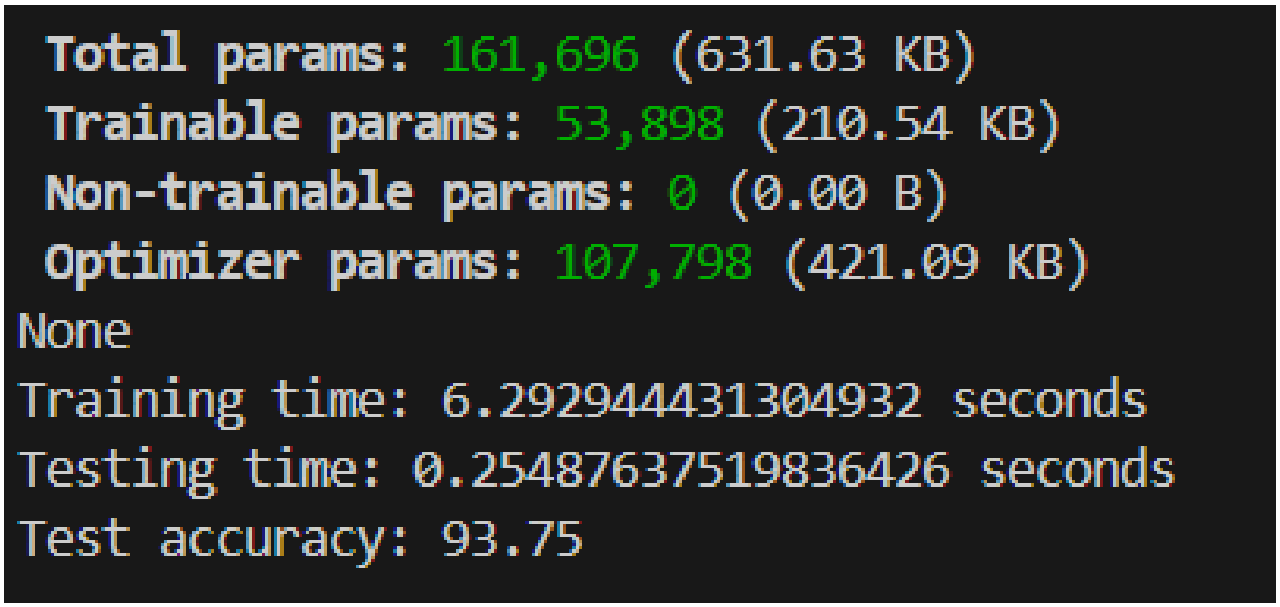


Figure 5: MLP Model using PCA Feature Extraction (3 Layers)

- PCA (5 Layers)

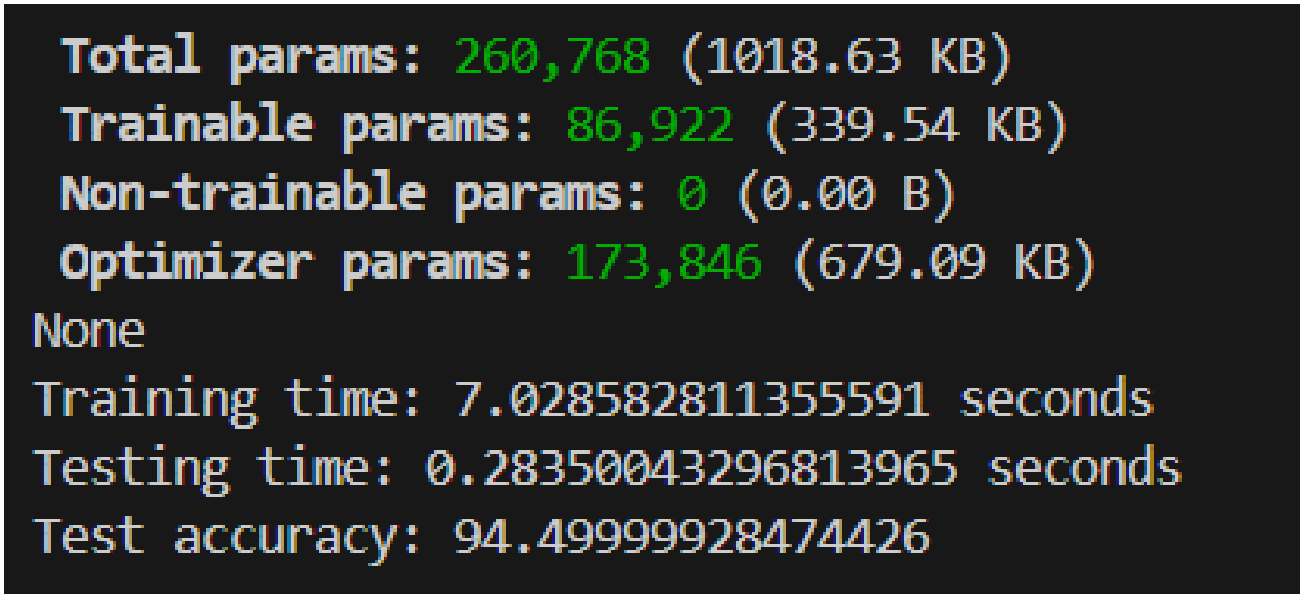
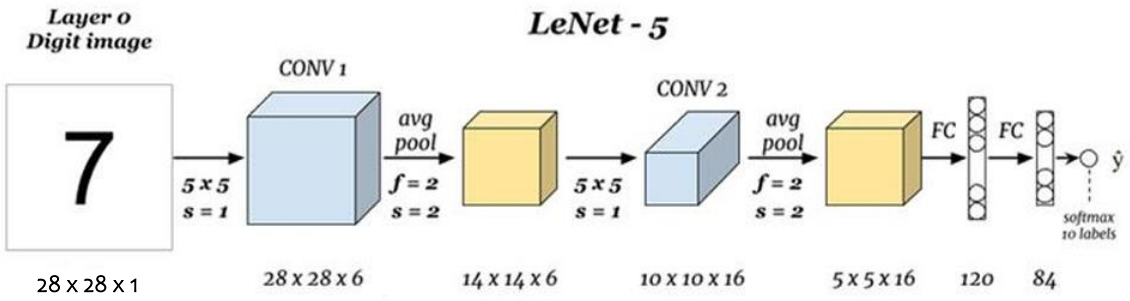


Figure 6: MLP Model using PCA Feature Extraction (5 Layers)

Problem 2: Convolutional Neural Network (CNN)

1. Network construction

LeNet-5 is a convolutional neural network (CNN) architecture designed for image classification. It consists of alternating convolutional and pooling layers followed by fully connected layers. This structure allows the network to automatically learn hierarchical features from input images, making it effective for tasks like digit recognition and image classification.



Kernel size and zero padding is calculated using the following equation:

$$Output\ size = \frac{Input\ size - kernel\ size + 2 * padding}{stride} + 1$$

With input and output sizes given the other parameters are put as follows to satisfy the equation:

- $Conv_1$: $Input\ size = 28, kernel\ size = 5, padding = 2, stride = 1$
- $Pool_1$: $Input\ size = 28, kernel\ size = 2, padding = 0, stride = 2$
- $Conv_2$: $Input\ size = 14, kernel\ size = 5, padding = 0, stride = 1$
- $Pool_2$: $Input\ size = 10, kernel\ size = 2, padding = 0, stride = 2$

2. Optimization algorithm

Adam (Adaptive Moment Estimation) is an optimization algorithm used in training deep learning models. It computes individual adaptive learning rates for each parameter by considering both the first moment (the mean of past gradients) and the second moment (the variance of past gradients). This adaptive approach helps the algorithm converge faster and more reliably compared to traditional stochastic gradient descent (SGD).

The primary parameter of Adam is the learning rate (α), which controls the step size during updates. The algorithm uses this learning rate to adjust the parameters at each step based on the gradients and moment estimates. In out code, learning rat is put to be = 0.001.

3. Results

- No Variation

```
Loading MNIST dataset from local storage...
Epoch 1, Loss: 0.9169, Acc: 70.36%
Epoch 2, Loss: 0.3949, Acc: 87.13%
Epoch 3, Loss: 0.3039, Acc: 90.06%
Epoch 4, Loss: 0.2634, Acc: 91.42%
Epoch 5, Loss: 0.2284, Acc: 92.48%
Epoch 6, Loss: 0.2051, Acc: 93.22%
Epoch 7, Loss: 0.1840, Acc: 93.85%
Epoch 8, Loss: 0.1687, Acc: 94.29%
Epoch 9, Loss: 0.1522, Acc: 94.86%
Epoch 10, Loss: 0.1433, Acc: 95.05%
Final Train Acc: 96.33%
Time taken to train LeNet-5: 42.595 seconds
Final Test Acc: 95.15%
Time taken to test LeNet-5: 0.06 seconds
```

Figure 7: LeNet-5 Model results with no variations

- Activation function changed to Tanh

```
Epoch 1, Loss: 0.8498, Acc: 72.72%
Epoch 2, Loss: 0.4053, Acc: 87.11%
Epoch 3, Loss: 0.3174, Acc: 89.72%
Epoch 4, Loss: 0.2720, Acc: 91.23%
Epoch 5, Loss: 0.2364, Acc: 92.28%
Epoch 6, Loss: 0.2104, Acc: 93.03%
Epoch 7, Loss: 0.1909, Acc: 93.58%
Epoch 8, Loss: 0.1758, Acc: 94.26%
Epoch 9, Loss: 0.1623, Acc: 94.46%
Epoch 10, Loss: 0.1522, Acc: 94.85%
Final Train Acc: 95.92%
Time taken to train LeNet-5: 41.717 seconds
Final Test Acc: 95.70%
Time taken to test LeNet-5: 0.07 seconds
```

Figure 8: LeNet-5 Model results with activation function changed to

- Depth increment (4 layers)

```
Loading MNIST dataset from local storage...
Epoch 1, Loss: 0.8987, Acc: 69.21%
Epoch 2, Loss: 0.3765, Acc: 88.03%
Epoch 3, Loss: 0.2675, Acc: 91.43%
Epoch 4, Loss: 0.2188, Acc: 92.89%
Epoch 5, Loss: 0.1907, Acc: 93.80%
Epoch 6, Loss: 0.1543, Acc: 95.11%
Epoch 7, Loss: 0.1405, Acc: 95.42%
Epoch 8, Loss: 0.1291, Acc: 95.82%
Epoch 9, Loss: 0.1096, Acc: 96.33%
Epoch 10, Loss: 0.1090, Acc: 96.49%
Final Train Acc: 97.52%
Time taken to train LeNet-5: 51.22 seconds
Final Test Acc: 96.45%
Time taken to test LeNet-5: 0.075 seconds
```

Figure 9: LeNet-5 Model results with extra two Hidden FC layers

- Increase number of filters (10, 25 instead of 6, 16)

```
Loading MNIST dataset from local storage...
Epoch 1, Loss: 0.7207, Acc: 76.60%
Epoch 2, Loss: 0.3467, Acc: 88.96%
Epoch 3, Loss: 0.2791, Acc: 91.01%
Epoch 4, Loss: 0.2388, Acc: 92.19%
Epoch 5, Loss: 0.2112, Acc: 93.05%
Epoch 6, Loss: 0.1891, Acc: 93.77%
Epoch 7, Loss: 0.1683, Acc: 94.38%
Epoch 8, Loss: 0.1559, Acc: 94.83%
Epoch 9, Loss: 0.1452, Acc: 95.16%
Epoch 10, Loss: 0.1293, Acc: 95.63%
Final Train Acc: 96.87%
Time taken to train LeNet-5: 69.843 seconds
Final Test Acc: 96.20%
Time taken to test LeNet-5: 0.105 seconds
```

Figure 10: LeNet-5 Model results with depth of each filter increased

Comparison of results with Assignment 1

Features						
Classifier		**DCT**		**PCA**		
		Accuracy	Processing Time	Accuracy	Processing Time	
K-means Clustering	1	80.50%	0.27 sec	80.55%	0.26 sec	Assign. 1
	4	87.10%	0.17 sec	87.40%	0.13 sec	
	16	91.95%	0.29 sec	92.35%	0.27 sec	
	32	92.55%	0.44 sec	92.65%	0.45 sec	
SVM	Linear	91.00%	1.31 sec	90.55%	1.11 sec	
	Nonlinear	95.55%	2.55 sec	96.45%	2.43 sec	
Multi-layer Perceptron (MLP)						
		DCT		**PCA**		Assign. 2
	Variations	Accuracy	Processing Time	Accuracy	Processing Time	
MLP	1-Hidden	95.20%	4.17 sec	93.70%	4.97 sec	
	3-Hidden	95.30%	5.49 sec	93.75%	6.29 sec	
	5-Hidden	95.90%	6.47 sec	94.50%	7.02 sec	
Convolutional Neural Network (CNN)						
		Training time		**PCA**		Assign. 2
	Variations	Accuracy	Processing Time	Accuracy	Processing Time	
CNN	No variations	96.3%	38.9 sec	95.15%	0.06 sec	
	Activision: Tanh	95.92%	41.7 sec	95.7%	0.07 sec	
	3-Hidden layers	97.5%	51.1 sec	96.45%	0.075 sec	
	Filters depth: 10, 25 (Instead of 6, 16)	96.87%	69.8 sec	96.2%	0.105 sec	

As we can observe, increasing the number of hidden layers in the model tends to improve testing accuracy. This is because additional layers allow the model to capture more complex patterns and relationships in the data, enhancing its ability to generalize. However, this comes at the cost of longer processing times. The reason for the increased computational time is the growth in the number of trainable parameters as more layers are added. Each layer introduces additional weights and biases that need to be learned during the training process. Consequently, the model requires more iterations to converge, leading to longer training times.

we can also observe that the accuracy of the Multi-Layer Perceptron (MLP) is higher than that of K-means Clustering and Support Vector Machine (SVM) classifiers, which are considered classical techniques. This higher accuracy can be attributed to the MLP's ability to model complex, non-linear relationships in the data through its multiple layers and neurons. However, it is important to note that the increased accuracy comes with a trade-off in processing time. The number of parameters in the MLP is significantly higher compared to the classical techniques, as MLPs contain multiple layers and neurons, each with its own set of weights and biases. As a result, the training process for MLP requires more computational resources, leading to longer processing times compared to simpler models like K-means and SVM.

Problem 3: Speech Recognition from Speech Spectrum with Augmentation

Program Flow:

Since there is no dataset specified for use for the speech data of the 10 digits, we are going to use the Audio Dataset of the AudioMNIST data available in the following github link: https://github.com/soerenab/AudioMNIST.git

By running the download_audio_mnist.py script, the AudioMNIST is automatically downloaded in the same folder.

We first generate the spectrograms of the AudioMNIST dataset in a folder called AudioMNIST_spectrograms and also generate the spectrograms of audio augmented versions "sped up version by 5%, sped down version by 5% and noise augmented version" of the same dataset in another folder called AudioMNIST_augmented_spectrograms. These can be done using the spectrograms.py script.

Finally, we train a CNN model based on the same LeNet 5 model described in Part 1. The model has four options "Training with no augmentation – Training with audio augmentation only – Training with image augmentation only – Training with both image and audio augmentation" which can be controlled using two Boolean variables: augmentAudio, and augmentImages. Also, the number of sample per digit from the dataset on which the model is trained can be modified using the parameter sample_per_digit. A large value for sample_per_digit results in better validation and testing accuracy but takes larger training time while a small value results in less training time. The results outlined in this report are made using 100 samples per digit.

Simulation Results:

The digital spectrograms for various scenarios involving two different speakers:

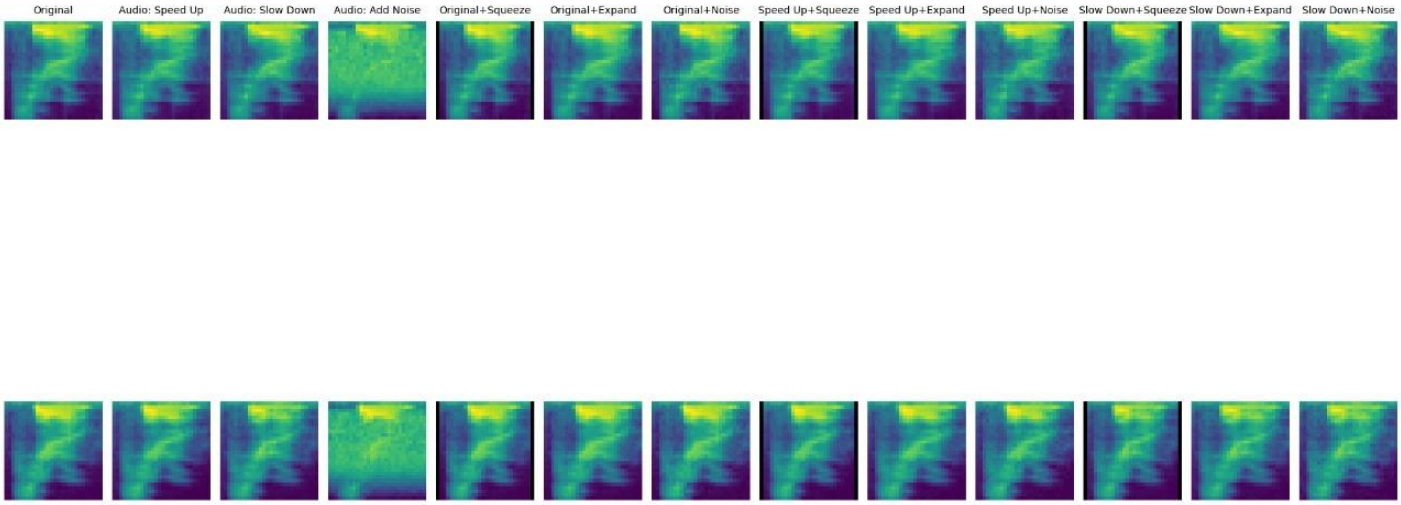
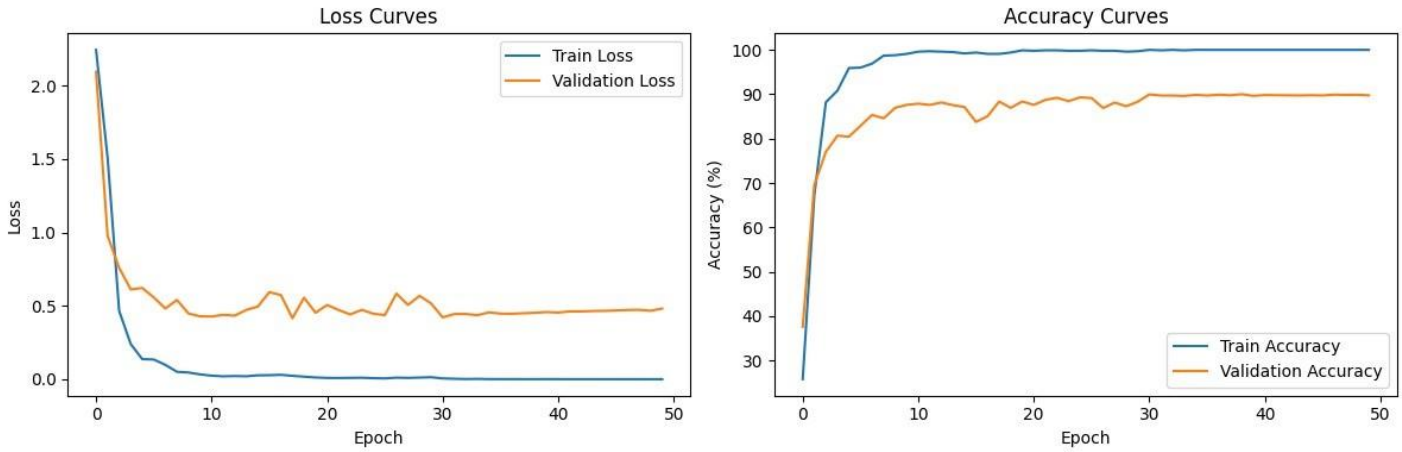


Figure 11: Spectrograms of sample audio files with different augmentations

No Augmentation Results:

```
Total training time: 0:04:46
Average time per epoch: 0:00:05
```

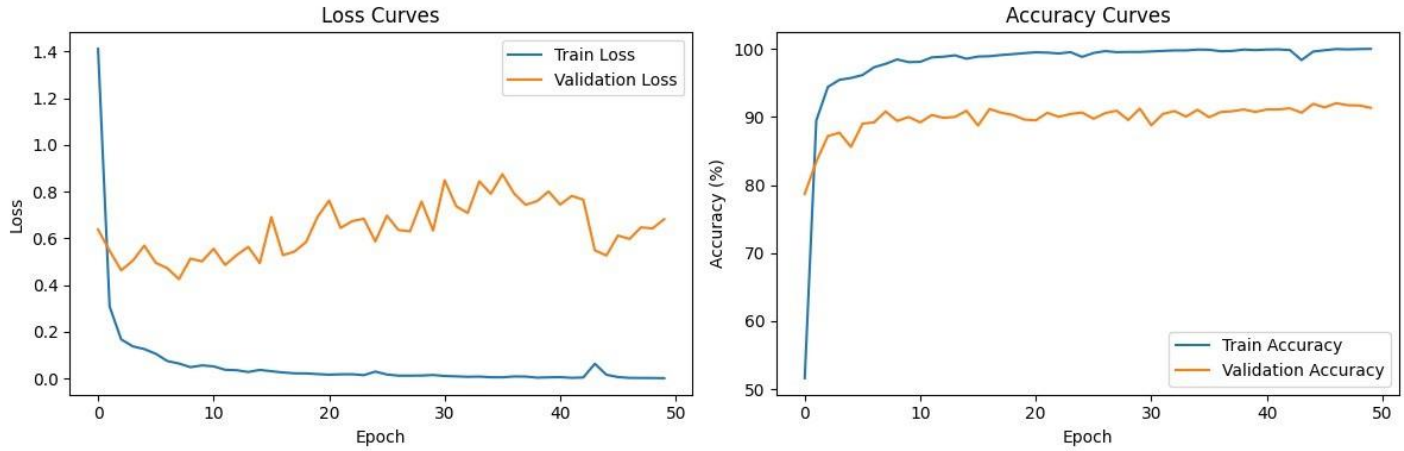
```
Epoch 50/50
Train Loss: 0.0005, Train Accuracy: 100.00%
Val Loss: 0.4817, Val Accuracy: 89.77%
```



Audio Augmentation Only Results:

```
Total training time: 0:07:07
Average time per epoch: 0:00:08
```

```
Epoch 50/50
Train Loss: 0.0013, Train Accuracy: 100.00%
Val Loss: 0.6816, Val Accuracy: 91.33%
```



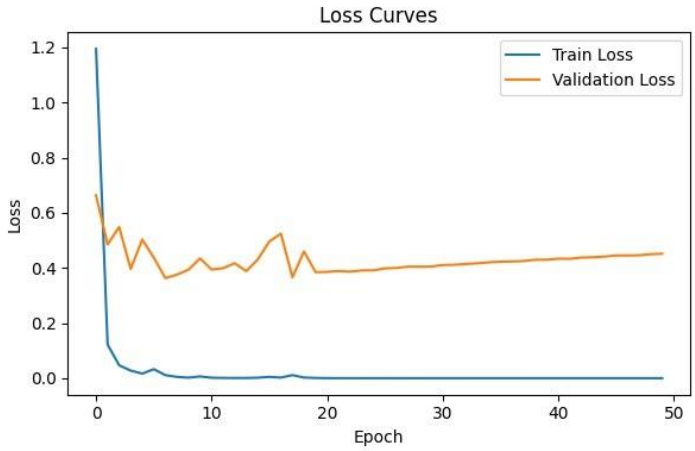
When trained with the entire data set:

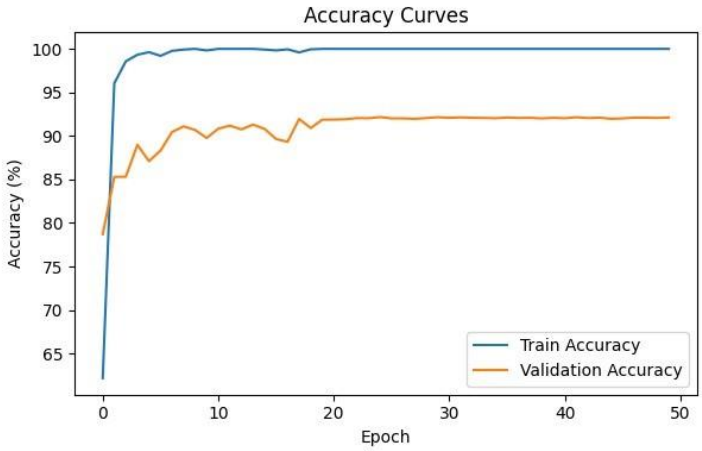
```
Epoch 1/3
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 2840s 1s/step - accuracy: 0.6824 - loss: 0.8876 - val_accuracy: 0.9116 - val_loss: 0.2589 - learning_rate: 0.0010
Epoch 2/3
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 157s 65ms/step - accuracy: 0.9212 - loss: 0.2270 - val_accuracy: 0.9345 - val_loss: 0.1871 - learning_rate: 0.0010
Epoch 3/3
2400/2400 ━━━━━━━━━━━━━━━━━━━━ 107s 45ms/step - accuracy: 0.9502 - loss: 0.1477 - val_accuracy: 0.9593 - val_loss: 0.1189 - learning_rate: 0.0010
750/750 ━━━━━━━━━━━━━━━━━━━━ 321s 428ms/step - accuracy: 0.9890 - loss: 0.0349

Test Accuracy: 98.72%
```

Image Augmentation Only Results:

```
Total training time: 0:05:30
Average time per epoch: 0:00:06
```

```
Epoch 50/50
Train Loss: 0.0000, Train Accuracy: 100.00%
Val Loss: 0.4515, Val Accuracy: 92.12%
```

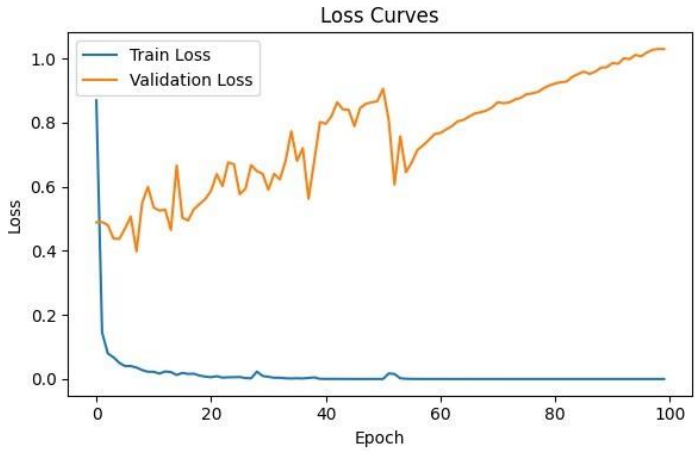


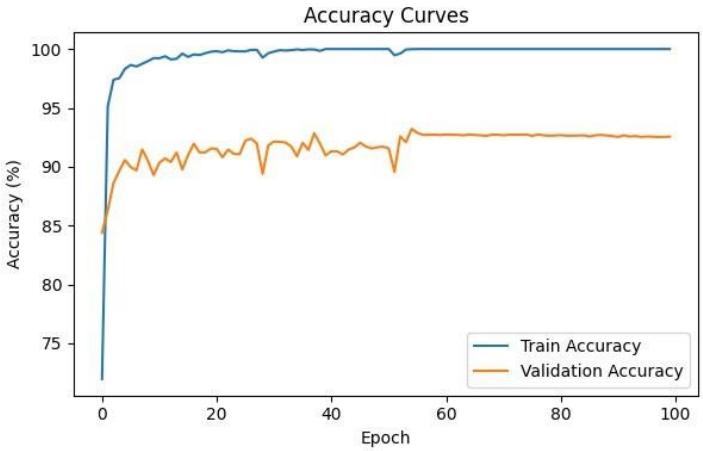


Both Audio and Image Augmentation Results:

```
Total training time: 0:17:59
Average time per epoch: 0:00:10
```

```
Epoch 100/100
Train Loss: 0.0000, Train Accuracy: 100.00%
Val Loss: 1.0299, Val Accuracy: 92.55%
```





When trained with the entire data set:

```
Epoch 1/3
7200/7200 ━━━━━━━━━━━━━━━━━━━━ 6006s 834ms/step - accuracy: 0.8270 - loss: 0.4869 - val_accuracy: 0.9615 - val_loss: 0.1178 - learning_rate: 0.0010
Epoch 2/3
7200/7200 ━━━━━━━━━━━━━━━━━━━━ 4923s 684ms/step - accuracy: 0.9771 - loss: 0.0680 - val_accuracy: 0.9692 - val_loss: 0.0998 - learning_rate: 0.0010
Epoch 3/3
7200/7200 ━━━━━━━━━━━━━━━━━━━━ 4715s 655ms/step - accuracy: 0.9855 - loss: 0.0429 - val_accuracy: 0.9696 - val_loss: 0.1038 - learning_rate: 0.0010
2250/2250 ━━━━━━━━━━━━━━━━━━━━ 141s 63ms/step - accuracy: 0.9920 - loss: 0.0239

Test Accuracy: 99.10%
```

Results Summary:

Test Scenario	Validation Accuracy
No Augmentation	**89.77%**
Audio Augmentation Only	**91.33%**
Image Augmentation Only	**92.12%**
Both Audio and Image Augmentation	**92.55%**

Comments and conclusion:

- It can be noticed that when testing with 100 samples per digit image augmentation increases the validation accuracy by about 1.5% from 89.77% to 92.12% while audio augmentation increases the validation accuracy to about 91.33%. Image augmentation accompanied with audio augmentation increases the validation accuracy to about 92.6% which shows how augmentation on the same dataset can increase the accuracy without the need for more original data.
- It can be noticed how image augmentation performs better than audio augmentation in the prospect of increasing the accuracy.
- In part (d), the accuracy did not improve as expected despite the increased amount of data. This outcome can be attributed to the high variability introduced by combining multiple augmentation techniques (e.g., noisy+noisy, fast+expand, slow+squeeze, etc.). While augmentation increases the dataset size, it also introduces significant diversity, which can degrade the overall data quality and lead to reduced model performance.