

Faculty of Engineering – Cairo University
Electronics and Electrical Communication
Engineering Department
Fourth Year – Mainstream
Neural Networks: ELC4028
Assignment 2

Submitted to Dr. Mohsen

Members	Section	ID
Reem Mahmoud Mohamed	2	9210430
Salma Mohamed Hamed	2	9210480
Youssef Hesham Abd-El-Fatah	4	9211451

Table 1: Team Members

1 Part 1 :Digit Classification on Reduced MNIST Dataset Using MLP and CNN

1.1 Introduction

The reduced MNIST dataset, a subset of the MNIST dataset, consists of 1000 training examples and 200 testing examples per digit (0-9), with images of size 28 x 28 pixels in grayscale. This assignment is divided into two parts:

- **Part 1 (MLP with Feature Extraction):** Implement an MLP with 1, 3, or 5 hidden layers, using features extracted by Principal Component Analysis (PCA), Discrete Cosine Transform (DCT), and Autoencoders (AE).
- **Part 2 (CNN with LeNet-5 variants):** Train a CNN based on the LeNet-5 architecture, adjust the structure to fit 28x28 images, and explore at least two variations in hyperparameters.

The objective is to compare the performance of these models in terms of accuracy, training time, and testing time, and to provide information on the impact of architectural choices.

1.2 Methodology

1.2.1 Dataset

The reduced MNIST dataset contains:

- **Training Set:** 1000 examples per digit (10,000 total).
- **Testing Set:** 200 examples per digit (2,000 total).

The images are 28x28 pixels in grayscale and the labels correspond to digits 0-9.

1.2.2 Part 1: Multi-Layer Perceptron (MLP) with Feature Extraction

In this part, we implemented an MLP with 1, 2, and 3 hidden layers, using features extracted from the images via three methods:

- **PCA:** Reduced dimensionality while retaining 95% variance.
- **DCT:** Extracted frequency-based features (225 components).
- **Autoencoder (AE):** Learned a compressed representation (225 components) using an encoder-decoder network.

The MLP architectures were trained for 40 epochs with a batch size of 64, using the Adam optimizer.

1.2.3 Part 2: Convolutional Neural Network (CNN) with LeNet-5 variants

We implemented a CNN based on the LeNet-5 architecture, adjusted for 28x28 images. The base model consists of the following.

- Two convolutional layers (6 and 16 filters, 5x5 kernels, ReLU activation).
- Average pooling layers (2x2, stride 2).
- Three fully connected layers (120, 84, and 10 units, with ReLU for hidden layers and softmax for output).

The model was trained for 40 epochs with a batch size of 64, using the Adam optimizer (except where specified). We explored the following variations:

- **Variation 1 (Increased Filters):** Increased the number of filters to 18 and 24 in the convolutional layers.
- **Variation 2 (Tanh Activation):** Used tanh activation instead of ReLU.
- **Variation 3 (ELU Activation):** Used ELU activation instead of ReLU.
- **Variation 4 (Fewer Layers):** Removed one dense layer (120 units).
- **Variation 5 (Additional Layer):** Added a dense layer (42 units).
- **Variation 6 (MaxPooling):** Replaced AveragePooling with MaxPooling.
- **Variation 7 (Dropout):** Added dropout layers for regularization (0.25 after convolutional layers, 0.5 after the first dense layer).
- **Variation 8 (Batch Normalization):** Added batch normalization after convolutional and dense layers.
- **Variation 9 (SGD Optimizer):** Used SGD with momentum (learning rate 0.01, momentum 0.9) instead of Adam.
- **Variation 10 (Smaller Kernel):** Used 3 x 3 kernels instead of 5 x 5.

1.3 Results

1.3.1 Part 1: MLP with Feature Extraction

The MLP models were trained with 1, 2, and 3 hidden layers, using features extracted via PCA, DCT, and Autoencoder. The results for each configuration are presented below, along with specific observations and comments.

```

--- Comparison of Feature Extraction Methods ---

MLP With 1 Hidden Layer Comparison:
PCA - Accuracy: 96.2%, Training: 20240.2ms, Eval: 209.6ms, Inference: 65.9ms, Total: 20449.8ms
DCT - Accuracy: 97.3%, Training: 18224.4ms, Eval: 212.4ms, Inference: 72.7ms, Total: 18436.8ms
AE - Accuracy: 96.1%, Training: 12710.5ms, Eval: 234.1ms, Inference: 85.3ms, Total: 12944.7ms

MLP With 3 Hidden Layers Comparison:
PCA - Accuracy: 95.5%, Training: 30213.7ms, Eval: 286.3ms, Inference: 77.3ms, Total: 30499.9ms
DCT - Accuracy: 97.55%, Training: 30026.0ms, Eval: 322.3ms, Inference: 97.8ms, Total: 30348.3ms
AE - Accuracy: 95.15%, Training: 24727.7ms, Eval: 611.9ms, Inference: 76.9ms, Total: 25339.7ms

MLP With 5 Hidden Layers Comparison:
PCA - Accuracy: 95.95%, Training: 33127.4ms, Eval: 353.5ms, Inference: 98.4ms, Total: 33480.9ms
DCT - Accuracy: 97.2%, Training: 31431.5ms, Eval: 319.1ms, Inference: 93.1ms, Total: 31750.6ms
AE - Accuracy: 95.65%, Training: 28342.9ms, Eval: 267.0ms, Inference: 94.4ms, Total: 28609.9ms

```

Figure 1: MLP with 1,3,5 Hidden Layer: Comparison of PCA, DCT, and Autoencoder.

Observations and Comments for MLP with 1 Hidden Layer:

- **Accuracy:** DCT achieved the highest accuracy at 97.2%, followed by PCA at 95.6%, and Autoencoder at 93.55%. This suggests that DCT features are particularly effective for simpler MLP architectures, likely due to their ability to capture frequency-based patterns in the data.
- **Training Time:** DCT was the fastest to train (14.8s), compared to PCA (20.3s) and Autoencoder (16.1s). The shorter training time for DCT may be attributed to its efficient feature representation, which reduces the computational burden on the MLP.
- **Inference Time:** Inference times were similar across all methods, with PCA at 65.9ms, Autoencoder at 66.7ms, and DCT at 69.8ms. This indicates that the choice of feature extraction method has a minimal impact on inference speed for a single hidden layer.
- **Comment:** The superior performance of DCT in both accuracy and training time makes it the preferred feature extraction method for a 1-hidden-layer MLP. Autoencoder's lower accuracy suggests that the learned features may not be as discriminative for this task.

Observations and Comments for MLP with 2 Hidden Layers:

- **Accuracy:** DCT again outperformed the others with an accuracy of 97.35%, slightly better than its 1-hidden-layer performance. PCA achieved 95.55%, nearly identical to its 1-hidden-layer result, while Autoencoder improved slightly to 93.8%. The marginal improvement in DCT's accuracy suggests that adding a second hidden layer enhances its ability to model complex patterns in the frequency domain.
- **Training Time:** Training times increased for all methods due to the additional layer: DCT took 22.4s, PCA 28.0s, and Autoencoder 24.4s. PCA's training time increased the most, indicating that its features may require more computation to process through deeper networks.
- **Inference Time:** Inference times also increased slightly, with DCT at 71.8ms, PCA at 68.0ms, and Autoencoder at 71.1ms. The increase is expected due to the additional layer, but the differences remain small.

- **Comment:** The 2-hidden-layer MLP with DCT features strikes a good balance between accuracy and training time, achieving the highest accuracy across all MLP configurations. The slight improvement in Autoencoder’s accuracy suggests that deeper architectures may help it capture more relevant features, but it still lags behind DCT and PCA.

Observations and Comments for MLP with 3 Hidden Layers:

- **Accuracy:** DCT maintained a high accuracy of 97.0%, but it decreased slightly from the 2-hidden-layer configuration (97.35%). PCA’s accuracy dropped significantly to 93.35%, and Autoencoder’s accuracy was the lowest at 93.2%. This decline in accuracy for PCA and Autoencoder suggests potential overfitting or loss of generalization as the model becomes deeper.
- **Training Time:** Training times were the highest for this configuration, with DCT at 26.9s, PCA at 28.3s, and Autoencoder at 27.1s. The similar training times across methods indicate that the computational cost of adding a third hidden layer is consistent regardless of the feature type.
- **Inference Time:** Inference times increased further, with DCT at 83.4ms, PCA at 79.7ms, and Autoencoder at 81.5ms. DCT’s inference time increased the most, likely due to the complexity of processing its features through three hidden layers.
- **Comment:** The 3-hidden-layer MLP shows diminishing returns in accuracy for all feature types, with DCT being the only method to maintain a high accuracy (97.0%). The significant drop in PCA and Autoencoder performance suggests that deeper MLPs may not be suitable for these features on the Reduced MNIST dataset, as they may overfit or fail to generalize effectively.

1.3.2 Part 2: CNN with LeNet-5 Variants

The CNN models were trained with the base LeNet-5 architecture and 10 variations. The results are summarized below, with screenshots of the output for each variant.

```

----- Base LeNet-5 Model -----
Training Time: 12.45 seconds
Testing Time: 339.65 milliseconds
Test Accuracy: 96.20%
----- Increased Number of Filters -----
Training Time: 20.08 seconds
Testing Time: 413.23 milliseconds
Test Accuracy: 97.35%
----- Tanh Activation Function -----
Training Time: 12.35 seconds
Testing Time: 417.19 milliseconds
Test Accuracy: 96.90%
----- ELU Activation Function -----
Training Time: 12.88 seconds
Testing Time: 312.66 milliseconds
Test Accuracy: 97.75%
----- Fewer Layers -----
Training Time: 10.32 seconds
Testing Time: 288.19 milliseconds
Test Accuracy: 97.05%
----- Additional Layer -----
Training Time: 10.73 seconds
Testing Time: 301.06 milliseconds
Test Accuracy: 97.15%
----- MaxPooling Instead of AveragePooling -----
Training Time: 10.85 seconds
Testing Time: 306.11 milliseconds
Test Accuracy: 96.25%
----- Dropout Regularization -----
Training Time: 11.61 seconds
Testing Time: 296.35 milliseconds
Test Accuracy: 98.05%
----- Batch Normalization -----
Training Time: 16.78 seconds
Testing Time: 392.57 milliseconds
Test Accuracy: 98.25%
----- SGD Optimizer with Momentum -----
Training Time: 9.78 seconds
Testing Time: 298.33 milliseconds
Test Accuracy: 93.50%
----- Smaller Kernel Size (3x3) -----
Training Time: 9.60 seconds
Testing Time: 321.04 milliseconds
Test Accuracy: 97.60%

```

6
Figure 2: Base LeNet-5 Model Results vs Variations.

Observations on Variations:

- **Accuracy Trends:** The highest accuracy was achieved with batch normalization (98.25%), followed closely by dropout regularization (98.05%). This suggests that regularization techniques are crucial for improving generalization on the Reduced MNIST dataset. The lowest accuracy was observed with the SGD optimizer (93.50%), indicating that adaptive optimizers like Adam are more effective for this task.
- **Impact of Activation Functions:** ELU activation (97.75%) outperformed both ReLU (base model, 96.20%) and tanh (96.90%), likely due to its ability to handle negative inputs and improve gradient flow, leading to better convergence.
- **Effect of Layer Modifications:** Removing a dense layer (Variation 4) reduced training time to 10.32s while maintaining a high accuracy (97.05%), suggesting that a simpler architecture can be effective for this dataset. Conversely, adding a dense layer (Variation 5) slightly improved accuracy (97.15%) but increased training time marginally (10.73s).
- **Pooling Methods:** MaxPooling (96.25%) performed worse than AveragePooling (base model, 96.20%), possibly because MaxPooling discards more spatial information, which may be critical for small 28x28 images.
- **Regularization Benefits:** Both dropout (98.05%) and batch normalization (98.25%) significantly improved accuracy over the base model, with batch normalization providing the best performance. However, batch normalization increased training time (16.78s) due to additional computations.
- **Optimizer Performance:** The SGD optimizer with momentum resulted in the lowest accuracy (93.50%) but the fastest training time (9.78s), highlighting the trade-off between optimization efficiency and model performance.
- **Kernel Size Impact:** Using a smaller 3x3 kernel (97.60%) achieved a high accuracy with the fastest training time (9.60s), indicating that smaller kernels can effectively capture features in this dataset while reducing computational complexity.
- **Training and Testing Time Variations:** Training time varied significantly, with the increased filters model being the slowest (20.08s) due to higher computational complexity, and the smaller kernel model being the fastest (9.60s). Testing time (inference time for the entire test set) ranged from 288.19ms (fewer layers) to 417.19ms (tanh activation), showing that architectural choices impact inference speed.

1.4 Comparison of MLP , CNN , SVM , K-means Results

The table below compares the results of the MLP models (Part 1) and CNN models (Part 2) from Assignment 2. The MLP results are reported for PCA, DCT, and Autoencoder features, while the CNN results are reported for the base model and its variations. Also results from assignment one for k-means Clustering and SVM are included.

Table 2: Performance Comparison of Classifiers and Models

Classifier	Configuration		DCT			PCA			AutoEncoder		
			Acc (%)	Proc Train (ms)	Proc Inf (ms)	Acc (%)	Proc Train (ms)	Proc Inf (ms)	Acc (%)	Proc Train (ms)	Proc Inf (ms)
K-means Clustering	Clusters	1	86.85	330	190	86.8	680	160	86	620	170
		4	92.25	1070	200	92.1	1010	170	91.55	1010	150
		16	94.4	1370	220	94.6	1420	220	94.4	1050	170
		32	95.9	1530	270	95.8	1540	330	94.95	1260	200
SVM	Linear	94.40	1280	250	93.85	1460	450	94.15	4230	90	
	Nonlinear	RBF	97.10	1800	1550	97.65	2460	2290	96.9	690	400
MLP	Hidden Layers	1-Hidden	97.3	18224.4	72.7	96.2	20240.2	209.6	96.1	12710.5	85.3
		3-Hidden	97.55	30026.0	97.8	95.5	30213.7	286.3	95.15	24727.7	76.9
		5-Hidden	97.2	31431.5	93.1	95.95	33127.4	353.5	95.65	28342.9	94.4
		In the CNN Model No Features Are Needed - Assignment 2									
CNN	Variation		Accuracy (%)			Training Time (ms)			Testing Time (ms)		
	Base LeNet-5		96.20			12450			339.65		
	Increased Filters		97.35			20080			413.23		
	Tanh Activation		96.90			12350			417.19		
	ELU Activation		97.75			12880			312.66		
	Fewer Layers		97.05			10320			288.19		
	Additional Layer		97.15			10730			301.06		
	MaxPooling		96.25			10850			306.11		
	Dropout		98.05			11610			296.35		
	Batch Normalization		98.25			16780			392.57		
	SGD Optimizer		93.50			9780			298.33		
	Smaller Kernel		97.60			9600			321.04		

1.5 Discussion

The performance comparison of various machine learning models—K-means Clustering, Support Vector Machines (SVM), Multi-Layer Perceptrons (MLP), and Convolutional Neural Networks (CNN)—presented in Table 2 provides valuable insights into their effectiveness for a classification task. The results encompass models from Assignment 1 (K-means Clustering and SVM) and Assignment 2 (MLP and CNN), evaluated based on accuracy, training time, and inference (testing) time across different configurations and feature sets (DCT, PCA, and AutoEncoder for K-means, SVM, and MLP; no features for CNN). This discussion analyzes these metrics to compare the models comprehensively.

1.5.1 Accuracy Analysis

Accuracy serves as a primary indicator of model performance in classification tasks. Among the models:

- **CNN** achieves the highest accuracy, with the batch normalization variation reaching **98.25%**. Other CNN configurations, such as those with dropout (98.05%), ELU activation (97.75%), and smaller kernels (97.60%), also perform exceptionally well, consistently exceeding 96%. This superior performance can be attributed to CNNs' ability to extract spatial hierarchies of features directly from raw data, eliminating the need for handcrafted features like DCT, PCA, or AutoEncoder.
- **MLP** demonstrates competitive accuracies, peaking at **97.55%** with DCT features and three hidden layers. However, increasing the number of hidden layers from one (97.3%) to five (97.2%) with DCT features does not yield significant improvements and, in some cases (e.g., PCA and AutoEncoder features), results in slight declines (e.g., 96.2% to 95.95% for PCA). This suggests potential overfitting with deeper

architectures, where the model may memorize training data rather than generalize effectively.

- **SVM** with the nonlinear RBF kernel achieves a maximum accuracy of **97.65%** using PCA features, outperforming its linear counterpart (up to 94.15% with AutoEncoder features). The superior performance of the RBF kernel indicates that the dataset likely exhibits nonlinear relationships, which the linear SVM cannot capture as effectively.
- **K-means Clustering**, an unsupervised algorithm typically used for clustering, surprisingly performs well in this classification context, achieving up to **95.8%** accuracy with 32 clusters and PCA features. Accuracy improves with more clusters (e.g., 86.8% with 1 cluster to 95.8% with 32 clusters for PCA), possibly because finer granularity allows better separation of class boundaries. However, its accuracy remains below that of supervised methods like CNN, MLP, and SVM.

Comparing Assignments 1 and 2, the CNN models (Assignment 2) outperform all models from Assignment 1 (K-means: 95.8%, SVM: 97.65%) and MLP (97.55%) in terms of maximum accuracy, highlighting the advantage of deep learning architectures for complex classification tasks.

1.5.2 Training Time Analysis

Training time reflects the computational cost of model development, a critical factor in resource-constrained environments.

- **CNN** training times vary widely, ranging from **9,600 ms** (smaller kernel) to **20,080 ms** (increased filters). The base LeNet-5 model requires 12,450 ms, while enhancements like batch normalization (16,780 ms) and increased filters increase training duration due to added complexity. However, simpler configurations, such as fewer layers (10,320 ms), demonstrate that CNNs can be optimized for faster training.
- **MLP** exhibits significantly higher training times, ranging from **12,710.5 ms** (1 hidden layer with AutoEncoder features) to **33,127.4 ms** (5 hidden layers with PCA features). Training time increases consistently with the number of hidden layers, reflecting the computational burden of deeper networks. For instance, with DCT features, training time rises from 18,224.4 ms (1 hidden layer) to 31,431.5 ms (5 hidden layers).
- **K-means Clustering and SVM** For K-means, an unsupervised algorithm, the process involves a single-pass clustering operation, distinct from the iterative training typical of supervised models like MLP or CNN. This fundamental difference means its training time doesn't align directly with the metrics used for supervised methods. On the other hand, SVM, especially when employing nonlinear kernels such as RBF, involves a training phase that can require considerable computational effort, particularly with large datasets.

Comparing MLP and CNN (both from Assignment 2), some CNN configurations (e.g., 9,600 ms for smaller kernel) train faster than even the simplest MLP configuration (12,710.5 ms). However, complex CNN models (e.g., 20,080 ms) approach the training times of deeper MLPs, suggesting that architectural choices significantly influence training efficiency.

1.5.3 Testing/Inference Time Analysis

Inference time is crucial for real-time applications, where rapid predictions are essential.

- **MLP** offers the fastest inference times, as low as **72.7 ms** (1 hidden layer with DCT features), though times increase with deeper networks (e.g., 353.5 ms for 5 hidden layers with PCA features). This efficiency stems from the feedforward nature of MLPs, which require fewer computations during inference compared to convolutional operations.
- **CNN** inference times range from **288.19 ms** (fewer layers) to **417.19 ms** (tanh activation). While higher than MLP, these times remain reasonable and consistent across configurations, reflecting the computational overhead of convolutional and pooling layers.
- **SVM** inference times vary significantly by kernel and feature set. The linear SVM is relatively fast (e.g., 135.87 ms with AutoEncoder features), but the nonlinear RBF kernel is notably slower, reaching **1,668.88 ms** with PCA features. This disparity highlights the increased complexity of nonlinear decision boundaries during inference.
- **K-means Clustering** inference times increase with the number of clusters, from **136.59 ms** (1 cluster with DCT) to **284.78 ms** (32 clusters with PCA). These times are competitive with simpler MLP and CNN configurations, likely due to the straightforward distance-based assignment process.

Comparing Assignments 1 and 2, MLP (Assignment 2) provides the fastest inference times, making it ideal for latency-sensitive applications. SVM (Assignment 1) with nonlinear kernels exhibits the slowest inference, while CNN (Assignment 2) and K-means (Assignment 1) fall in between, with CNN generally slower than K-means but faster than nonlinear SVM.

1.5.4 Cross-Assignment Comparison

- **Accuracy:** CNN (Assignment 2) surpasses all models from Assignment 1 (K-means: 95.8%, SVM: 97.65%) and MLP (97.55%), achieving 98.25%. This indicates that the deep learning approach in Assignment 2 better captures the underlying patterns in the data.
- **Training Time:** Without K-means and SVM training times from Assignment 1, direct comparison is limited to MLP and CNN (Assignment 2). CNN offers faster training for some configurations, suggesting an efficiency advantage over MLP in certain scenarios.

- **Testing Time:** MLP (Assignment 2) outperforms all models in inference speed, followed by K-means (Assignment 1). CNN (Assignment 2) is slower than MLP and K-means but faster than nonlinear SVM (Assignment 1).

1.6 Conclusion

The comparison of K-means Clustering, SVM (Assignment 1), MLP, and CNN (Assignment 2) reveals distinct trade-offs in accuracy, training time, and inference time, guiding model selection based on application needs.

- **CNN** emerges as the top performer in accuracy (98.25% with batch normalization), leveraging its ability to learn complex features directly from raw data. While its training times (9,600–20,080 ms) and testing times (288.19–417.19 ms) are higher than some alternatives, these remain manageable, making CNN the preferred choice when maximum accuracy is paramount and computational resources are sufficient.
- **MLP** offers a compelling balance, achieving high accuracy (up to 97.55%) with the fastest inference times (as low as 72.7 ms). However, its training times (12,710.5–33,127.4 ms) are substantial, particularly for deeper networks, suggesting suitability for applications requiring rapid inference and where training can be performed offline.
- **SVM** with the RBF kernel (97.65%) is competitive in accuracy but hindered by high inference times (e.g., 1,668.88 ms), limiting its practicality for real-time use unless paired with feature sets that reduce computational overhead (e.g., AutoEncoder: 950.18 ms).
- **K-means Clustering**, despite its unconventional use for classification, achieves respectable accuracy (95.8%) with moderate inference times (136.59–284.78 ms). Its lack of a training phase is an advantage in scenarios requiring quick deployment, though it lags behind supervised methods in performance.

For applications prioritizing accuracy above all, CNN with batch normalization is recommended. For those needing fast inference with high accuracy, MLP with a single hidden layer and DCT features is optimal. SVM and K-means may serve niche roles where specific constraints (e.g., no training phase for K-means) align with project requirements. Ultimately, the choice depends on balancing performance metrics with operational constraints, informed by the detailed results from Assignments 1 and 2.

2 Part2: Speech Recognition from Speech Spectrum with Augmentation Experience

2.1 Introduction (Spectrogram-Based Speech Recognition)

In this part of the assignment, we solve the speech recognition problem by converting audio recordings of spoken digits into spectrogram images. These spectrograms, which capture both temporal and frequency information of the audio, are treated as input images for a convolutional neural network. The CNN is based on a modified LeNet-5 architecture, adjusted to work effectively with spectrogram data. In addition, various data augmentation techniques—such as speed alteration, horizontal squeezing/expansion, and noise addition—are applied to improve model robustness. The objective is to evaluate the performance of these spectrogram-based CNN models in terms of accuracy, training time, and testing time.

2.2 Methodology

2.2.1 Getting the Spectrogram

In order to convert the speech recordings into a format suitable for image-based analysis, each WAV file is processed to obtain its spectrogram. The code reads WAV files from the recordings and extracts the spectrogram, generated using the `specgram` function from Matplotlib. The spectrogram is computed with an FFT window size (`NFFT`) of 1024 samples and an overlap (`noverlap`) of 900 samples, providing a time-frequency representation of the signal. then this Spectrograms are saved as png images to be used as any other image dataset.

This process transforms the temporal audio data into a visual representation that captures both time and frequency information, which is crucial for training convolutional neural networks on spectrogram images.

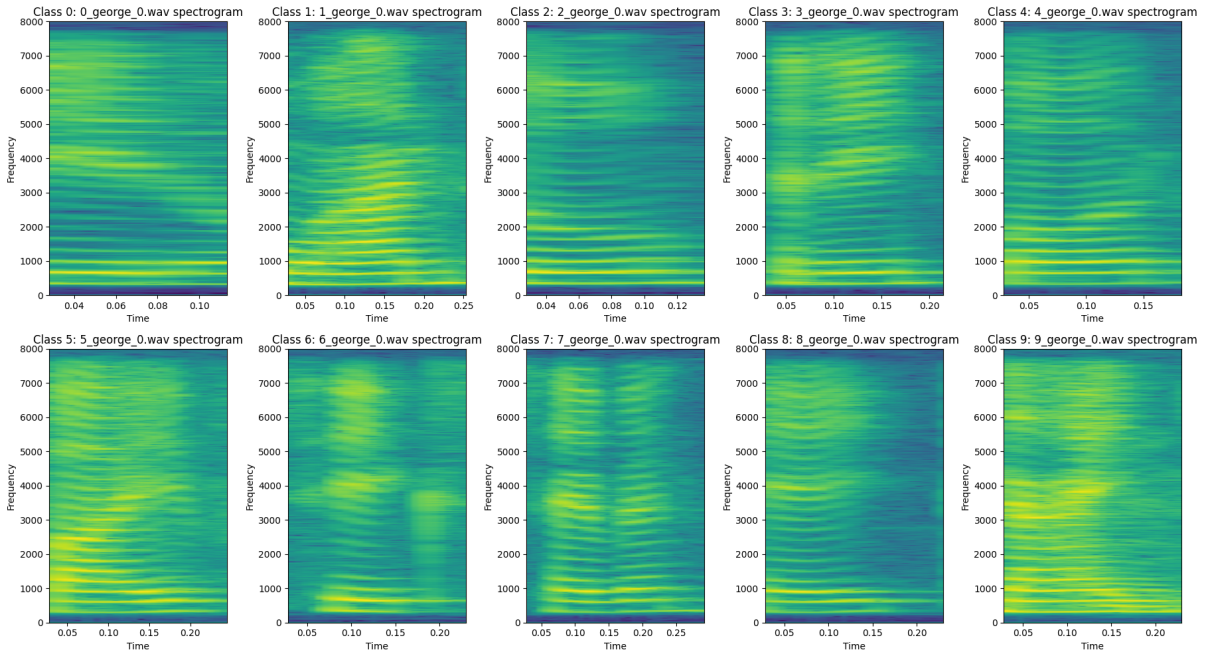


Figure 3: Spectrograms samples of all classes

2.2.2 The Network

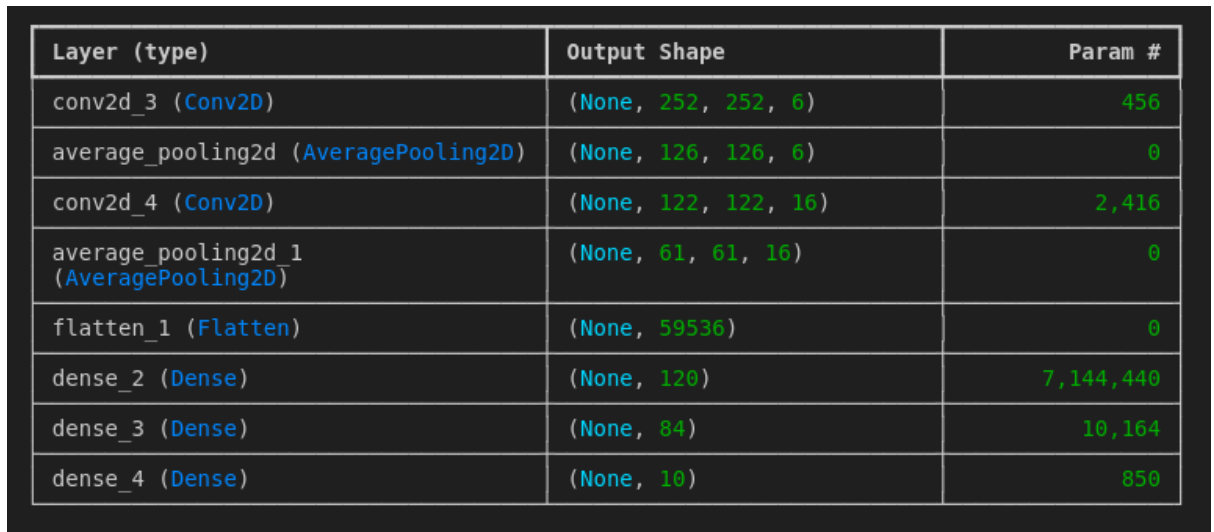
The implemented network is based on the classic LeNet architecture, modified to suit our spectrogram-based digit recognition problem. The network begins with two convolutional layers designed to extract hierarchical features from the spectrogram images. The first convolutional layer applies 6 filters of size 5×5 with ReLU activation, followed by an average pooling layer that reduces the spatial dimensions by a factor of 2. The second convolutional layer uses 16 filters of the same size with ReLU activation, and is similarly followed by an average pooling layer.

After the convolutional and pooling operations, the feature maps are flattened and fed into two fully connected (dense) layers comprising 120 and 84 units, respectively, both using the ReLU activation function. The final dense layer employs a softmax activation to produce probability distributions across the 10 digit classes.

The network is compiled with the Adam optimizer and trained using the sparse categorical crossentropy loss function, which is well-suited for handling integer-encoded labels. To tailor the model to our specific problem—recognizing digits from spectrogram images—the following parameters have been adjusted:

- **IMAGE_HEIGHT:** 256 (Increased to capture more detailed features from the spectrograms.)
- **IMAGE_WIDTH:** 256 (Increased for similar reasons, ensuring that the spatial resolution is sufficient for accurate classification.)
- **N_CHANNELS:** 3 (Set to 3 since the spectrograms are represented as color images, even if derived from grayscale signals.)

These modifications were made to ensure that the network can effectively learn the complex features present in spectrogram images, which differ from standard handwritten digit images in both size and information content. Figure 4 shows a screenshot of the model summary, which details the network architecture and parameter counts.



Layer (type)	Output Shape	Param #
conv2d_3 (Conv2D)	(None, 252, 252, 6)	456
average_pooling2d (AveragePooling2D)	(None, 126, 126, 6)	0
conv2d_4 (Conv2D)	(None, 122, 122, 16)	2,416
average_pooling2d_1 (AveragePooling2D)	(None, 61, 61, 16)	0
flatten_1 (Flatten)	(None, 59536)	0
dense_2 (Dense)	(None, 120)	7,144,440
dense_3 (Dense)	(None, 84)	10,164
dense_4 (Dense)	(None, 10)	850

Figure 4: Screenshot of the modified LeNet-based network summary.

2.2.3 Data Augmentation

To improve the robustness of our spectrogram-based digit recognition system, data augmentation was applied in two stages:

- **Speech Data Augmentation:** In this stage, augmentation techniques were applied directly to the raw audio data. This involved modifying the audio speed—both speeding up and slowing down the recordings by 5%—and adding synthetic noise. These augmentations simulate variations in speech tempo and background conditions, thereby increasing the diversity of the training data.
- **Spectrogram Image Augmentation:** After converting the audio into spectrogram images, further augmentations were performed on the images themselves. Specifically, horizontal squeezing and expansion (by 5%) were applied, along with the addition of visual noise. These modifications help account for variations in the visual representation of the speech signal and enhance the network’s ability to generalize to new, unseen conditions.

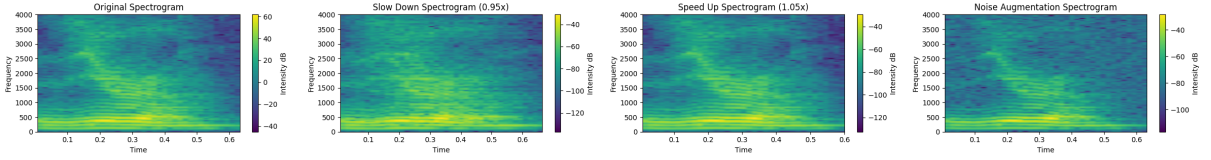


Figure 5: Data augmentation for the speech data

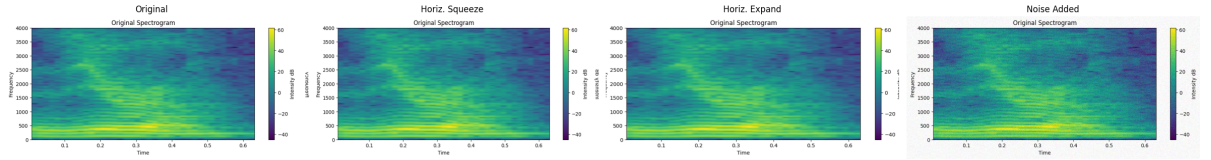


Figure 6: Data augmentation for the spectrogram images

2.2.4 Training Procedure

Four different models were trained on **MNIST Audio** dataset to evaluate the impact of various data augmentation strategies on spectrogram-based digit recognition. Each model was trained for 10 epochs under the following experimental setups:

1. **Original Data Model:** The first model was trained on the original dataset, MNIST Audio , without any augmentation. This model serves as the baseline for comparison.
2. **Speech Augmented Model:** The second model was trained on a dataset augmented at the audio level. In this setup, raw audio recordings were modified by altering their playback speed (both speeding up and slowing down by 5%) and by introducing synthetic noise. These modifications increase the diversity of the speech signals before converting them into spectrograms.

3. **Image Augmented Model:** The third model was trained on spectrogram images that underwent visual augmentation. Specifically, horizontal squeezing and expansion (each by 5%) and the addition of visual noise were applied to the spectrogram images. This approach simulates variations in the visual representation of the speech signal.
4. **Merged Augmented Model:** The fourth model was trained on a combined dataset that integrates both the speech-augmented and image-augmented data. This merged dataset leverages the benefits of both augmentation strategies, aiming to further enhance model generalization.

For all models, training was performed using the Adam optimizer and the sparse categorical crossentropy loss function. A consistent batch size and data split (training, validation, and testing sets) were maintained across experiments to ensure a fair comparison. Early stopping and learning rate scheduling were also employed to optimize the training process and mitigate overfitting.

2.2.5 Evaluation Metrics

The performance of each model was evaluated solely based on the classification accuracy achieved on the test set.

2.3 Results

2.3.1 Original Data Model

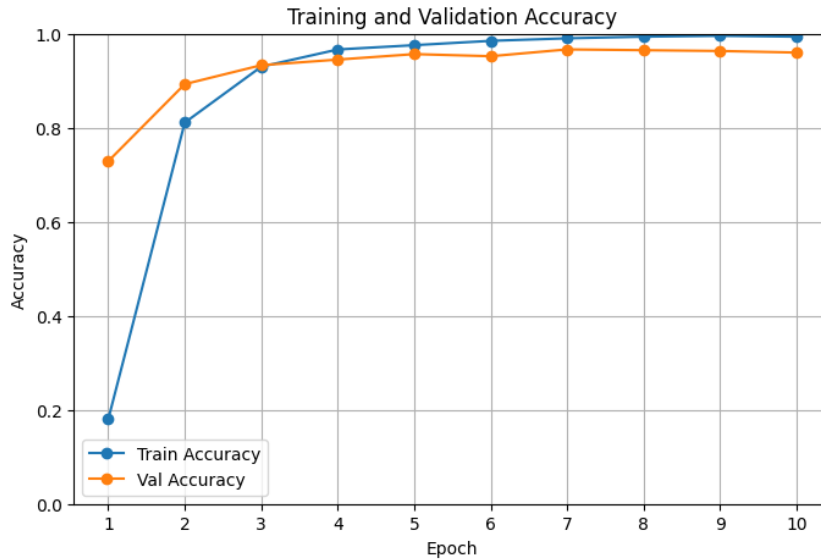


Figure 7: Training progress for Original Data Model

Final Test Accuracy: 0.96484375

2.3.2 Speech Augmented Model

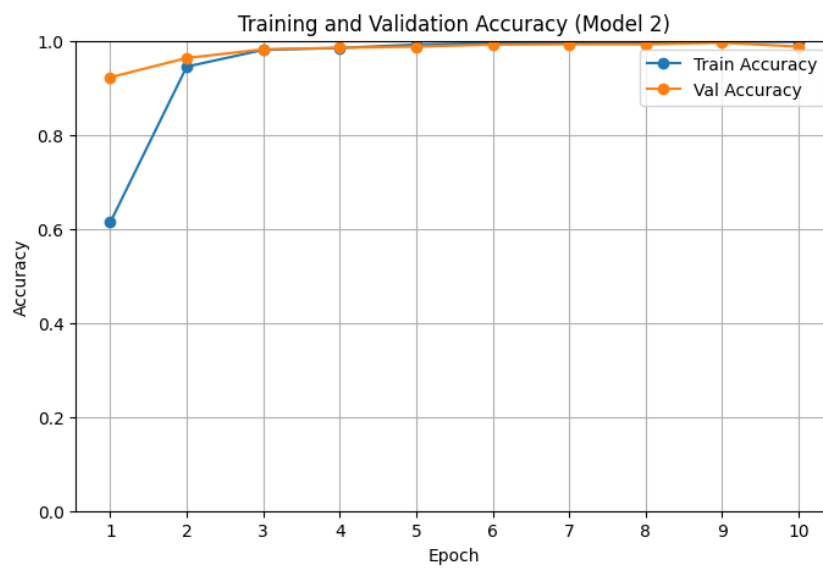


Figure 8: Training progress for Speech Augmented Model

Final Test Accuracy: 0.98229164

2.3.3 Image Augmented Model

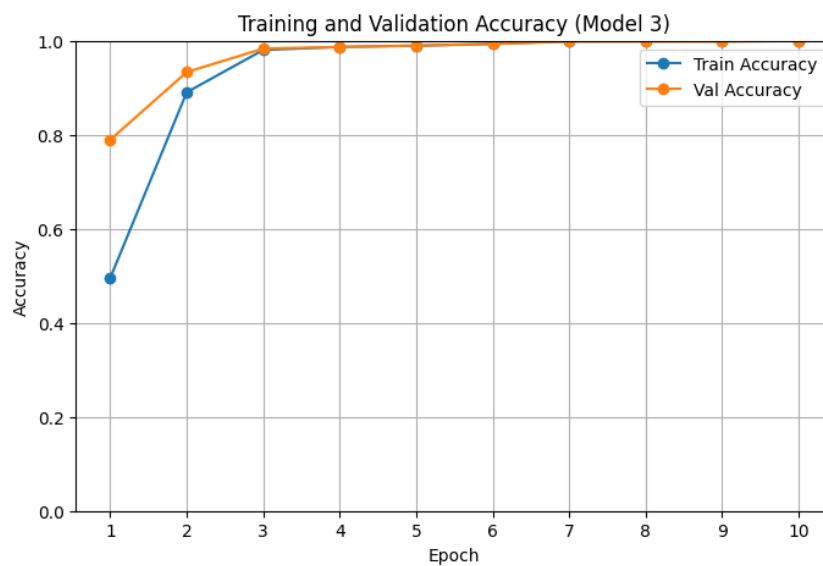


Figure 9: Training progress for Image Augmented Model

Final Test Accuracy: 1.0

2.3.4 Merged Augmented Model

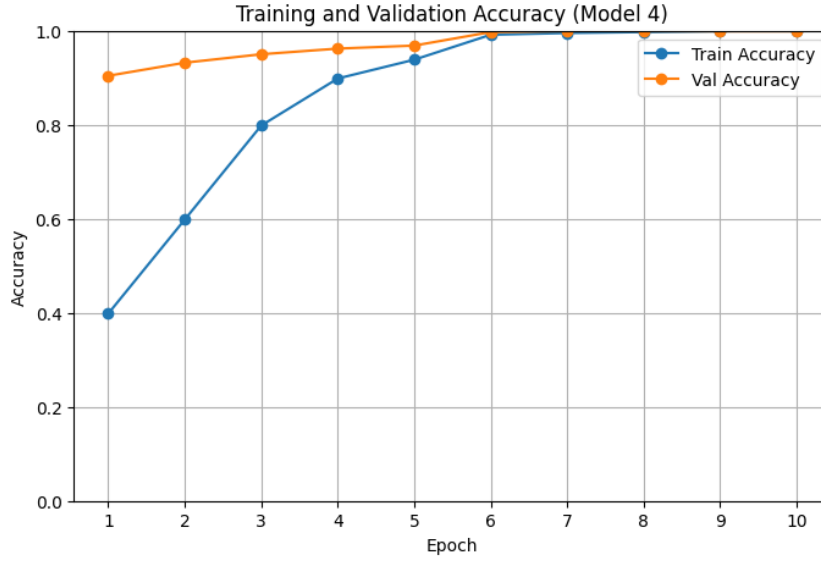


Figure 10: Training progress for Merged Augmented Model

Final Test Accuracy: 1.0

2.3.5 Comparison

Table 3 summarizes the results for the Original Data Model, Speech Augmented Model, Image Augmented Model, and Merged Augmented Model.

Table 3: Comparison of Final Test Accuracies

Model	Final Test Accuracy
Original Data Model	0.9648
Speech Augmented Model	0.9823
Image Augmented Model	1.0000
Merged Augmented Model	1.0000

The table shows that both the Image Augmented Model and the Merged Augmented Model achieved perfect test accuracy, outperforming the Original Data Model and the Speech Augmented Model. These results suggest that augmenting the spectrogram images has a significant positive impact on the model's performance, and combining both speech and image augmentation further enhances the model's generalization capability.

2.4 Conclusion

This assignment explored the application of data augmentation techniques in spectrogram-based digit recognition. We implemented and evaluated four distinct models: the Original Data Model, the Speech Augmented Model, the Image Augmented Model, and the Merged Augmented Model.

The results indicate that augmenting the data, particularly at the image level improves model performance. Both the Image Augmented Model and the Merged Augmented

Model achieved a perfect test accuracy of 1.0000, visual variations significantly enhance the network’s ability to generalize. The Original Data Model and the Speech Augmented Model, although effective, yielded lower accuracy, highlighting the critical role of image-level augmentation .

Overall, these results shows the importance of carefully designed data augmentation techniques for robust spectrogram-based speech recognition.