Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Computer Vision

ENCS5343

Course Project

Arabic Handwritten Text Identification Using Deep Learning Techniques

Prepared by:

Qusay Taradeh **1212508.**

Karim Marayta **1211610.**

Instructor: **Dr. Aziz Qaroush.**

Section: **1.**

Date: **Saturday, January 25, 2025.**

Table of Contents

Table of Contents i

Table of Figures ii

List of Tables iii

Introduction 1

Experiments And Results 1

Data Preprocess and Split: 1

Task 1: Build Custom CNN. 2

CNN 1: 3

CNN 2: 5

CNN 3: 8

Task 1: Results Discussion 10

Task 2: Retrain Selected CNN After Doing Data Augmentation. 11

Data Augmentation 11

Task 2: Results Discussion 13

Task 3: Train Published CNN With Data Augmentation. 14

Data Augmentation and 3 Channels Conversion 14

ResNet-18 Setup 14

Squeeze and Excitation Blocks 15

Task 3: Results Discussion 16

Task 4: Use Pre-trained CNN. 17

Task 4: Results Discussion 17

CNN's Validation and Test Accuracy Comparison 18

Conclusion: 19

Table of Figures

Figure 1: Samples of pre-processed images..... 2
Figure 2: CNN 1 validation accuracy and loss at 15 patience and 200 max epochs 4
Figure 3: CNN 1 validation accuracy and loss at 25 patience and 250 max epochs 4
Figure 4: CNN 1 validation accuracy and loss at 50 patience and 500 max epochs 5
Figure 5: CNN 2 validation accuracy and loss at 15 patience and 200 max epochs 6
Figure 6: CNN 2 validation accuracy and loss at 25 patience and 250 max epochs 7
Figure 7: CNN 2 validation accuracy and loss at 50 patience and 500 max epochs 7
Figure 8: CNN 3 validation accuracy and loss at 15 patience and 200 max epochs 9
Figure 9: CNN 3 validation accuracy and loss at 25 patience and 250 max epochs 9
Figure 10: CNN 3 validation accuracy and loss at 50 patience and 500 max epochs 10
Figure 11: Samples of augmented images 11
Figure 12: CNN 3 retrained validation accuracy and loss at 15 patience and 200 max epochs ... 12
Figure 13: CNN 2 retrained validation accuracy and loss at 25 patience and 250 max epochs ... 12
Figure 14: CNN 2 retrained validation accuracy and loss at 50 patience and 500 max epochs ... 13
Figure 15: ResNet 18 + SE validation accuracy and loss 16
Figure 16: Resnet-18 + SE with Transfer validation accuracy and loss. 17
Figure 17 Best Validation Accuracy of all models. 18
Figure 18 Testing Accuracy of all models. 18

List of Tables

Table 0-1: All CNNs' Accuracies Summary 18

Introduction

Handwritten text recognition is a challenging problem, especially for scripts like Arabic, which have complex shapes and styles. In the earlier assignment, traditional methods such as SIFT, ORB, and AKAZE were used to extract and compare image features. Building on that work, this project focuses on using deep learning techniques, specifically Convolutional Neural Networks (CNNs), to address the same problem.

CNNs are powerful tools in computer vision because they can automatically learn patterns from raw data, such as handwritten text, without the need for handcrafted features. They are particularly effective at recognizing patterns that stay consistent despite changes in scale, rotation, or illumination.

This project involves designing custom CNN models, applying data augmentation to improve performance, and using pre-trained networks with transfer learning to fine-tune results. The AHAWP dataset, which contains Arabic handwritten word samples, will be used to evaluate how well these models perform. Key metrics like accuracy and robustness will be analyzed, and the results will be compared to understand the benefits of different approaches.

By exploring CNN-based methods, this project aims to demonstrate how deep learning can handle complex handwriting recognition tasks while addressing the limitations of traditional methods.

Experiments And Results

Data Preprocess and Split:

In this section, the dataset of images uploaded and preprocessed by the torch-vision library to be ready for training any CNN we build on them as follows:

Resize: each image is resized to **224x224** pixels which is suit for each CNN we built, and to ensure all images have the same dimensions, which is required for feeding them into any CNN.

RGB Conversion: The dataset is grayscale, but some prebuilt models take RGB (3 channels). The input was duplicated to get 3 channels for such models.

PyTorch Tensor Conversion: Convert each image into a format (**tensors**) suitable for PyTorch that was used in three CNNs.

Normalization: Normalize each image tensor by adjusting its pixel values with **0.5** value for both mean and standard deviation to ensure that scaled values are between **-1** and **1**, and this step is done for faster and more stable training. For prebuilt models and transfer learning, the normalization parameters have been matched to the ones used during transfer learning.

Then, the dataset was divided into **70%** training, **15%** validation, and **15%** test sets with batch size **64** which is suit for our dataset for each one.

There is below some samples of pre-processed images:

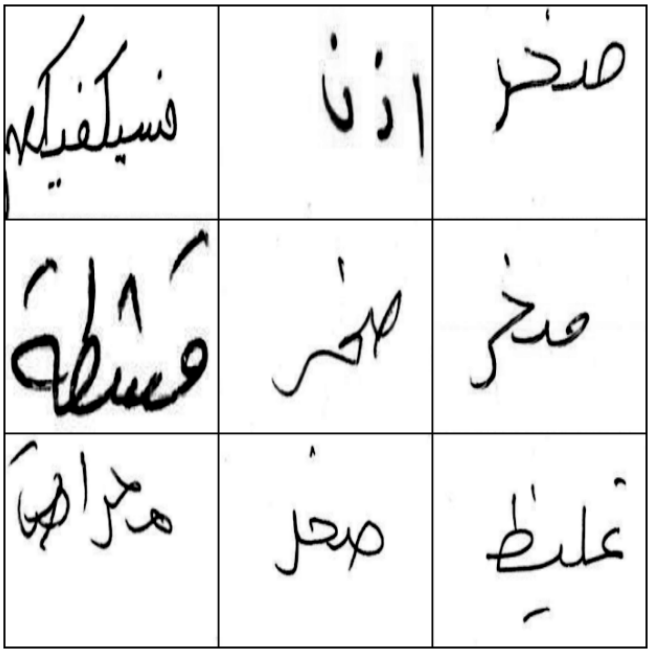


Figure 1: Samples of pre-processed images

Task 1: Build Custom CNN.

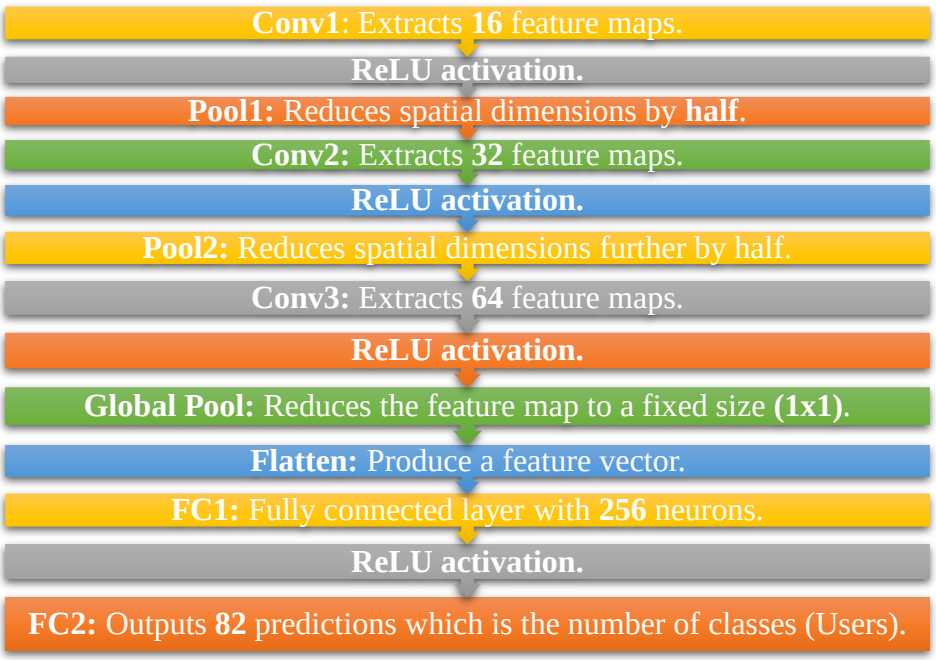
In this section, utilizing **Torch** Library Methods we built three several CNN architectures with different hyperparameters for each one, selecting the best CNN with the highest accuracy and lowest loss.

We built a **Base CNN** class to receive any CNN with its hyperparameters. We built and did training and evaluation together, such that the learning rate default value is $\mathbf{10^{-3}}$, and the loss function used is **Cross Entropy Loss**. Other hyperparameters are taken from global configuration values such as the learning rate. Class functions involve training, validation, and testing steps such that in both validation and testing the loss and accuracy were measured but in training one the loss was measured only. It also contains a configure optimizer function that utilizes **AdamW** Optimizer for efficient optimization with regularization to reduce

overfitting by adjusting weight decay hyperparameter value with $\mathbf{10^{-4}}$. To add to that, it utilizes a **CosineAnnealingWarmRestarts** scheduler from the torch library schedular that contains the optimizer and restarts after 10 epochs, doubling this period as growing with a minimum learning rate equal to $\mathbf{10^{-6}}$.

CNN 1:

In this CNN we used **LightningModule** from **pytorch_lightning** library to build the CNN, then we identified the number of **convolutional layers** to be **3** with **kernel size** equals **3** and **padding** equals **1** with **stride** equals **1** for each one to extract hierarchical spatial features, **one Max-pooling layer** with **pooling window size** equals **2** to reduce spatial dimensions by half, **one Adaptive average pooling layer** to reduce the feature map to a fixed size (1x1), and number of **fully-connected layers** to be **2** for classification. In the Forward Pass mechanism, we passed input through convolutional layers with **ReLU activation** and **max pooling**. Then we used global average pooling before flattening the feature map to a vector to be ready for a fully connected layer. Finally, we applied fully connected layers and produced the final output. To summarize this CNN Architecture the process is done as follows:



Then the training was at a **learning rate** equal to **0.001** with **maximum epochs** equal to **200** and an **early stopping patience** value equal to **50** which is used when the validation loss didn't improve for **50 sequential epochs** to stop the learning early and save the best model

hyperparameters. In this step, the optimizer and schedular mentioned previously are used to reduce overfitting and find a good minimum of loss function.

After the training step is done, both Loss and Validation Accuracy Vs. The number of Epochs is shown in the figures below:

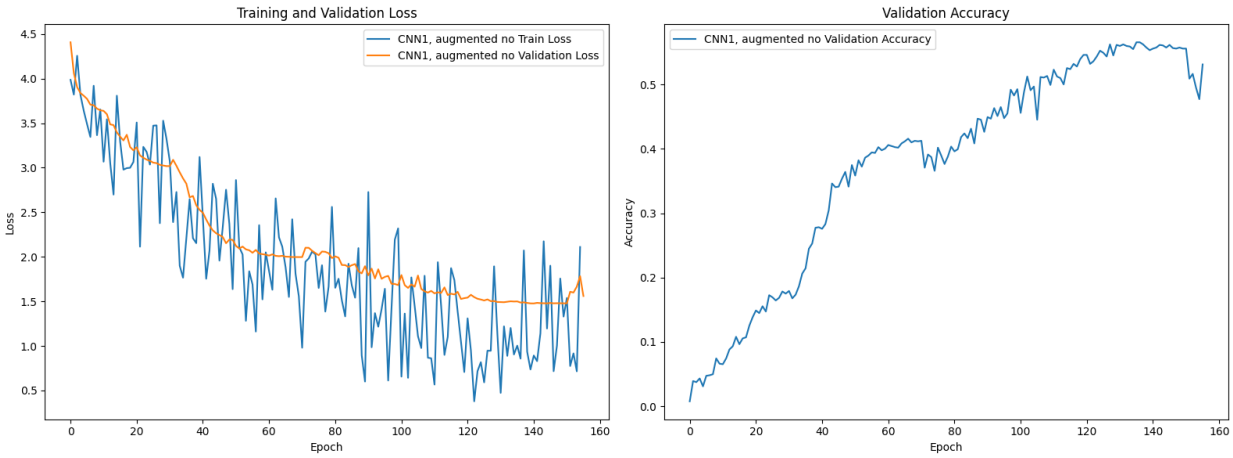


Figure 2: CNN 1 validation accuracy and loss at 15 patience and 200 max epochs

At the **144th epoch**, the best **validation accuracy** was **52%**, the **test accuracy** was **48%**, the best minimum **validation loss** was **1.62**, and the **total number of parameters** was **61.0K**.

Now change the values of **epochs** equal to **250** and an **early stopping patience** value equal to **25** the results are shown in the figures below**:**

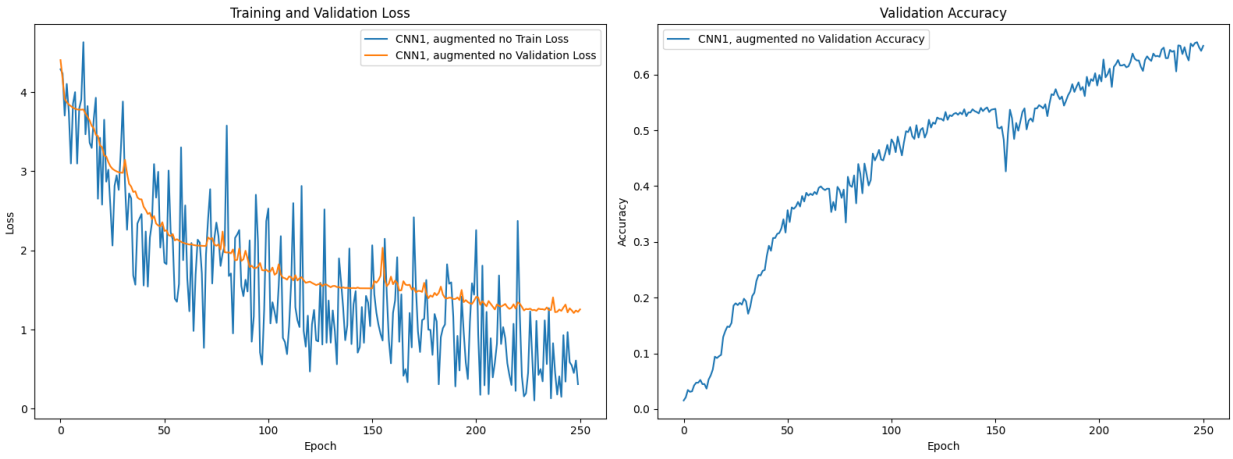


Figure 3: CNN 1 validation accuracy and loss at 25 patience and 250 max epochs

At the **245th epoch**, the best **validation accuracy** was **65.8%**, the **test accuracy** was **61.6%**, the best minimum **validation loss** was **1.2**, and the **total number of parameters** was **61.0K**.

Finally, change the values of **epochs** equal to **500** and an **early stopping patience** value equal to **50** the results are shown in the figures below**:**

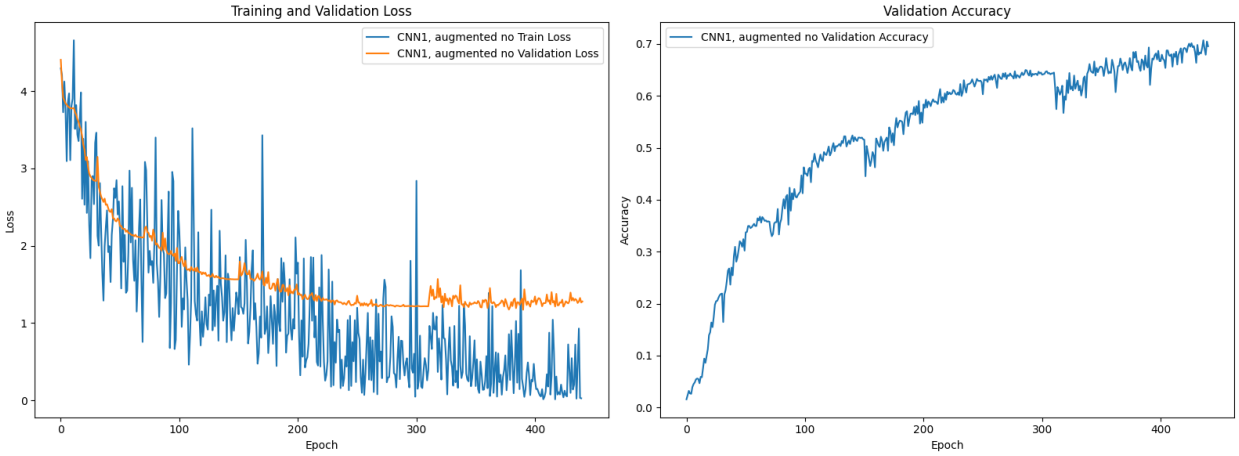


Figure 4: CNN 1 validation accuracy and loss at 50 patience and 500 max epochs

At the **391st epoch**, the best **validation accuracy** was **69.3%**, the **test accuracy** was **68.4%**, the best minimum **validation loss** was **1.17**, and the **total number of parameters** was **61K**.

CNN 2:

In this CNN we used **LightningModule** from **pytorch_lightning** library to build the CNN, then we identified the number of **convolutional layers** to be **4** with **kernel size** equals **3** and **padding** equals **1** with **stride** equals **1** for each one, the number of **Normalization Batch layers** to be **4**, **one Max-pooling layer** with **pooling window size** equals **2**, **one Adaptive average pooling layer**, and the number of **fully-connected layers** to be **2**. In the Forward Pass mechanism, we passed input through convolutional layers with **ReLU activation** and **max pooling**. Then we used global average pooling before flattening the feature map to a vector to be ready for a fully connected layer then using **Drop out** with **50%** to reduce the number of neurons through the training process. Finally, we applied fully connected layers and produced the final output. To summarize this CNN Architecture the process is done as follows:

In this step, learning rate, number of epochs, early stopping patience and both the optimizer and schedular remained the same as they were in CNN 1.

After the training step is done, both Loss and Validation Accuracy Vs. The number of Epochs is shown in the figures below:

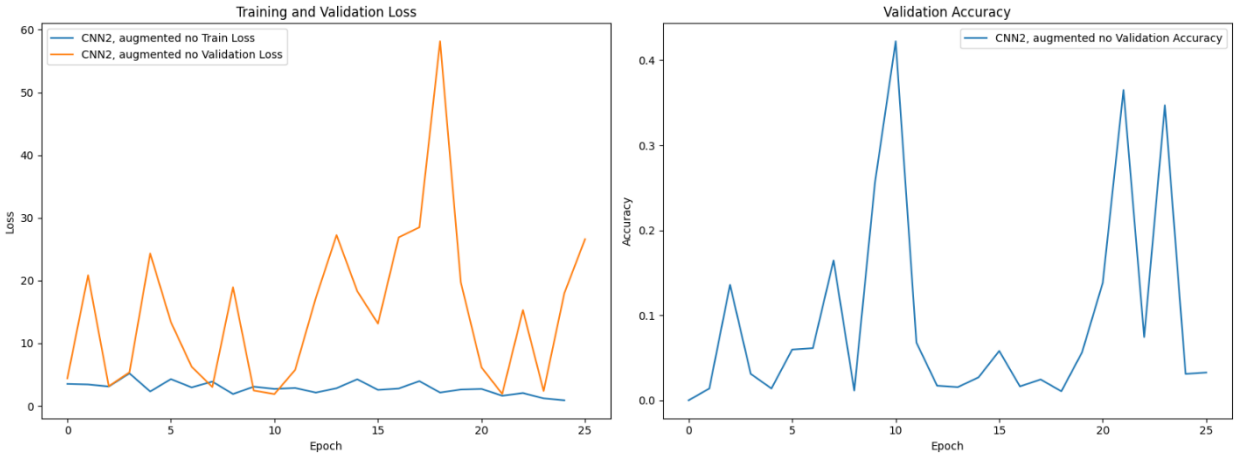


Figure 5: CNN 2 validation accuracy and loss at 15 patience and 200 max epochs

At the **8th epoch**, the best **validation accuracy** was **43%**, the **test accuracy** was **42%**, the best minimum **validation loss** was **1.9**, and the **total number of parameters** was **562.0K**.

Now change the values of **epochs** equal to **250** and an **early stopping patience** value equal to **25** the results are shown in the figures below**:**

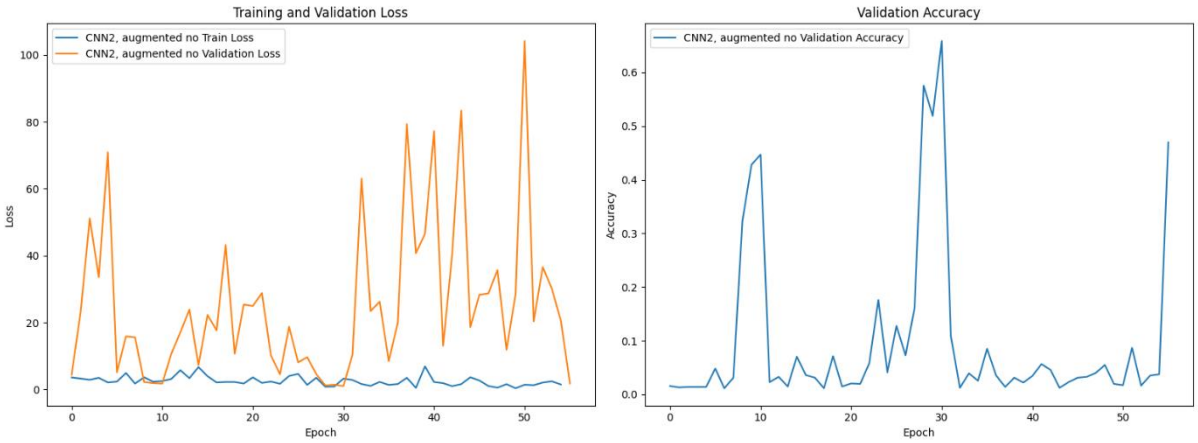


Figure 6: CNN 2 validation accuracy and loss at 25 patience and 250 max epochs

At the **28th epoch**, the best **validation accuracy** was **65.9%**, the **test accuracy** was **63.1%**, the best minimum **validation loss** was **1.03**, and the **total number of parameters** was **562.0K**.

Finally, change the values of **epochs** equal to **500** and an **early stopping patience** value equal to **50** the results are shown in the figures below**:**

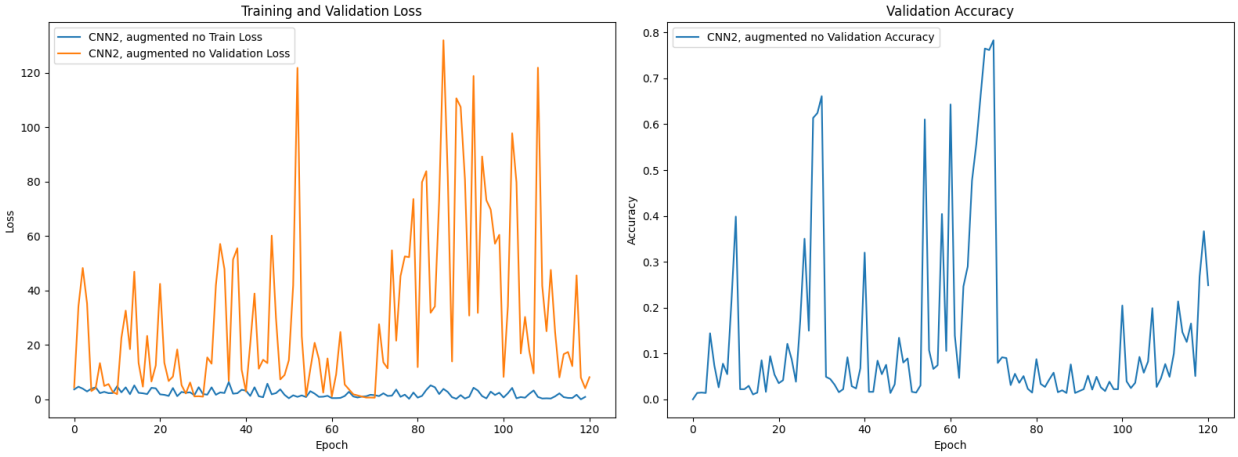
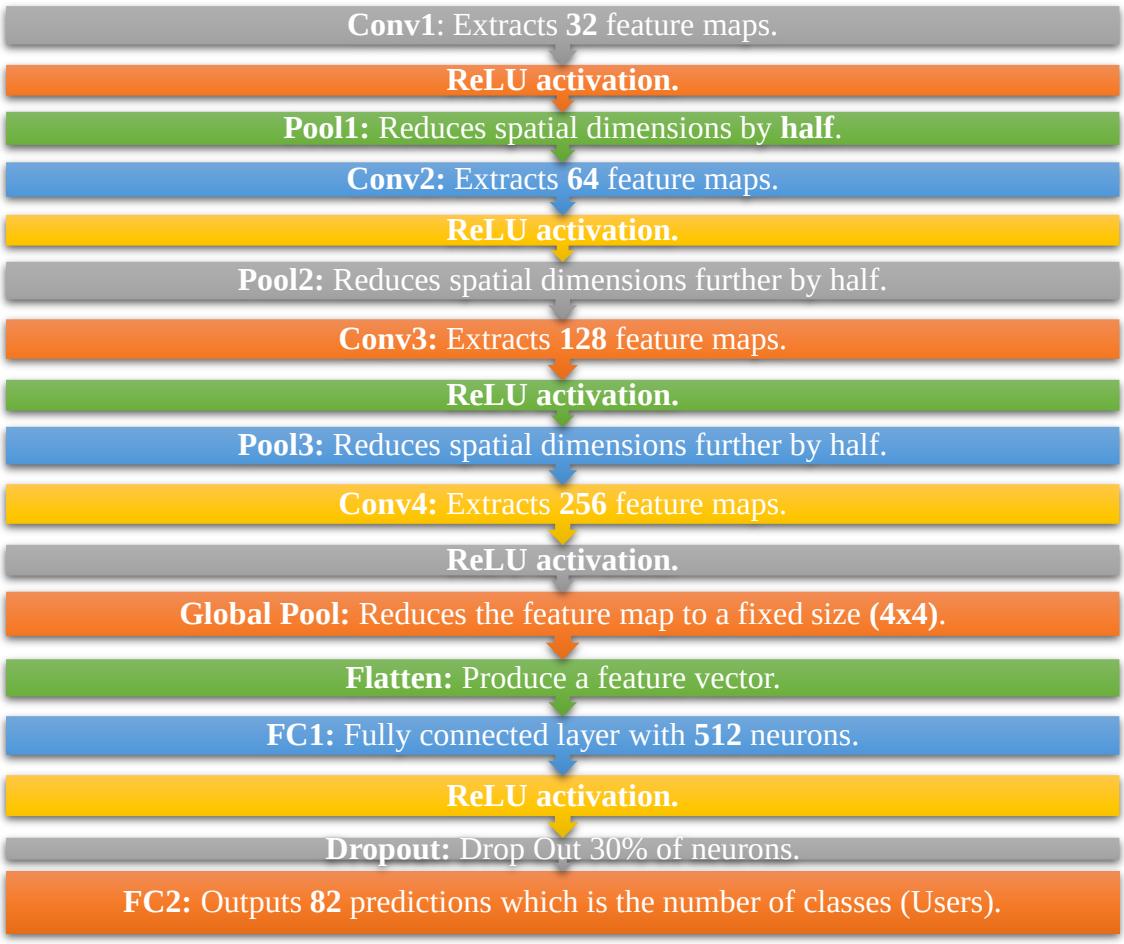


Figure 7: CNN 2 validation accuracy and loss at 50 patience and 500 max epochs

At the **71st epoch**, the best **validation accuracy** was **78.3%**, the **test accuracy** was **80.9%**, the best minimum **validation loss** was **0.61**, and the **total number of parameters** was **562K**.

CNN 3:

In this CNN we used **LightningModule** from **pytorch_lightning** library to build the CNN, then we identified the number of **convolutional layers** to be **4** with **kernel size** equals **3** and **padding** equals **1** with **stride** equals **1** for each one, the number of **Normalization Batch layers** to be **4**, **one Max-pooling layer** with **pooling window size** equals **2**, **one Adaptive average pooling layer** to reduce the feature map to a fixed size **(4x4)**, and the number of **fully-connected layers** to be **2**. In the Forward Pass mechanism, we passed input through convolutional layers with **ReLU activation** and **max pooling**. Then we used global average pooling before flattening the feature map to a vector to be ready for a fully connected layer then using **Drop out** with **30%**. Finally, we applied fully connected layers and produced the final output. To summarize this CNN Architecture the process is done as follows:



In this step, learning rate, number of epochs, early stopping patience and both the optimizer and schedular remained the same as they were in CNN 1.

After the training step is done, both Loss and Validation Accuracy Vs. The number of Epochs is shown in the figures below:

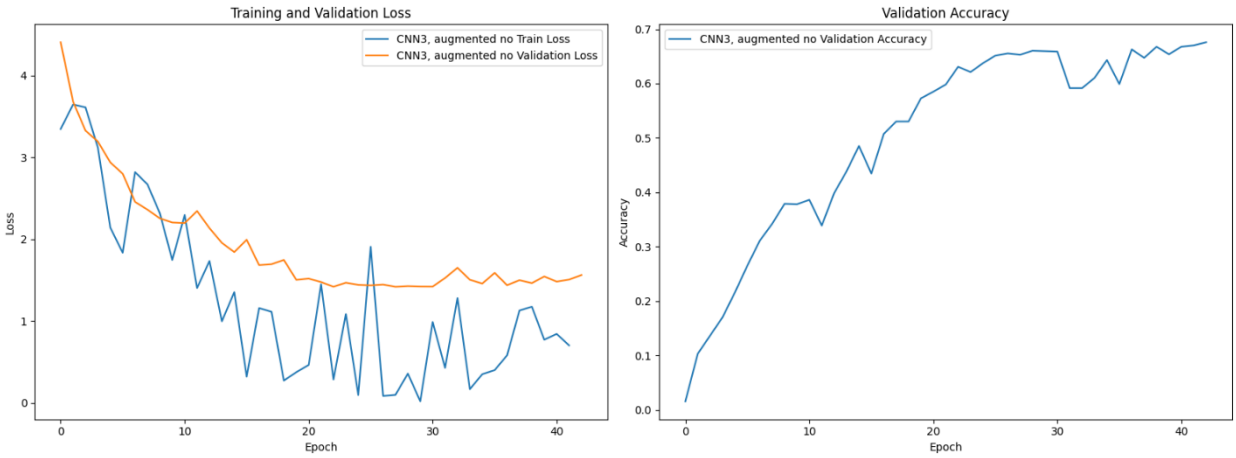


Figure 8: CNN 3 validation accuracy and loss at 15 patience and 200 max epochs

At the **25th epoch**, the best **validation accuracy** was **65%**, the **test accuracy** was **64%**, the best minimum **validation loss** was **1.4**, and the **total number of parameters** was **2.5M**.

Now change the values of **epochs** equal to **250** and an **early stopping patience** value equal to **25** the results are shown in the figures below**:**

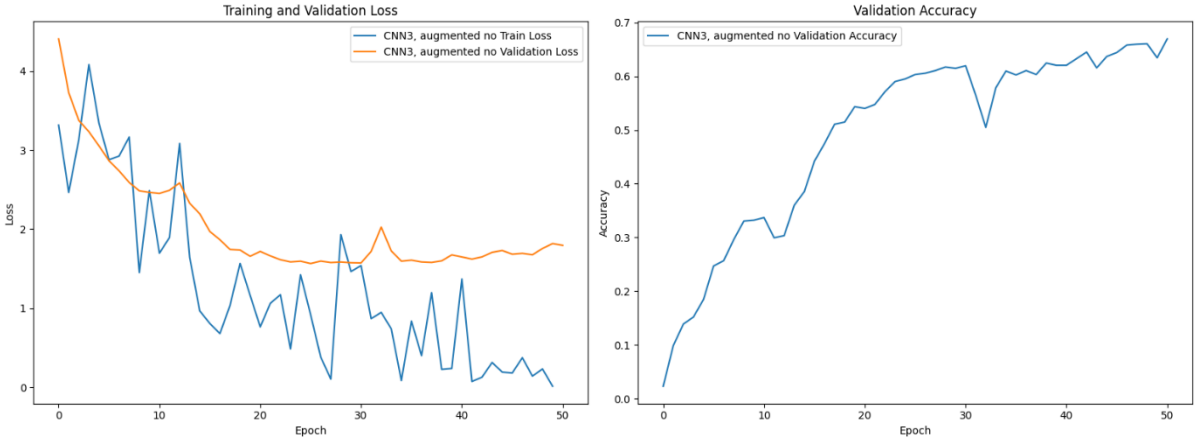


Figure 9: CNN 3 validation accuracy and loss at 25 patience and 250 max epochs

At the **23rd epoch**, the best **validation accuracy** was **60.3%**, the **test accuracy** was **61.5%**, the best minimum **validation loss** was **1.56**, and the **total number of parameters** was **2.5M**.

Finally, change the values of **epochs** equal to **500** and an **early stopping patience** value equal to **50** the results are shown in the figures below**:**

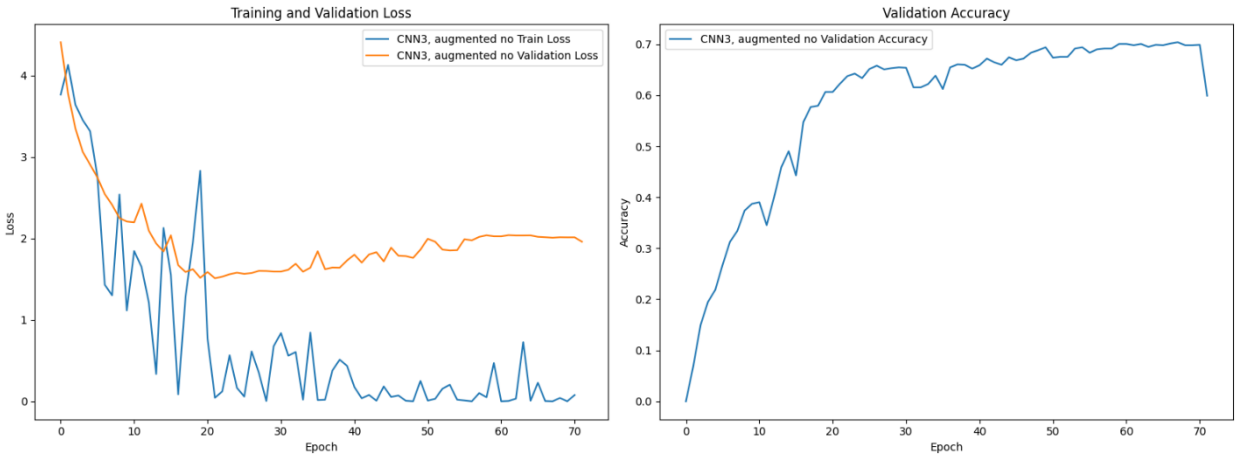


Figure 10: CNN 3 validation accuracy and loss at 50 patience and 500 max epochs

At the **22nd epoch**, the best **validation accuracy** was **62.3%**, the **test accuracy** was **61.2%**, the best minimum **validation loss** was **1.51**, and the **total number of parameters** was **2.5M**.

Task 1: Results Discussion

Impact of Model Complexity: We Notice that increasing model complexity through additional convolutional and fully connected layers (as seen in **CNN 2** and **CNN 3**) improved initial accuracy but introduced risks of overfitting. On the other hand, regularization techniques like **dropout** and **weight decay** helped mitigate this to some extent.

Training Stability: We found that **CNN 1** showed the most stable training curve, gradually improving over epochs, whereas **CNN 2** and **CNN 3** exhibited more rapid convergence but suffered from performance variations, especially with extended training epochs.

Optimization Techniques: We noticed that using both the optimizer and scheduler significantly improved the models' convergence and helped reduce overfitting. Specifically, we found that the restarting schedule allowed the models to explore local minima during training effectively.

Final Attempt: During the final attempt, we increased the maximum epochs to **500** and set the early stopping patience to **50**, which yielded improved results for all three CNNs.

Generalization Performance: We found that **CNN 2** achieved the best test accuracy of **80.9%** and a validation accuracy of **78.3%**. This highlights how batch normalization and dropout played a key role in reducing overfitting compared to **CNN 1** and **CNN 3**.

Task 2: Retrain Selected CNN After Doing Data Augmentation.

In this section depending on Task 1 results the best CNN was CNN 2 so we chose to retrain it with augmented images.

Data Augmentation

The List of Augmentations that were applied is:

Resizing: Resize each image to **224x224** to ensure all images is the same size.

Rotation: Rotate each image to **-15 and +15** to cover the variations in text orientation to improve robustness to minor rotations.

Translation: Horizontal and vertical shifts are **up to 5%** of the image size to handle minor positional changes.

Perspective Distortion: Limits the amount of distortion **up to 5%** and **20%** is the probability of applying this distortion to be robust to slight perspective changes.

Erasing: Randomly erases a rectangular region of the image with **a 10% probability** for generalization.

Tensor Conversion: Converts the image to a PyTorch tensor with **1 Greyscale** Channel.

Normalization: Normalize each image tensor by adjusting its pixel values with **0.5** value for both mean and standard deviation to ensure that scaled values are between **-1** and **1**, and this step helps improve model convergence during training.

There is below some samples of augmented images:

Figure 11: Samples of augmented images

In this step, learning rate, number of epochs, early stopping patience and both the optimizer and schedular remained the same as they were in CNN 1.

After the training step is done, both Loss and Validation Accuracy Vs. The number of Epochs is shown in the figures below:

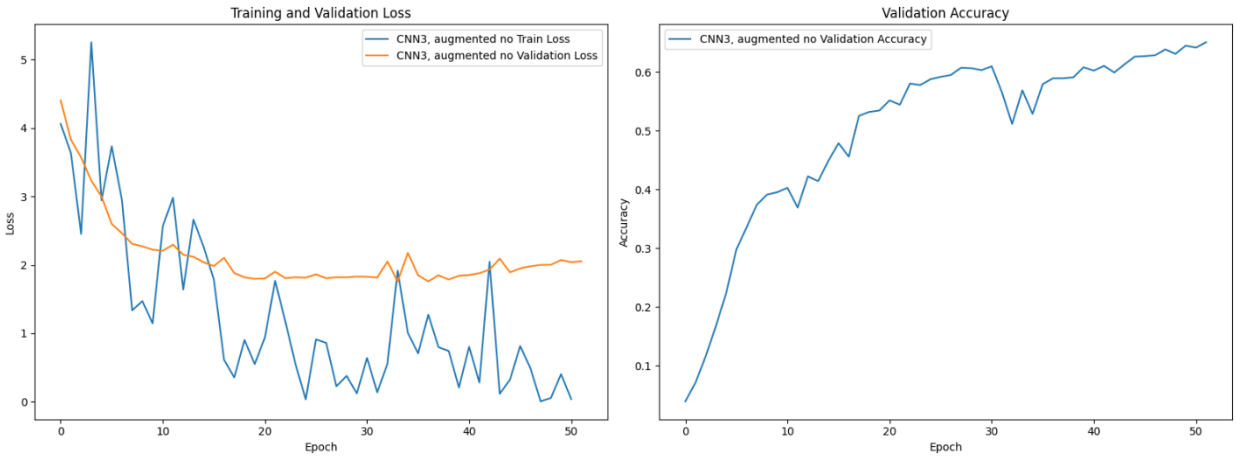


Figure 12: CNN 3 retrained validation accuracy and loss at 15 patience and 200 max epochs

At the **34th epoch**, the best **validation accuracy** was **59%**, the **test accuracy** was **54%**, the best minimum **validation loss** was **1.7**, and the **total number of parameters** was **2.5M**.

Now after changing the values of **epochs** equal to **250** and an **early stopping patience** value equal to **25,** the best CNN was **CNN2,** and the results are shown in the figures below**:**

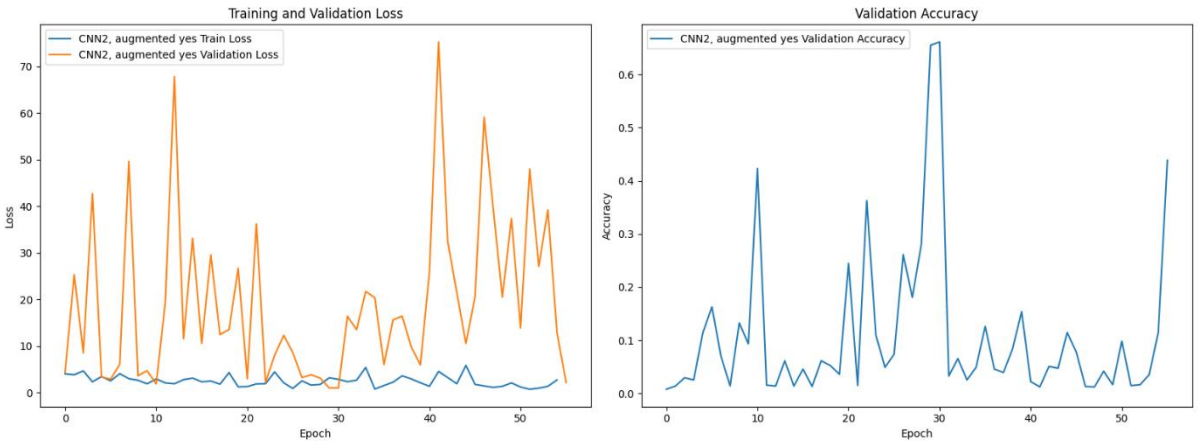


Figure 13: CNN 2 retrained validation accuracy and loss at 25 patience and 250 max epochs

At the **28th epoch**, the best **validation accuracy** was **66.1%**, the **test accuracy** was **65.8%**, the best minimum **validation loss** was **1.0**, and the **total number of parameters** was **562.0K**.

Finally, after changing the values of **epochs** equal to **500** and an **early stopping patience** value equal to **50,** the best CNN was **CNN2,** and the results are shown in the figures below**:**

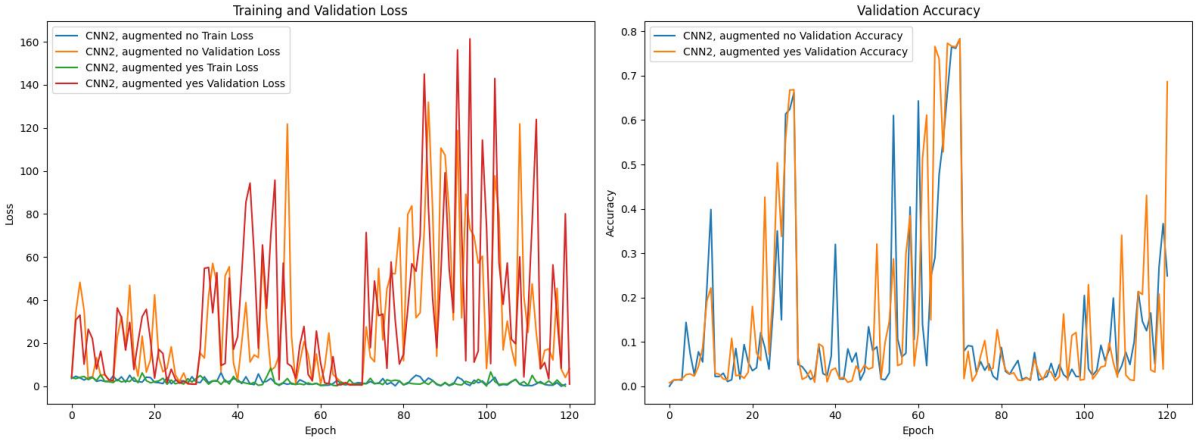


Figure 14: CNN 2 retrained validation accuracy and loss at 50 patience and 500 max epochs

At the **71st epoch**, the best **validation accuracy** was **78.3%**, the **test accuracy** was **79.4%**, the best minimum **validation loss** was **0.59**, and the **total number of parameters** was **562K**.

Task 2: Results Discussion

Effectiveness of Data Augmentation: We noticed that the data augmentation techniques we applied significantly increased the diversity of the training data. This, in turn, improved the model's ability to handle variations in **handwritten Arabic text**. We found this improvement reflected in the higher validation and test accuracies achieved by **CNN 2**.

CNN 2 vs. CNN 3: While **CNN 3** initially showed promise, we found that its high parameter count caused **overfitting**, especially when retrained with the augmented dataset. On the other hand, **CNN 2**'s more balanced architecture enabled it to perform much better and achieve superior results overall.

Improved Generalization: Retraining **CNN 2** with the augmented data decreased its test accuracy from **80.9%** to **79.4%** but also reduced its minimum validation loss to **0.59** at 500 epochs with 50 early stopping patience. This is a significant improvement compared to the non-augmented results in **Task 1**, indicating that the augmentation techniques enhanced CNN 2's ability to generalize to unseen data.

Optimization and Convergence: We found that the combination of the optimizer and scheduler allowed **CNN 2** to achieve its best performance in **fewer epochs** compared to **CNN 3**.

Task 3: Train Published CNN With Data Augmentation.

Data Augmentation and 3 Channels Conversion

The augmentation applied here is identical to Task 2, but the number of channels has been increased to 3 by duplication. Additionally, the normalization parameters have been changed to mean [0.4914, 0.4822, 0.4465], std [0.2023, 0.1994, 0.2010] to match ImageNet dataset for using it in transfer learning in Task 4.

ResNet-18 Setup

As the dataset is small, the choice fell onto the smallest Resnet model **Resnet-18**.

ResNet-18 is a convolutional neural network designed for image classification and feature extraction, known for its **residual connections**, which help mitigate the vanishing gradient problem in deep networks. The architecture consists of **18 layers**, including **convolutional layers, batch normalization, ReLU activations, and residual blocks**. The defining feature of ResNet-18 is its **skip connections**, which allow the network to learn identity mappings, improving gradient flow and enabling deeper architectures to train effectively.

The network begins with a **7×7 convolutional layer with 64 filters**, followed by **batch normalization and a 3×3 max pooling layer with a stride of 2**, reducing spatial dimensions early in the network. The core of ResNet-18 is composed of **four stages of residual blocks**, each containing two **3×3 convolutional layers**. The number of filters increases at each stage: **64, 128, 256, and 512 channels**, with downsampling performed using a stride of 2 at the beginning of each stage. Each residual block includes an **identity shortcut connection**, which allows the input to bypass the two convolutional layers, ensuring better gradient propagation.

At the final stage, the network applies **global average pooling**, reducing the spatial dimensions to **1×1 per channel**. This structure provides a balance between depth and efficiency, making ResNet-18 a strong feature extractor while maintaining a relatively low computational cost. Given the small dataset size, this model was chosen as the base architecture before incorporating modifications such as the SE block to enhance its feature representation capabilities.

Squeeze and Excitation Blocks

Squeeze-and-Excitation (SE) blocks introduce a channel-wise attention mechanism that enhances feature representations by adaptively recalibrating the importance of different channels. Each SE block consists of two main operations: **squeeze** and **excitation**. The squeeze operation applies global average pooling to aggregate spatial information into a single descriptor per channel. The excitation operation then passes this descriptor through two fully connected layers, with a ReLU activation between them, and a sigmoid activation at the end. This results in a set of learned channel-wise weights that are applied multiplicatively to the input feature map, allowing the network to emphasize more informative channels while suppressing less useful ones. This mechanism improves feature selectivity and enhances network performance with minimal computational overhead.

To integrate SE blocks into ResNet-18, each convolutional block is extended with a channel attention mechanism. The SE block first applies **adaptive average pooling with a (1,1) kernel size**, which condenses the spatial dimensions of each feature map into a single value per channel. This results in a **(C × 1 × 1) feature vector**, where **C** is the number of channels in the given convolutional layer. To reduce computational complexity and introduce non-linearity, this vector is passed through a **fully connected layer that reduces the channel count by a factor of 16** (i.e., **C/16**). This reduction ratio is a common choice, balancing expressiveness and efficiency. A ReLU activation function is then applied, followed by a second **fully connected layer that restores the original number of channels (C)**. The output of this layer is passed through a **sigmoid activation function**, ensuring that the learned scaling factors remain between 0 and 1. These per-channel scaling factors are then **reshaped to (C × 1 × 1) and broadcasted across the original feature map**, effectively reweighting each channel's contribution. By introducing this recalibration mechanism, the network can emphasize more informative feature channels while suppressing less relevant ones, leading to improved feature representation and classification performance.

Then the training was at a **learning rate** equal to **0.001** with several **epochs** equal to **500** and an **early stopping patience** value equal to **50**, the results are shown in the figures below:

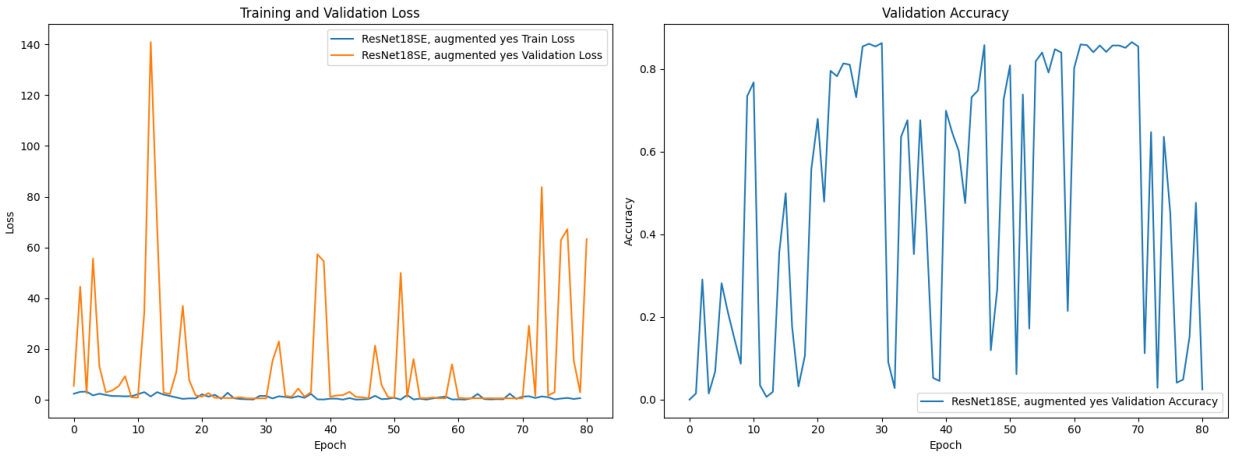


Figure 15: ResNet 18 + SE validation accuracy and loss

At the **31st epoch**, the best **validation accuracy** was **86.25%**, the **test accuracy** was **85.50%**, the best minimum **validation loss** was **0.4592**, and the **total number of parameters** was **11.2 M**.

Task 3: Results Discussion

When comparing the performance of **ResNet-18 with SE** in **this task** and **CNN 2** in **Task 2**, several observations that we found can be made. Both models benefited from data augmentation, which improved their robustness to variations in the dataset. However, we notice that **ResNet-18 with** SE **outperformed CNN2 in terms of validation and test accuracy**. As shown in the results, ResNet-18 with SE achieved a best validation accuracy of **86.25%** and test accuracy of **85.50%**, while CNN2 reached a validation accuracy of **78.31%** and test accuracy of **79.36%**. This improvement in accuracy comes at a cost of 20 fold increase in the number of parameters and heavier computation.

In summary, ResNet-18 with SE outperformed CNN2, achieving higher validation (**86.25%**) and test accuracy (**85.50%**) compared to CNN2 (**78.31%** and **79.36%**, respectively). However, this improvement came with a **20-fold increase in parameters** and higher computational costs. Given these trade-offs, ResNet-18 with SE is recommended when accuracy is the priority, while CNN2 remains a viable option for resource-constrained environments.

Task 4: Use Pre-trained CNN.

In this task, transfer learning has been applied to the model in Task 3. The pretrained weights on ImageNet-10K for the ResNet-18 have been loaded and all the layers have been trained on the augmented dataset described in Task 3 with the same hyperparameters.

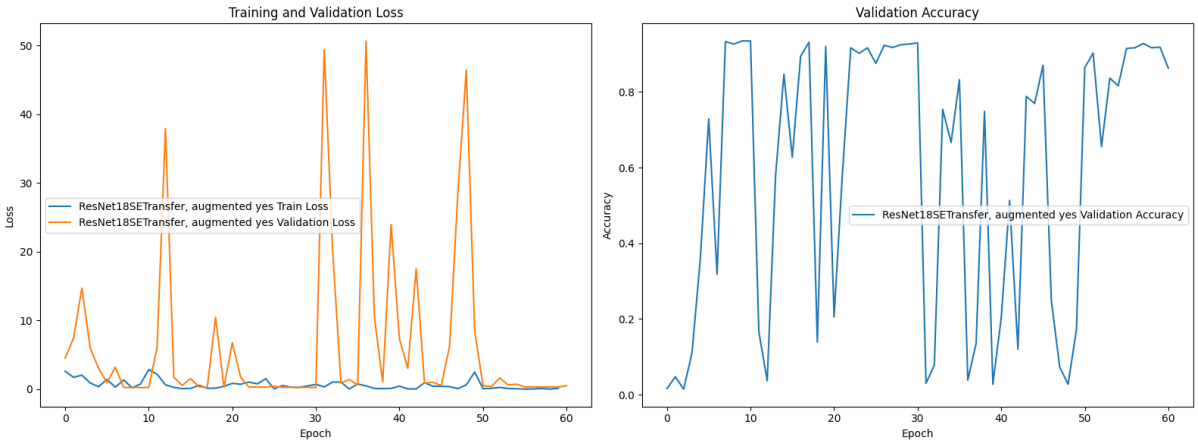


Figure 16: Resnet-18 + SE with Transfer validation accuracy and loss.

At the **10th** epoch, the best **validation accuracy** of **93.45%** was achieved, the **validation loss** is **0.2373, test accuracy** is **93.36%.**

Task 4: Results Discussion

Even though, ImageNet_10K is different from our dataset, transfer learning achieved a higher accuracy in a lower number of epochs compared to Task 3, where the model was trained from scratch (random initialization).

This result demonstrates the power of transfer learning in terms of both accuracy, training time and generalization. The result could be improved by finding a pretrained model on a dataset more related to handwriting, but we did not find anything other than CIFAR10 pretrained models that expect 32x32 input image. Resizing our dataset to 32x32 results in catastrophic loss of information.

CNN's Validation and Test Accuracy Comparison

In this section utilizing Bar Plot the validation and test accuracies of all trained CNNs mentioned in all tasks are shown in the figures below:

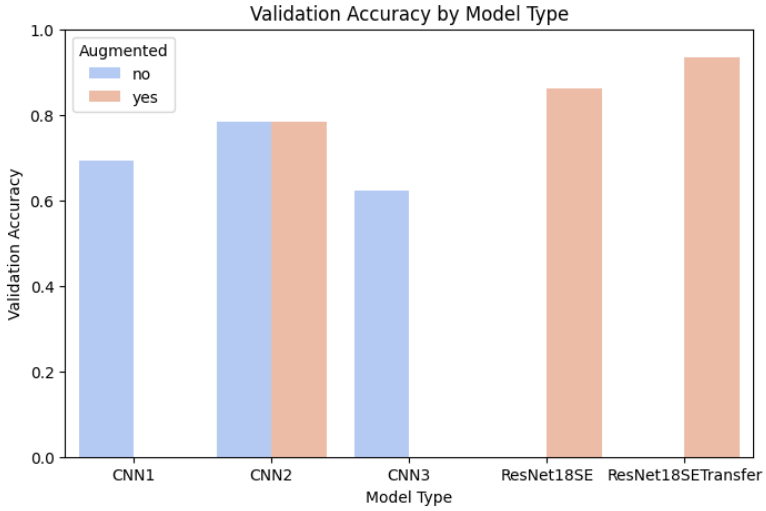


Figure 17 Best Validation Accuracy of all models.

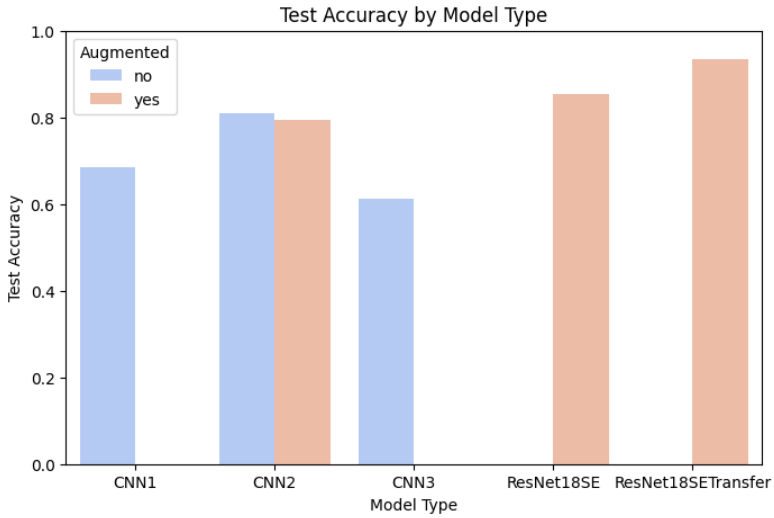


Figure 18 Testing Accuracy of all models.

Model	Train Accuracy	Validation Accuracy	Test Accuracy	Augmented
CNN1	0.904754	0.693126	0.683866	no
CNN2	0.922996	0.783142	0.809173	no
CNN3	0.947553	0.622750	0.611794	no
CNN2	0.927732	0.783142	0.793612	yes
ResNet18SE	1.000000	0.862520	0.855037	yes
ResNet18SETransfer	1.000000	0.934534	0.933661	yes

Table 0-1: All CNNs' Accuracies Summary

Conclusion:

In this study, **ResNet18SETransfer** achieved the highest validation accuracy of **93.45%**, with a test accuracy of **93.36%**, and was selected based on its superior validation performance. The transfer model demonstrated a **17%** improvement in validation accuracy compared to the same architecture without transfer learning, alongside significantly faster convergence (10 epochs compared to 31 epochs), highlighting the effectiveness of transfer learning. Even when pre-trained weights are unavailable, fine-tuning a model trained on large datasets such as ImageNet can lead to substantial gains in accuracy, making transfer learning a valuable approach.

For cases where model size is constrained, **CNN2 with augmentation** offers a lightweight alternative, achieving **78.31% validation** and **79.36% test accuracy**. However, CNN2 did not benefit from data augmentation, as its validation accuracy remained unchanged, and test accuracy slightly decreased. This suggests that **CNN2** may be underfitting the augmented data due to its lower number of parameters, which might limit its ability to capture the additional patterns introduced by augmentation. As a result, the model fails to improve with the added data, likely because it lacks the capacity to leverage the augmented images effectively.

An essential aspect of model performance is the careful selection and tuning of hyperparameters. In this study, we utilized the **AdamW optimizer**, which helps with regularization and improves generalization, and employed the **Cosine Annealing with Warm Restarts** scheduler to adjust the learning rate dynamically for better convergence. Hyperparameters such as the learning rate and the decision to implement early stopping were also tuned to optimize training. These choices, combined with the careful tuning of parameters, played a crucial role in achieving the high performance of **ResNet18SETransfer**.

Building a model completely from scratch, as seen with **CNN1 to CNN3**, can be time-consuming and often yields suboptimal results. In contrast, using pre-trained models or modifying them (e.g., by adding SE blocks) can significantly improve performance and speed up training. Therefore, for more reliable results and efficiency, utilizing pre-trained or modified models is recommended over starting from scratch.