Faculty of Engineering & Technology

Electrical & Computer Engineering Department

Computer Vision – ENCS5343

Course Project

Prepared by:

Mohammed Owda	1200089
Osaid Hamza	1200875

Instructor: Dr. Aziz Qaroush

Section: 2

Date: 25-1-2025

Birzeit

Abstract

This report addresses the challenge of Arabic handwritten word recognition using the AHAWP dataset, which contains 8,144 samples of 10 Arabic words written by 82 different individuals. Our objective is to identify the writer of each word image (an 82-class classification). The project follows four main tasks. First, we design and tune custom convolutional neural networks (CNNs) of varying depth and complexity, achieving notable performance after systematically exploring hyperparameters such as learning rate, batch size, optimizer, and dropout. Second, we demonstrate that applying data augmentation—through random rotation, scaling, and shifting—further improves accuracy by enhancing model generalization. Third, we train a well-known published architecture (ResNet-18) from scratch, adapted to grayscale input, and show that it surpasses our best custom CNN results. Finally, we employ transfer learning using a ResNet-50 model pretrained on ImageNet, partially fine-tuning the later layers while freezing earlier layers to leverage learned image features.

Experiments reveal a progressive improvement at each stage: the best custom CNN without augmentation attains around 65% test accuracy, while augmenting data pushes it to roughly 76%. Training ResNet-18 from scratch on augmented data elevates performance to approximately 90%. Fine-tuning a pretrained ResNet-50 yields the highest accuracy, nearing 99%, significantly outperforming the other approaches. These findings underscore the impact of data augmentation in reducing overfitting and highlight how transfer learning on robust pretrained backbones can substantially boost accuracy for handwriting classification tasks. This work demonstrates a promising approach for writer identification in Arabic handwriting, and the methods described can be extended to broader applications in handwriting analysis and recognition.

Table of Contents

List of Figures

List of Tables

1. Introduction

Handwritten text recognition is an important field of computer vision. It is more challenging than printed text recognition because writing styles, stroke pressure, and other characteristics differ greatly among writers. This becomes even more complex for scripts such as Arabic, which use connected letters with shapes that change according to position. Because of these variations, robust methods for recognizing handwritten Arabic words and identifying writers are needed. Such methods support applications in forensics, document management, and digital archiving.

In this project, the AHAWP (Arabic Handwritten Automatic Word Processing) dataset is used. This dataset contains 10 Arabic words, written by 82 individuals, with 10 samples per word, resulting in 8,144 total images. The dataset provides a wide range of handwriting styles, making it a good benchmark for studying local feature extraction and testing deep learning methods. The main objective is to classify each image according to its writer, leading to an 82-class classification problem.

Convolutional Neural Networks (CNNs) are employed because they learn complex features from images directly, which reduces the need for crafted features. However, it is necessary to pay attention to model architecture, hyperparameters, and overfitting, especially when using datasets that have variation in writing styles and limited samples per writer.

The experiments in this report are divided into four tasks. In the first task, different custom CNN architectures are built and tuned, and the one with the best performance is selected. In the second task, data augmentation is introduced to increase the effective size of the dataset and enhance generalization. In the third task, the selected custom CNN is compared to a well-known CNN architecture (ResNet-18) trained from scratch on grayscale input. In the final task, transfer learning is performed with a ResNet-50 model that was pretrained on ImageNet, which further increases accuracy by adapting powerful existing features to the handwriting domain.

2. Background

Handwritten word recognition tasks are often tackled using Convolutional Neural Networks (CNNs), which have demonstrated great success in image classification due to their ability to learn hierarchical features from raw pixel data. In the context of handwritten word recognition, CNNs learn local patterns and features such as edges, strokes, and shapes that are crucial for understanding the structure of characters and words. A CNN typically consists of several layers, including convolutional layers for feature extraction, activation layers to introduce non-linearity, pooling layers for downsampling, and fully connected layers for classification.

Convolutional Neural Networks (CNNs) are widely used in image classification tasks due to their ability to learn hierarchical features directly from raw pixel data. In handwritten word recognition, CNNs are particularly effective as they can automatically extract important features like edges, curves, and strokes, which are essential for distinguishing different characters. However, deeper

networks can suffer from challenges such as vanishing gradients, which hinder the learning process as the network depth increases.

Residual Networks (ResNet) offer a solution to this problem by introducing residual connections, which allow the model to bypass certain layers and focus on learning the residual, or the difference, between the input and output. This helps in maintaining gradient flow across deeper networks and enables the successful training of very deep models without performance degradation. ResNet architectures, such as ResNet-18 and ResNet-50, differ in the number of layers they contain, with ResNet-18 being a shallower model (18 layers) and ResNet-50 being deeper (50 layers). ResNet-50, with its increased depth, is able to capture more complex features and is therefore often preferred for tasks that involve high variability in input data, such as handwritten word recognition.

In the context of this project, transfer learning with a pre-trained ResNet-50 model is used. Transfer learning leverages a model that has already been trained on a large and diverse dataset, like ImageNet, and adapts it to the specific task at hand. This approach is particularly useful when the available dataset, such as the AHAWP dataset for Arabic handwritten word classification, is relatively small, as it allows the model to benefit from knowledge learned from a much larger dataset.

To further enhance model performance and prevent overfitting, data augmentation techniques are applied. This involves generating multiple variations of each image by applying transformations like rotation, scaling, flipping, and color adjustments. These augmented images effectively increase the dataset size, providing more diverse examples for the model to learn from. This helps improve generalization and ensures that the model does not memorize specific details of the training images, making it more robust when applied to new, unseen data.

3. Experimental Setup, Results, and Discussion

3.1. Experimental Setup

3.1.1. Hardware/Software

The experiments were conducted on Kaggle using two NVIDIA T4 GPUs, leveraging their parallel computing power for training deep learning models. The primary software stack consisted of PyTorch, OpenCV, NumPy, and Matplotlib. The code was implemented in Python, with training and validation routines managed using PyTorch's DataLoader objects for efficient batching and parallel data loading.

3.1.2. Dataset Loading and Preprocessing

The Arabic Handwritten Automatic Word Processing (AHAWP) dataset was used in these experiments. This dataset contains 10 distinct Arabic words written by 82 individuals, each providing 10 samples per word, resulting in 8,144 total images. The images were organized such that each user's samples reside in a dedicated directory. A custom function, load_images_custom_split, was employed to resize the images to 256×128 pixels, convert them to

grayscale, normalize pixel intensities to the [0,1] range, and perform an 80–20 train-test split. This split was done on a per-user-word basis, ensuring that each user's images for each word were divided proportionally between training and testing subsets. PyTorch's Dataset and DataLoader classes were then used to handle the final data preparation for the models.

3.1.3. Evaluation Methodology

The evaluation of the models was based on accuracy and loss. Accuracy measures the percentage of correct predictions made by the model on the test set, while loss measures the difference between the model's predictions and the true values. The Cross-Entropy Loss was used as the loss function, which is commonly used for multi-class classification tasks.

During training, the models were evaluated after every epoch by calculating both the training and validation (test) accuracy. The training accuracy shows how well the model performs on the training data, while the validation accuracy shows how well it generalizes to unseen data. By tracking both of these metrics, the effectiveness of the model could be monitored, and potential overfitting could be detected if there was a large gap between training and validation accuracy.

The training procedure involved running each model for several epochs, where an epoch refers to one full pass through the entire training dataset. The training time, loss, and accuracy were recorded after each epoch to evaluate how well the model was learning over time. At the end of each training session, the final test accuracy was calculated, which was the main metric used to compare different models and configurations.

3.2. Task 1: Build Custom CNN Architectures

3.2.1. Architecture Details

Three custom Convolutional Neural Network (CNN) architectures for writer identification (CustomCNNv1, CustomCNNv2, and CustomCNNv3) were designed to explore different depths and numbers of convolutional filters. Each variant employed standard convolutional blocks that included a 3×3 convolution, a ReLU activation, and a 2×2 max-pooling operation to reduce spatial dimensions. After the final convolutional stage, an adaptive average pooling layer was inserted to produce a fixed-size representation. Fully connected layers then projected these feature vectors to the 82-class output. Table 1 shows the architecture details of the three custom CNN variants.

Table 1: Architecture details for the three custom CNN variants used in the project.

Architecture	Layers	Channels	Fully Connected Layers	Additional Details
CustomCNNv1	3 convolutional blocks (Conv → ReLU → Pool)	[1 → 16 → 32 → 64]	FC(64×4×4 → 128) → FC(128 → 82)	AdaptiveAvgPool2d used before FC layers to pool feature maps to 4×4
CustomCNNv2	Similar to v1 but with more filters	[1 → 32 → 64 → 128]	FC(128×4×4 → 256) → FC(256 → 82)	Designed to learn richer features while retaining a 3-block structure
CustomCNNv3	Adds a fourth convolution block	[1 → 32 → 64 → 128 → 256]	FC(256×4×4 → 256) → FC(256 → 82)	Larger model capacity to capture more complex handwriting variations

3.2.2. Hyperparameter Tuning and Training Procedure

Various sets of hyperparameters were tested to optimize the performance of the three custom CNN architectures. Hyperparameters included the learning rate (ranging from 5e−5 to 1e-3), optimizer type (Adam, SGD, or RMSprop), weight decay values (0, 1e-4, or, 1e-3), batch sizes (32 or 64), epoch counts (15, 20, 25, or 30), and dropout probabilities (0.0, 0.2, or 0.5). Each set of hyperparameters was applied to all three architectures. The training process tracked both training and validation losses and accuracies to evaluate model performance.

An optimizer is an algorithm used to adjust the weights of the model during training in order to minimize the loss function. Weight decay is a regularization technique that adds a penalty term to the loss function to discourage large weights, helping to prevent overfitting. Dropout is a technique where random neurons are "dropped" or turned off during training to prevent the model from relying too heavily on specific neurons, thus improving generalization.

After experimenting with multiple combinations, the best-performing hyperparameters for each model were selected based on the highest final test accuracy. The following table summarizes the best hyperparameter set for each architecture, along with the corresponding final test accuracy and training time.

Table 2: Hyperparameter Tuning on the custom CNN architectures results

Model	**Best Hyperparameter Set**	**Final Test Accuracy (%)**	**Training Time (sec)**	**Train Loss**	**Test Loss**
CustomCNNv1	Learning Rate: 0.001, Batch Size: 32, Epochs: 25, Optimizer: Adam, Dropout: 0.0	56.35%	158.36	0.8864	1.7287
CustomCNNv2	Learning Rate: 0.001, Batch Size: 32, Epochs: 30, Optimizer: Adam, Dropout: 0.2	65.07%	218.24	0.8561	1.2682
CustomCNNv3	Learning Rate: 0.001, Batch Size: 32, Epochs: 20, Optimizer: RMSProp, Dropout: 0.2	53.71%	162.27	1.6309	1.6807

The following figures display the respective loss and accuracy curves for all tested hyperparameter combinations, helping to compare the training behavior and generalization abilities of each model.

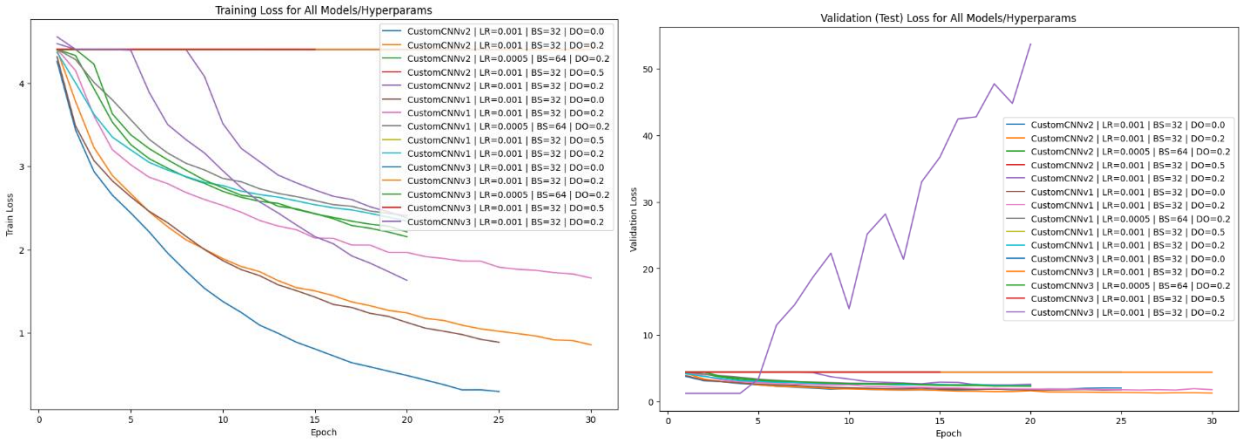

Figure 1: Training Loss and Validation Loss curves for all custom models and hyperparameters.

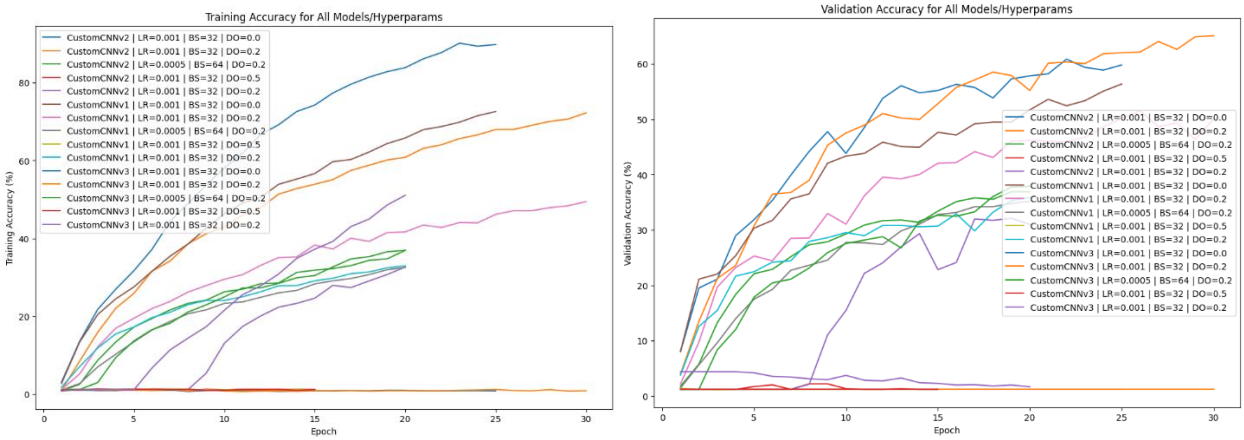

Figure 2: Training Accuracy and Validation Accuracy curves for all custom models and hyperparameters.

The hyperparameter tuning results show that CustomCNNv2, despite being the simplest among the three architectures, produced the best results with a final test accuracy of 65.07%. This suggests that deeper models may not always be the best choice for handwritten word recognition tasks. CustomCNNv2 achieved the highest performance with a moderate dropout of 0.2, 0.001 learning rate, 32 batch size, 30 epochs, and Adam optimizer, indicating that it struck an optimal balance between complexity and regularization. In contrast, CustomCNNv1, with its simpler design, reached a lower accuracy of 56.35%, while CustomCNNv3, despite being the most complex model, underperformed with an accuracy of 53.71%, highlighting the challenges of overfitting and inefficient capacity utilization in deeper networks.

3.3. Task 2: Retrain Selected Network with Data Augmentation

In Task 1, CustomCNNv2 yielded the best accuracy (65.07%) when trained without any data augmentation. To further enhance model generalization, an offline data augmentation pipeline was introduced. And in order to maintain a fair comparison, the same best hyperparameters from Task 1 (for CustomCNNv2) were used in this task.

3.3.1. Data Augmentation Methodology

Data augmentation is used to artificially enlarge the training dataset by applying random transformations to the original images. This helps the model generalize better to unseen data by exposing it to various perturbations. Given the characteristics of handwritten Arabic text (where small rotations, shifts, and minor shape distortions are realistic variations), the following transformations were applied:

- Rotation: A random rotation of up to ±10 degrees.
- Scaling: A random scale factor between 0.90 and 1.10.
- Translation: A random shift up to 2% of the image dimensions in both x and y directions.
- Shear: A random shear of up to 2°.
- Fill Color: Any region outside the original image boundary due to these transformations is filled with white (grayscale value 255).

These transformations were applied using a RandomAffine transform in PyTorch. Each original image was augmented twice (n_aug=2), effectively tripling the original training set size (the original image plus two augmented versions).

3.3.2. Data Augmentation Methodology

Instead of applying augmentation on-the-fly (dynamically each epoch), an offline approach was used:

Original Dataset:

- Training samples: 6,515 images
- Testing samples: 1,629 images

Offline Augmentation:

- Each training image is saved to disk along with its two augmented versions.
- The final augmented training set contains 19,545 images (3× the size of the original train set).

By generating the augmented dataset offline, we avoid recomputing transformations each epoch, at the cost of additional storage space. However, this approach makes training faster since reading augmented images from disk is typically faster than applying multiple geometric transforms on-the-fly during training.

3.3.3. Results and Discussion

To quantify the impact of data augmentation, we compare the key metrics obtained from Task 1 (no augmentation) and Task 2 (with augmentation). Table 3 summarizes the final test accuracies and other relevant statistics.

Table 3: Comparison of training the model with and without augmentation

Model Setup	Final Test Accuracy (%)	Training Set Size	Training Time (sec)
No Augmentation	65.07	6,515 images	~218
With Augmentation	76.55	19,545 images (3×)	~611

- **Accuracy:** Incorporating offline data augmentation boosted the final test accuracy from 65.07% to 76.55%, demonstrating significantly better generalization.
- **Dataset Size:** The training set was effectively tripled (from 6,515 to 19,545 images), giving the model exposure to a wider variety of handwriting distortions.
- **Training Time:** Training took longer (611 seconds vs. 218 seconds) due to the enlarged training dataset. This trade-off is typically acceptable given the substantial gain in performance.

Figures below illustrate the training and validation curves for both the non-augmented (Task 1) and augmented models (Task 2):

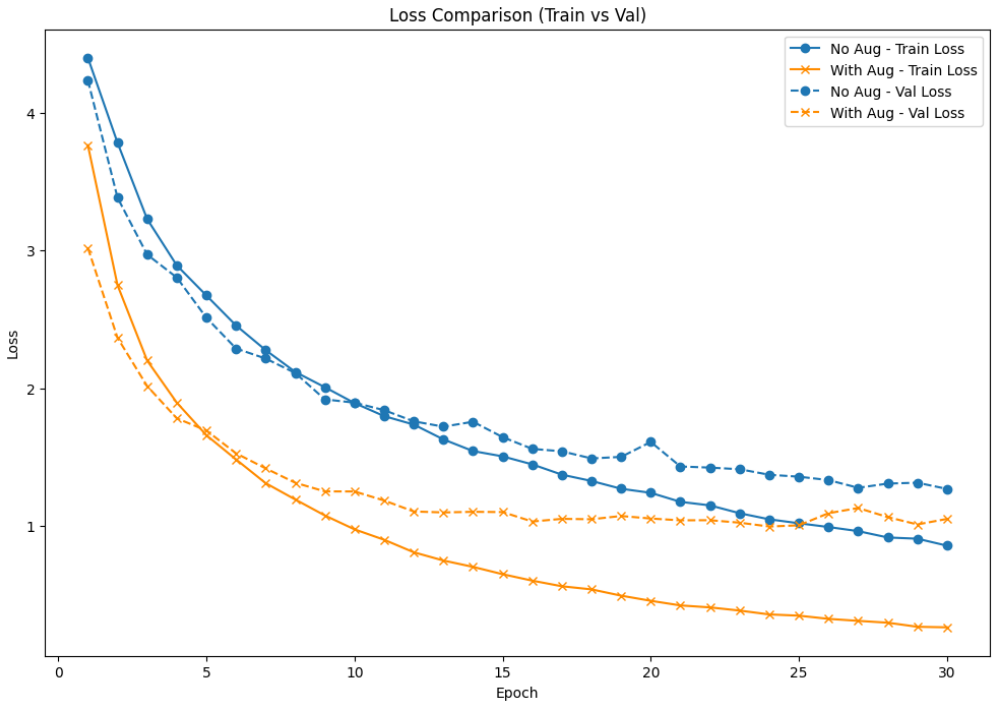

Figure 3: Loss Comparison (Train vs. Val) for No Aug vs. With Aug)

Figure 3 shows that the model with augmentation converges to a lower train and validation loss compared to the non-augmented model.

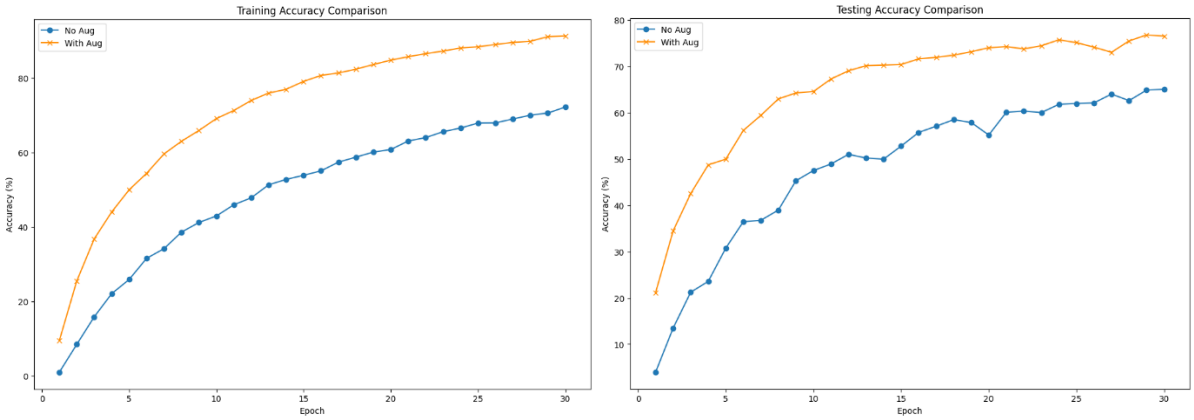

Figure 4: Training and Testing Accuracy Comparison for No Aug vs. With Aug)

Figure 4 Illustrates that training and testing accuracy improves more rapidly and reaches a higher final value with augmentation.

Overall, the use of offline data augmentation led to a substantial improvement in the model's generalization capabilities. By tripling the size of the training set and introducing realistic perturbations (rotations, translations, scaling, and shearing), the network learned more robust features, resulting in an 11.48% increase in test accuracy (from 65.07% to 76.55%). While training time increased from approximately 218 seconds to 611 seconds, the trade-off was justified by the significantly enhanced performance.

These results underscore the critical role of data augmentation in addressing the challenges of limited and varied handwriting datasets. By generating synthetic variations, data augmentation offers a cost-effective solution to expand datasets and improve model robustness, especially in scenarios where data collection is expensive or constrained. This approach helps to mitigate overfitting, and delivers a more accurate and reliable model for writer identification tasks.

3.4. Task 3: Training a Well-Known CNN Architecture (ResNet-18)

In the previous tasks, a custom CNN (CustomCNNv2) achieved a 76.55% test accuracy after applying data augmentation. In this task, we compare the performance of a well-known, published CNN architecture - ResNet18 - trained from scratch on the augmented dataset (grayscale images).

3.4.1. Adapting ResNet-18 for Grayscale

ResNet-18 in PyTorch is designed for 3-channel (RGB) images. To handle 1-channel (grayscale) images, the first convolution layer is modified to have only one input channel. The final fully connected (FC) layer is also replaced to produce 82 output classes, matching the number of writers in the dataset. Other components (residual blocks, batch normalization, skip connections) remain as in the standard ResNet-18 design.

3.4.2. Training Setup

To ensure a fair comparison, we use the same offline-augmented dataset generated in Task 2. However, images are now resized to 224×224 and normalized with mean=0.5 and std=0.5 to match the typical ResNet preprocessing.

Hyperparameters used: 5e-5 Learning Rate, 25 Epochs, Adam Optimizer, 1e-4 Weight Decay, and 32 Batch Size.

We also apply a learning rate scheduler that reduces the learning rate during training to aid convergence.

3.4.3. Results and Discussion

Below, Table 4 compares the performance of ResNet-18 (trained on augmented data) with the custom CNN from Task 2.

Table 4: Custom CNN vs ResNet-18 on the Augmented dataset

Model	Final Test Accuracy (%)	Training Time (sec)
Custom CNN + Aug (Task 2)	76.55	~611
ResNet-18 + Aug (Task 3)	90.12	~1,208

- **Accuracy:** ResNet-18 achieved 90.12% test accuracy on the augmented dataset, surpassing the 76.55% of the custom CNN, demonstrating the advantage of deeper residual architectures in capturing complex handwriting features.
- **Training Time:** Training ResNet-18 took ~1,208 seconds, longer than the custom CNN due to its deeper architecture and higher-resolution images (224×224). Despite the added cost, the performance gains make it worthwhile.

As shown in Figures below, ResNet-18 converges to a lower loss and a higher accuracy on both training and validation sets, confirming the advantage of a well-optimized, deeper architecture when sufficient data (including augmented samples) is available.

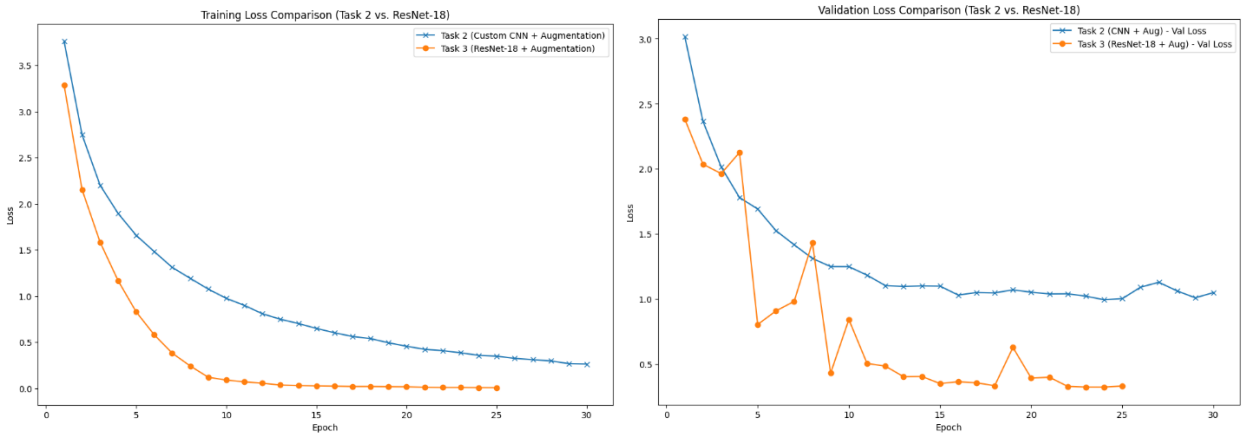

Figure 5: Training and Validation Loss Comparison (Custom CNN vs. ResNet-18)

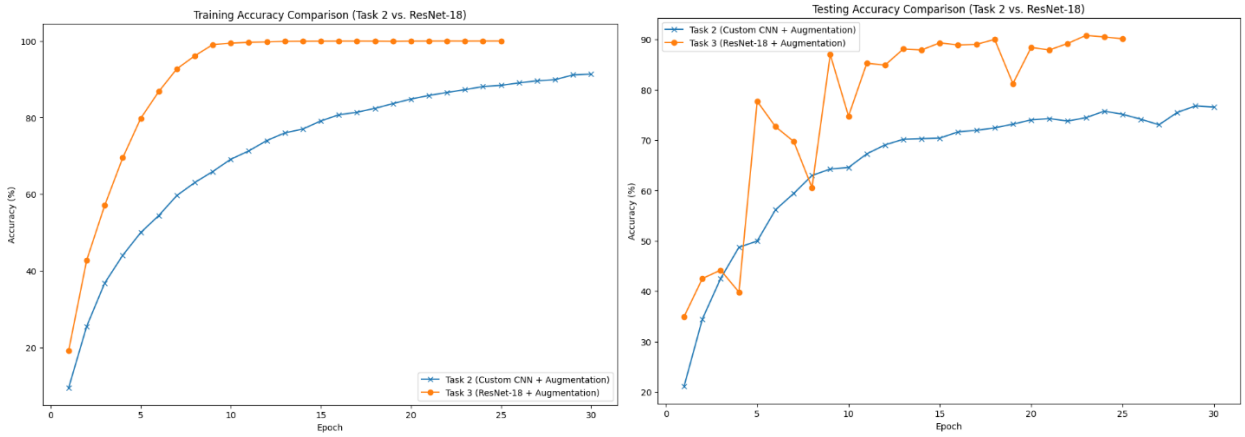

Figure 6: Training and Validation Accuracy Comparison (Custom CNN vs. ResNet-18)

The results indicate that ResNet-18, as a well-established published CNN architecture, significantly outperforms the custom CNN in terms of final test accuracy, reaching 90.12% compared to 76.55%. This demonstrates the effectiveness of published CNN networks in extracting richer, more robust features from complex handwriting data, especially when trained on an augmented dataset. While ResNet-18 requires a longer training time due to its deeper structure and higher input resolution (224×224), the notable accuracy gains justify the increased computational cost. Overall, leveraging a mature, deeper architecture proves advantageous in this handwritten writer-identification task, confirming that state-of-the-art CNN designs are well suited to capturing subtle variations present in Arabic handwriting.

3.5. Task 4: Transfer Learning with a Pretrained Network (ResNet-50)

In this final task, we explore the use of transfer learning to further improve performance beyond the scratch-trained models tested in previous tasks. Specifically, we employ a ResNet-50 model that has been pre-trained on the large ImageNet dataset, then adapt (fine-tune) it to the 82-class writer-identification problem on our augmented AHAWP dataset.

3.5.1. Building and Freezing ResNet-50

A pre-trained ResNet-50 architecture was loaded from PyTorch's model library, which provides weights originally trained on ImageNet. We then replaced the final fully connected layer with a new classifier for 82 output classes. Additionally, to reduce the risk of overfitting on our moderate-sized handwriting dataset, we partially froze the network:

1. **Frozen Layers**: Early layers (e.g., conv1, layer1, layer2) remain frozen, keeping their ImageNet-trained weights intact.
2. **Trainable Layers**: We unfroze the deeper layers (layer3, layer4) plus the final FC layer to allow the model to adapt high-level features to Arabic handwriting.

By doing so, the model retains general visual features learned from a massive dataset (ImageNet) while fine-tuning only a subset of parameters for the new domain. This approach is particularly

effective when data is not as extensive as in ImageNet, balancing the benefits of pretraining with the need to specialize in handwriting features.

This partial fine-tuning allows the model to retain the general visual features learned from ImageNet while still adapting to the specific characteristics of Arabic handwriting.

3.5.2. Preprocessing and Data Loading

To fully leverage the pretrained weights, we did not modify ResNet-50's first convolution layer to 1 channel. Instead, we converted each grayscale image into a 3-channel format, replicating the single channel into R, G, and B. This preserves the consistency of the model's first-layer filters, allowing it to use its pretrained parameters effectively. We also normalized the data according to ImageNet standards: Mean of 0.485, and standard deviation of 0.229 on each channel.

Images were resized to 224×224 to match the ResNet-50 input dimensions. By combining these steps with the same offline-augmented dataset from Task 2, the model was primed to leverage its learned ImageNet features effectively.

3.5.3. Fine-Tuning Setup

Key training hyperparameters included:

- Learning Rate: 1e-4
- Epochs: 25
- Optimizer: Adam
- Weight Decay: 1e-4
- Batch Size: 32

A learning rate scheduler was also used to decay the learning rate over epochs, ensuring stable convergence.

3.5.4. Results and Discussion

Table 5 compares the performance of the pretrained ResNet-50 (Task 4) with the ResNet-18 from Task 3, both trained on the augmented dataset.

Table 5: Comparison of the pretrained ResNet-50 with the ResNet-18 from scratch

Model	**Final Test Accuracy (%)**	**Training Time (sec)**
ResNet-18 + Aug (Task 3)	90.12	~1,208
ResNet-50 (Pretrained) + Aug (T4)	98.77	~2,292

- **Accuracy:** Fine-tuning a ResNet-50 pretrained on ImageNet achieved 98.77% test accuracy, outperforming the 90.12% of scratch-trained ResNet-18, highlighting the advantage of leveraging pretrained models for Arabic handwriting.

- **Training Time:** Fine-tuning ResNet-50 took ~2,292 seconds, nearly double ResNet-18's time. Despite the higher cost, the near-perfect accuracy justifies the trade-off.

The following figures show both training/validation loss comparisons and training/validation accuracy comparisons for Task 3 (ResNet-18) and Task 4 (Pre-Trained ResNet-50).

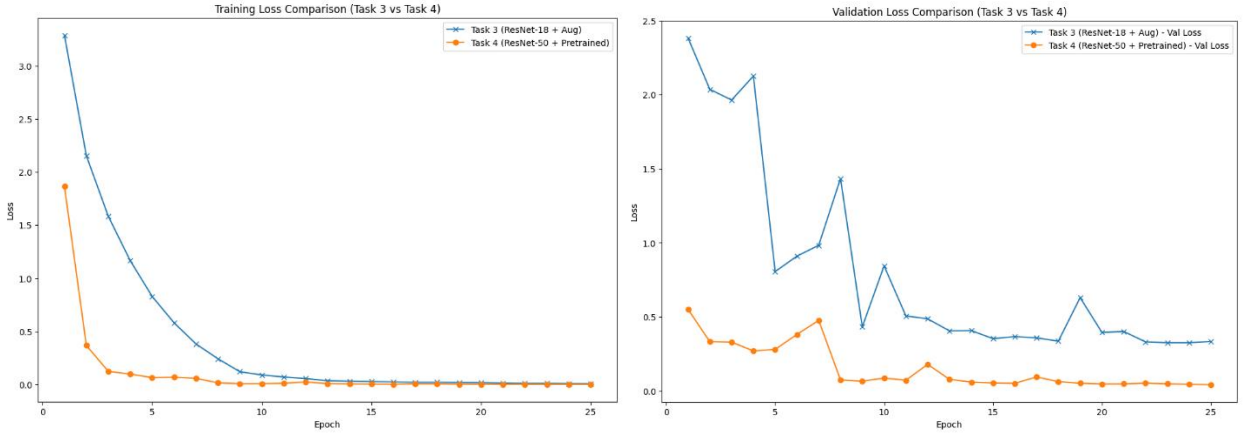

Figure 7: Training and Validation Loss Comparison (ResNet-18 vs pre-trained ResNet-50)

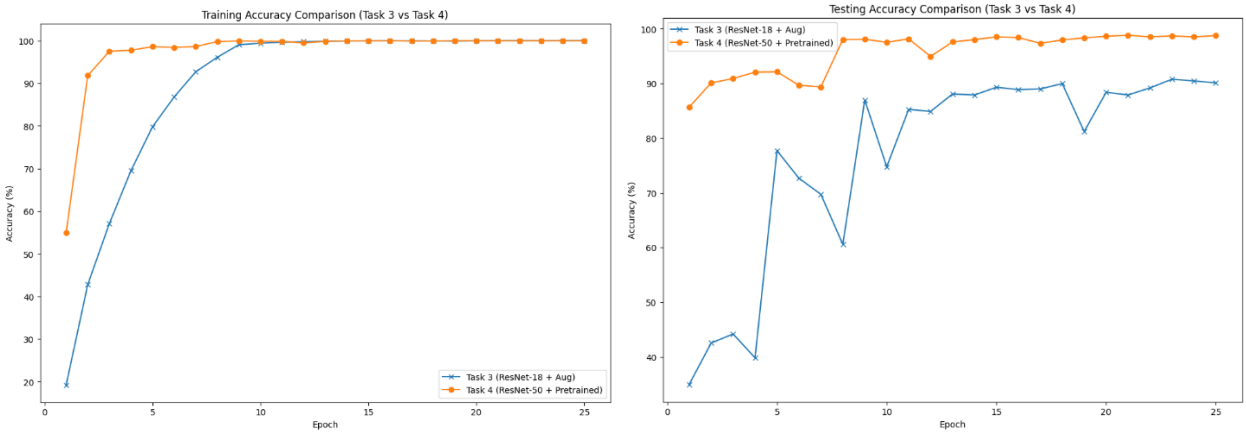

Figure 8: Training and Validation Loss Comparison (ResNet-18 vs pre-trained ResNet-50)

Leveraging a pretrained ResNet-50 offers a clear advantage over training a deep network from scratch, particularly when the available dataset is moderate in size. By reusing the robust features learned from a massive dataset like ImageNet, the model starts from a highly effective baseline for recognizing generic visual patterns—edges, shapes, and textures—without having to learn these from the ground up. The partial fine-tuning strategy enables specialized adaptation to Arabic handwriting in the final layers while preserving core feature-extraction capabilities in the earlier layers. This approach dramatically boosts performance, achieving a 98.77% final accuracy compared to 90.12% with a scratch-trained ResNet-18, showcasing the effectiveness of using a pretrained network. Although fine-tuning a larger architecture like ResNet-50 demands more computation time, the near-perfect results illustrate that it is usually more beneficial to start with a pretrained model. Doing so accelerates convergence, reduces overfitting risks, and capitalizes on knowledge already embedded in the network's weights from extensive prior training.

4. Conclusion and Future Work

This project explored multiple approaches for Arabic handwriting-based writer identification using the AHAWP dataset. Four main tasks were undertaken to progressively improve model performance. First, three custom CNN architectures were designed and evaluated, with CustomCNNv2 striking the optimal balance between depth and complexity, yielding a 65.07% accuracy. Second, introducing an offline data augmentation pipeline (including rotations, scaling, translation, and shearing) significantly boosted accuracy from 65.07% to 76.55% by effectively tripling the size of the training set and exposing the model to more handwriting variations. Third, a ResNet-18 model trained from scratch on the augmented dataset outperformed the custom CNN by achieving 90.12% accuracy, highlighting the advantage of deeper residual architectures. Finally, transfer learning with a ResNet-50 pretrained on ImageNet achieved the highest accuracy at 98.77%, demonstrating the power of reusing robust, generic visual features for specialized tasks like Arabic writer identification.

Collectively, these results emphasize the critical importance of data augmentation for addressing the limited and diverse nature of handwriting datasets, as well as the substantial benefits of transferring knowledge from large-scale, pretrained models. By combining these strategies, the project achieved a near-perfect level of performance on the AHAWP dataset, underscoring the state-of-the-art capabilities of deep learning in handwritten text recognition tasks.

While the outcomes are promising, there remain several avenues for further improvement. First, expanding or diversifying the dataset—either by incorporating additional Arabic handwriting collections or gathering more samples—could bolster model robustness and mitigate overfitting. Second, experimenting with fine-grained layer-freezing strategies in pretrained networks may uncover an even better balance between computational cost and accuracy. Third, exploring advanced data augmentation techniques such as elastic deformations or adversarial training could help the model learn more invariant features. Fourth, ensemble methods that combine predictions from multiple models (e.g., a custom CNN, ResNet-18, and ResNet-50) may enhance overall accuracy and resilience to noise. Lastly, testing these systems on real-world documents or in practical contexts like forensic analysis would offer deeper insights into their reliability and potential for broader deployment. By pursuing these directions, future research can further advance Arabic handwritten word recognition and push the boundaries of deep learning–based writer identification.