BIRZEIT UNIVERSITY

FACULTY OF ENGINEERING AND TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING
Computer Vision
ENCS5343

Course Project

Arabic Handwritten Text Identification Using deep learning techniques

Prepared by: Shereen Ibdah – Amr Halahla
IDs: 1200373 – 1200902

Instructor: Dr. Aziz Qaroush

Jan - 25, 2025

Contents

1. Introduction

Handwritten text recognition is a critical challenge in the field of computer vision, with applications spanning from automated document processing to preserving cultural heritage. Arabic handwriting, in particular, presents unique difficulties due to its cursive nature, contextual character shapes, and a rich set of diacritics. These challenges necessitate robust machine learning solutions capable of effectively handling variations in handwriting styles and stroke formations.

This project aims to identify handwritten Arabic words at the word level by utilizing deep learning techniques to develop robust recognition models. The study is based on the AHAWP dataset, with the primary goal of designing, training, and evaluating convolutional neural networks (CNNs) to achieve high accuracy in recognizing Arabic handwritten words. The project encompasses several key tasks, including the development of custom CNN architectures, the implementation of data augmentation techniques, and the application of pre-trained models through transfer learning. Each phase is designed to assess and compare the effectiveness of different approaches in enhancing classification accuracy while maintaining computational efficiency.

Throughout this project, performance metrics such as accuracy and loss curves will be carefully analyzed to determine the effectiveness of each approach. The results will offer insights into the potential of deep learning-based solutions for Arabic handwritten text recognition and pave the way for future enhancements and applications.

Data Set

The AHAWP dataset is used for this study, focusing on word-level Arabic handwriting. It includes 10 unique words handwritten by 82 individuals, with each contributing 10 samples per word, totaling 8,144 images.

Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are a type of deep learning neural network architecture that is particularly well suited to image classification and object recognition tasks. A CNN works by transforming an input image into a feature map, which is then processed through multiple convolutional and pooling layers to produce a predicted output.

Working of Convolutional Neural Network:

A convolutional neural network (CNN) processes an input image by progressively transforming it into a feature map through a sequence of convolutional and pooling layers. In the convolutional layers, a set of filters is applied to the input image, with each filter extracting specific features that highlight distinct aspects of the image. Pooling layers then follow to down sample the feature maps, reducing their dimensions while preserving essential information.

As the feature maps pass through multiple convolutional and pooling layers, the network learns increasingly complex and abstract features of the input image. Ultimately, the final output of the network provides a predicted class label or a probability score for each class, depending on the specific task at hand.

The architecture of Convolutional Neural Network:

A typical convolutional neural network (CNN) architecture consists of three key components: the input layer, hidden layers, and output layer. The input layer takes in the image data and forwards it to the hidden layers, which consist of multiple convolutional and pooling layers. The output layer then generates the predicted class label or probability scores for each class based on the learned features.

The hidden layers play a crucial role in a CNN's performance, and their configuration—such as the number of layers and filters—can be fine-tuned to achieve optimal results. A commonly used CNN architecture involves multiple convolutional layers that extract hierarchical features, followed by one or more pooling layers to reduce dimensionality, and finally, a fully connected layer that produces the final classification output.

Layers of Convolutional neural network:

The layers of a Convolutional Neural Network (CNN) can be broadly classified into the following categories:

1. **Convolutional Layer:** The convolutional layer is responsible for extracting features from the input image. It performs a convolution operation on the input image, where a filter or kernel is applied to the image to identify and extract specific features.
2. **Pooling Layer:** The pooling layer is responsible for reducing the spatial dimensions of the feature maps produced by the convolutional layer. It performs a down-sampling operation to reduce the size of the feature maps and reduce computational complexity.
3. **Activation Layer:** The activation layer applies a non-linear activation function, such as the ReLU function, to the output of the pooling layer. This function helps to introduce non-linearity into the model, allowing it to learn more complex representations of the input data.
4. **Fully Connected Layer:** The fully connected layer is a traditional neural network layer that connects all the neurons in the previous layer to all the neurons in the next layer. This layer is responsible for combining the features learned by the convolutional and pooling layers to make a prediction

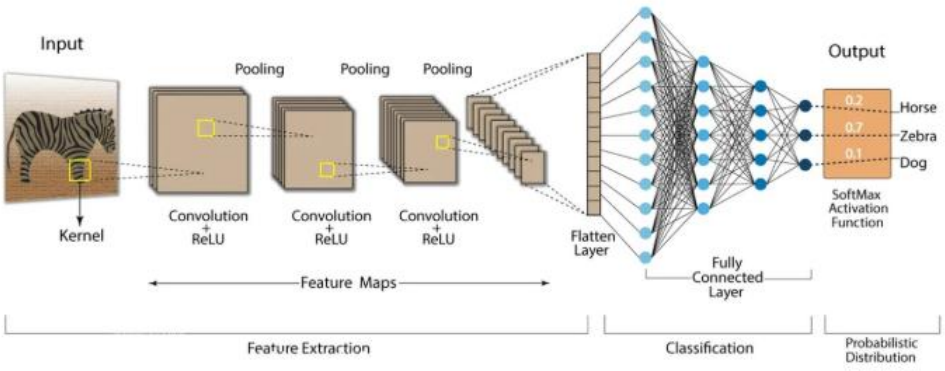

Figure 1: Convolutional Neural Network

Benefits of Convolutional Neural Network:

1. **Feature extraction:** CNNs are capable of automatically extracting relevant features from an input image, reducing the need for manual feature engineering.
2. **Spatial invariance:** CNNs can recognize objects in an image regardless of their location, size, or orientation, making them well-suited to object recognition tasks.
3. **Robust to noise:** CNNs can often handle noisy or cluttered images, making them useful for real-world applications where image quality may be variable.
4. **Transfer learning:** CNNs can leverage pre-trained models, reducing the amount of data and computational resources required to train a new model.
5. **Performance:** CNNs have demonstrated state-of-the-art performance on a range of computer vision tasks, including image classification, object detection, and semantic segmentation.

2. Methodology

2.1. Dataset Preparation

The dataset consists of grayscale images of handwritten Arabic text, preprocessed differently for the tasks:

- **Task 1 and Task 2**: Images were resized to 128×128 pixels, and pixel values were normalized to the range [0,1][0, 1][0,1].
- **Task 3 and Task 4**: Images were resized to 224×224 pixels to ensure compatibility with the MobileNetV1 architecture. The pixel values were similarly normalized to [0,1][0, 1][0,1].

Data augmentation was applied to the training dataset for Tasks 2, 3, and 4 to increase diversity and improve generalization. The validation and test datasets were left unaugmented to provide an unbiased assessment of model performance.

Task 1: Custom CNN Development

In Task 1, a custom CNN architecture was designed and trained on the dataset preprocessed to 128×128dimensions. The CNN architecture included:

- Convolutional and max-pooling layers for feature extraction.
- Fully connected dense layers for classification.
- Dropout layers to mitigate overfitting.

The training process involved **dynamic learning rate adjustment** and **early stopping** to optimize model convergence and prevent overfitting. These fine-tuning techniques allowed the model to adapt effectively to the dataset:

1. **Dynamic Learning Rate Adjustment**:
    - The learning rate was dynamically reduced by a factor of 0.5 if the validation loss plateaued for a specified number of epochs.
2. **Early Stopping**:
    - Training was halted if the validation loss did not improve for a pre-defined number of epochs, ensuring efficient training.

Experimentation and Final Architecture

Multiple experiments were conducted to evaluate various architectures for the custom CNN, including variations in:

- The number of convolutional layers and their filter sizes.
- The number of filters per layer to balance feature extraction and computational cost.
- The depth and size of fully connected layers.

Through these experiments, the following architecture demonstrated the best performance and generalization capabilities:

1. **Feature Extraction Layers**:
    - **Five convolutional blocks** with increasing filter sizes (32, 64, 128, 256, and 512).
    - Each block consisted of:
        - A **3×3** convolutional layer with ReLU activation.
        - Batch Normalization to stabilize training and improve convergence.
        - Max-Pooling to reduce spatial dimensions while retaining important features.
2. **Global Average Pooling**:
    - Replaced the traditional Flatten layer to reduce overfitting by minimizing the number of trainable parameters.
3. **Classification Layers**:
    - Two fully connected dense layers with 512 and 256 neurons, respectively, each followed by Dropout for regularization.
    - A final **softmax** layer with 82 neurons for classification.

This architecture was fine-tuned using dynamic learning rate adjustment and early stopping to optimize training. The combination of convolutional layers for feature extraction and fully connected layers for classification allowed the model to achieve superior accuracy and performance compared to other configurations.

Task 2: Training Custom CNN with Data Augmentation

In Task 2, the custom CNN developed in Task 1 was retrained using an augmented training dataset. The augmentation techniques included rotation, shifting, zooming, and adding noise to mimic real-world variations in handwritten text.

The augmented dataset increased the training set size, enabling the CNN to learn more robust features. The validation and test datasets remained unaugmented for unbiased evaluation.

Task 3: MobileNetV1 Trained from Scratch

In Task 3, a **MobileNetV1** architecture was initialized with random weights and trained from scratch. The input images were resized to 224×224 to match the network's requirements. Since MobileNetV1 expects RGB input, a 3×3 convolutional layer was added to convert grayscale images to three channels.

The custom classification head included:

1. A **Global Average Pooling** layer to reduce spatial dimensions.
2. Two **fully connected dense layers** with 256 and 128 neurons, respectively, followed by **Dropout** layers for regularization.
3. A final **softmax** output layer with 82 neurons corresponding to the dataset classes.

The training process employed early stopping and a learning rate scheduler for optimization.

Task 4: Transfer Learning with MobileNetV1

In Task 4, a **pre-trained MobileNetV1** model with ImageNet weights was used for transfer learning. The model was adapted for grayscale input using the same preprocessing step as in Task 3.

The transfer learning workflow included:

1. Training the classification head with the base model layers frozen to retain pre-trained features.
2. **Fine-tuning**:
    - Unfreezing all layers of the base model.
    - Retraining the entire model with a reduced learning rate ($1\times10-5$) to align pre-trained weights with the dataset's domain.

Evaluation and Metrics

All tasks used the same evaluation metrics:

1. **Categorical Crossentropy Loss**: To measure the difference between predicted and true class probabilities.
2. **Accuracy**: To evaluate the percentage of correct predictions.

The performance was compared across tasks, and accuracy/loss curves were plotted to assess model generalization and convergence.

Results and Discussion

Task 1: Custom CNN

The goal of Task 1 was to design and evaluate a custom convolutional neural network (CNN) for handwritten Arabic text classification. Several experiments were conducted to optimize the architecture, focusing on:

- The number of convolutional layers and their configurations.
- The choice between **Global Average Pooling (GAP)** and **Flatten**, with GAP consistently yielding better generalization.
- The use of **Batch Normalization** after each convolutional layer, followed by **Max Pooling**, which outperformed average pooling in terms of accuracy.
- The application of **Dropout** for regularization, with various values tested. While dropout did not significantly impact accuracy across different trials, the optimal value was selected for the final architecture.

Dynamic learning rate adjustment and early stopping were applied in all trials to ensure efficient convergence and to prevent overfitting. The performance of each architecture is discussed in the following subsections, with a focus on accuracy, loss, and generalization. The experiments culminated in selecting the best-performing architecture, which balanced simplicity and accuracy.

Overview of Architectures and Performance

In Task 1, six different architectures were designed and tested to evaluate their performance on the dataset. The architectures varied in the number of convolutional and fully connected layers, as well as additional configurations such as wider filters and deeper networks. The results, summarized in the table below, highlight the test loss, test accuracy, and the number of epochs required for convergence for each architecture.

The table below summarizes the performance of the various CNN architectures tested:

Table 1: Comparison Table between different custom CNN architectures.

Architecture	Test Loss	Test Accuracy	Epochs Taken
4Conv + 2FC	0.584185	82.87%	74
4Conv + 3FC	1.323367	65.32%	39
5Conv + 2FC	0.514813	86.00%	87
5Conv + 3FC	0.787411	79.68%	91
Wider 5Conv + 2FC	0.315841	90.79%	89
Deeper 6Conv + 2FC	0.670207	81.95%	30

Insights from the Table

- **Best Performance:** The **Wider 5Conv + 2FC architecture** achieved the best accuracy at **90.79%** with the lowest test loss of **0.3158**, demonstrating the benefits of wider filters in extracting meaningful features while maintaining generalization.
- **Effect of Depth:** The **Deeper 6Conv + 2FC architecture** achieved good accuracy at **81.95%**, but its test loss was higher than that of the wider model. This indicates that increasing depth without a corresponding increase in data or regularization can lead to overfitting.
- **Fully Connected Layers:** Adding a third fully connected layer (e.g., in 4Conv + 3FC and 5Conv + 3FC) generally did not improve performance. Instead, it increased the risk of overfitting and caused higher test losses.
- **Baseline Performance:** The **4Conv + 2FC architecture** provided a solid baseline with an accuracy of **82.87%**, showing stable convergence during training.

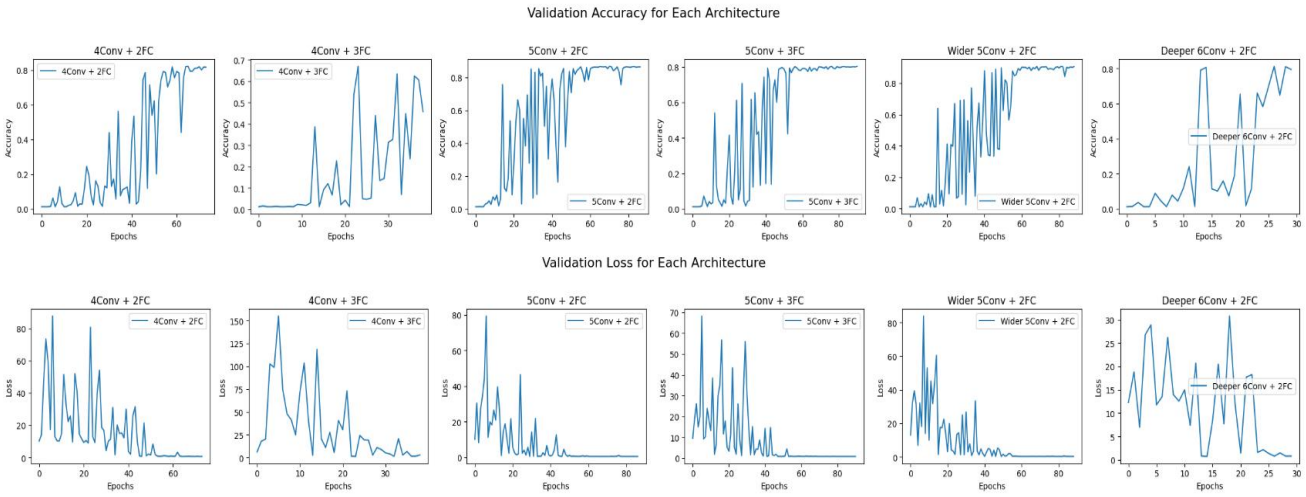

Figure 2: Plots of the results for different CNN architectures.

Insights from the Figures

1. Validation Accuracy Plots

The validation accuracy plots for all architectures illustrate the progression of learning and convergence:

- The **Wider 5Conv + 2FC architecture** shows the smoothest improvement in accuracy, reaching a peak performance with minimal fluctuations.
- Architectures like **4Conv + 3FC** and **Deeper 6Conv + 2FC** exhibit instability, indicating challenges in generalization or convergence.

- **5Conv + 2FC** and **5Conv + 3FC** show promising trends but are slightly less stable compared to the wider configuration.

2. Validation Loss Plots

The validation loss plots highlight how well each architecture minimizes loss over epochs:

- The **Wider 5Conv + 2FC architecture** achieves the lowest validation loss consistently, demonstrating its ability to generalize effectively.
- Architectures with more fully connected layers, such as **4Conv + 3FC**, show erratic loss trends, indicating potential overfitting or instability.
- The **Deeper 6Conv + 2FC architecture** initially minimizes loss but struggles with stability as training progresses.

In conclusion, in Task 1, we evaluated six CNN architectures to determine the best-performing model. The **Wider 5Conv + 2FC** emerged as the most effective, achieving the highest accuracy (90.79%) and lowest loss (0.3158), balancing depth and width for robust feature extraction. Deeper architectures, such as **Deeper 6Conv + 2FC**, showed moderate performance but struggled with stability. Additional fully connected layers, as in **4Conv + 3FC** and **5Conv + 3FC**, often led to overfitting and poorer results. The analysis underscores the importance of balancing complexity and regularization, with **Wider 5Conv + 2FC** being the most suitable model for this task.

Task2: Data Augmentation Integration

Following the results of Task 1, which demonstrated the effectiveness of different convolutional neural network architectures, the **Wider 5Conv + 2FC architecture** was selected for Task 2. This architecture showed superior performance in Task 1 and was deemed the most suitable candidate for further evaluation on an augmented dataset. Task 2 aimed to assess the impact of data augmentation on the model's generalization and performance, specifically for the chosen architecture.

Methodology and Dataset Augmentation

The dataset for Task 2 underwent extensive augmentation to introduce greater variability and mimic real-world scenarios. The augmentation techniques included random rotations, shifts, flips, and zoom transformations. These augmentations aimed to enhance the model's robustness by training it on a more diverse set of input data. The augmented dataset was then resized to 128x128 pixels to match the preprocessing of Task 1.

The **Wider 5Conv + 2FC architecture** was retrained from scratch on the augmented dataset using the same training configurations as in Task 1. This configuration included the use of dynamic learning rates, early stopping, and batch normalization to ensure efficient training and

prevent overfitting. The training process was monitored over several epochs, and the results were recorded.

Results and Insights

The training results showed that the model adapted effectively to the augmented dataset. It achieved a high training accuracy, demonstrating its ability to learn complex patterns in the data, while the validation accuracy indicated stable generalization to unseen samples. The test evaluation yielded an accuracy of **92.69%** and a test loss of **0.33**, highlighting the effectiveness of the architecture in handling the augmented data.

```
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 14ms/step - accuracy: 0.9194 - loss: 0.3405
Original Test Loss: 0.3339027762413025
Original Test Accuracy: 0.9269490242004395
```

Figure 3: Task2 =>Accuracy and Loss of Test Results (Our Custom CNN Network).

The validation and training plots provided insights into the training dynamics. The training accuracy curve steadily increased, with the validation accuracy stabilizing after the early epochs, suggesting that the model successfully avoided overfitting despite the increased variability in the data. Similarly, the training and validation loss plots demonstrated smooth convergence, with minimal fluctuations after the initial learning stages.

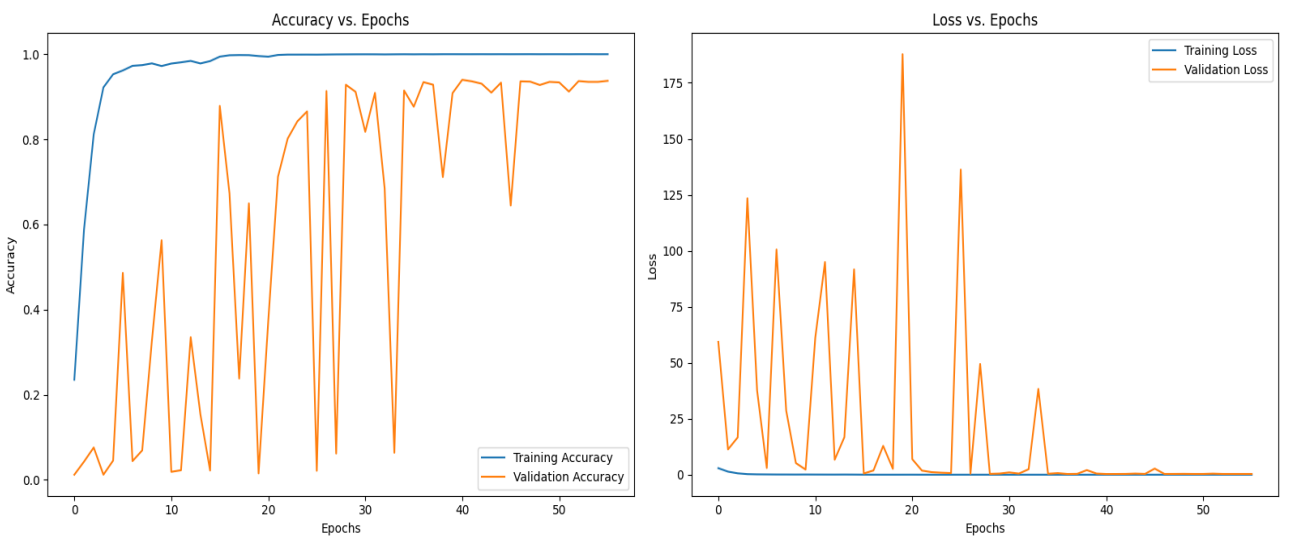

Figure 4: Task2 => (Accuracy & Loss) vs. Epochs (Our Custom CNN Network)

Conclusion

Task 2 showcased the benefits of data augmentation in improving the generalization capabilities of the Wider 5Conv + 2FC architecture. By exposing the model to augmented variations, it was better equipped to handle the complexities of real-world scenarios. These results reinforce the

importance of both architecture selection and dataset preprocessing in achieving optimal performance.

Task3: Training Using a Well-Known CNN Architecture (MobileNetV1)

Overview
In Task 3, the MobileNetV1 architecture was selected to train the model from scratch using the same dataset preprocessed in grayscale format, resized to 224x224 pixels. MobileNetV1 is a lightweight and efficient architecture known for its balanced performance between accuracy and computational efficiency. The objective of this task was to compare the results obtained from a state-of-the-art architecture with those from a custom-built model in Task 2 and analyze its advantages and challenges.

Methodology
The dataset underwent preprocessing to resize the input images to 224x224 pixels, making them compatible with MobileNetV1's input requirements. The model was implemented using the Keras library's MobileNet class, initialized with random weights (i.e., no pre-trained weights were used). The training process incorporated the following:

- **Optimization and Regularization**: Adam optimizer with an initial learning rate of 0.0005.
- **Callbacks**: Early stopping to prevent overfitting and learning rate reduction on plateau to optimize convergence.
- **Training Configuration**: The model was trained for a maximum of 100 epochs, with the training process halting at 63 epochs due to early stopping.

Results and Insights

```
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 24ms/step - accuracy: 0.9210 - loss: 0.3125
Task 3 - MobileNetV1 (Scratch): Test Loss: 0.3005315661430359, Test Accuracy: 0.9238796830177307
```

Figure 5: Test Accuracy and Test Loss Results (MobileNetV1).

The MobileNetV1 model achieved the following:

- **Test Loss**: 0.3085
- **Test Accuracy**: 92.39%

The results indicate that MobileNetV1 performed slightly lower in accuracy compared to Task 2's custom wider network, which achieved **92.69%** accuracy. While both tasks demonstrated

exceptional performance, Task 2 surpassed Task 3 by a small margin despite leveraging a simpler custom-built model.

Challenges and Limitations
This experiment faced resource constraints that restricted the ability to:

1. **Expand Dataset Size**: Due to computational and memory limitations, extensive data augmentation to increase dataset diversity was not feasible.
2. **Utilize More Complex Architectures**: The computational overhead of training deeper and more sophisticated architectures made it impractical within the given resource environment.

These limitations underscore the importance of balancing available resources with model complexity to achieve optimal results under constrained environments.

Visual Observations

1. **Accuracy Trends**: The training accuracy quickly reached 100%, while validation accuracy plateaued at around 92.33%, suggesting effective generalization with minor signs of potential overfitting mitigated by callbacks.
2. **Loss Behavior**: Training loss approached zero, while validation loss stabilized at around 0.3448, indicating successful convergence.

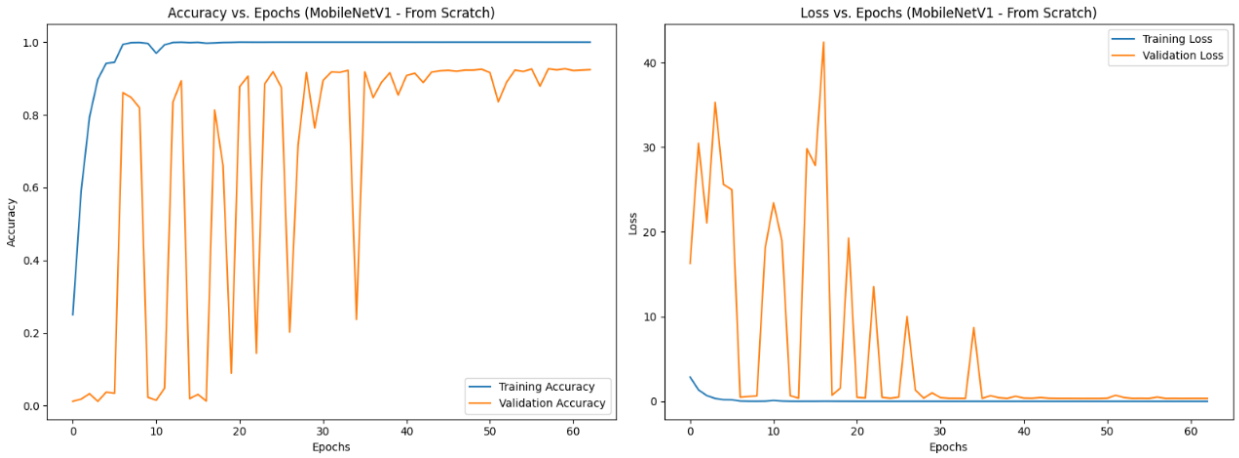

Figure 6: (Accuracy & Loss) vs. Epochs (MobileNetV1)

Comparison with Task 2
Despite using a more advanced architecture, Task 3's MobileNetV1 fell slightly behind Task 2's custom wider network in accuracy. The **custom network achieved 92.69% test accuracy**, while MobileNetV1 reached 92.39%. Furthermore, Task 3 required significantly more training time due to the complexity of MobileNetV1, demonstrating that simpler models can sometimes outperform or match state-of-the-art architectures when designed effectively for specific datasets.

Conclusion

While MobileNetV1 proved to be a powerful architecture, the slightly higher accuracy of Task 2's custom network and its shorter training time highlight the effectiveness of tailored solutions in resource-constrained settings. Task 3 reinforced the potential of leveraging state-of-the-art models but also emphasized the necessity of adapting choices to computational and resource limitations.

Task4: Transfer Learning with Pre-Trained MobileNetV1

Task 4 leverages a pre-trained MobileNetV1 model, adapted for grayscale images, to classify the dataset using transfer learning. The base layers, pre-trained on ImageNet, were frozen, and custom classification layers were added to suit the task. This approach reduces training time and enhances performance by reusing learned feature representations. The results, detailed below, demonstrate the effectiveness of transfer learning in achieving high accuracy efficiently.

```
51/51 ━━━━━━━━━━━━━━━━━━━━ 1s 25ms/step - accuracy: 0.9414 - loss: 0.2981
Task 4 - MobileNetV1 (Transfer Learning): Test Loss: 0.23815320432186127, Test Accuracy: 0.9496623873710632
```

Figure 7: Accuracy vs. Epochs (MobileNetV1 - Transfer Learning)

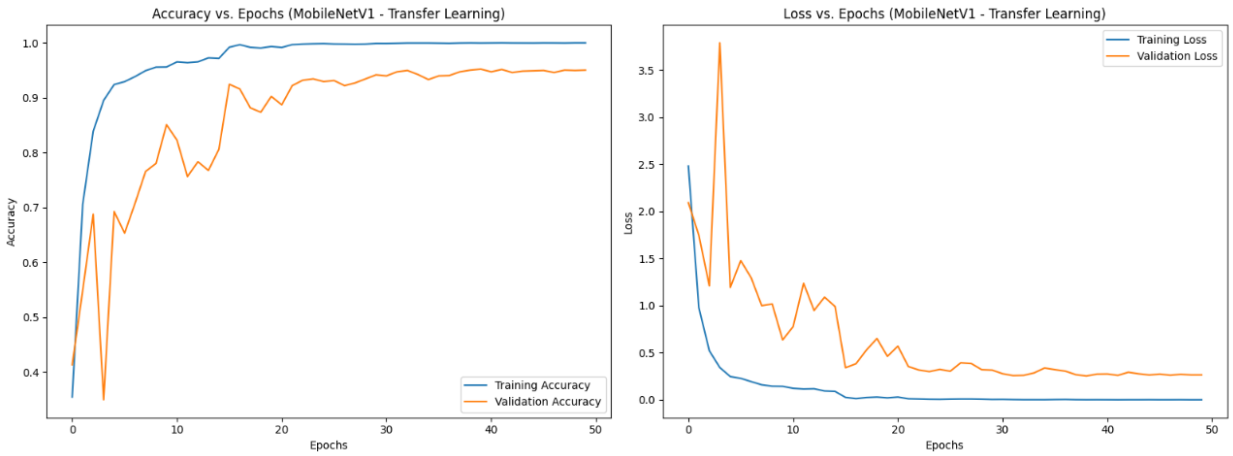

Figure 8: Loss vs. Epochs (MobileNetV1 - Transfer Learning)

The results for Task 4 demonstrate the performance of the MobileNetV1 architecture with transfer learning, trained on the given dataset:

- **Test Loss**: 0.23815320432186127
- **Test Accuracy**: 94.96%

The plots showcase the training and validation accuracy, as well as the loss, over the epochs:

1. **Accuracy vs. Epochs**:
   The training accuracy shows a rapid improvement in the early epochs, stabilizing around 100% in less than 20 epochs. The validation accuracy improves steadily, plateauing around 93-95%, indicating robust generalization of the model to unseen data.
2. **Loss vs. Epochs**:
   The training loss decreases consistently and approaches near-zero values, reflecting effective optimization. Validation loss also decreases but stabilizes with minor oscillations, suggesting a well-regularized model with no significant overfitting.

The results of Task 4 clearly show that utilizing transfer learning with the pre-trained MobileNetV1 model has yielded the highest accuracy among all tasks. The model achieved excellent generalization while requiring fewer computational resources compared to training from scratch. The efficient pre-trained features provided by MobileNetV1 likely contributed to the rapid convergence and superior accuracy.

These results also highlight the potential of transfer learning for handling complex classification tasks, even with limited resources. The steady improvement in validation accuracy confirms the capability of transfer learning to leverage pre-trained knowledge for similar datasets, outperforming models trained from scratch.

Conclusion and Future Work

In this project, we systematically explored various deep learning architectures and methodologies to classify the given dataset effectively. Starting with custom-designed convolutional neural networks (CNNs) in Task 1 and Task 2, we progressively improved model performance through experimentation and analysis. Task 3 introduced the MobileNetV1 architecture trained from scratch, achieving competitive accuracy but with higher computational costs. Finally, Task 4 leveraged transfer learning with MobileNetV1 pre-trained on ImageNet, significantly improving accuracy and efficiency.

Conclusion: The results demonstrated that transfer learning can effectively enhance model performance while reducing training time. The wider architectures in Task 2 also proved efficient in balancing model complexity and accuracy. However, constraints such as limited computational resources and dataset size restricted the exploration of more complex models and larger datasets.

Future Work: Future efforts will focus on addressing these limitations by:

1. Expanding the dataset through advanced data augmentation or collecting additional samples.
2. Exploring state-of-the-art architectures, such as EfficientNet or Vision Transformers, for potentially better performance.
3. Incorporating techniques like model pruning and quantization to optimize models for deployment.
4. Investigating ensemble learning approaches to combine the strengths of multiple architectures for improved generalization.

This experiment lays a solid foundation for further advancements in leveraging deep learning for classification tasks in constrained resource environments.