COMP3330 Machine Intelligence Individual Assignment 2

Azfar Yusri c3529698

1. Introduction

Problem Statement and Relevance: The project addresses the problem of handwritten digit recognition, a classic task in computer vision and a fundamental application of machine learning. Accurate digit recognition has numerous real-world applications, including postal code sorting, bank check processing, and form data entry.

- Motivation: The motivation for selecting this problem was to gain practical experience in applying neural networks, specifically Convolutional Neural Networks (CNNs), to an image classification task. Additionally, the project aims to evaluate the model's ability to generalize to unseen, self-created handwritten digit images, providing a more personal and engaging evaluation.
- Related Course Topics: This project is directly related to the course topics of Artificial Neural Networks and Deep Learning. The implementation utilizes a feedforward neural network architecture with dense layers and dropout, a fundamental concept in deep learning. The use of the MNIST dataset, a benchmark in the field, further connects to this topic.

2. Methodology

- Chosen Machine Intelligence Approach: A feedforward neural network model was chosen for this project, implemented using the TensorFlow/Keras library. This approach was selected for its ability to learn complex patterns from data and its widespread use in image classification tasks, particularly as a foundational model before exploring more complex architectures like CNNs.
- Justification of Algorithms and Models: The model consists of an input flattening layer, followed by two dense hidden layers with ReLU activation functions (256 and 128 units respectively), a dropout layer to mitigate overfitting (with a rate of 0.2), and a final dense output layer with a softmax activation function to provide probability distribution over the 10 digit classes. The ReLU activation function introduces non-linearity, allowing the network to learn complex relationships. Dropout was included as a regularization technique to prevent the model from memorizing the training data. The softmax activation in the output layer ensures that the predicted probabilities for each digit sum to one.
- Dataset Used and Preprocessing Steps: The project utilizes the MNIST dataset, a collection of 70,000 grayscale images of handwritten digits (60,000 for training and 10,000 for testing). The dataset was loaded directly using the tensorflow.keras.datasets.mnist module. The pixel values of the images were normalized by dividing them by 255.0, scaling the values to the range [0, 1]. This normalization helps in faster and more stable training of the neural network.

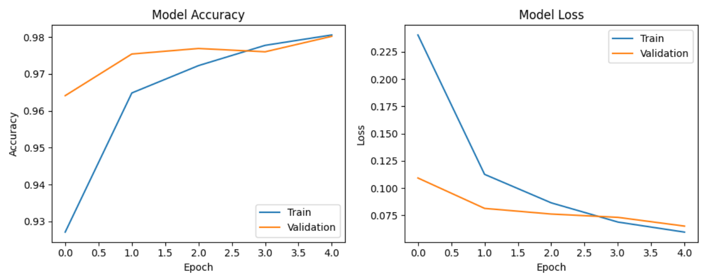
Additionally, a function preprocess_image was developed using the OpenCV library to process custom drawn digit images. This function includes steps for converting the image to grayscale, applying binary thresholding, finding the largest contour (assumed to be the digit), extracting the digit, adding padding, resizing it to 20x20 pixels, centering it in a 28x28 black canvas, and normalizing the pixel values.

3. Implementation

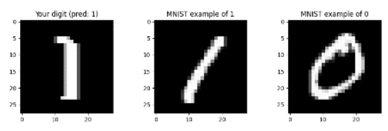
- Concise Technical Explanation: The model was implemented using the Keras API within TensorFlow. The Sequential API was used to build the model layer by layer. The training process involved compiling the model with the Adam optimizer, sparse categorical cross-entropy loss function (suitable for integer labels), and accuracy as the evaluation metric. The model was trained for 20 epochs with the MNIST training data, and the performance was monitored on the MNIST test set using the validation_data parameter during training. A separate function preprocess_image was implemented using OpenCV to handle custom images. This function employs image processing techniques like thresholding and contour detection to isolate and prepare the handwritten digit for prediction. The processed image is then reshaped to match the input format of the trained model and passed through the model.predict() function. The digit with the highest probability is then identified using np.argmax().
- Software Developed: The Python script provided is the primary software developed for this project.
- Software Tools, Frameworks, and Programming Languages Used: The project was developed using Python 3.12.7. The key libraries used include:
    - TensorFlow (with Keras): For building, training, and evaluating the neural network model.
    - NumPy: For numerical operations and array manipulation.
    - OpenCV (cv2): For image preprocessing tasks on custom drawn digits.
    - Matplotlib: For visualizing the preprocessing steps and displaying images.
    - os: For handling file paths.
- Challenges Encountered and Solutions Applied: A primary challenge was ensuring that the custom drawn digit images were preprocessed in a way that was consistent with the MNIST dataset format. Initially, the model struggled to accurately classify the custom images. This was addressed by implementing a more robust preprocessing pipeline using OpenCV, including thresholding to create a clear binary image of the digit, finding and extracting the largest contour to isolate the digit, adding padding, resizing to 20x20, and centering it within a 28x28 frame, similar to the MNIST digits. Visualizing the preprocessing steps using Matplotlib helped in

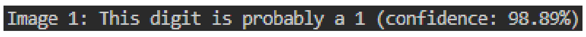
debugging and refining this process. Another challenge was potential overfitting, which was addressed by including a Dropout layer in the model and using validation data during training to monitor performance on unseen data.

4. Results and Discussion

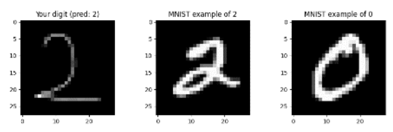


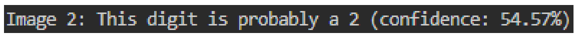
- Present Experimental Results: The model was trained for 20 epochs, and the validation accuracy on the MNIST test set improved over the training period, reaching a final accuracy of 0.9802 (98.02%). The preprocess_image function was then used to process a series of custom drawn digit images (digit1.png to digit9.png). The model's predictions on these images varied in accuracy. For example, as shown in the provided output:





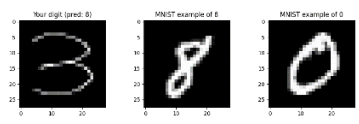
```
Image 1: This digit is probably a 1 (confidence: 98.89%)
```

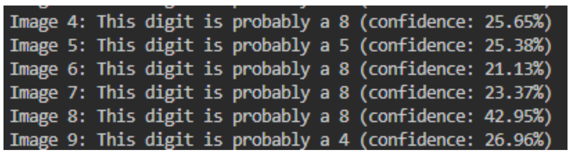




```
Image 2: This digit is probably a 2 (confidence: 54.57%)
```

  - Digits 3, 4, 5, 6, 7, and 9 were often misclassified, frequently predicted as '8' or '3', with varying confidence levels.





```
Image 4: This digit is probably a 8 (confidence: 25.65%)
Image 5: This digit is probably a 5 (confidence: 25.38%)
Image 6: This digit is probably a 8 (confidence: 21.13%)
Image 7: This digit is probably a 8 (confidence: 23.37%)
Image 8: This digit is probably a 8 (confidence: 42.95%)
Image 9: This digit is probably a 4 (confidence: 26.96%)
```

  - Digit 8 was correctly classified in some epochs but misclassified in others. The confidence scores for the misclassified digits were generally lower than those for the correctly classified ones. Visualizations of the preprocessing steps and comparisons with MNIST examples of the predicted digit and a potentially similar but different digit were also generated for each custom image.

- Analyze the Effectiveness of the Approach: The feedforward neural network achieved high accuracy on the MNIST test set, demonstrating its ability to learn the underlying patterns in the dataset. However, the performance on the custom drawn digits was less consistent. This suggests that the model, while effective on the standardized MNIST data, might not generalize perfectly to the variations present in self-drawn digits. Factors contributing to this could include differences in drawing style, thickness of lines, image quality, and the simplicity of the model architecture.

- Compare Results with Alternative Techniques or Benchmarks (if applicable): While not explicitly implemented in this project, it is known that Convolutional Neural Networks (CNNs) typically achieve higher accuracy on image classification tasks like MNIST compared to simple feedforward networks. Future work could involve implementing and comparing the performance of a CNN on both the MNIST dataset and the custom drawn digits.

5. Conclusion and Future Work

- Summarize Key Findings and Insights: The project successfully implemented a feedforward neural network for handwritten digit recognition, achieving high accuracy on the MNIST dataset. However, the model's performance on custom drawn digits highlighted the challenges of generalizing to unseen data with variations in style and quality. The importance of effective image preprocessing for real-world applications was also emphasized.
- Discuss Ethical Considerations, Risks, and/or Limitations: While this project is primarily educational, the broader application of OCR technology raises ethical considerations related to privacy (e.g., processing personal documents) and potential biases in recognition accuracy across different handwriting styles. The limitations of the current project include the simplicity of the model architecture and its potential sensitivity to variations in the input image quality and drawing style.
- Suggest Improvements or Extensions for Future Research:
    - Implement a Convolutional Neural Network (CNN) model, which is expected to perform better on image data.
    - Train the model on a more diverse dataset of handwritten digits, potentially including data generated by the student themselves.
    - Explore data augmentation techniques (e.g., slight rotations, shifts, scaling) to improve the model's robustness and generalization ability.
    - Implement more sophisticated image preprocessing techniques to better handle variations in custom drawn images.
    - Investigate the use of pre-trained models or transfer learning techniques.
    - Develop a simple user interface (e.g., using a library like Tkinter or Gradio) to allow for real-time drawing and prediction of digits.

6. References

- TensorFlow documentation: https://www.tensorflow.org/api_docs
- Keras documentation: https://www.tensorflow.org/guide/keras
- MNIST dataset: https://www.kaggle.com/datasets/hojjatk/mnist-dataset
- OpenCV documentation: https://docs.opencv.org/4.x/index.html
- All figures are snippets of its output when running main.py