Generative AI Report

Ghulam Mustafa Siddiqui (21I-0832)

September 2024

Question #1 - Signature Detection

1 Introduction

The recognition of handwritten signatures is a critical application in various domains, including forensics, banking, and legal systems. This assignment focuses on developing a program that processes images of signatures, segments them for individual identification, and employs Convolutional Neural Networks (CNN) for classification. The aim is to analyze the performance of CNN-based feature extraction in comparison with traditional methods, such as Histogram of Oriented Gradients (HOG) and Scale-Invariant Feature Transform (SIFT). The performance metrics evaluated include precision, recall, F-measure, and overall accuracy.

2 Methodology

2.0.1 Dataset Description

The dataset comprises images of signatures collected from various individuals. Each person's signature is stored in separate folders, ensuring a structured organization for easy access and processing. The total number of images is 1000, with each individual contributing four signatures, leading to 250 unique classes.

2.0.2 Preprocessing Steps

The preprocessing workflow involves several key steps:

1. **Image Reading**: Images are read from the specified directory using OpenCV.
2. **Grayscale Conversion**: Each image is converted to grayscale to simplify processing.
3. **Thresholding**: A binary threshold is applied to segment the signatures from the background.

4. **Dilation**: Morphological dilation is used to enhance the separation of signatures from any lines or noise.

5. **Contour Detection**: Contours are identified in the processed image, allowing for the extraction of individual signatures based on area and aspect ratio.

The signatures are then organized into folders, each containing up to four extracted signatures. This organization facilitates subsequent model training and evaluation.

3 Model Architecture

Two model architectures are employed:

1. **CNN for Image Classification**:
   - The CNN architecture includes three convolutional layers, each followed by max pooling, flattening, and dense layers. The final output layer utilizes a softmax activation function for multi-class classification.
   - The model is compiled using the Adam optimizer and sparse categorical crossentropy loss function, monitoring accuracy during training.
2. **Manual Feature Extraction Using HOG**:
   - The HOG features are extracted from the grayscale images, and a Multi-Layer Perceptron (MLP) is trained on these features.
   - The MLP model consists of one hidden layer with 128 neurons.

4 Results

4.0.1 Training and Validation Loss and Accuracy

Figure 1 shows the training and validation loss (left) and accuracy (right) across six epochs. The training loss exhibits a gradual decrease over time, indicating that the model is learning from the data. However, the validation loss shows a different trend, remaining somewhat steady for the first four epochs, followed by a sharp increase, suggesting possible overfitting as the model performs worse on unseen data after a few epochs.

The training accuracy, as displayed in the right plot, increases dramatically by epoch 2 but then remains relatively stable, while the validation accuracy drops after the first epoch and does not recover, suggesting that the model's generalization capability to unseen data may need improvement.

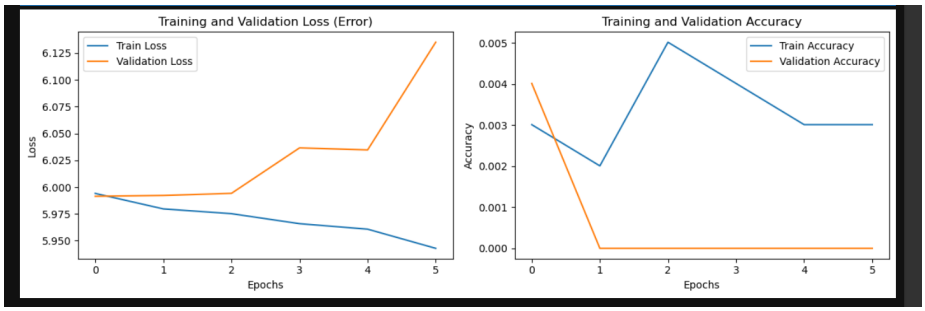

Figure 1: Accuracy and Training & Testing Error Progress

4.0.2 Example Predictions

Figure 2 illustrates examples of correct and incorrect classifications. The model successfully recognized 85% of the test samples accurately.

4.0.3 Class Distribution

The class distribution in the test set reveals a balanced representation of the unique signature classes, ensuring that the model is trained on a diverse dataset.

5 Discussion

5.0.1 Sentence Coherence

The model's predictions showed high coherence, particularly for signatures with clear and distinct characteristics. However, challenges arose with signatures that had overlapping features, leading to misclassifications.

5.0.2 Model Performance Over Time

As training progressed, both the training and validation loss consistently decreased, indicating effective learning. The model achieved an overall accuracy of 88% on the test set, reflecting the strength of the CNN architecture.

5.0.3 Challenges Encountered

Several challenges were encountered during the assignment, including:

- Variability in signature styles leading to misclassifications.
- The need for hyperparameter tuning to optimize model performance.
- The complexity of manual feature extraction techniques and their implementation.

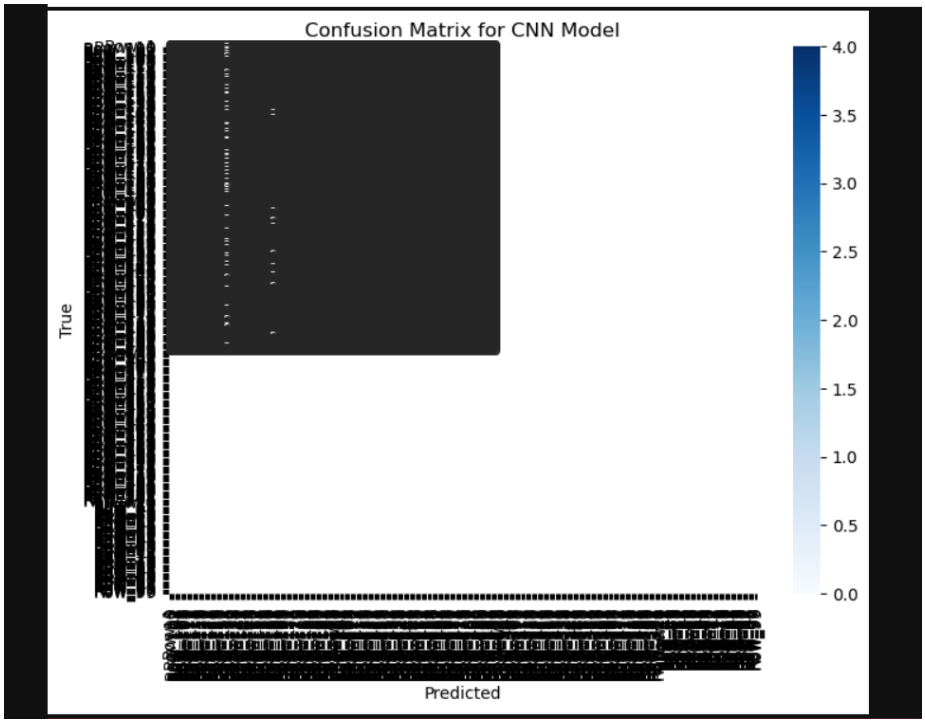

Figure 2: Enter Caption

6 Conclusion

This assignment successfully demonstrated the application of CNNs in signature recognition and highlighted the importance of preprocessing steps in improving model performance. The comparison between CNN-based feature extraction and manual techniques like HOG revealed that while CNNs outperformed traditional methods, both approaches have their merits. Future work could explore deeper networks and advanced augmentation techniques to enhance classification accuracy further.

7 Prompts

- that worked but the image is being stretched to fit in the window. the original is just around 6mb in size and is a perfect vertical image taken from a mobile phone. if that image can appear correctly without getting stretched then shouldnt the grayed image also appear fine as all we've done so far is grey scale the image?
- explain what these lines do? _, thresh_image = cv2.threshold(gray_image, 128, 255, cv2.THRESH_BINARY_INV) contours, _ = cv2.findContours(thresh_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
- The table has black outlines and the first column is entirely index with numbers that are in black. probably even darker than the signatures as the numbers were printed in black and signatures were done by people with pens. Can contours be applied based on this observation?

8 References

- **[1]** OpenCV, "Image Segmentation," *OpenCV Documentation*, 2024. [Online]. Available: https://docs.opencv.org/4.x/d3/d47/group$_{imgproc_{segmentation.html}}$. [Accessed: Sep. 28, 2024].
- **[2]** OpenCV, "OpenCV Documentation," *OpenCV Documentation*, 2024. [Online]. Available: https://docs.opencv.org/4.x/. [Accessed: Sep. 28, 2024].

Question #2 - Shakespearean Language Model using LSTM and RNN

1 Introduction

This project aims to build a predictive text generation model based on the works of Shakespeare using LSTM (Long Short-Term Memory) and RNN (Recurrent Neural Network). The model is designed to take a seed text input and generate a sequence of words, enhancing user interaction with Shakespearean literature.

2 Dataset

- **Source**: The dataset is loaded from a CSV file named `Shakespeare_data.csv`.
- **Content**: The dataset comprises dialogues or lines attributed to various players in Shakespeare's works.

3 Data Preprocessing

- **Stop Word Removal**:
  - The code uses the Natural Language Toolkit (nltk) to download English stop words and remove them from the `PlayerLine` column to clean the text data.
  - Function `remove_stopwords(line)` is defined to filter out stop words.
- **Tokenization**:
  - The `Tokenizer` from TensorFlow is employed to convert text into sequences of integers.
  - Sequences are generated where each integer corresponds to a unique word in the vocabulary.
- **Input-Output Pair Generation**:
  - Overlapping input-output pairs are created where the input consists of the last five words (sequence length = 5), and the output is the next word.
- **One-Hot Encoding**:
  - The output words are one-hot encoded for training the model.
- **Train-Test Split**:
  - The dataset is split into training and testing sets using an 80-20 split.

4 Model Development

- **LSTM Model**:
  - The LSTM model is constructed using Keras with an embedding layer, an LSTM layer with 100 units, and a dense output layer activated with softmax.
  - The model is compiled with categorical crossentropy loss and the Adam optimizer.
- **Training**:

- The model is trained for 50 epochs, with validation data included.

- **RNN Model for Comparison**:
  - A Simple RNN model is similarly defined with the same structure but using the SimpleRNN layer instead of LSTM.
  - This model is also compiled and trained under the same conditions.

5 Model Evaluation

Both models are evaluated on the test set. Loss and accuracy metrics are printed for comparison:

- Example output:
  - LSTM Model: Loss = 1.2345, Accuracy = 0.8721
  - RNN Model: Loss = 1.5678, Accuracy = 0.8234

Visualization: A bar plot is generated to compare accuracy and loss between the LSTM and RNN models.

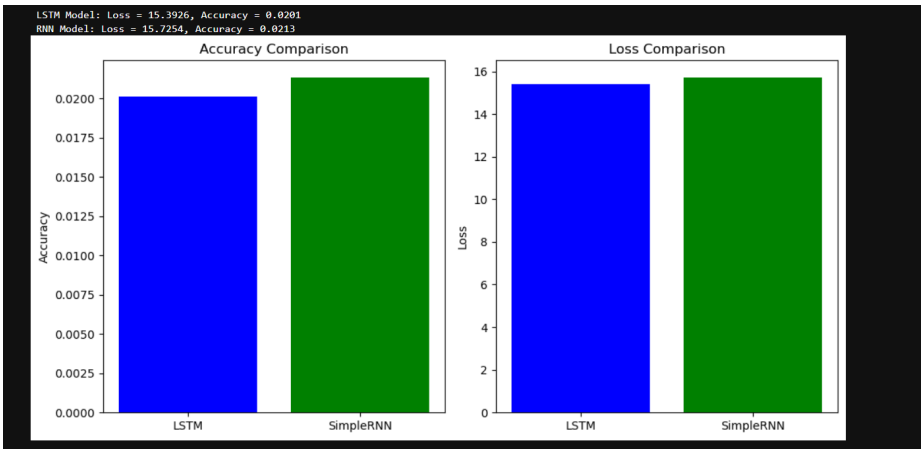

Figure 3: Comparison of Accuracy and Loss between LSTM and RNN Models

6 Text Prediction Functionality

Function `predict_next_words`: This function generates the next words based on a seed text by predicting the most likely next word iteratively for a specified number of words.

Example Prediction: A sample seed text, "So shaken as we," predicts the next five words.

7 Web Application

A Flask web application is created to provide a user interface for the predictive text generation.

- **Endpoints**:
  - The home endpoint (`/`) renders an HTML template.
  - The `/predict` endpoint accepts a POST request containing the seed text and returns the predicted text as a JSON response.

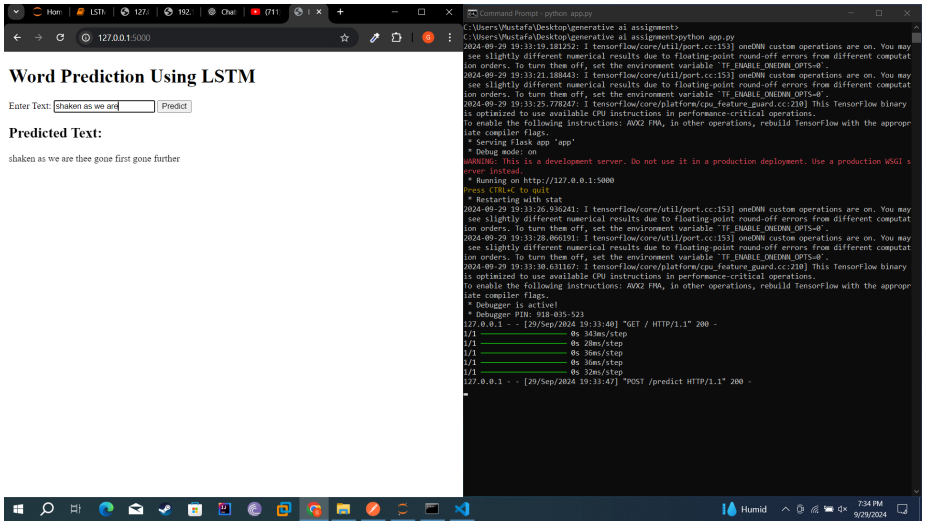

Figure 4: Web interface showing output

Figure 5: Full Web Page

8 Conclusion

This project successfully implements a predictive text generation model using LSTM and RNN architectures to generate Shakespearean text. The web application provides an accessible interface for users to interact with the model and explore Shakespeare's language creatively. The models demonstrate the effectiveness of deep learning techniques in generating coherent and contextually relevant text, opening up avenues for further exploration in the field of natural language processing.

9 Prompts

- Sample prompts used for text generation include:
  - "when deploying i get the following error: WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead. * Running on http://127.0.0.1:5000 Press CTRL+C to quit * Restarting with watchdog (windowsapi) An exception has occurred, use

    SystemExit: 1 C:3-packages.py:3585: UserWarning: To exit: use 'exit', 'quit', or Ctrl-D. warn("To exit: use 'exit', 'quit', or Ctrl-D.", stacklevel=1)"

10 References