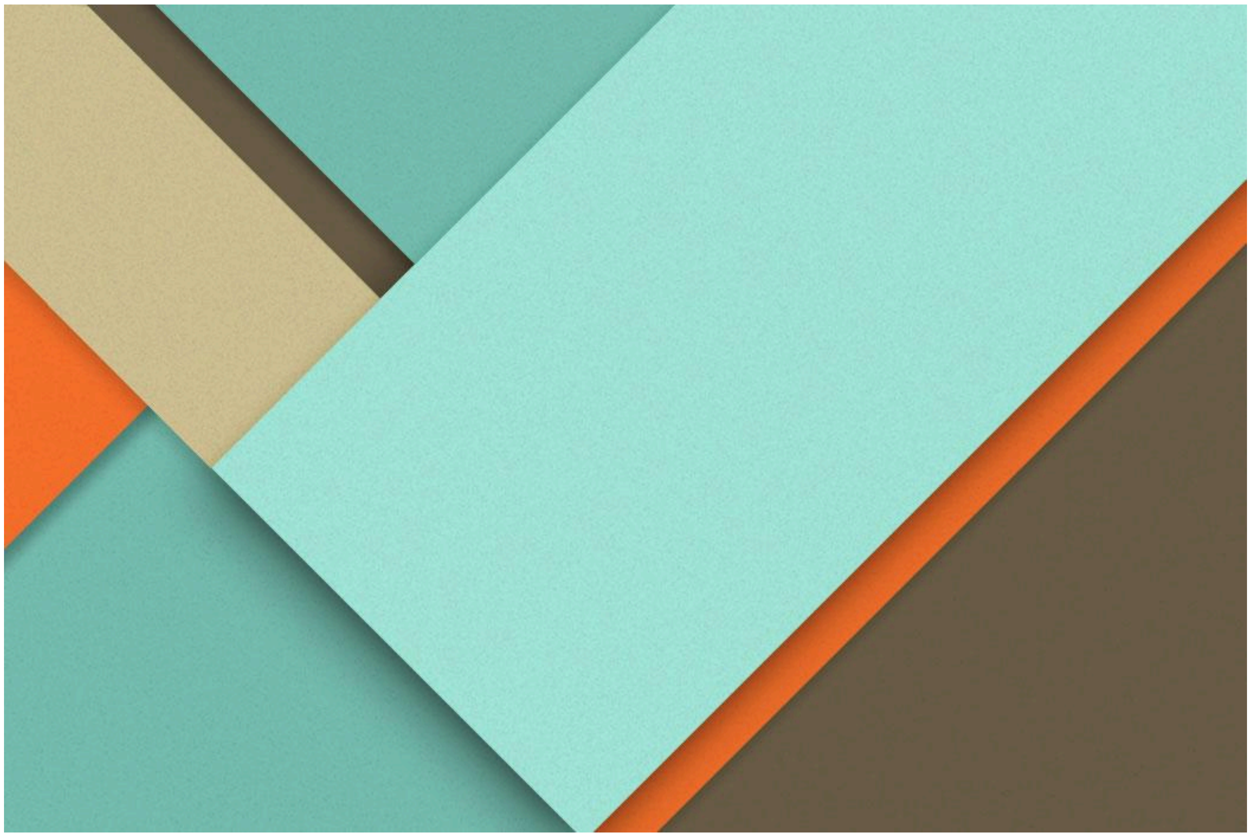


Handwritten Text Recognition

CS 354 PROJECT

—

Members:
Sana Presswala (210001062)
Agrima Bundela (210002009)
Kirtan Kushwah (210001030)

Title

Handwritten Text Recognition using Deep Learning

Problem Definition

In today's digital age, the transformation of handwritten text into editable and searchable formats is a vital yet challenging task. Handwritten documents, such as historical manuscripts, personal notes, and forms, contain valuable information that often remains inaccessible due to the limitations of traditional text recognition methods. This project addresses the pressing need for accurate and efficient handwritten text recognition (HTR) systems.

The primary objective of this project is to develop a robust and accurate handwritten text recognition system capable of transcribing handwritten documents into machine-readable text. This system should be able to handle various handwriting styles, languages, and document formats to ensure broad applicability and usability.

Types of handwriting recognition

Optical Character Recognition (OCR): Focuses on recognizing printed or handwritten characters within images or documents.

Handwriting Text Recognition (HTR): Specifically designed to decipher diverse handwriting styles and convert handwritten text into digital format.

Problems Faced with Handwriting Recognition:

I. ### Variability in Handwriting Styles:
Handwriting can vary significantly between individuals, making it challenging for recognition systems to accurately interpret diverse writing styles.

II. ### Noise and Distortions:
Handwritten text may contain noise, smudges, or distortions due to imperfect writing conditions or scanning artifacts, which can hinder recognition accuracy.

III. ### Contextual Understanding:
Handwritten text often lacks punctuation and may be part of larger documents

where context is crucial for accurate interpretation, posing challenges for recognition systems to understand the intended meaning.

IV. Ambiguity and Similar Characters:

Some characters or symbols may appear similar or ambiguous, especially in cursive or stylized handwriting, leading to errors in recognition.

V. Data Scarcity and Training:

Insufficient training data, especially for recognizing diverse handwriting styles, can limit the effectiveness of recognition models and hinder their ability to generalize to new samples.

VI. Computational Complexity:

Handwriting recognition algorithms can be computationally intensive, especially when dealing with large datasets or complex recognition tasks, requiring substantial processing power and resources.

Applications

1. **Digital Note-taking:** Handwriting recognition facilitates seamless conversion of handwritten notes into digital format, simplifying storage and sharing for users across various platforms.
2. **Form Processing**: By automating the extraction of data from handwritten forms, handwriting recognition streamlines administrative tasks, reducing manual data entry errors and enhancing efficiency in industries reliant on paper-based documentation.
3. **Signature Verification:** Leveraging handwriting recognition technology, signature verification processes ensure the authenticity of handwritten signatures, bolstering security and trust in legal and financial transactions while mitigating the risk of fraud.

Analysis and Design

Data Collection/Preprocessing

Data Collection: IAM Dataset

The IAM (English handwriting) dataset serves as a benchmark dataset for handwritten text recognition tasks. It encompasses a wide array of handwritten text samples sourced from various documents, including forms, letters, and handwritten notes. Each sample within the IAM dataset is accompanied by ground truth annotations, providing the correct transcription of the handwritten text. This rich dataset offers a diverse range of handwriting styles, vocabulary, and writing conditions, making it an ideal choice for training and evaluating handwriting recognition models.

Key Features of the IAM Dataset:

1. **Diverse Handwriting Styles:** The IAM dataset encompasses a wide variety of handwriting styles, ranging from neat and legible to cursive and intricate. This diversity reflects real-world scenarios and ensures that models trained on the dataset are capable of recognizing and transcribing a broad spectrum of handwriting styles.
2. **Large Vocabulary:** The dataset includes a substantial vocabulary of English words and phrases, reflecting the natural variability of written language. This comprehensive vocabulary enables models to recognize and transcribe a wide range of textual content accurately.
3. **Ground Truth Annotations:** Each handwritten text sample in the IAM dataset is accompanied by ground truth annotations, providing the correct transcription of the handwritten text. These annotations serve as the reference standard for training and evaluating handwriting recognition models.
4. **Document Variety:** The dataset contains handwritten text samples sourced from various documents, including forms, letters, and handwritten notes. This diversity of document types ensures that models trained on the dataset are robust and capable of handling different document formats.

Data Preparation:

Upon downloading the IAM Handwriting Dataset, the dataset files were extracted to access the images and associated metadata. The "words.txt" file, containing details about each handwritten word, including its status, ID, and the actual word, was utilized to read label information. This step ensured access to accurate ground truth annotations necessary for training and evaluating the handwriting recognition model.

Image Preprocessing:

Custom preprocessing functions were implemented to prepare the images for the handwriting recognition task. Images were resized to a standardized shape of (32, 128, 1) to ensure uniformity across the dataset. Additionally, pixel values were normalized to a range between 0 and 1 to facilitate consistency in feature scaling. Aspect ratio adjustments were carefully handled to preserve the original proportions of the handwriting samples, ensuring minimal distortion during resizing.

Label Encoding:

Text labels were encoded into numerical values based on a predefined character list. Each character in a word was mapped to its corresponding index in the character list, facilitating compatibility with the model architecture. This encoding process prepared the textual labels for training the handwriting recognition model, enabling the model to recognize and transcribe handwritten text accurately.

Data Loading and Splitting:

The dataset was loaded, and images along with their corresponding labels were extracted for model training. To assess model performance, the dataset was split into training and validation sets using a stratified splitting approach. This ensured an even distribution of samples across the training and validation sets, preventing bias and facilitating robust evaluation of the model's performance.

Sequence Padding:

To accommodate deep learning models requiring fixed-length input sequences, sequences of labels were padded to ensure uniform length. The ‘pad_sequences’ function from Keras was employed to pad sequences with zeros up to the maximum sequence length. This step was crucial for maintaining compatibility with the model architecture and facilitating seamless training.

Data Verification:

Comprehensive data verification checks were conducted to ensure the integrity and consistency of the processed data. Specific indices in the training and validation datasets were examined to confirm successful preprocessing and padding. Additionally, the shapes of the training and validation data arrays were verified to validate the preprocessing steps, ensuring the dataset's readiness for model training and evaluation.

By following these steps, the IAM Handwriting Dataset is properly prepared and formatted for training a handwriting recognition model, ensuring that the model can effectively learn from the data and generalize well to unseen samples.

Study & Understanding of Algorithm:

The algorithm used is a combination of Convolutional Neural Networks (CNNs) and Long Short-Term Memory (LSTM) networks.

1. Convolutional Neural Networks (CNNs):

Convolutional Neural Networks (CNNs) are foundational in image processing tasks, including handwritten text recognition (HTR). In this context, CNNs are adept at capturing spatial features from input images, which is crucial for recognizing patterns in handwritten characters. CNN architectures typically consist of convolutional layers, which apply learnable filters to input images to extract features such as edges, corners, and textures. Subsequent pooling layers help reduce the dimensionality of extracted features while retaining important spatial information. By leveraging CNNs in HTR, we preprocess and extract meaningful features from handwritten text images, enabling subsequent layers to effectively transcribe the text.

2. Long Short-Term Memory (LSTM) Networks:

Long Short-Term Memory (LSTM) networks are a specialized type of recurrent neural network (RNN) architecture designed to address the vanishing gradient problem and capture long-term dependencies in sequential data. In the context of HTR, LSTM networks excel at processing variable-length sequences, making them well-suited for transcribing handwritten text. LSTM layers utilize a system of gates, including input, forget, and output gates, to selectively update and propagate information over time. This enables the model to maintain context and effectively transcribe sequential inputs, such as handwritten characters. By incorporating LSTM layers after the feature extraction stage, we empower the model to decode the sequence of features extracted by the CNNs and output accurate textual representations of the handwritten text.

3. Connectionist Temporal Classification (CTC) Loss Function:

The Connectionist Temporal Classification (CTC) loss function is instrumental in training sequence-to-sequence models, such as those used in speech recognition and HTR. Unlike traditional loss functions, CTC loss allows the model to learn from sequences of variable lengths by aligning input sequences with target sequences without requiring explicit alignment information. This makes it particularly well-suited for transcribing handwritten text, where the length of input sequences may vary. By minimizing the CTC loss during training, the model learns to accurately transcribe handwritten text images into corresponding textual representations, effectively capturing the variability and complexity of handwriting styles.

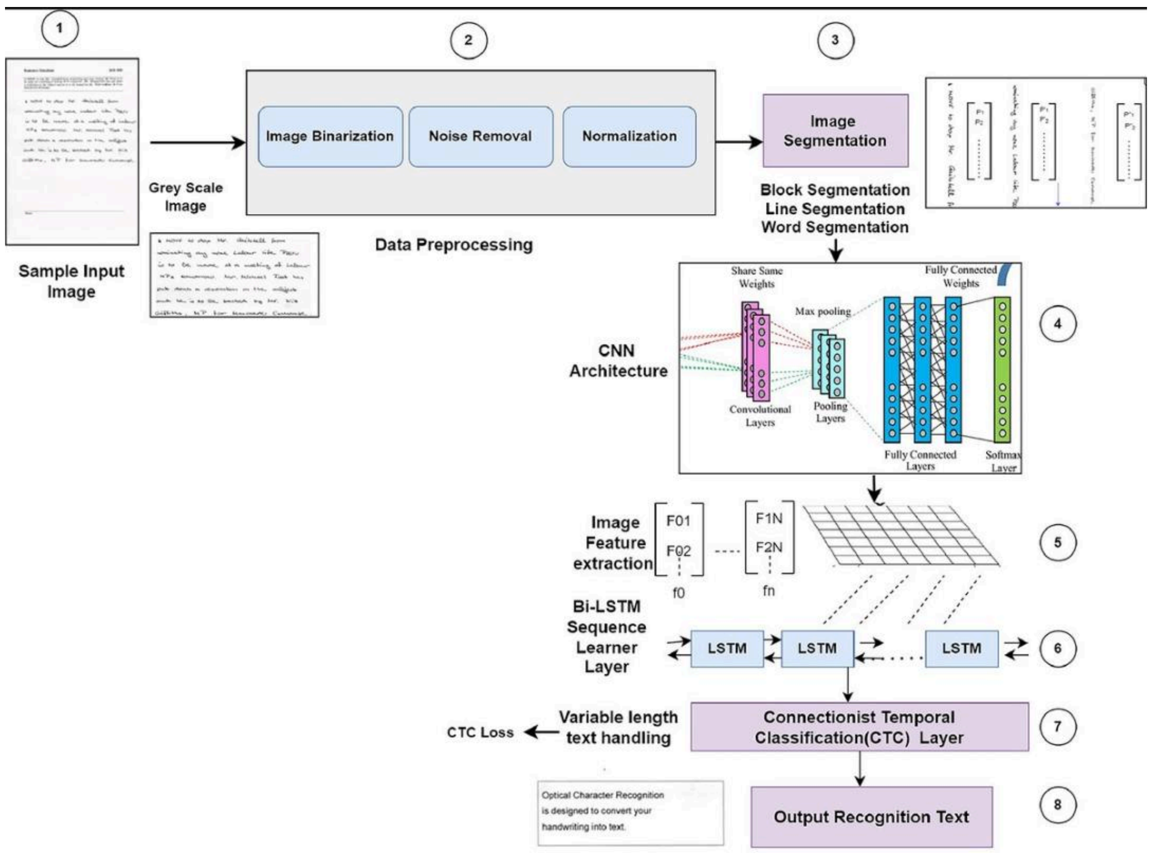
4. Optimization and Training:

Optimization and training play a pivotal role in fine-tuning the parameters of the CNN-LSTM architecture for optimal performance. Stochastic gradient descent (SGD) is a commonly used optimization algorithm that iteratively adjusts the model's parameters to minimize the CTC loss function. Hyperparameters such as learning rate, batch size, and number of epochs are carefully tuned to optimize the model's performance while preventing overfitting. Throughout the training process, the model learns to extract relevant features from handwritten text images and generate accurate textual representations through iterative optimization, ultimately enhancing its ability to transcribe handwritten text effectively.

In conclusion, the integration of CNNs, LSTM networks, and the CTC loss function enables our model to preprocess input images, extract meaningful features, and transcribe handwritten text into textual representations accurately. Through careful optimization and training, the model learns to recognize diverse handwriting styles and effectively transcribe handwritten text, showcasing its potential for various applications in text recognition and digitization.

The model we have used:

1. The input shape for our architecture has an input image of a height of 32 and a width of 128.
2. Here we used seven convolution layers, of which 6 have kernel size (3,3) and the last one is of size (2.2). The number of filters is increased from 64 to 512 layer by layer.
3. Two max-pooling layers are added with size (2,2), and then two max-pooling layers of size (2,1) are added to extract features with a larger width to predict long texts.
4. Also, we used batch normalization layers after the fifth and sixth convolution layers, which accelerates the training process.
5. Then we used a lambda function to squeeze the output from the convolution layer and make it compatible with the LSTM layer.
6. Then two Bidirectional LSTM layers, each of which has 128 units. This RNN layer gives the output of size (batch_size, 31, 79). Where 79 is the total number of output classes, including blank characters.



Study of Performance Measurement Criteria:

Jaro similarity is a performance measurement criterion commonly used in tasks where the similarity between two strings needs to be evaluated. In the context of handwritten text recognition, Jaro similarity can serve as a valuable metric for assessing the similarity between the predicted text generated by the model and the ground truth text derived from the handwritten images.

Methodology:

1. Prediction Generation: Utilize the trained model to generate predictions for the handwritten images in the dataset.

2. Ground Truth Extraction: Extract the ground truth text corresponding to each handwritten image.

3. Jaro Similarity Calculation: Calculate the Jaro similarity between each predicted text and its corresponding ground truth text. Implement the Jaro similarity calculation algorithm, considering the number of matching characters and transpositions between the two strings.

4. Aggregation of Scores: Aggregate the Jaro similarity scores obtained for all predictions to derive an overall performance measure.

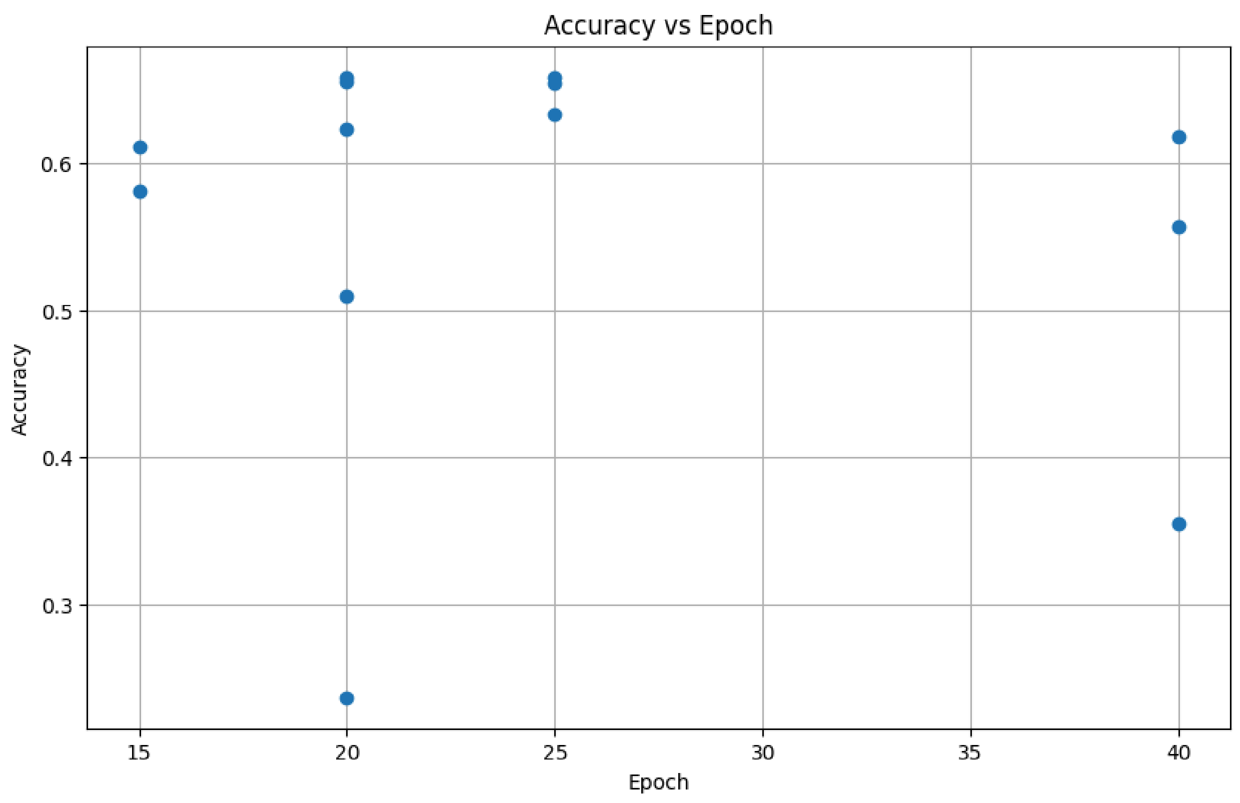
Interpretation:

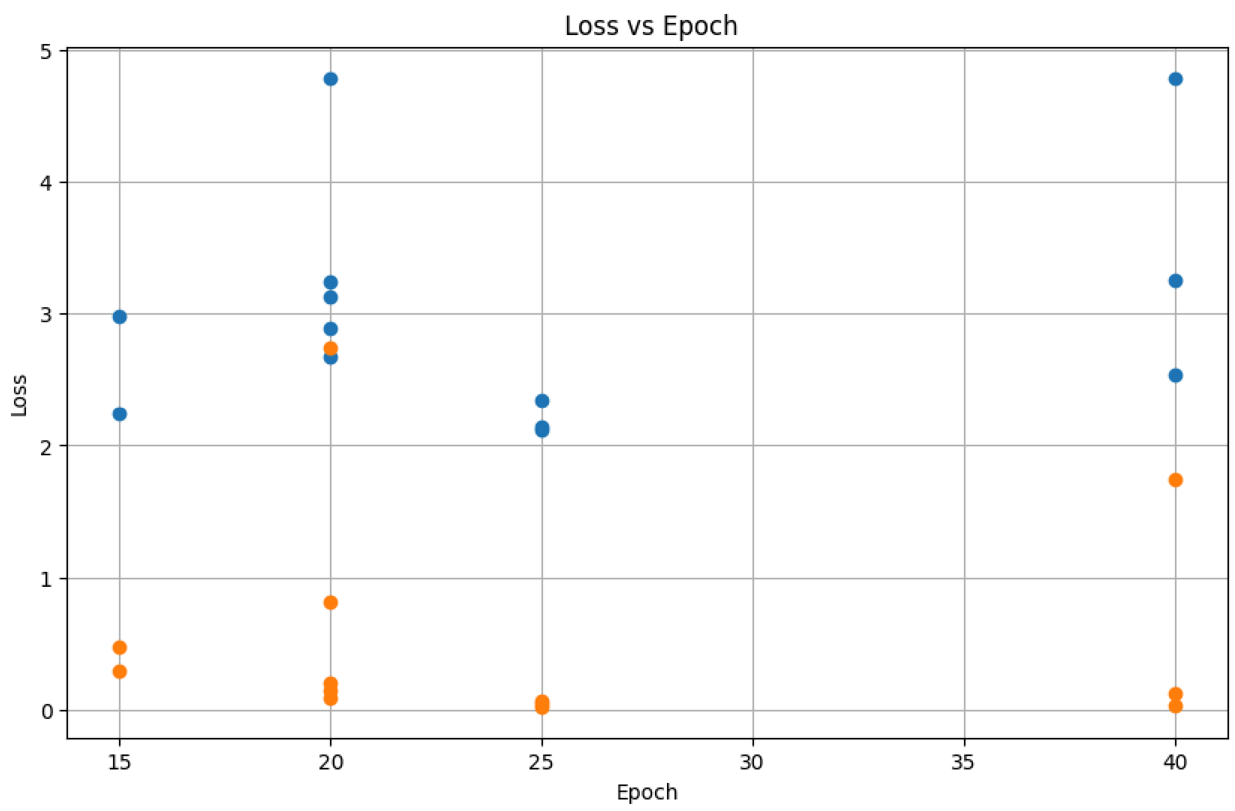
The Jaro similarity scores obtained provide insights into the accuracy of the model's predictions in transcribing handwritten text. A higher Jaro similarity score indicates a closer resemblance between the predicted text and the ground truth text, suggesting better performance in recognizing and transcribing handwritten characters. Conversely, a lower Jaro similarity score suggests discrepancies between the predicted and ground truth text, indicating potential areas for improvement in the model.

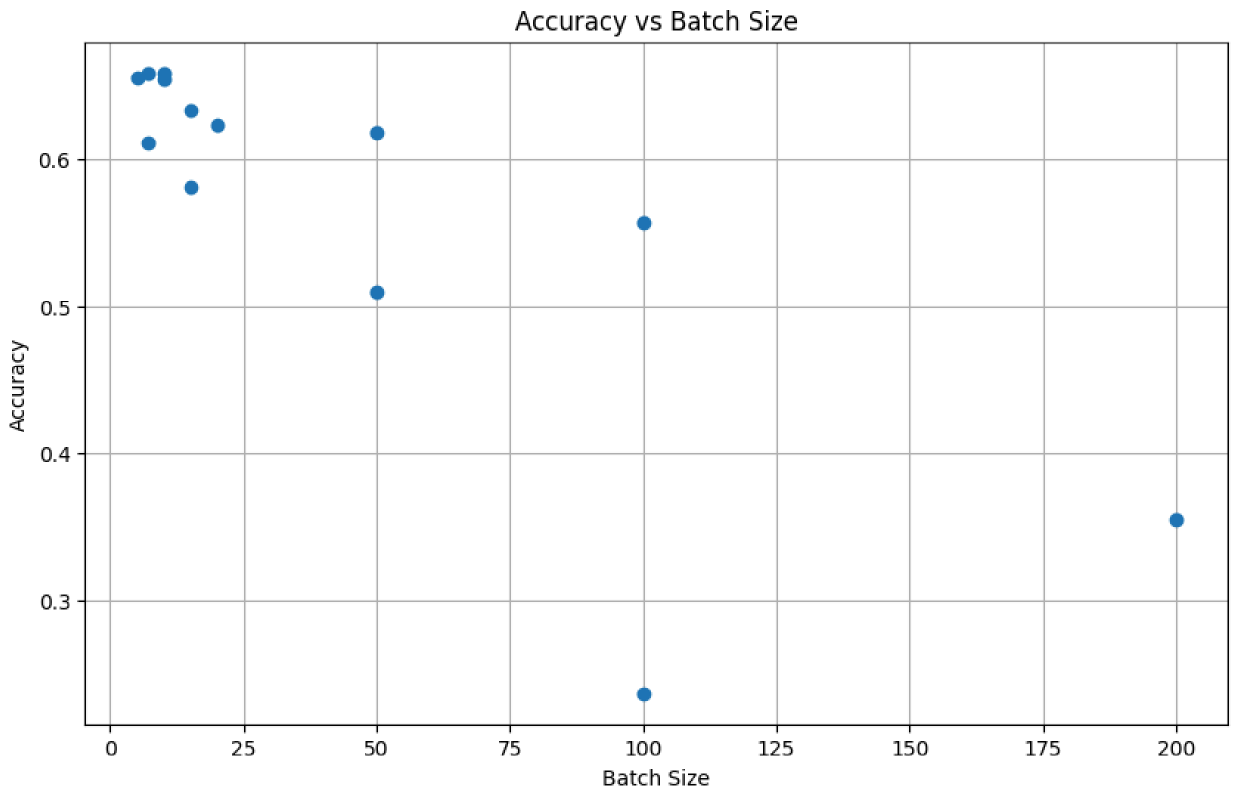
Coding, Experimentation & Results

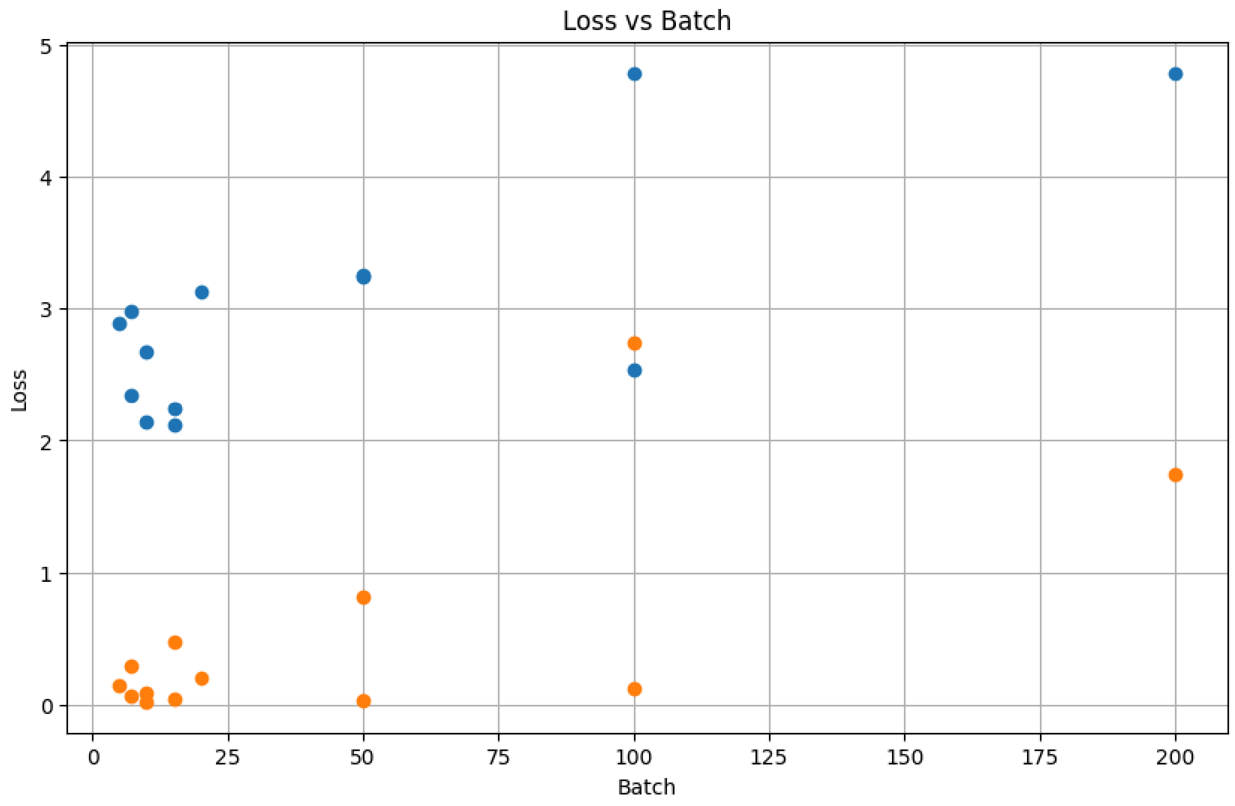
We trained our model over different batch sizes and number of epochs. Each training and related testing took about 5-6 hours to complete, depending on the parameters used.

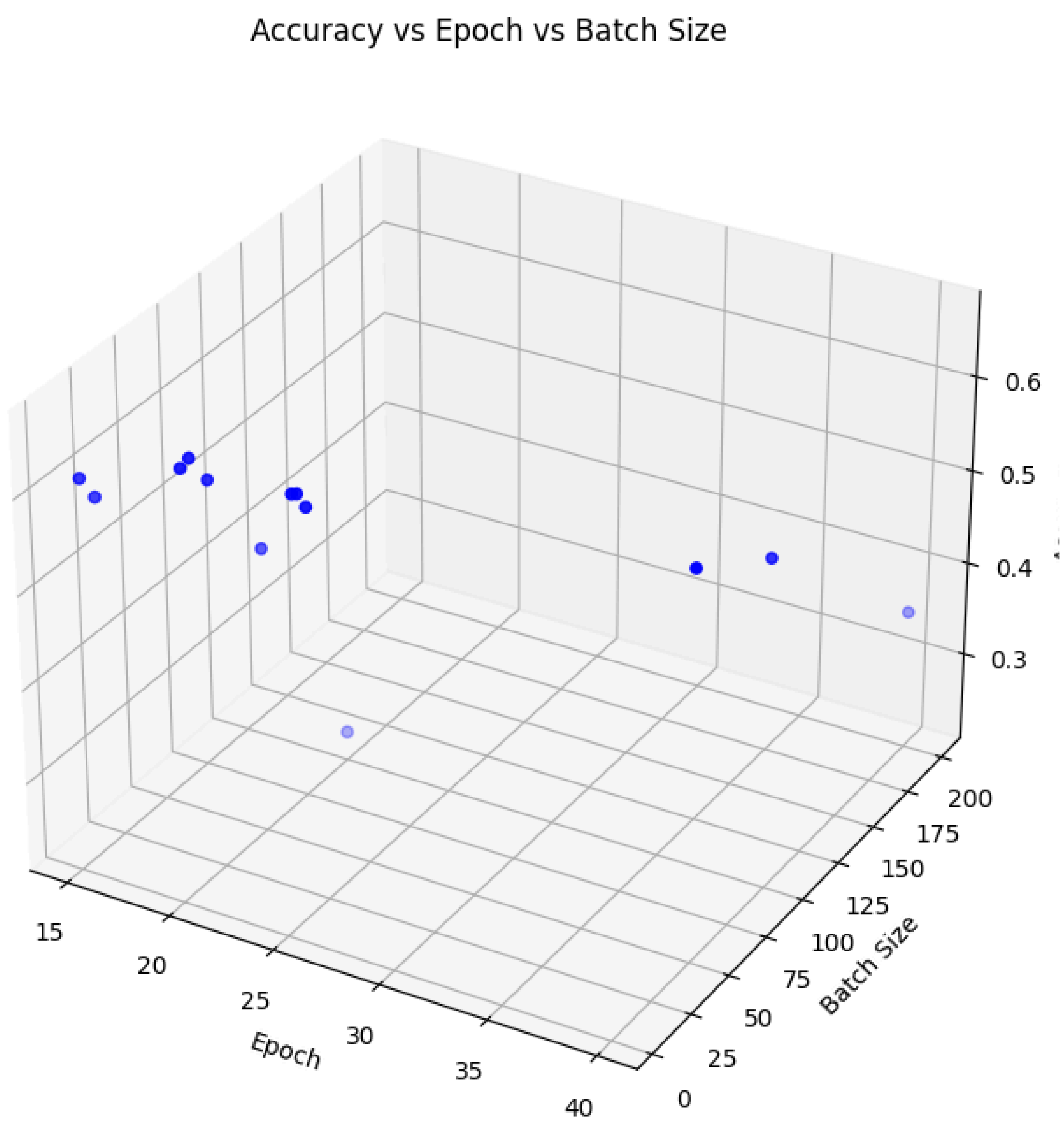
The link to the data we collected during this is: Experiment_Data

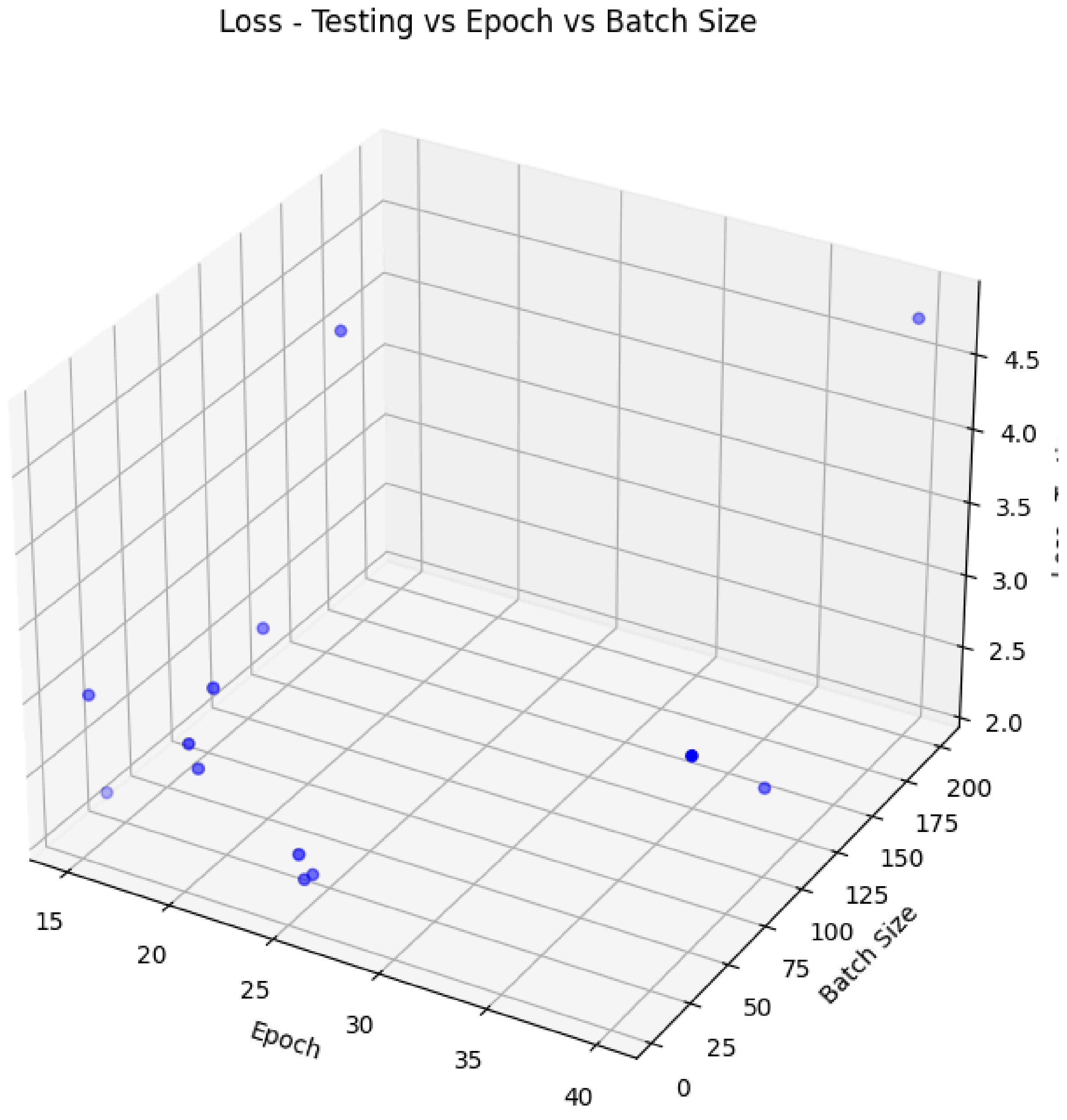


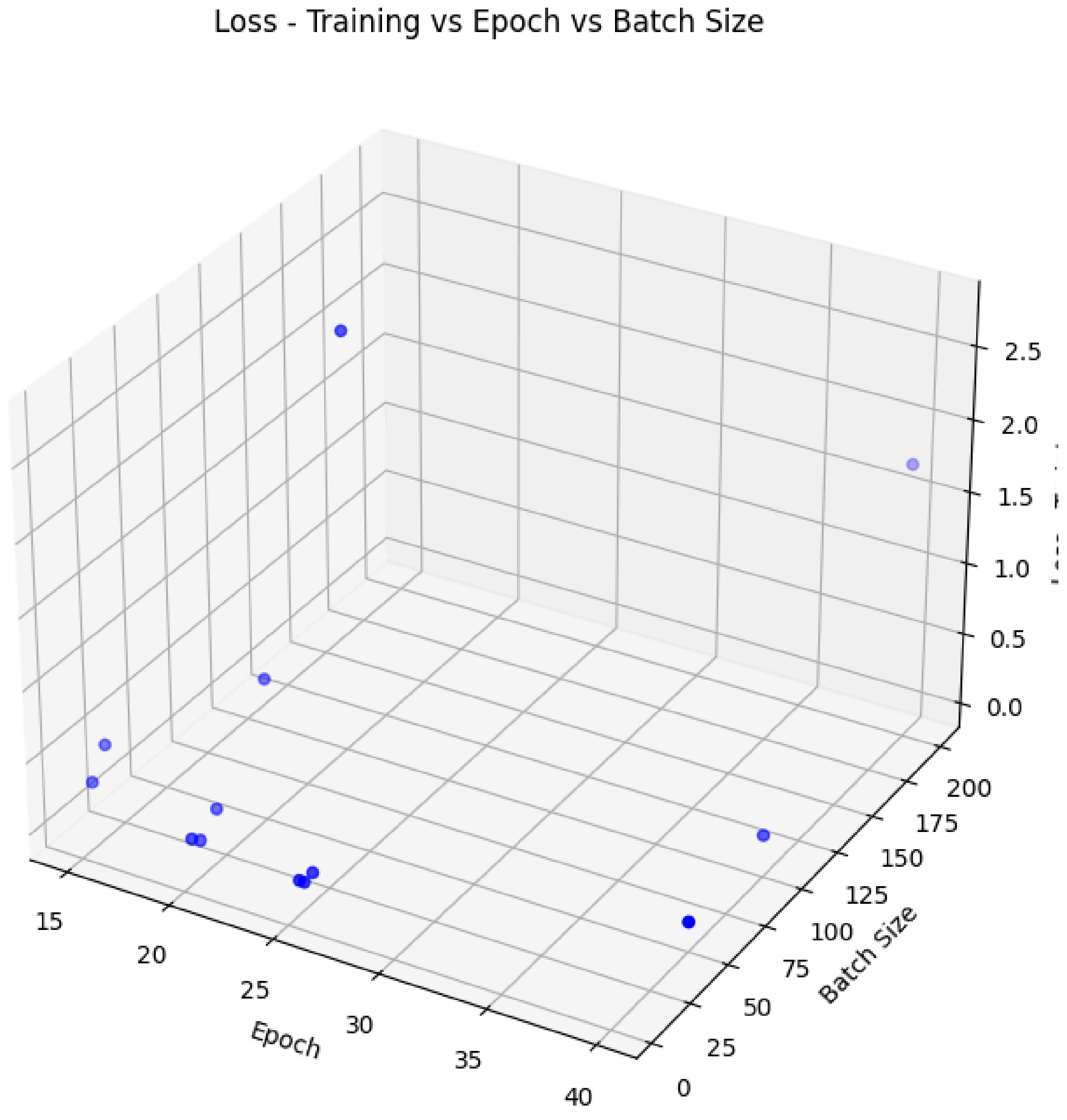

Loss vs Epoch
Loss
Epoch
0
1
2
3
4
5
15
20
25
30
35
40

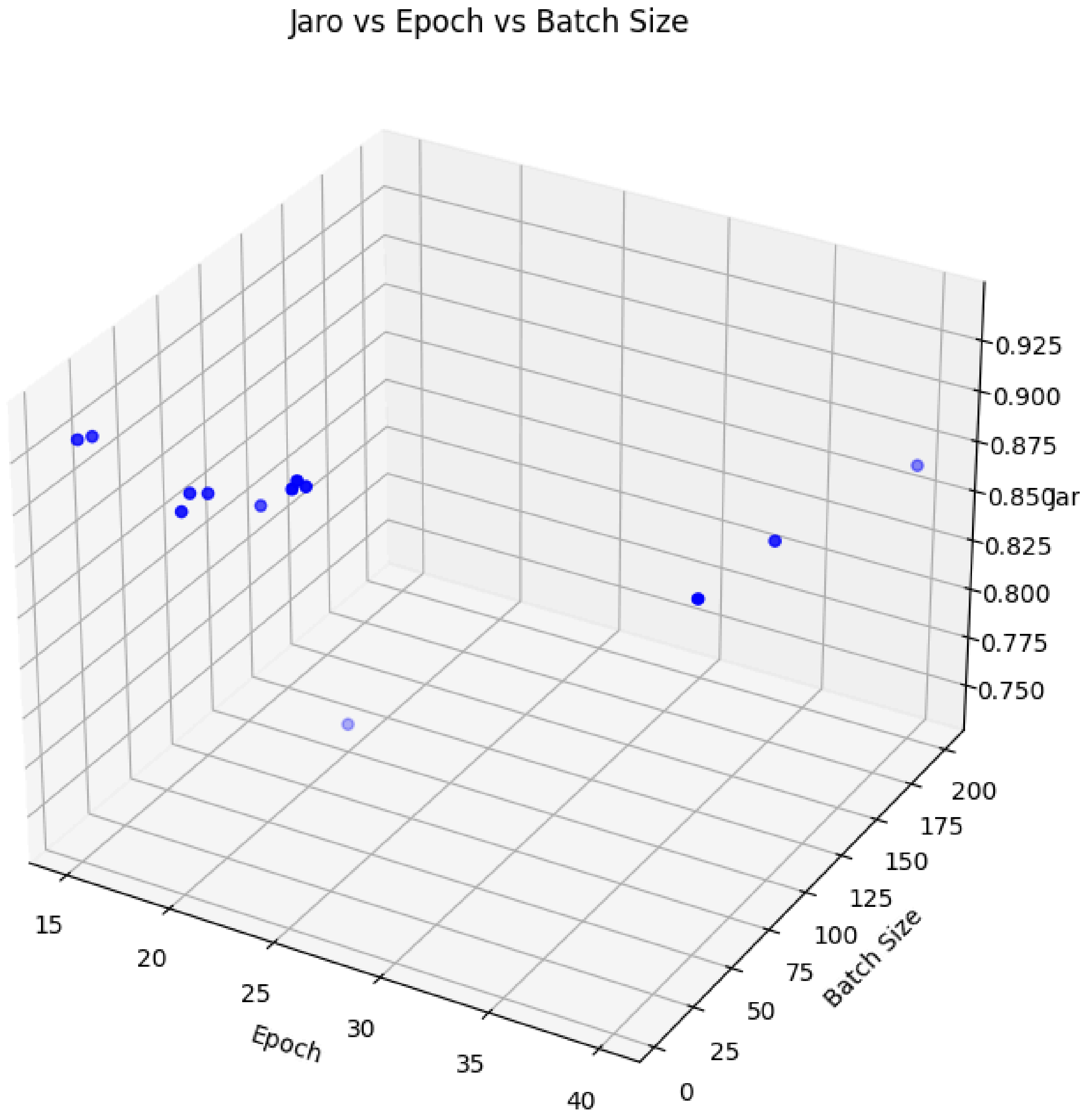

Accuracy vs Batch Size
Accuracy
Batch Size
0.3
0.4
0.5
0.6
0
25
50
75
100
125
150
175
200


Loss vs Batch
Loss
Batch
0
1
2
3
4
5
0
25
50
75
100
125
150
175
200


Accuracy vs Epoch vs Batch Size
Epoch
15
20
25
30
35
40
Batch Size
0
25
50
75
100
125
150
175
200
0.3
0.4
0.5
0.6


Loss - Testing vs Epoch vs Batch Size
Epoch
15
20
25
30
35
40
Batch Size
0
25
50
75
100
125
150
175
200
2.0
2.5
3.0
3.5
4.0
4.5


Loss - Training vs Epoch vs Batch Size
Epoch
Batch Size
15
20
25
30
35
40
0
25
50
75
100
125
150
175
200
0.0
0.5
1.0
1.5
2.0
2.5


Jaro vs Epoch vs Batch Size
0.925
0.900
0.875
0.850
0.825
0.800
0.775
0.750
Jar
15
20
25
30
35
40
Epoch
0
25
50
75
100
125
150
175
200
Batch Size

Conclusion

This study presents an innovative approach to offline sentence recognition utilizing convolutional neural networks (CNNs) and bidirectional long short-term memory (BiLSTM) networks, augmented by a connectionist temporal classification (CTC) decoder. Departing from traditional methodologies, which directly input paragraph images into the neural network for recognition, our proposed system first preprocesses the images using OpenCV contour algorithms to segment them into line images. Subsequently, the line images undergo further processing to extract word images. These extracted images are then fed into the CNN and BiLSTM layers for feature extraction and sequential processing, respectively. The CTC decoder decodes the output sequence generated by the BiLSTM layers to produce the final recognized text. Experimental results exhibit noteworthy enhancements in recognition accuracy, achieving accuracies of 65% BiLSTM with CTC, respectively.

Future Improvements and Implementations

Integration of Hybrid Datasets:

1. Incorporate diverse datasets from different sources to enhance the model's ability to recognize a broader range of text styles, fonts, and languages.
2. Combining datasets with varying characteristics can improve the model's robustness and generalization capability.

Exploration of Various Activation Functions:

1. Experiment with different activation functions (e.g., Leaky ReLU, ELU, Swish) to optimize the performance of neural network layers.
2. Selecting appropriate activation functions can lead to faster convergence and improved model accuracy.

Incorporation of Additional Neural Network Layers:

1. Explore the addition of new layers such as attention mechanisms or self-attention mechanisms to enhance the model's ability to focus on relevant parts of the input data.
2. Incorporating residual connections or skip connections can facilitate the training of deeper networks and improve information flow.

Enrichment of Language Support:

1. Extend language support beyond the current scope by training the model on datasets containing texts from multiple languages.
2. Incorporate multilingual embeddings or language-specific pre-trained models to enhance the model's understanding of different languages.

Integration of Online Recognition:

1. Extend the system's capabilities to support online handwriting recognition, enabling real-time processing of handwritten input.
2. Implement techniques such as stroke-based recognition or dynamic time warping to handle online handwriting data effectively.

Handling Poor Quality or Fragmented Characters:

1. Develop strategies to preprocess and augment training data to simulate poor quality or fragmented characters.
2. Explore data augmentation techniques such as rotation, scaling, and noise addition to make the model more robust to imperfect input.

Adaptation to Specific Domains:

1. Customize the model for specific domains or applications by fine-tuning on domain-specific datasets.
2. Incorporate domain knowledge or constraints into the model architecture or training process to improve performance in specific scenarios.

Deployment and Evaluation in Real-world Scenarios:

1. Deploy the system in real-world scenarios to evaluate its performance and usability in practical applications.
2. Collect feedback from users and stakeholders to identify areas for further improvement and refinement.

References

1. Enhancement of handwritten text recognition using AI-based hybrid approach
2. https://www.tensorflow.org/tutorials/images/cnn
3. https://medium.com/@raghavaggarwal0089/bi-lstm-bc3d68da8bd0
4. https://pytorch.org/docs/stable/generated/torch.nn.CTCLoss.html