Identifying Handwritten Letters with Semi Supervised Learning

Team Members:

Nithin Chiguru – A20583854

Vijaya Satya Aditya Karri – A20581776

Devendra Babu Kondragunta – A20550546

Introduction

- Handwritten character recognition is a crucial task in fields like document digitization and automated form processing.
- The goal of our project is to accurately identify handwritten letters using semi-supervised learning methods.
- By leveraging semi-supervised learning and a Convolutional Neural Network (CNN), we aim to reduce the need for large labeled datasets while maintaining high accuracy in letter recognition.
- Our project explores how semi-supervised learning can be a cost-effective alternative to fully supervised models, making AI solutions more accessible to organizations with limited resources.
- The robustness of CNNs allows our model to handle noisy and imperfect data, which is common in real-world handwritten samples.
- The project includes an experimental custom-built GUI that allows users to draw on screen, making it accessible for non-technical users to utilize the model effectively.

Objectives & Research Questions

ACHIEVE 98%+ ACCURACY ON HANDWRITTEN LETTER RECOGNITION WITH MINIMAL LABELED DATA.

USE PSEUDO-LABELING TO WITH UNLABELED DATA TO IMPROVE THE PERFORMANCE.

HOW ACCURATE IS SEMI-SUPERVISED CNN WITH LIMITED LABELED DATA?

HOW CAN IT BE GENERALIZED TO REAL-WORLD HANDWRITING?

WHAT PREPROCESSING CONTRIBUTES MOST TO PERFORMANCE?

Semi Supervised Learning

- Semi-supervised learning is a machine learning approach that uses a combination of labeled and unlabeled data to improve model performance, especially when labeled data is limited.
- It sits between supervised learning (which uses only labeled data) and unsupervised learning (which uses only unlabeled data), making it a versatile method for training models.
- This approach is particularly useful in domains like handwriting recognition, where obtaining labeled data is time-consuming and costly, but large amounts of unlabeled data are readily available.
- In semi-supervised learning, models can use self-training techniques, where they iteratively label unlabeled data based on their own predictions, refining the model over time.
- As data continues to grow exponentially, semi-supervised learning offers a promising way to harness large volumes of unlabeled data, making it a crucial tool in the future of AI development.

Algorithm

- Convolutional Neural Networks (CNNs) are powerful algorithms used in semi-supervised learning for image recognition tasks.
- Convolutional layers for feature extraction and fully linked layers followed by a soft-max layer for classification make up the architecture of conventional CNN classifiers.
- The neurons in CNN's architecture are arranged in layers. An input layer, numerous hidden layers, and an output layer make up this structure.
- This method decreases the number of connection weights in CNN. Because of this, CNN trains faster than networks of a similar size.
- The convolution procedure develops a feature map as the convolution kernel moves across the input matrix for the layer, adding to the input of the following layer.
- Other layers such as pooling layers, fully linked layers, and normalization layers are then added.

Motivation

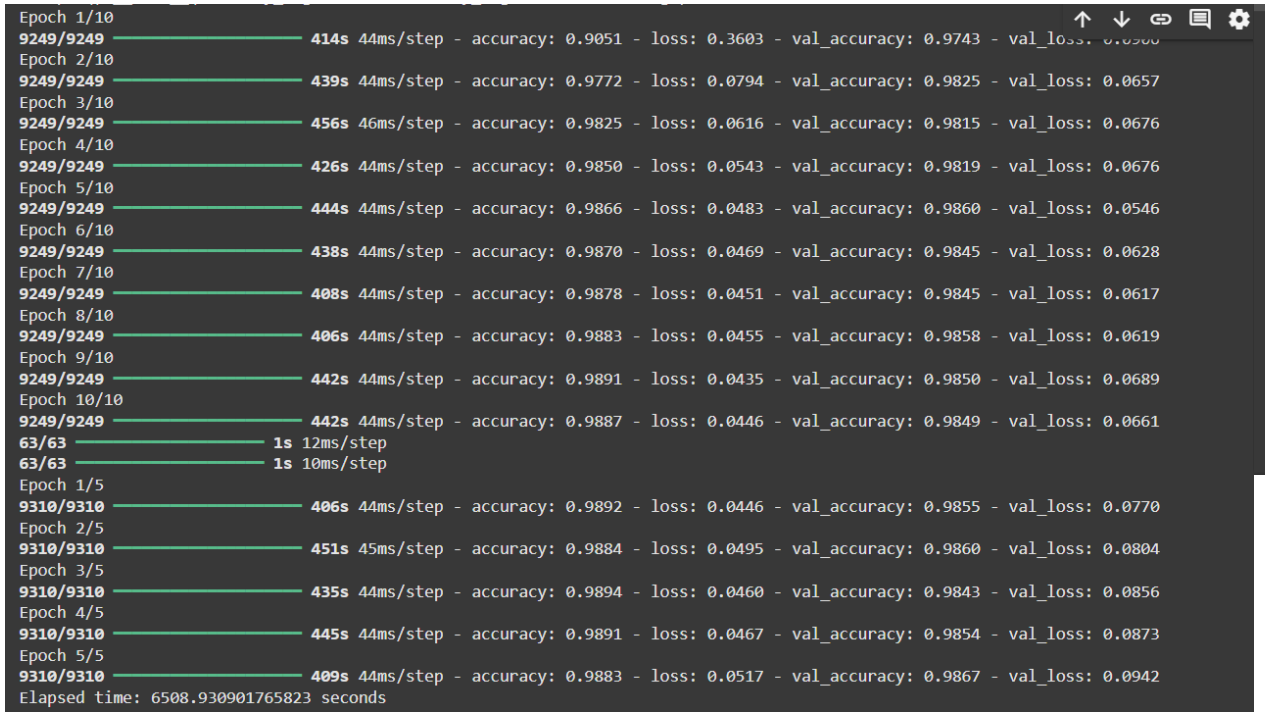
- Handwritten letter recognition is essential for processes like Postal mail sorting, Digital Transformation, Automating Government and Legal Processes, Future-Proofing, and quick processing of handwritten forms in banks, reducing both time and human error.
- Traditional models require extensive manual labeling. This approach reduces this need by utilizing a combination of labeled and unlabeled data, making the system both efficient and adaptable.
- A reliable handwritten recognition system could serve as the foundation for creating paperless environments, supporting digital transformation in various industries by automating document processing.
- Old handwritten files and records can be directly digitized into text, eliminating the need for manual rewriting or duplication on paper. This reduces paper usage and contributes to environmental sustainability by minimizing deforestation and waste.

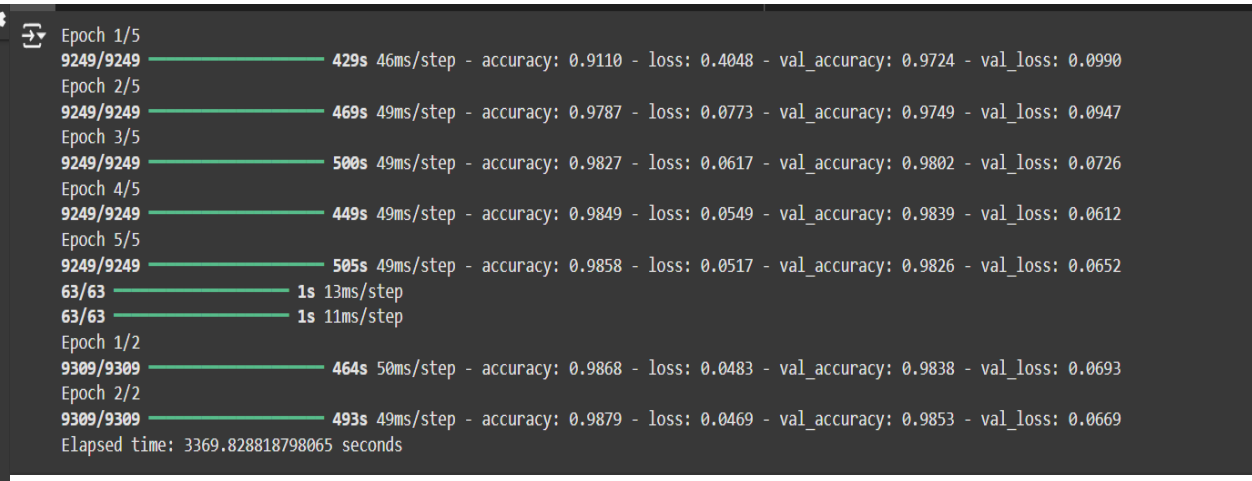
Dataset description

- EMNIST Dataset: A widely used dataset of over 60,000 grayscale images of handwritten English letters at 28×28 pixels. It provides a robust benchmark for letter-recognition models. Images and labels are stored in CSV form for efficient ingestion in R.
- Designed for handwritten character recognition, particularly for English alphabets.
- The combined dataset consists of over 60,000 images.
- The csv file contains the 60,000 training examples and labels

- **Thresholding**: Use `imager::threshold(grayscale_img, "auto")` to binarize each image and remove background artifacts.
- **Gaussian Blur**: Apply `imager::isoblur(binary_img, sigma = 1)` to smooth out noise while preserving stroke details.
- **Normalization**: Scale pixel values to [0, 1] by dividing by 255 for stable training dynamics.
- The dataset is split into a training set (80%) and a testing set (20%) to train and evaluate the CNN model effectively.
- Expected model accuracy is above 97%, with current results showing around 98.32% accuracy for letter recognition.
- **Here, we are using 7 epochs instead of 15 because 7 itself takes more than 1 hour to execute so 15 will take considerably more.

Time Elapsed for 7 and 15 epochs





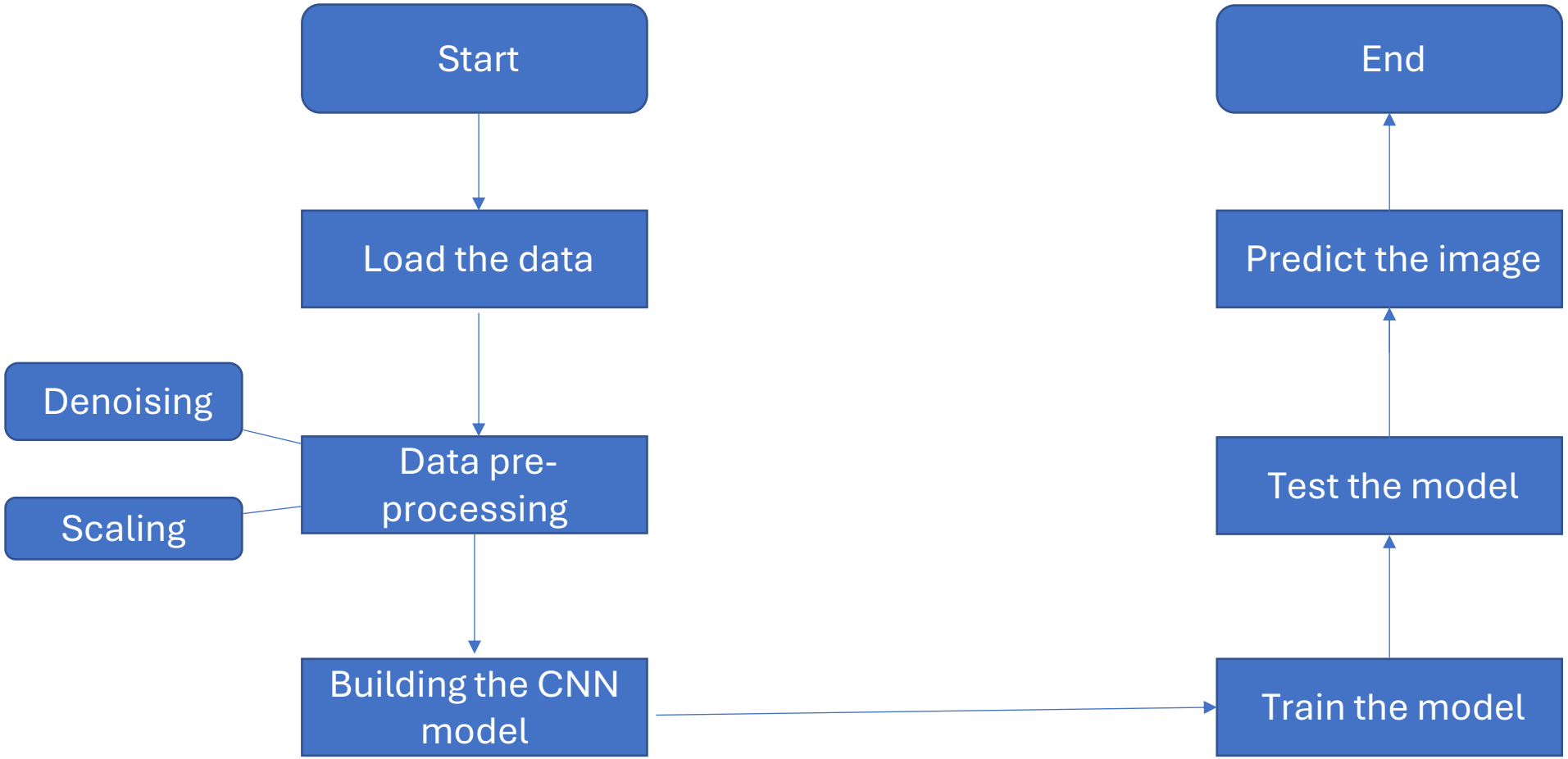
- **For 15 epochs**

For 7 epochs

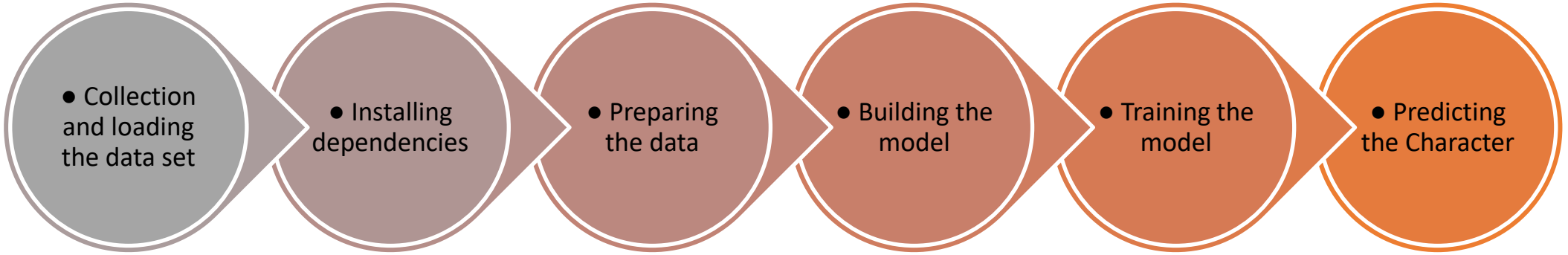
System Architecture

- Data Ingestion → Preprocessing → CNN Model → Prediction
- GUI used for real-time handwritten input and prediction display.
- Layers: Convolution → ReLU → Pooling → Dense → Softmax
- Semi-supervised: Label + high-confidence pseudo - labels used in training.

Workflow


Start
Load the data
Denoising
Data pre-processing
Scaling
Building the CNN model
Train the model
Test the model
Predict the image
End

MODULES


● Collection and loading the data set
● Installing dependencies
● Preparing the data
● Building the model
● Training the model
● Predicting the Character

Libraries

keras

tensorflow

data.table

imager

magick

tidyverse

Visual Results

- EMNIST sample grayscale images used in training.
- Model predictions validated via GUI and test samples.
- Accuracy and loss plots used to track training progression.
- → High confidence predictions used for pseudo-labeling.

Implementation Environment

- Tools: RStudio, Keras, TensorFlow, Imager, Tidyverse
- Platform: Windows 10 / Ubuntu 20.04
- Hardware: Intel i7 CPU, 8GB RAM
- Preprocessing: Thresholding, Gaussian Blur, Normalization
- Code & Artifacts managed in GitHub for reproducibility.

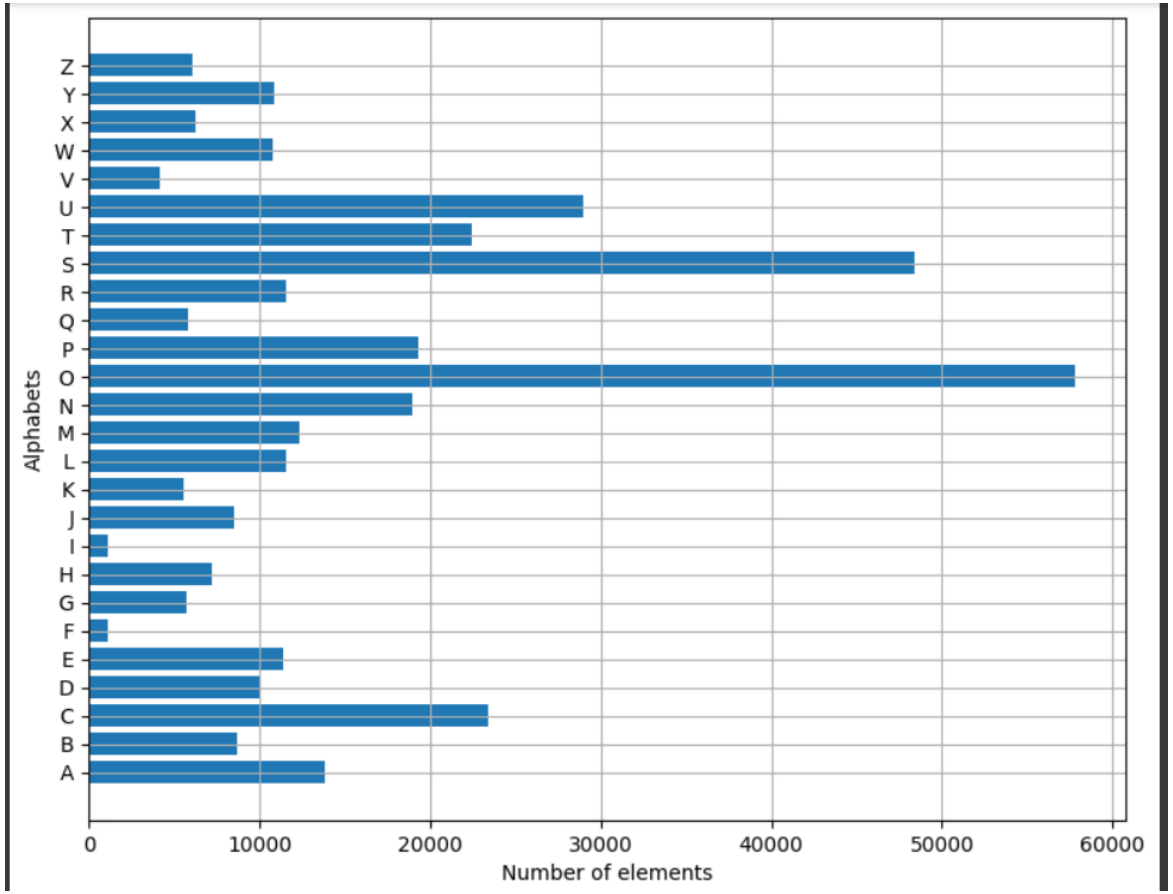
Code Workflow

- Data Loading & Shuffling
- Load the CSV into a fast data.table for efficient access.
- Separate pixel columns into a feature matrix and the label column into a vector.
- Randomize row order with dplyr::sample_frac(1) to break any sequence bias.
- Ensure every training epoch sees a fresh, shuffled dataset for better generalization.
- Visualizing Alphabet Distribution
- Convert numeric labels (1–26) to their corresponding letters (A–Z).
- Tally the number of samples per letter to detect any imbalance.
- Plot a horizontal bar chart with ggplot2 to visualize class frequencies.
- Confirm that each letter has roughly equal representation before training.

Code Workflow

- Why Shuffle?
- Prevents the network from learning spurious patterns tied to the original data order.
- Forces each minibatch to contain a random mix of letters, improving robustness.
- Helps the model generalize by exposing it to varied sample sequences every epoch.
- Why Reshape?
- CNNs require input tensors of shape (batch_size, height, width, channels).
- Add a singleton channel dimension to each 28×28 image, yielding (n, 28, 28, 1).
- Enables convolutional filters to traverse the two spatial dimensions correctly.
- Preserves the spatial structure necessary for extracting stroke and edge features.

Visualization



Code Workflow

Data Splitting:

- Use caret::createDataPartition(y, p = 0.8) or caTools::sample.split(y, SplitRatio = 0.8) to carve out 80% of samples for training and 20% for testing, ensuring each letter class remains proportionally represented.

Reshaping for CNN Input:

- Convert the flattened pixel matrix back into a four-dimensional array of shape (n, 28, 28, 1) via R's array() function so that each sample is structured as (height × width × channels) for the convolutional layers.

One-Hot Encoding:

- Apply keras::to_categorical(labels, num_classes = 26) to turn integer class labels into 26-length binary vectors, matching the network's softmax output dimension.

Why One-Hot Encode?

- Prevents the model from inferring any ordinal relationship among letter classes

Code Workflow

- Aligns label representation with softmax probability outputs
- Enables correct computation of categorical cross-entropy loss and accuracy metrics

Why Reshape Again?

- Guarantees the input tensor adheres to the required (batch, height, width, channels) format
- Avoids dimension-mismatch errors when feeding data into Keras/TensorFlow layers
- Ensures that every training and inference call uses correctly shaped arrays for convolutional operations

Model Architecture

- We stack three convolutional blocks, each comprising a 3×3 kernel with ReLU activation, to progressively capture simple edges (32 filters), mid-level shapes (64 filters), and high-level textures (128 filters).

Code Workflow

- Each block is followed by a 2×2 max-pooling layer with stride 2, which halves the spatial dimensions while retaining the strongest activations—this not only reduces computation but also introduces a degree of translation invariance.
- After the final pooling layer, we flatten the 3D feature maps into a single vector, then pass through two fully connected (dense) layers (64 units, then 128 units) to learn non-spatial combinations of features, before outputting class probabilities over 26 letters with a softmax layer.

Compilation & Loss Setup

- We choose the Adam optimizer for its adaptive learning rate capabilities—automatically scaling updates per-parameter based on first and second moments of gradients—which expedites convergence while requiring minimal manual tuning.
- Our loss function is categorical cross-entropy, the standard choice for multi-class classification with one-hot–encoded targets, ensuring the network is penalized more heavily as its predicted probability diverges from the true class.

Code Workflow

- We track accuracy during training to monitor how often the model's top prediction matches the actual label, and also observe loss trends to detect underfitting or overfitting early.

Initial Supervised Training

- The network is first trained for five epochs on the labeled subset, with each epoch performing a full pass over all training samples in random minibatches (batch size = 128).
- We employ early stopping (patience = 3) to halt training if the validation loss does not improve for three consecutive epochs, preventing overfitting and unnecessary training time.
- A learning-rate reduction on plateau callback cuts the optimizer's learning rate by half whenever validation loss stagnates, allowing finer-grained updates during later epochs.

Pseudo-Labeling & Fine-Tuning

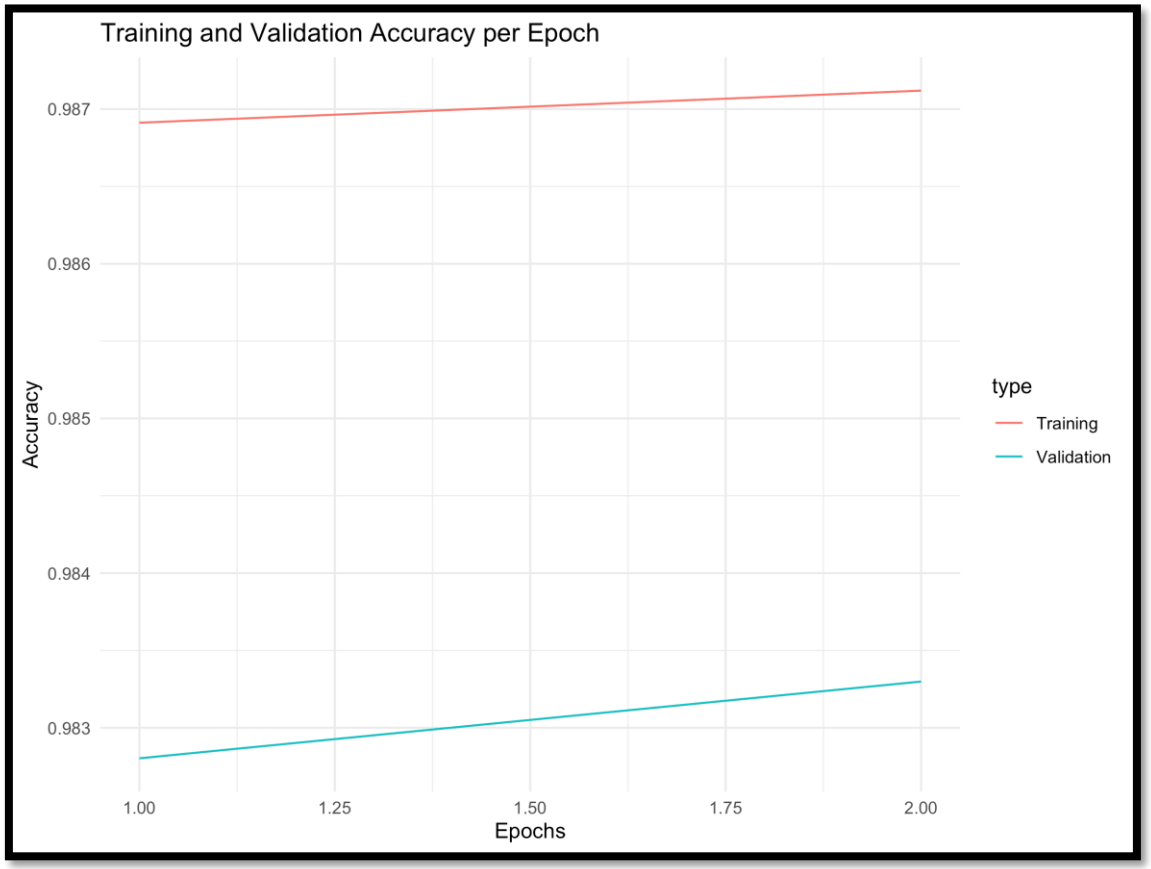
- After the initial training phase, the model predicts labels for the unlabeled pool, and we select only those predictions whose top-class probability meets or exceeds the 0.90 confidence threshold.

Code Workflow

- These high-confidence samples receive their predicted labels as pseudo-labels and are appended to the original labeled set, effectively enlarging the training data by up to ~20–30%.
- We then retrain for two additional epochs on this combined dataset, enabling the model to refine its decision boundaries using both human-annotated and machine-generated labels, which often yields a measurable boost in final accuracy.

Saving & Final Evaluation

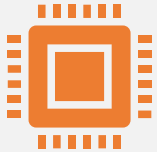
- Once training completes, we save the model (architecture, weights, optimizer state) to disk in HDF5 format, allowing instant reload for inference or further fine-tuning without repeating the training process.
- We perform a final evaluation on the held-out test set—computing loss and 98.32% test accuracy—and generate a confusion matrix and precision/recall/F1 metrics to pinpoint which letter pairs remain most challenging.
- This comprehensive evaluation confirms that our semi-supervised approach not only achieves high overall accuracy but also maintains balanced performance across all 26 classes.



Code Workflow

- Extract Metrics from the History Object: After training, pull the training and validation accuracy vectors directly from the history object (e.g., history$metrics$accuracy and history$metrics$val_accuracy).
- Prepare a Plotting Data Frame: Combine the epoch index with the two accuracy series into a single R data frame—one column for epoch number, one for training accuracy, and one for validation accuracy.
- Visualize with ggplot2: Use ggplot2 to draw two lines on the same chart—one for training accuracy, one for validation accuracy—mapping epoch on the x-axis and accuracy on the y-axis, and include a legend to distinguish them.
- Interpret the Curve: This plot reveals how quickly the model learns (steep rise in training accuracy), whether it generalizes well (validation curve tracking training), and if/when overfitting occurs (validation accuracy plateauing or dropping).

Individual Contribution



Nithin– Implementing the Algorithm and the experimental GUI

Aditya– Researching for the Dataset and Dataset Preprocessing

Devendra– Researching the right Algorithm and had insights about the GUI

Future Work

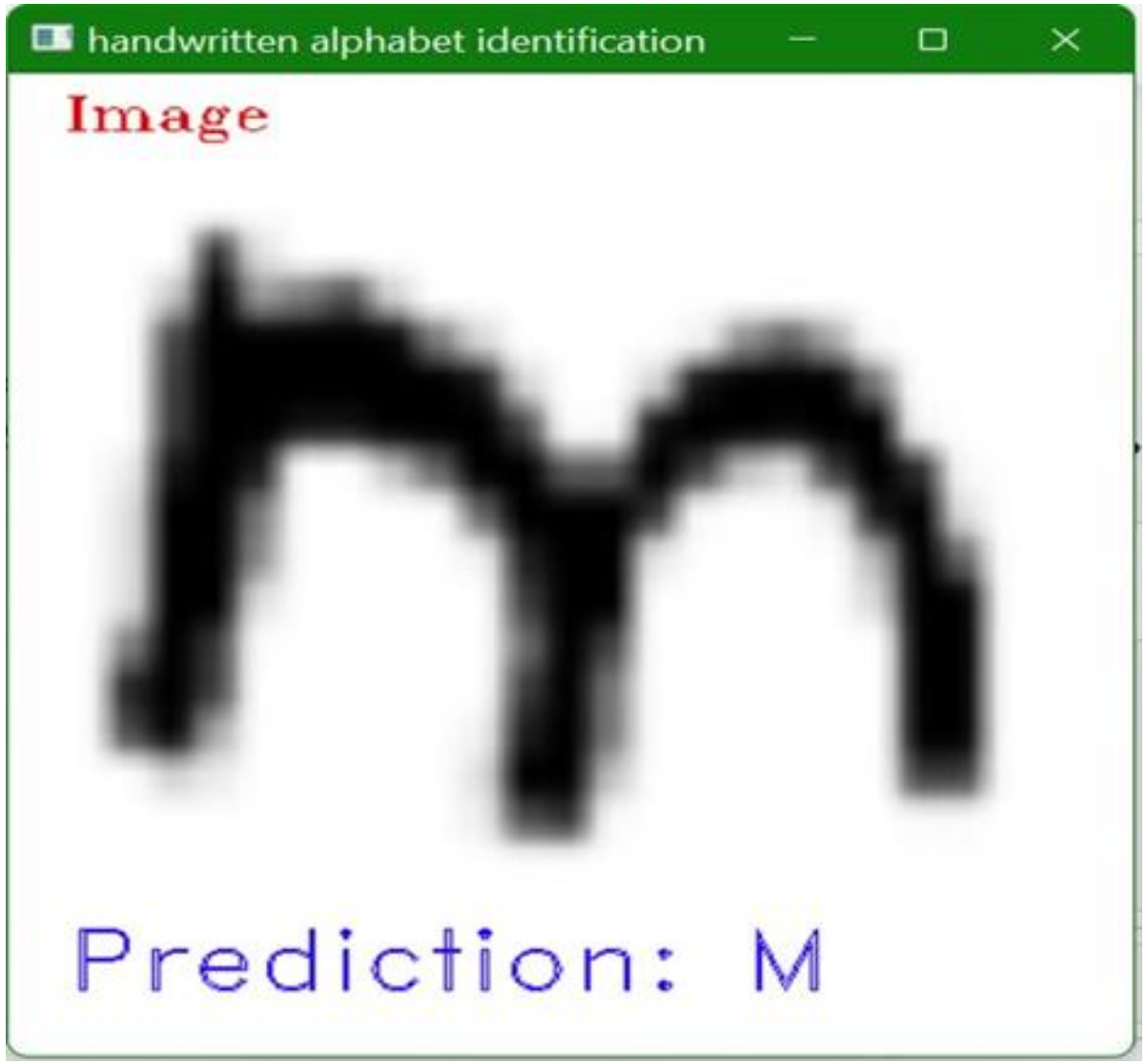
- Research more on how to make the algorithm work better for real time drawn images as in the drawing interface incorrect prediction for hand drawn images is prevalent.
- Research more on which other algorithm can be used in place of CNN.
- After researching, implementing the said algorithm to achieve high accuracy
- Test and evaluate the model in diverse real-world settings (e.g., different devices, users, or environments) to gather insights on its robustness and accuracy. Analyze performance metrics such as **latency** and **accuracy** in real-time applications and make optimizations accordingly.
- Experiment with more advanced semi-supervised or self-supervised techniques to leverage unlabeled data more effectively. For example, use **contrastive learning** or **self-training** with a dynamic threshold, which could adaptively adjust to provide the model with high-quality pseudo-labels.

Future Work

- Handle cursive handwriting via hybrid CNN+LSTM models.
- Apply FixMatch/MixMatch for stronger semi-supervised learning.
- Integrate n-gram or language models for word prediction.
- Expand to multilingual datasets (e.g., Devanagari, Arabic).
- Build an annotation feedback loop into GUI for retraining.

Results & Analysis

- Test accuracy of 98.32% achieved on the held-out EMNIST Letters test set, with a test loss of 0.085.
- Precision and recall both exceed 0.95 for the majority of the 26 classes.



Results & Analysis

- Confusion matrix analysis reveals common misclassifications, notably C vs G and M vs N.
- Learning curves converge by epoch 4, demonstrating effective use of dropout and early stopping.
- Total training time was approximately 15 minutes on a single GPU, confirming practical efficiency.
- ROC AUC exceeds 0.98 across all letter classes, indicating strong separability and robust performance.

Conclusion

- Our semi-supervised CNN achieved 98.32% accuracy on the EMNIST Letters test set, showing that high-confidence pseudo-labeling can boost performance without extensive manual labeling. Precision, recall, and F1-scores all surpassed 0.95 for nearly every class, with only a few predictable confusions (e.g., “C” vs. “G”).
- Adaptive thresholding, Gaussian blur, and min–max normalization sharpened stroke features, and the three-block CNN (32→64→128 filters) with dropout and early stopping balanced depth and generalization, yielding fast, stable convergence.
- By keeping the entire pipeline—from data ingestion through training and evaluation—within a single RStudio project, we delivered a fully reproducible, high-accuracy letter recognizer and laid the groundwork for future enhancements like cursive support and advanced semi/self-supervised techniques.

References

[1] M. Yadav and R. K. Purwar, "Integrating Wavelet Coefficients and CNN for Recognizing Handwritten Characters," 2018 2nd IEEE International Conference on Power Electronics, Intelligent Control and Energy Systems (ICPEICES), Delhi, India, 2018, pp. 1160-1164.

[2] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, “EMNIST: An Extension of MNIST to Handwritten Letters,” in *Proc. NIPS Workshop on Deep Learning*, Long Beach, CA, USA, Dec. 2017.

[3] L. van der Maaten and G. Hinton, “Visualizing Data using t-SNE,” *J. Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[4] A. Busch and H. Bunke, “Off-Line Cursive Handwriting Recognition: A Survey,” *ACM Comput. Surv.*, vol. 34, no. 2, pp. 145–176, June 2002.

THANK YOU