Handwriting Recognition using Backpropogation

Team Number 4

Team Members

01 Adithya C (CB.SC.U4AIE24005)

02 Architha Rajasekar (CB.SC.U4AIE24009)

03 Bhuvaneswaran S (CB.SC.U4AIE24012)

04 Harshith Reddy (CB.SC.U4AIE24018)

Introduction

In this project, we built a deep learning model to recognize handwritten English names. The model combines Convolutional Neural Networks (CNN) to extract features from images and Recurrent Neural Networks (RNN) to understand the sequence of letters. We used a special loss function called CTC loss, which helps the model learn without needing exact letter positions. The training was done using GPU for better performance, and the model showed descent accuracy when tested on new handwriting samples.

OBJECTIVES

1

To build a deep learning model that recognizes handwritten English names.

3

To design a CNN-RNN architecture using CTC loss for sequence prediction.

2

- **To** preprocess handwriting images for consistent input to the model.

4

To evaluate model performance on character and word level accuracy

DATASET

1. Training Dataset

Contains images of handwritten names.

Each image is labeled in the CSV with:

- FILENAME (e.g., 00001.jpg)
- IDENTITY (e.g., JEBASTIN)

2. Validation Dataset

Same structure as the training set.

Used to validate model performance during training.

DATASET

3. Test Dataset

•Used to evaluate model predictions.

•Labels may be present but are not used for training.

•**Initial Checks:**

  •It checks for **NaN values** in the 'IDENTITY'

  •Removes rows with 'IDENTITY' == 'UNREADABLE'.

•**Sizes:**

  •**Training data used:** train_size = 30000

  •**Validation data used:** valid_size = 3000

  •Test size is not explicitly set, but 6 samples are visualized in the final section.

MODEL IN DETAIL

Why CNN?

- **Convolutional Neural Networks (CNNs)** are used to extract spatial features from handwritten images — such as edges, curves, and patterns — making them ideal for capturing the visual structure of characters and strokes in text.

Why RNN?

- **Recurrent Neural Networks (RNNs)**, especially Bidirectional LSTMs, are excellent for sequence modeling. They help the model understand the **temporal order and dependencies** in handwriting, enabling it to interpret sequences of characters over time (left-to-right and right-to-left).

Why CTC Loss?

- **Connectionist Temporal Classification (CTC) Loss** is used because handwriting inputs and corresponding text labels can vary in length. CTC allows the model to **learn alignments automatically** between image sequences and text labels without needing character-level segmentation.

IMPLEMENTATION

1. **Importing** – Load all required Python libraries and TensorFlow/Keras modules
2. **Loading** – Read dataset CSV files and corresponding image files
3. **Visualizing** – Display sample images from the dataset to understand handwriting styles and detect anomalies.
4. **Cleaning** – Remove entries with missing or unreadable labels
5. **Preprocessing** – Resize, rotate, and normalize images to a fixed input shape suitable for the model.
6. **Reshaping** – Convert image and label arrays into the correct format required by the neural network.
7. **Encoding** – Transform character labels into numerical format for CTC training.
8. **Modeling** – Construct a CNN-RNN hybrid architecture with CTC loss
9. **Training** – Train the model using the CTC loss function with input images.
10. **Predicting** – Use the trained model to generate output sequences from input images.
11. **Decoding** – Convert model output into readable text using the CTC decoding method.
12. **Evaluating** – Measure character-wise and word-wise prediction accuracy to assess model performance.

METHODOLOGY

1. Importing Libraries

- All required libraries (like TensorFlow, NumPy, OpenCV, etc.) are imported to build, train, and evaluate the neural network model.

2. Loading the Dataset

- The dataset of handwritten word images and their labels (true text) is loaded into the program. Each image corresponds to a handwritten word.

3. Visualizing the Data

- Some sample images are displayed to understand how the data looks — this helps in checking the quality of the images.

4. Cleaning the Data

- Unwanted or badly formatted data is removed. This includes filtering out images with empty labels or those not matching the required format.

METHODOLOGY

5.Preprocessing:

- Resized or padded to **64×256 pixels**
- Rotated vertically
- Pixel values normalized to the range **[0, 1]**

6. Reshaping the Data

- The images are reshaped to a format that the model can understand

7. Encoding the Labels

- The actual text (labels) is converted into numbers using a character-to-index mapping. This allows the neural network to work with them.

8. Creating the Neural Network Model

- A Convolutional Neural Network (CNN) is created

9. Training the Model

- The model is trained over 60 epochs using the training data

METHODOLOGY

10. Predicting on Test Images

Once training is complete:

- Test images are passed through the model
- The model outputs predicted character sequences

11. Decoding the Output

- The predicted sequences (in numbers) are converted back to readable text using the index-to-character mapping.

MODEL ARCHITECTURE

1. Input Layer

- **Shape:** (256, 64, 1)
- Receives grayscale images of handwritten names.
- Each image is resized and rotated to maintain consistent input format.
- Acts as the entry point to the model.

2. Convolutional Layers (CNN)

These layers extract spatial features like edges, curves, and strokes from the image.

- **Conv2D Layer 1:**
  - 32 filters of size 3×3
  - Captures low-level features (edges, lines)
  - Followed by **Batch Normalization** and **ReLU** activation
  - **RELU : f(x)=max(0,x)**
  - **MaxPooling (2×2):** Reduces spatial dimensions and computation

MODEL ARCHITECTURE

Conv2D Layer 2:

- 64 filters of size 3×3
- Detects more complex patterns like corners and contours
- Followed by BatchNorm, ReLU, and **MaxPooling (2×2)**
- **Dropout (0.3):** Prevents overfitting by randomly disabling 30% of neurons during training

Conv2D Layer 3:

- 128 filters of size 3×3
- Extracts deeper and more abstract visual features
- Followed by BatchNorm, ReLU, and **MaxPooling (1×2)** to compress only one axis
- Another Dropout (0.3) applied

MODEL ARCHITECTURE

3. Reshape Layer

- Converts 3D output from CNN into a 2D sequence format suitable for RNNs
- New shape: **(64 time steps, 1024 features)**
- Each time step represents a vertical slice of the image, simulating how text is read

4. Dense Layer (Fully Connected)

- **Units:** 64
- Applies transformation across all features per time step
- Helps in condensing feature representation before passing to LSTM
- Uses ReLU activation

MODEL ARCHITECTURE

5. Recurrent Layers (BiLSTM)

Processes image as a sequence, learning patterns over time steps (like reading text from left to right).

Bidirectional LSTM 1:

- 256 units
- Processes input in both forward and backward directions
- Captures dependencies across time steps from both past and future

Bidirectional LSTM 2:

- 256 units
- Further refines temporal understanding
- Helps in identifying complete letter sequences and structure

MODEL ARCHITECTURE

6. Output Layer

Dense Layer:

- Number of units = total characters (A–Z, space, hyphen, apostrophe) + 1 for CTC blank
- Converts LSTM outputs into character probabilities at each time step

Softmax Activation:

- Converts raw scores into a probability distribution
- Indicates the likelihood of each character at every position

MODEL ARCHITECTURE

7. CTC Loss Layer (Connectionist Temporal Classification)

- Used because labels do not align directly with image pixels
- Computes loss by comparing predicted character sequences with actual labels
- Allows model to learn the correct order of characters without explicit alignment

Optimizer & Training Details

- **Optimizer:** Adam (learning rate = 0.0001)
- **Loss Function:** Custom CTC loss using Lambda layer
- **Epochs:** 60
- **Batch Size:** 128

ADVANTAGES AND DISADVANTAGES

ADVANTAGES

Automation of Manual Data Entry:

Helps in digitizing handwritten forms, cheques, medical prescriptions, and historical documents—saving time and reducing human error.

Streamlined Document Management:

Organizations can automatically index and search handwritten documents, making document retrieval faster and more efficient.

DISADVANTAGES

Struggles with Diverse Handwriting Styles:

Handwriting varies drastically from person to person—cursive, sloppy, or unconventional writing can cause incorrect predictions.

Limited Accuracy in Noisy Environments:

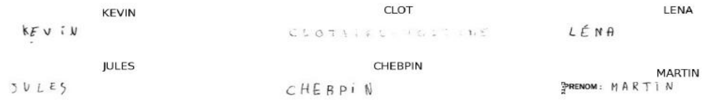
Low-quality scans, smudges, lighting issues, or background noise in images can significantly reduce the model's accuracy in real-world scenarios.

RESULTS

```
235/235 ━━━━━━━━━━━━━━━━━━━━ 421s 2s/step - loss: 1.2583 - val_loss: 2.0776
Epoch 58/60
235/235 ━━━━━━━━━━━━━━━━━━━━ 418s 2s/step - loss: 1.2629 - val_loss: 2.1134
Epoch 59/60
235/235 ━━━━━━━━━━━━━━━━━━━━ 357s 2s/step - loss: 1.2135 - val_loss: 2.1269
Epoch 60/60
235/235 ━━━━━━━━━━━━━━━━━━━━ 357s 2s/step - loss: 1.2064 - val_loss: 2.1228
94/94 ━━━━━━━━━━━━━━━━━━━━ 12s 118ms/step
Correct characters predicted : 90.79%
Correct words predicted      : 76.83%
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
```

OUTPUT

OUTPUT



1	FILENAME	IDENTITY
2	TEST_0001	KEVIN
3	TEST_0002	CLOTAIRE
4	TEST_0003	LENA
5	TEST_0004	JULES
6	TEST_0005	CHERPIN
7	TEST_0006	MARTIN

DICUSSION ON RESULTS

Accuracy Metrics

Correct Characters Predicted: **90.79%**

- This is a **very good character-level accuracy** for handwriting recognition.
- It means that out of all characters in the validation set, over 90% were recognized correctly.

- **Correct Words Predicted**: **76.83%**
  - This metric is stricter, as it considers a word correct **only if all characters** are predicted correctly.
  - A ~77% word accuracy is a **strong result** in sequence modeling, especially for handwritten text.

Prediction Speed

- The model predicts each test image in **35–45 milliseconds**, which is very fast.
- This means it can be used in **real-time handwriting recognition applications**.

FUTURE SCOPE

1.Support for Multilingual Handwriting:
Extend the system to recognize handwritten text in multiple languages or scripts (e.g., Hindi, Arabic), increasing usability across regions.

2.Integration with Real-Time Applications:
Deploy the model in real-time systems such as note-taking apps, digital forms, or smart classroom tools using mobile or web platforms.

3.Accuracy with Larger Datasets:
Train the model on more diverse and larger handwriting datasets to boost generalization and performance across varied writing styles.

THANK YOU!!☺