HANDWRITING DETECTION/RECOGNITION

Introduction:

Handwriting detection involves the development of algorithms and models capable of understanding and interpreting handwritten text, enabling computers to recognize and convert handwritten characters into digital text.

This capability has numerous practical applications across various industries, including document digitization, automatic form filling, postal services, and historical documents.

This report delves into a machine-learning project focused on handwriting detection.

Dataset:

The IAM dataset is a collection of handwritten text specifically designed to train and test models that deal with handwriting. It contains over 13,000 images of handwritten lines from hundreds of different people.

These lines are labeled at the word, line, and sentence level, making it useful for tasks like converting handwriting to digital text (Handwritten Text Recognition) or identifying who wrote a particular sample.

Methodology:

We've divided our methodology into the following steps for easy understanding:
Here's a detailed explanation of the methodology along with the functions used in each step:

1. Data Preparation:
   - Function: `get_paths_and_gts(partition_split_file)`
   - Explanation: This function parses a text file (`words.txt`) containing information about image paths and corresponding ground truth texts. It extracts image paths and ground truth texts, forming a list of pairs (`image_location`, `ground_truth_text`).

2. Image Preprocessing:
   - Functions:
     - `add_padding(img, old_w, old_h, new_w, new_h)`
     - `fix_size(img, target_w, target_h)`
     - `preprocess(path, img_w, img_h)`
   - Explanation:
     - `add_padding`: Adds padding to an image to match a specified size.
     - `fix_size`: Resizes an image to match a specified size while maintaining the aspect ratio. If necessary, padding is added to ensure the target size.
     - `preprocess`: Reads an image from the specified path, and applies resizing, padding, grayscale conversion, and normalization.

3. Model Architecture:
   - Functions:
     - Model construction using TensorFlow/Keras layers.
   - Explanation:

- The model architecture is defined using convolutional (`Conv2D`) and recurrent (`GRU`) layers in Keras.
- Convolutional layers capture spatial features from input images, while recurrent layers process these features to recognize sequences of characters.
- Batch normalization and activation functions (e.g., ReLU) are applied after convolutional layers for feature normalization and non-linearity.

4. Loss Function:
- Function: `ctc_lambda_func(args)`
- Explanation:
  - This function defines the CTC loss function used for training the model.
  - It computes the CTC loss based on the predicted outputs, ground truth labels, input sequence lengths, and label lengths.
  - CTC loss is suitable for sequence recognition tasks where the alignment between input and output sequences is not known.

5. Training:
- Function: `model.compile()`
- Explanation:
  - Compiles the model for training, specifying the loss function (`ctc_lambda_func`) and optimizer (e.g., Adam).
  - Training is performed using the `fit()` method, where training data is fed in batches, and the model parameters are updated to minimize the loss.

6. Evaluation:
- Functions:
  - `ModelCheckpoint`, `EarlyStopping`, `EpochTimeHistory` (custom callback)
- Explanation:
  - ModelCheckpoint saves the best model weights during training based on specified criteria (e.g., validation loss).
  - EarlyStopping stops training if the monitored metric (e.g., validation loss) stops improving.
  - EpochTimeHistory records the time taken for each epoch during training and validation.

7. Inference:
- Functions:
  - `predict()` method of the trained model
- Explanation:
  - After training, the model can be used for inference on new images by calling the `predict()` method.
  - Preprocessed images are fed into the model, and the output sequences of characters are decoded using the CTC decoding algorithm.

8. Evaluation Metrics:
- Functions:
  - Various metrics computed from the model's predictions (e.g., loss, accuracy, CER, WER).

- Explanation:
  - Model performance is evaluated based on metrics such as loss (both training and validation), accuracy, character error rate (CER), or word error rate (WER).
  - These metrics provide insights into the model's ability to recognize characters and words accurately.

Results:

On training for 10 epochs, the gathered result is as follows:

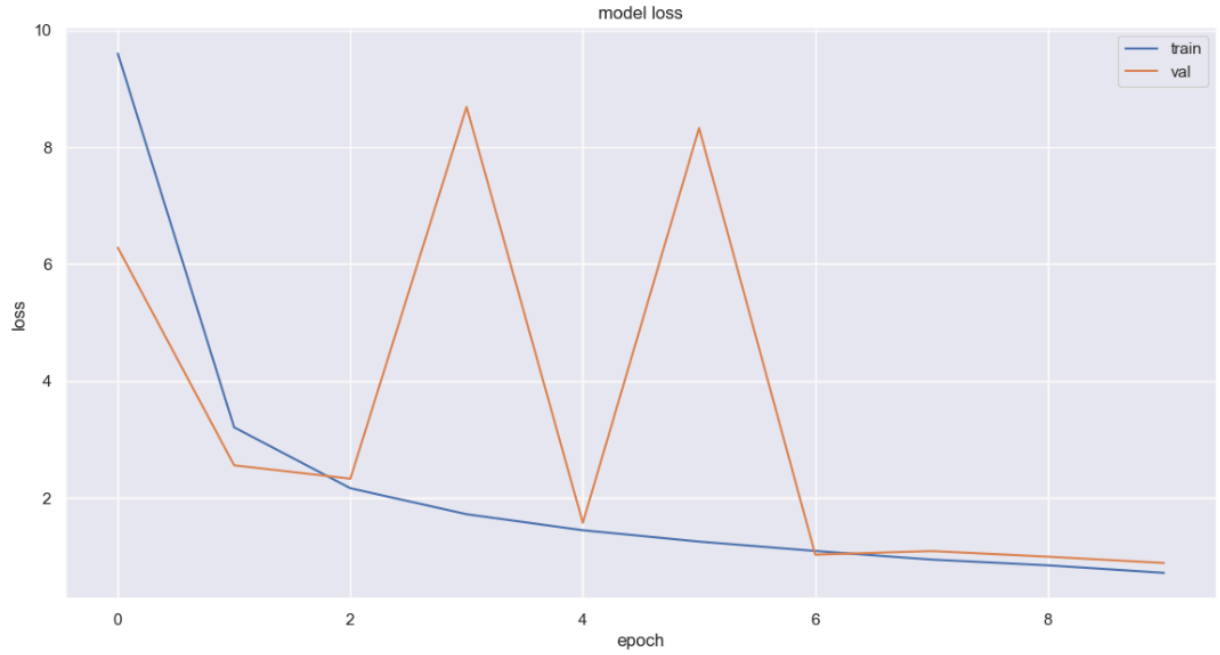

Fig. 1: Graph representing Model Loss

The predictions were as follows:

```
original_text =  certain
predicted text =  Ccertain

original_text =  biographies
predicted text =  Bbrograpties

original_text =  all
predicted text =  Oall

original_text =  It
predicted text =  It

original_text =  I
predicted text =  I
```

```
original_text =  show
predicted text =  wshow

original_text =  Kings
predicted text =  HKMings

original_text =  A
predicted text =  A

original_text =  the
predicted text =  Hthe

original_text =  and
predicted text =  Gand
```

Conclusion:

To summarise the results, in our opinion the model prediction, although not 100% accurate is pretty close to the actual handwritten text, we think that this can be improved by:

a. increasing the number of epochs during training which we could not do due to the time constraint
b. Hyperparameter tuning
c. Feedback loop
d. Using Ensemble Methods
e. L1/L2 Regularization

- Lopamudra Dalai (2105897)
- Shivansh Mani Tripathi (2105665)
- Ankit Choudhary (2105354)