University of Regina

Final Report: Handwriting Recognition System using Deep Learning

Archisha Bhattacharya

ENSE 480: Knowledge Base and Information System

Dr. Christine Chan

April 11, 2023

Table of Contents

1.0 Introduction

1.1 Background

Handwritten text recognition (HTR) is a system where computers can process and identify handwritten text to convert it to digital text. The process includes using image processing algorithms on a handwritten document to transcribe it into computer-readable text.

HTR is essential because it converts handwritten text into digital text, which computers can save, search, and process.

HTR systems can be applied in various industries and fields as it helps reduce manual labour and improve data accuracy and accessibility. The following is a list of different applications of HTR.

1. **Education**: HTR can digitize students' notes, making it easier to organize, analyze and search through their work.
2. **Healthcare**: HTR can transcribe prescriptions and doctors' notes into electronic records that can be stored in a patient's files, reducing medical errors.
3. **Banking**: HTR can be used to process cash electronically, cheques and other forms of physical currency, minimizing human error and improving the efficiency of banking operations
4. **Law enforcement**: HTR can be used to transcribe statements, reports, and other legal documents, improving the accuracy and speed of investigations.
5. **Historical Preservation**: HTR can create digital copies of important historical documents, thus preserving them and making them accessible to a broader audience.

1.2 Scope

The project aims to develop a system that can recognize handwritten sentences by developing an Artificial Intelligence (AI) model utilizing deep learning algorithms using a combination of Convolutional and Recurrent Neural Networks trained on standardized datasets.

Further, upon developing a successful model that can accurately recognize handwritten text, the model can be used in a front-end visual interface where users can upload images of their handwritten text, and the model will attempt to output the predicted text.

1.3 Challenges

According to Bortolozzi et al. (2005) HTR systems consist of various challenges, such as:

1. **Variability in handwriting style**: Handwriting can vary significantly between individuals, making it challenging to develop HTR models that can recognize text accurately across different writers.
2. **Overlapping characters:** Some handwriting styles involve overlapping characters, making it challenging to segment text accurately into individual characters.
3. **Multilingual recognition**: Recognizing handwriting in multiple languages and scripts is challenging due to the differences in character shapes and styles.
4. **Writer adaptation and personalization**: HTR models should be able to adapt to individual writers' handwriting styles to improve recognition accuracy.
5. **Document layout analysis:** HTR models should be able to analyze the document layout to identify text lines, text blocks, and other elements to improve recognition accuracy.

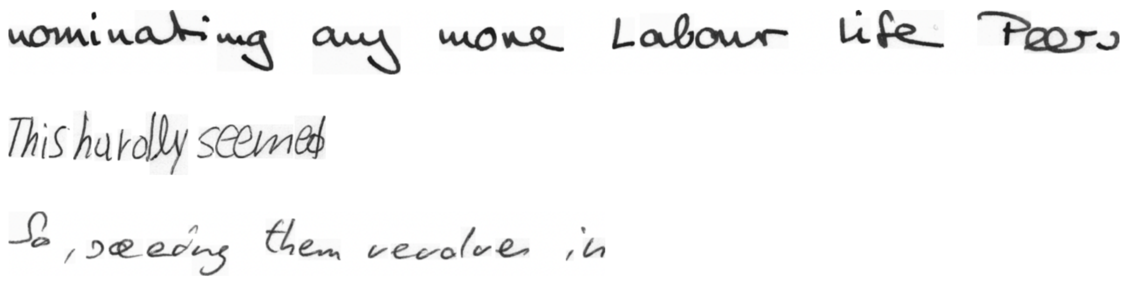
6. **Scalability**: HTR models should be scalable to efficiently handle large volumes of data, which is essential for applications such as postal address recognition and digitizing historical documents.

Need for large training sets: HTR models require large amounts of training data to learn to recognize different handwriting styles accurately. Creating such datasets can be time-consuming and expensive.

2.0 Knowledge and Data Representation

2.1 Dataset

The dataset chosen to train the model for the HTR system in this project is the IAM Dataset. The dataset consists of 16,752 images of handwritten text, scanned at a resolution of 300 dpi and saved as PNG images with 256 gray levels. Figure 1 below shows examples of handwritten text in the IAM dataset.



nominating any more Labour life Peers

This hardly seemed

So, seeing them revolve, in

Figure 1. Examples of handwritten sentences in the IAM dataset (IAM handwriting database)

According to M. Liwicki and H. Bunke (2005), using the IAM dataset to train a handwriting recognition model has several advantages:

1. **Size and Diversity**: The IAM dataset contains a large and diverse set of handwritten text samples, including cursive and printed text, making it suitable for training HTR models that can recognize a wide range of handwriting styles.

2. **Segmentation Information**: The dataset provides detailed segmentation information, such as text lines and word boundaries, which can be used to develop more accurate HTR models.
3. **Ground Truth Transcription**: The IAM dataset provides accurate ground truth transcription of the handwritten text, which can be used for training and evaluating HTR models.
4. **Benchmarking**: The IAM dataset is widely used for evaluating the performance of HTR models, making it easier to compare the performance of different models.
5. **Availability**: The IAM dataset is freely available online, which makes it accessible to researchers and practitioners around the world.

2.2 Processing and Preparing Dataset

The dataset images should be prepared and processed prior to training the model. Firstly, the images are extracted and converted to a suitable format to be used by the system. Secondly, they are resized to the pre-defined dimensions to ensure consistency, padding is added to images and indexing labels are added. Lastly, the dataset is split into training and validating sets to train and validate the model simultaneously.

Also, the images are processed wherein the brightness, dilation and sharpness are adjusted for better readability. Though rotating the images was considered to improve the prediction results of the model, it was later dismissed as rotating long sentences made the images unreadable.

3.0 Approach, Techniques and Algorithms

3.1 Approach

The approach used in developing the HTR model was a deep learning algorithm using a combination of convolutional neural network (CNN) and recurrent neural network (RNN) layers. Since handwriting styles varies vastly from person to person and can change over time, the model was trained using Connectionist Temporal Classification (CTC) loss, which is well-suited for problems where the alignment between input and output sequences is unknown.

For this project, deep learning was preferred to machine learning algorithms because machine learning relies on manually extracted features, whereas deep learning automatically extracts features and performs classification using neural networks such as CNN (Gautam, 2021). The following figure shows the performance of deep learning algorithms compared to other learning algorithms.

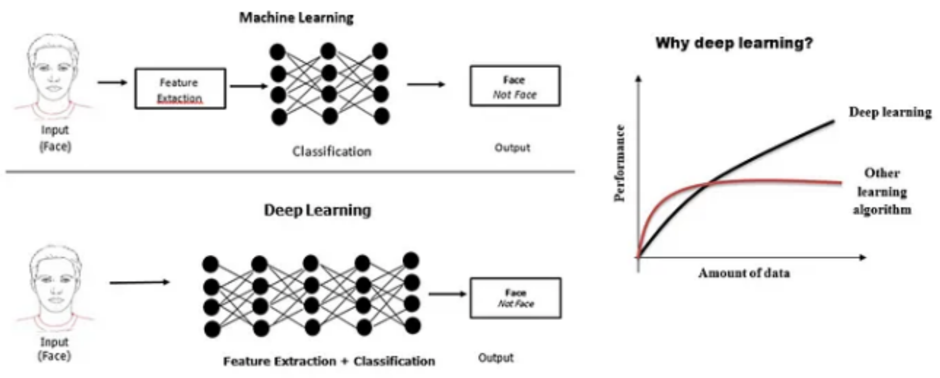


Figure 2. Performance of deep learning algorithm in comparison with other algorithms (Gautam, 2021)

The above figure shows that the deep learning algorithm's performance is higher than other learning algorithms when the amount of data increases.

3.2 Techniques and Algorithm

The model architecture was defined using Keras and TensorFlow. The final model architecture prepared for the project can be divided into CNN, RNN, and CTC Loss layers.

The CNN layers are responsible for extracting features from the images. The RNN layers used a Bidirectional Long Short-Term Memory (BiLSTM) network to perform sequence labelling by processing the sequence of features and predicting the characters in the provided text. BiLSTM was chosen over Long Short-Term Memory (LSTM) because BiLSTM combines LSTM layers from both directions, which improves its performance and accuracy (Augustyniak et al., 2019). Finally, the model is compiled using the CTC Loss function to attempt to eliminate alignment differences between the input and output sequences. The CTC loss function is based on a "blank" symbol, a special symbol that can be inserted between characters in the output sequence. The blank symbol is used to align the input and output sequences and allows the model to predict characters in the correct order, even if there are skips or repetitions in the input sequence.

The front-end application was deployed using Flask and HTML and Bootstrap (CSS library) was used to display the front-end contents.

4.0 A Structural Diagram and Explanation

4.1 Model Definition

The model prepared and trained for the HTR system consists of multiple layers.

The input layer takes in images with pre-defined dimensions, where the Lambda layer is used for image normalization to normalize the pixel values from 0-255 to 0-1.

Next, the CNN layers are added, which consist of a series of residual blocks used to extract the features from the images. Each residual block consists of multiple convolutional layers, batch normalizations, activation (leaky ReLU) and dropout layers. The residual blocks use skip connections and varying numbers of channels (32, 64, 128) and strides (1, 2) for different levels of feature extraction. The output of the last residual block is then reshaped and passed on to the RNN layers.

The RNN layers comprise two BiLSTM with 256 and 64 hidden units, respectively. Dropout is applied after each LSTM layer to prevent overfitting. A dense layer with a softmax activation is used, which outputs probabilities for each character in the vocabulary, including a unique character for CTC blank token. The ground truth text is the predicted text.

Figure 3 below shows the structural diagram of the model architecture used for the HTR system.

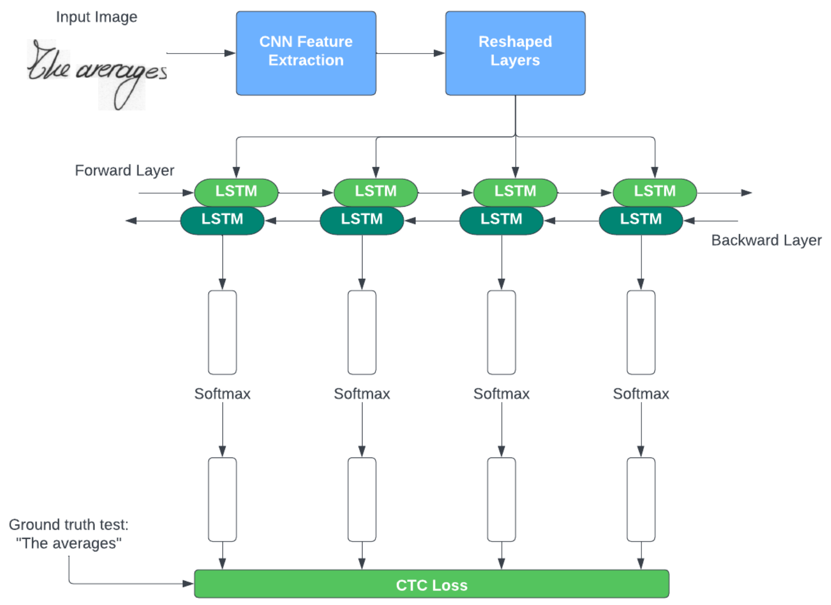


Figure 3. Structural Diagram of the model architecture

4.2 Front-end Application Deployment

The front-end application consists of a form allowing users to upload their handwritten text images. The uploaded image will be saved locally on the device, and the saved trained model will then be used on the uploaded image, and the predicted text will be shown to the user on the application. Figure 4 below shows the structural diagram of the front-end application for the HTR system.

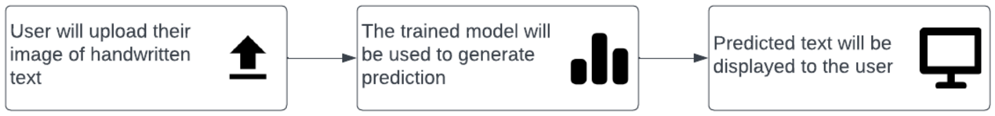

User will upload their image of handwritten text
The trained model will be used to generate prediction
Predicted text will be displayed to the user

Figure 4. Structural Diagram of the front-end application

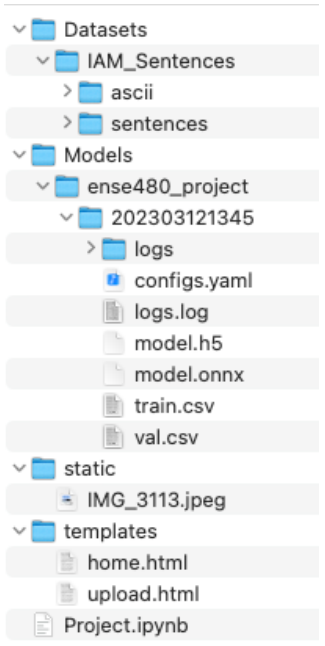
5.0 "Modules" Description and Sample Listing

5.1 Environment

The project consists of a single Jupyter Notebook which consists of all the modules. The recommended environment to run the Jupyter Notebook is on a Python virtual environment hosted on Miniconda. The system prerequisites required before running the modules are Python 3, TensorFlow (v2.11.0) and mltu (v1.0.0).

5.2 File Structure

Figure 5 below shows the file structure that users should follow in order to run this project.



```
Datasets
    IAM_Sentences
        ascii
        sentences
Models
    ense480_project
        202303121345
            logs
            configs.yaml
            logs.log
            model.h5
            model.onnx
            train.csv
            val.csv
static
    IMG_3113.jpeg
templates
    home.html
    upload.html
Project.ipynb
```

Figure 5. Project file structure

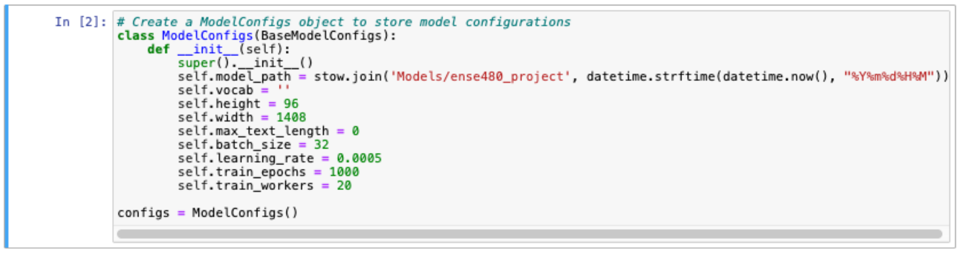
In the above picture, it can be seen that there five primary objects in the folder:

- Folder 'Datasets' - It consists of the dataset images as well as the summarized meta information in ascii format.
- Folder 'Models' - In this folder, the model is trained and saved, training logs are stored here as well as training and validation datasets are saved as CSV files.
- Folder 'static' - Images of handwritten text uploaded by users on the front-end application are saved in this folder.
- Folder 'templates' - This folder consists of the HTML files which are used to render the front-end components on a web browser.

- 'Project.ipynb' - Jupyter Notebook that can be run in the environment specified in section 5.1. It consists of different modules to extract and prepare the dataset, define and train the model, perform validation on the model and deploy the front-end application.

5.3 Sample Listing

5.3.1 Model Configuration



```
In [2]: # Create a ModelConfigs object to store model configurations
        class ModelConfigs(BaseModelConfigs):
            def __init__(self):
                super().__init__()
                self.model_path = stow.join('Models/ense480_project', datetime.strftime(datetime.now(), "%Y%m%d%H%M"))
                self.vocab = ''
                self.height = 96
                self.width = 1408
                self.max_text_length = 0
                self.batch_size = 32
                self.learning_rate = 0.0005
                self.train_epochs = 1000
                self.train_workers = 20

        configs = ModelConfigs()
```

Figure 6. Code the snippet for the model configuration definition.

Function Name: ModelConfigs

Function Description: The ModelConfigs class is used to store configurations related to the model. It stores various settings and parameters required for the model training and evaluation process.

Function Attributes:

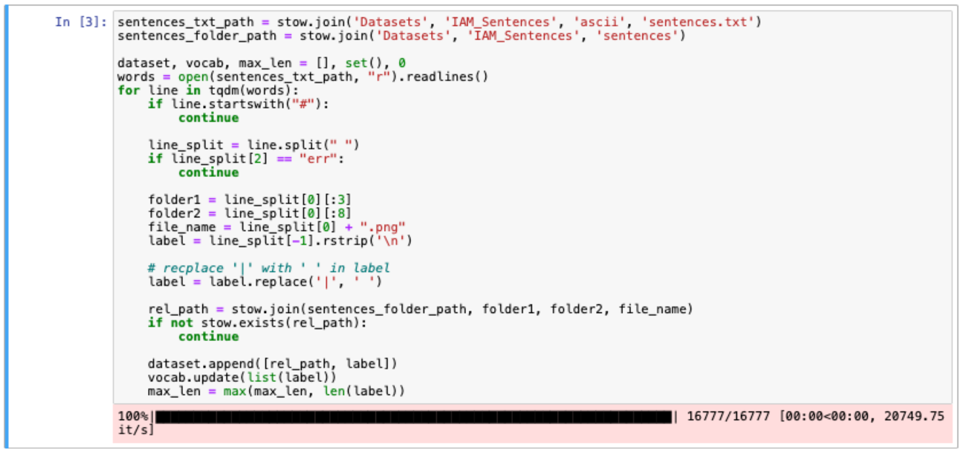
- model_path (str): The path where the model files will be saved.
- vocab (str): The vocabulary for the model.
- height (int): The height of the input images for the model.
- width (int): The width of the input images for the model.
- max_text_length (int): The maximum length of text allowed for the model.
- batch_size (int): The batch size to be used for training.
- learning_rate (float): The learning rate to be used for model optimization.
- train_epochs (int): The number of epochs for training the model.

- train_workers (int): The number of workers to be used for loading the training data.

Function Input: None.

Function Output: An instance of ModelConfigs with the attributes initialized to their default values.

5.3.2 Preparing Dataset



```
In [3]: sentences_txt_path = stow.join('Datasets', 'IAM_Sentences', 'ascii', 'sentences.txt')
        sentences_folder_path = stow.join('Datasets', 'IAM_Sentences', 'sentences')

        dataset, vocab, max_len = [], set(), 0
        words = open(sentences_txt_path, "r").readlines()
        for line in tqdm(words):
            if line.startswith("#"):
                continue

            line_split = line.split(" ")
            if line_split[2] == "err":
                continue

            folder1 = line_split[0][:3]
            folder2 = line_split[0][:8]
            file_name = line_split[0] + ".png"
            label = line_split[-1].rstrip('\n')

            # recplace '|' with ' ' in label
            label = label.replace('|', ' ')

            rel_path = stow.join(sentences_folder_path, folder1, folder2, file_name)
            if not stow.exists(rel_path):
                continue

            dataset.append([rel_path, label])
            vocab.update(list(label))
            max_len = max(max_len, len(label))

        100%|████████████████████████████████████████████████████| 16777/16777 [00:00<00:00, 20749.75
        it/s]
```

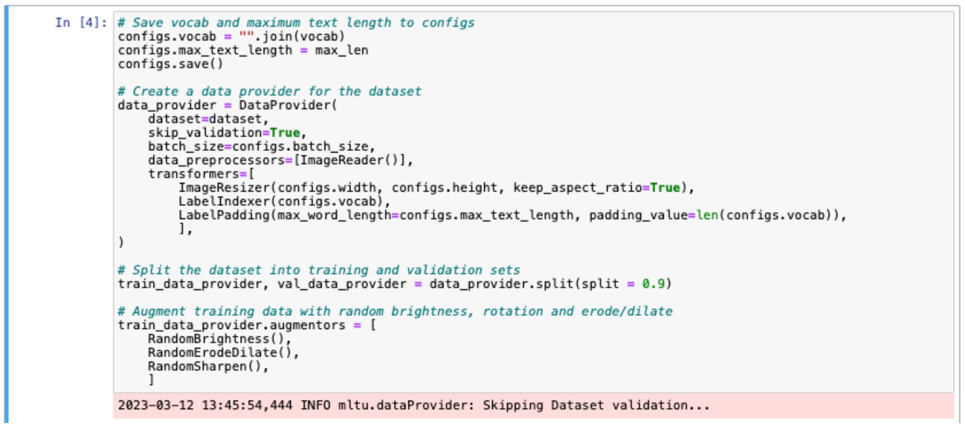
Figure 7. Code snippet for the extracting dataset.

Function Description: The above script processes the dataset of sentences and their corresponding images from the IAM Handwriting Database. It reads the sentences from a text file, filters out invalid entries, and stores the relative paths of the images along with their labels (sentences).

Variables:

- sentences_txt_path (str): Path to the text file containing sentence information.
- sentences_folder_path (str): Path to the folder containing sentence images.
- dataset (list): List of tuples containing relative image paths and labels.
- vocab (set): Set containing all unique characters found in the labels.

- max_len (int): Length of the longest label found in the dataset.
- words (list): List of lines from the sentences.txt file.

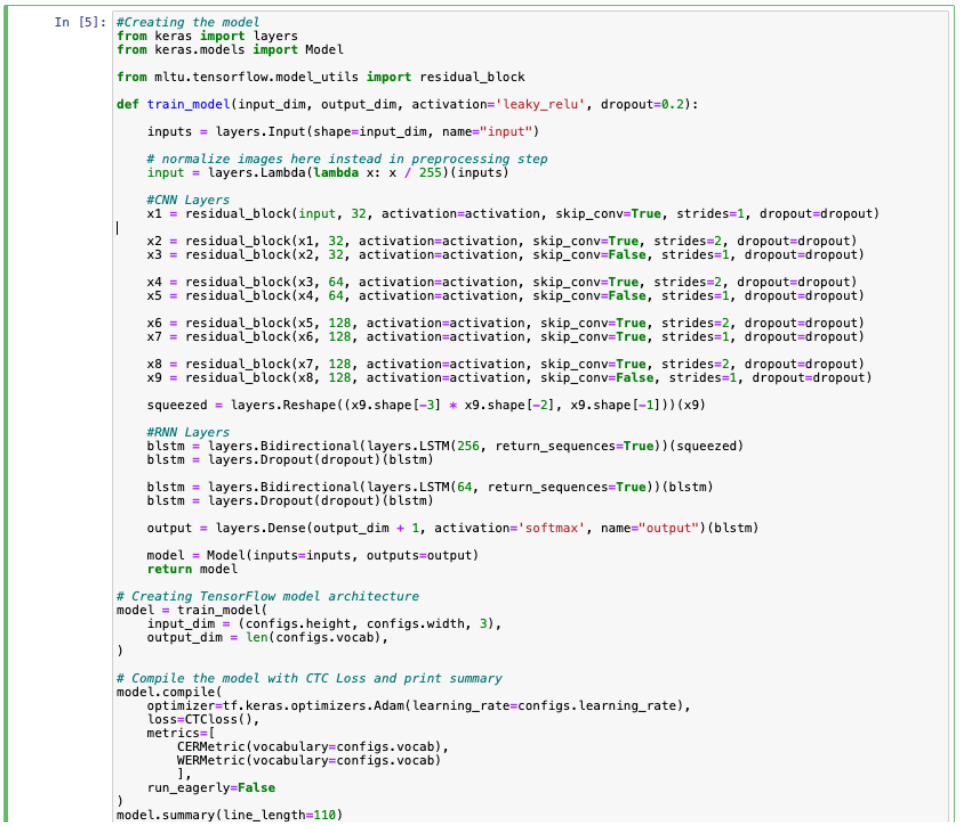


```
In [4]: # Save vocab and maximum text length to configs
        configs.vocab = "".join(vocab)
        configs.max_text_length = max_len
        configs.save()

        # Create a data provider for the dataset
        data_provider = DataProvider(
            dataset=dataset,
            skip_validation=True,
            batch_size=configs.batch_size,
            data_preprocessors=[ImageReader()],
            transformers=[
                ImageResizer(configs.width, configs.height, keep_aspect_ratio=True),
                LabelIndexer(configs.vocab),
                LabelPadding(max_word_length=configs.max_text_length, padding_value=len(configs.vocab)),
                ],
        )

        # Split the dataset into training and validation sets
        train_data_provider, val_data_provider = data_provider.split(split = 0.9)

        # Augment training data with random brightness, rotation and erode/dilate
        train_data_provider.augmentors = [
            RandomBrightness(),
            RandomErodeDilate(),
            RandomSharpen(),
            ]

        2023-03-12 13:45:54,444 INFO mltu.dataProvider: Skipping Dataset validation...
```

Figure 7. Code snippet for processing the dataset

Function Description: The above script saves the vocabulary and maximum text length to the ModelConfigs object, creates a DataProvider object to handle the dataset, and splits the dataset into training and validation sets. It also applies data augmentation to the training data.

5.3.3 Defining and Training Model



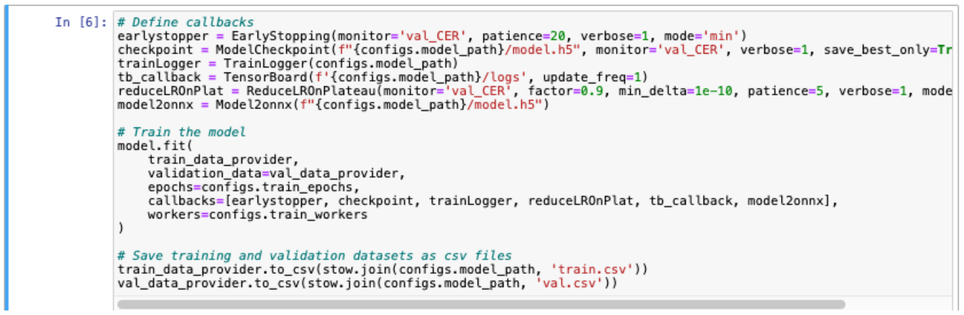
```
In [5]: #Creating the model
        from keras import layers
        from keras.models import Model

        from mltu.tensorflow.model_utils import residual_block

        def train_model(input_dim, output_dim, activation='leaky_relu', dropout=0.2):

            inputs = layers.Input(shape=input_dim, name="input")

            # normalize images here instead in preprocessing step
            input = layers.Lambda(lambda x: x / 255)(inputs)

            #CNN Layers
            x1 = residual_block(input, 32, activation=activation, skip_conv=True, strides=1, dropout=dropout)

            x2 = residual_block(x1, 32, activation=activation, skip_conv=True, strides=2, dropout=dropout)
            x3 = residual_block(x2, 32, activation=activation, skip_conv=False, strides=1, dropout=dropout)

            x4 = residual_block(x3, 64, activation=activation, skip_conv=True, strides=2, dropout=dropout)
            x5 = residual_block(x4, 64, activation=activation, skip_conv=False, strides=1, dropout=dropout)

            x6 = residual_block(x5, 128, activation=activation, skip_conv=True, strides=2, dropout=dropout)
            x7 = residual_block(x6, 128, activation=activation, skip_conv=True, strides=1, dropout=dropout)

            x8 = residual_block(x7, 128, activation=activation, skip_conv=True, strides=2, dropout=dropout)
            x9 = residual_block(x8, 128, activation=activation, skip_conv=False, strides=1, dropout=dropout)

            squeezed = layers.Reshape((x9.shape[-3] * x9.shape[-2], x9.shape[-1]))(x9)

            #RNN Layers
            blstm = layers.Bidirectional(layers.LSTM(256, return_sequences=True))(squeezed)
            blstm = layers.Dropout(dropout)(blstm)

            blstm = layers.Bidirectional(layers.LSTM(64, return_sequences=True))(blstm)
            blstm = layers.Dropout(dropout)(blstm)

            output = layers.Dense(output_dim + 1, activation='softmax', name="output")(blstm)

            model = Model(inputs=inputs, outputs=output)
            return model

        # Creating TensorFlow model architecture
        model = train_model(
            input_dim = (configs.height, configs.width, 3),
            output_dim = len(configs.vocab),
        )

        # Compile the model with CTC Loss and print summary
        model.compile(
            optimizer=tf.keras.optimizers.Adam(learning_rate=configs.learning_rate),
            loss=CTCloss(),
            metrics=[
                CERMetric(vocabulary=configs.vocab),
                WERMetric(vocabulary=configs.vocab)
                ],
            run_eagerly=False
        )
        model.summary(line_length=110)
```

Figure 8. Code snippet for model architecture.

Function Description: The above function creates and returns a deep learning model for handwriting recognition.

Function Parameters:

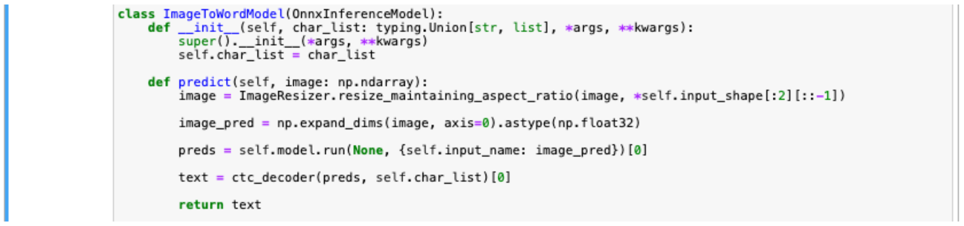
- input_dim (tuple): Input dimensions (height, width, channels) of the input images.
- output_dim (int): The number of unique characters in the dataset.
- activation (str, optional): The activation function to use in the residual blocks. Default is 'leaky_relu'.
- dropout (float, optional): The dropout rate to use in the dropout layers. Default is 0.2.



```
In [6]: # Define callbacks
        earlystopper = EarlyStopping(monitor='val_CER', patience=20, verbose=1, mode='min')
        checkpoint = ModelCheckpoint(f"{configs.model_path}/model.h5", monitor='val_CER', verbose=1, save_best_only=Tr
        trainLogger = TrainLogger(configs.model_path)
        tb_callback = TensorBoard(f'{configs.model_path}/logs', update_freq=1)
        reduceLROnPlat = ReduceLROnPlateau(monitor='val_CER', factor=0.9, min_delta=1e-10, patience=5, verbose=1, mode
        model2onnx = Model2onnx(f"{configs.model_path}/model.h5")

        # Train the model
        model.fit(
            train_data_provider,
            validation_data=val_data_provider,
            epochs=configs.train_epochs,
            callbacks=[earlystopper, checkpoint, trainLogger, reduceLROnPlat, tb_callback, model2onnx],
            workers=configs.train_workers
        )

        # Save training and validation datasets as csv files
        train_data_provider.to_csv(stow.join(configs.model_path, 'train.csv'))
        val_data_provider.to_csv(stow.join(configs.model_path, 'val.csv'))
```

Figure 9. Code snippet for training model

5.3.4 Validating Model



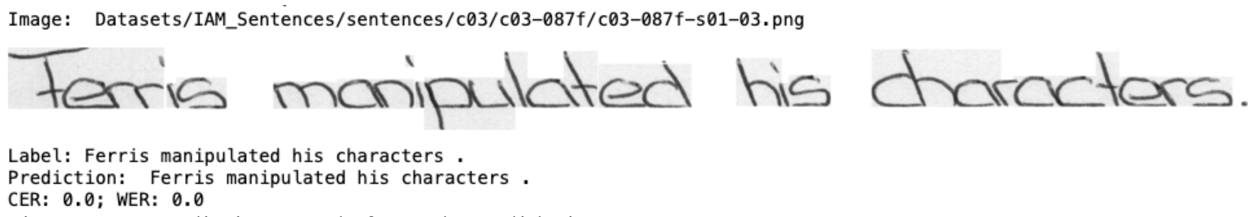
```
class ImageToWordModel(OnnxInferenceModel):
    def __init__(self, char_list: typing.Union[str, list], *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.char_list = char_list

    def predict(self, image: np.ndarray):
        image = ImageResizer.resize_maintaining_aspect_ratio(image, *self.input_shape[:2][::-1])

        image_pred = np.expand_dims(image, axis=0).astype(np.float32)

        preds = self.model.run(None, {self.input_name: image_pred})[0]

        text = ctc_decoder(preds, self.char_list)[0]

        return text
```

Figure 10. Code snippet for converting image to text using the trained model

Function Description: The above function defines a class for handwriting recognition that inherits from the OnnxInferenceModel class. It predicts the text in a given input image using a trained model.

6.0 Sample Sessions

Figure 11 below shows the prediction results when the model is run on one of the images from the validation set.

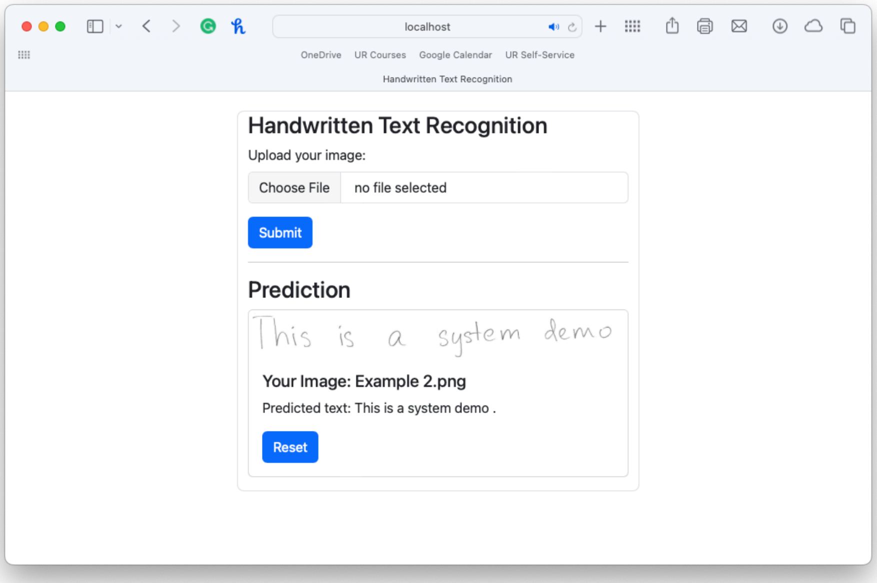


```
Image:  Datasets/IAM_Sentences/sentences/c03/c03-087f/c03-087f-s01-03.png
```

```
Label: Ferris manipulated his characters .
Prediction:  Ferris manipulated his characters .
CER: 0.0; WER: 0.0
```

Figure 11. Prediction result from the validation set.

Figure 12 below shows prediction result on the front-end application when the user uploads their own image to the system.



Handwritten Text Recognition

Upload your image:

Choose File no file selected

Submit

Prediction

This is a system demo

Your Image: Example 2.png

Predicted text: This is a system demo .

Reset

Figure 12. Prediction result on user uploaded image on the front-end application

7.0 Discussion

Using a combination of neural networks helped us achieve a very accurate model that can be used to predict handwritten text. Through developing this model, the system was able to overcome the challenges stated in section 1.3.

While evaluating the model, the model is then tested using the following parameters:

- **Word Error Rate (WER)** - percentage of words incorrectly guessed in a given image.
- **Character Error Rate (CER)** - percentage of characters incorrectly guessed in a given image.

The graphs for the above metrics were generated using TensorBoard. Figure 13 below depicts the graph for WER.

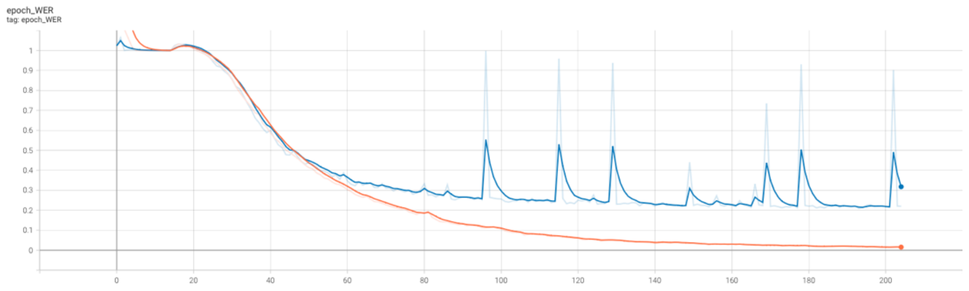


Figure 13. WER performance of the trained model

Figure 14 below depicts the graph for CER.

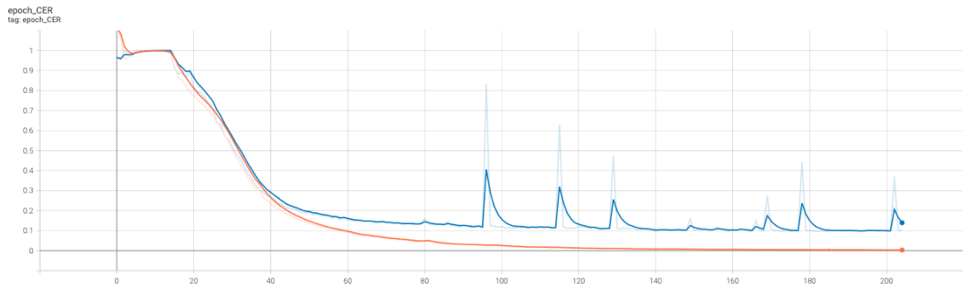
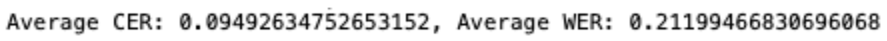


Figure 14. CER performance of the trained model

In figure 13 and 14, the orange line represents the training set while blue represents the validation set. While the WER and CER shows a steady decrease for the training set, there are some spikes in the validation sets, indicating that the model can be improved to produce more accurate results.

While performing validation on the model, the average CER and WER is depicted in the figure below.


```
Average CER: 0.09492634752653152, Average WER: 0.21199466830696068
```
Figure 15. Average CER and WER on validation set

However, the model is reliable enough to be used in front-end applications where users can upload their own images. The model has been able to perform prediction on images out of dataset. However, the resolution and sizing of the image has to be optimal in order for the model to extract its features and run predictions.

8.0 Conclusion

Implementing a combinational neural network yielded a highly accurate model which is able to perform predictions on images not only from the dataset it was trained on but also from external sources.

This project was a good introduction into Artificial Intelligence as well as developing deep learning algorithms to achieve the scope and overcome the challenges stated in section 1.0.

9.0 Future Work

Further improvements that can be implemented on the system include:

- Expanding the dataset to include other datasets such as EMNIST to improve the accuracy of the model.
- Including other variety of scripts in the dataset to train the model to predict other languages.
- Training the model to detect large paragraph or entire documents at once.

10.0 References

Bortolozzi, Flavio, et al. "Recent Advances in Handwriting Recognition." ResearchGate, Jan. 2005, https://www.researchgate.net/profile/Flavio-Bortolozzi/publication/266161055_Recent_Advances_in_Handwriting_Recognition/links/552419ad0cf2caf11bfcbfd9/Recent-Advances-in-Handwriting-Recognition.pdf.

Gautam, Sushant. "How to Make Real-Time Handwritten Text Recognition with Augmentation and Deep Learning." Medium, The Startup, 20 Dec. 2021, https://medium.com/@susant/learn-and-use-handwritten-line-text-recognition-using-deep-learning-with-tensorflow-b661434b5e3b.

"IAM Handwriting Database." Research Group on Computer Vision and Artificial Intelligence - Computer Vision and Artificial Intelligence, https://fki.tic.heia-fr.ch/databases/iam-handwriting-database.

Liwicki, M., and H. Bunke. "IAM-Ondb - an on-Line English Sentence Database Acquired from Handwritten Text on a Whiteboard." Eighth International Conference on Document Analysis and Recognition (ICDAR'05), 2005, https://doi.org/10.1109/icdar.2005.132.

Augustyniak, Łukasz, et al. "Aspect Detection Using Word and Char Embeddings with (Bi) LSTM and CRF." ResearchGate, June 2019, https://www.researchgate.net/publication/335072274_Aspect_Detection_using_Word_and_Char_Embeddings_with_Bi_LSTM_and_CRF.

"Miniconda." Miniconda - Conda Documentation, https://docs.conda.io/en/latest/miniconda.html.

"Tensorflow." TensorFlow, https://www.tensorflow.org/.

"Welcome to Flask." Welcome to Flask - Flask Documentation (2.2.x), https://flask.palletsprojects.com/en/2.2.x/.