FINAL REPORT PART A: PERCEPTOR

Aman Patra
Student No. 1007301420
aman.patra@mail.utoronto.ca

Gitansh Kothari
Student No. 1006686819
gitansh.kothari@mail.utoronto.ca

Hitarth Desai
Student No. 1007376796
hitarth.desai@mail.utoronto.ca

Lalith Vaishnav Elangovan
Student No. 1006916884
lalith.elangovan@mail.utoronto.ca

ABSTRACT

This project aims to discern a handwritten word and convert it to digital text. We begin by describing the problem we are fixing and our scope for the project. Next, we discuss the background & related works regarding our project. Following that, we describe the model and the baseline model, as well as the comparison between the two. Finally, we discuss our results and evalute the final model. —-Total Pages: 9

1 INTRODUCTION

The digital era is on the rise and with it we slowly leave behind the world of handwritten text. Although many are getting on this trend, there remain a large number of us writing with pens on paper. Today, there are still people transitioning into the digital advent, businesses maintaining physical records and even forms needing to be filled out by hand. Although we can't change the basis of handwritten text, we can address the need for human involvement in converting it to digital text. By doing so, we can enhance the security, efficiency and accessibility of handwritten data with the help of automation. The goal of this project is to support this technological advent by creating a Deep Learning model that reads and deciphers handwritten text (Figure 1).

'Handwritten text' is quite vague, and as such we limit our scope to **deciphering a single handwritten word**. The project can be scaled in the future by incorporating a model which segments words from a handwritten document. Moving a step further, to preserve the "handwritten feel" of the document, we could utilize Generative Adversarial Networks to rewrite the same document in a more legible and standard handwritten font.

In this project, we approach the problem through Deep Learning since non-deep learning models such as Hidden Markov Models perform weakly due to manual feature extraction and learning capability limitations. In fact, when working with different handwriting styles and languages, regular Machine Learning models have poor accuracy and fail to scale.

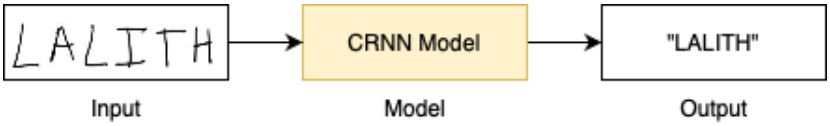

Figure 1: *Overview of the Project*

2 BACKGROUND & RELATED WORKS

The aim of this project is to discern a handwritten word from a given input image. Multiple machine learning models and software applications attempt to do the same. We explore a few below.

2.1 HMM: HIDDEN MARKOV MODEL

The first prominent machine learning model to recognize characters was the Hidden Markov Model. Their approach was to define each character as a *state* that provides context while identifying the rest of the characters in the word. While this model proved to perform better compared to its non-machine learning counterparts, it had its own limitations. These include limitations caused by manual feature extraction, biases given to the current state and the inability to scale due to the high requirement of prior knowledge (Cady, 2022) (Balci et al.).

2.2 MDLSTM: MULTI-DIMENSIONAL LONG-SHORT TERM MEMORY

Long-Short Term Memory models (LSTMs) are traditionally used with 1-Dimensional data. Since the input is an image of a word, a multi-dimensional LSTM architecture was proposed and developed. This multi-dimensional LSTM now captures states from every dimension of the input data. For a 2-dimensional image, these dimensions are the height and width. Although this approach encapsulates both the classification and segmentation within a single model, it fails to take into account the sequential order of the characters in the word (Nan, 2022) (Balci et al.).

2.3 MODERN SOFTWARE SOLUTIONS

There are many Big Data companies exploring the domain of handwritten text recognition. Google Lens, Apple Notes and Microsoft OneNote are some well known software applications that produce digital text equivalents from images of handwritten text.

3 THE MODEL

3.1 DATASET AND DATA PROCESSING

We decided to use the Kaggle IAM Handwriting Word Database [Vidyadharan] for our project. We chose this dataset since it contains a large number of examples of commonly used words and is easy to import on Google Colab using the Kaggle API. Other datasets that we found either had a lack of labels, or the images provided were blurry, making it difficult even for a human to read. Some datasets required us to upload huge amounts of data on Google Drive which is an extremely slow process. Hence, the Kaggle IAM Handwriting Word Database best fits our project.

The dataset provided by Kaggle API returns a zip file containing the main folder with all the images divided into multiple sub-folders, and a text file containing information about these images. While the original documentation claims that there are 115,320 we could only access 44,563 words using the text file in the latest version.

1. We first parse the text file to obtain all the images' path and their associated ground truth labels.
2. We filter out the data based on the following three conditions:
   (a) The word in the image was correctly segmented, given by status = "ok"
   (b) The label only contains alphabets (a-z, A-Z)
   (c) the length of the label is between 5 and 9 inclusive.

   If the given data satisfies these three conditions, we store its path and label as tuples inside a list. This filters the dataset from 44,563 examples to 24,537
3. Then, we convert each image to grayscale, and resize them to (128, 32, 1). We encode the labels as indices, where 0 represents unknown characters and space, 1-26 represent uppercase and 26-52 represents lowercase letters. We also pad the labels such that it has a length of 9.
4. We then split the dataset into training, validation and testing using a 80-10-10 split.

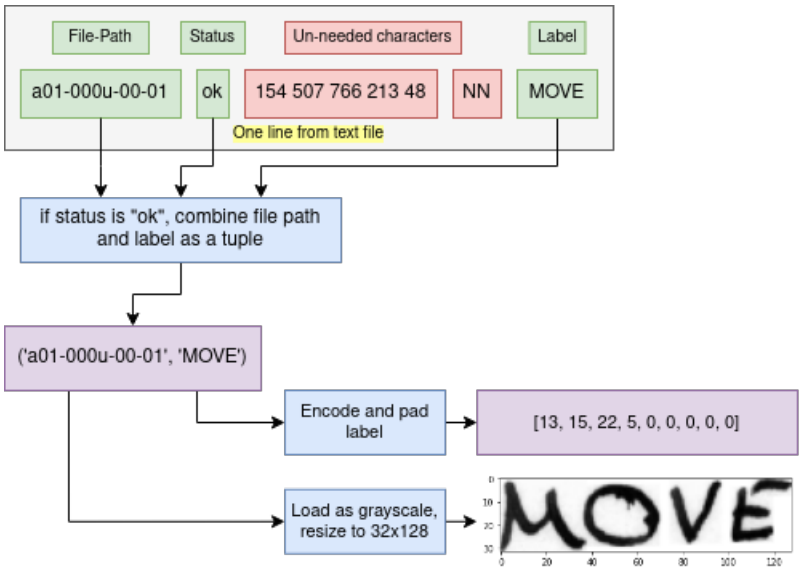

Figure 2: *Data Pre-Processing Flowchart*

The above pre-processing step is performed on a total of 22,082 training, 1,226 validation and 1,228 testing images. Figure 2 shows an example of the original image from the dataset.

3.2 Primary Model Architecture

All the dimensions of an image are given in the form of (height, width, channels). As described initially in the project proposal, our primary model architecture can be split into three major sections. These are the Convolutional, Recurrent and Transcription blocks.

Th Convolutional block takes a batch of grey-scale images of size (32, 128, 1) as input and returns convolutional feature maps of size (1, 31, 512). The output consists of 512 output features, each of which have a condensed height of 1 and width of 31. This shape is ideal for our next block which uses LSTMs to predict the characters in sequential order. This block consists of 7 Conv2d layers, each of which uses a ReLU activation function. Alternative convolutional layers are followed by appropriate MaxPooling layers. After the 5th and 6th Conv2D layers, we also add Batch normalization layers.

The Recurrent block takes the 512 convolutional feature maps as input and returns per-frame predictions of the input word. This block consists of 2 Bidirectional LSTM layers each with 128 hidden features, and 0.2 dropouts rates (which was decided after hyperparameter tuning). This returns 256 features which are then passed into a Fully Connected layer which maps these 256 features to 54 classes and computes the Log-Softmax of these values.

In the Transcription block we use the Log-Softmax values over the 54 classes for 31 values (the output width of the convoutional block). Hence this takes a tensor of size (256, 31, 54) as input. We find the index of the highest probability for each character for all the words, and pass this into the CTC Loss criterion to compute the loss and backpropagate. We then decode the predicted word and the ground truth label and calculate the similarity between the two using a Sequence Matcher algorithm. A detailed description of the primary model architecture is given below in Figure 3.

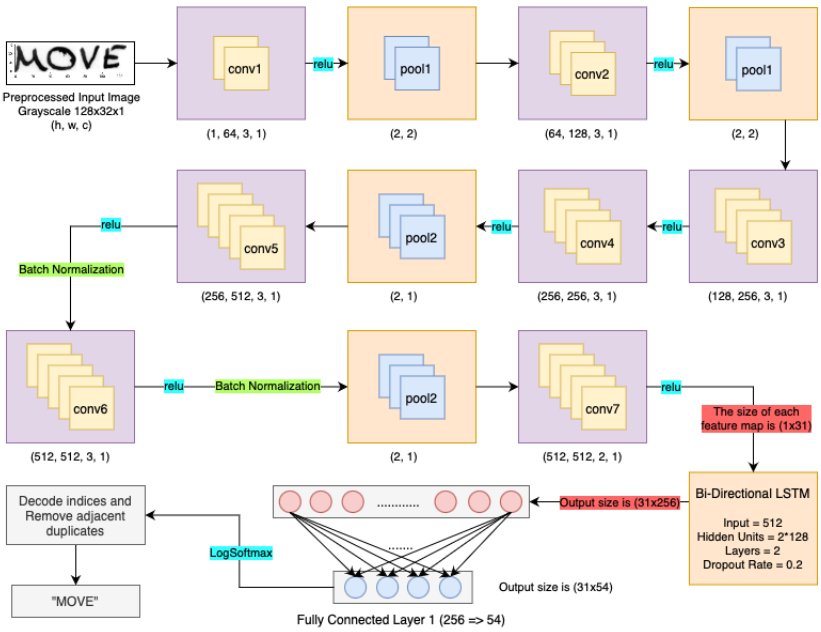

Figure 3: *Summary of the Architecture of the Main Model*

3.3 Baseline Model Architecture

The goal of this project is to discern handwritten words. One way to do this is by training a model to perform character level recognition using CNNs, and use RNNs to predict the correct sequence of the word. This is what the Primary model that we proposed tries to accomplish. Another way is to map each and every word in a dictionary to a class in the output layer using a classification model, which is the basis of the idea behind our proposed baseline model. In this case, each class would be the ground truth label of the word.

One might already notice that the baseline model has glaring issues in scaling. A single language would have hundreds of thousands of words, which in turn requires an exorbitant number of classes. In fact, adding a new language to this model could double the number of classes, a serious drawback to this model.

To make our baseline model simple, we plan to use a Kaggle dataset [kag] which contains a total of 6 unique words either written in uppercase and lowercase. Thus, there are a total of 12 classes. When comparing to the primary model, the differences with classification models and dataset limitations will be accounted for.

The baseline model consists of 2 convolution layers, both of which are followed by the ReLU activation function and MaxPooling layers. The convolutional feature maps are then transposed to a certain dimension and passed into a Fully Connected section which maps these feature maps to one of the possible classes. This section consists of 2 Fully Connected layers with a softmax activation function on the last layer applied internally by the Cross Entropy criterion function. The baseline model architecture is shown below in Figure 4.

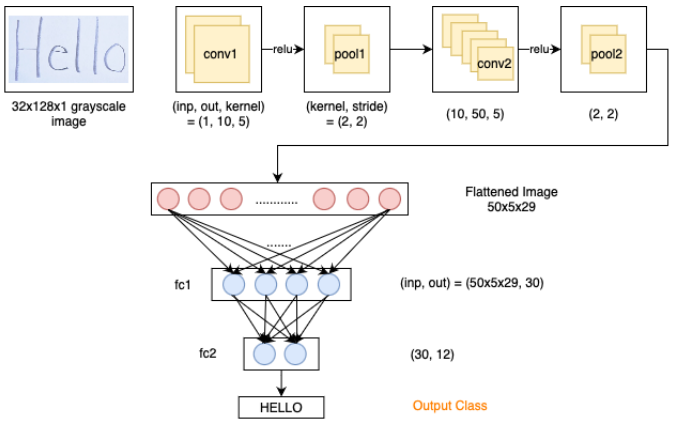

Figure 4: *Summary of the Architecture of the Baseline Model*

After training the baseline model and tuning our hyper parameters, we find the results in Figure 5. As can be seen, the classification model with limited layers is capable of gaining about 63 percent accuracy. On further analysis, we also notice that in specific cases, our baseline often fails is identifying the letter 'r' with handwritten cursive writing.

4 Analysis

In this section, we discuss the results of the baseline model and our primary model both qualitatively and quantitatively. We proceed to describe our primary model's performance in testing and the ethical consideration we needed to examine while implementing the model.

4.1 Baseline Model

Quantitative Results:
After building, evaluating and tuning the baseline model, the results can be seen in Figure 5. From the graphs, it becomes apparent that somewhere after the 15^{th} epoch, the model starts to over fit and memorize the training data. It fails to generalize the data further since we see the validation accuracy stagnates at approximately 70%. The loss also deflects around the same epoch, emphasizing the baseline's inability to generalize the data.

Qualitative Results:
As we know, the Baseline Model is a CNN Classification model and works with 12 different classes of handwritten words. CNN classification models are known to preform well and given the limited scope of the baseline model, the resulting accuracy is somewhat expected. As we dig deeper into the model's performance, we notice the model is better at classifying longer words. This most likely

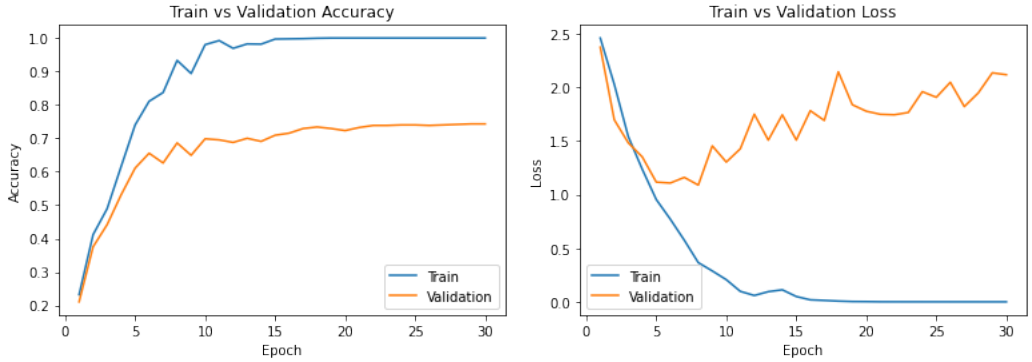

Figure 5: *Training vs Validation Accuracy and Loss for Baseline Model*

can be attributed to the increase in features the model can use for identification. This model however would not be able to extrapolate it's knowledge in our given problem since we are recognizing any character sequence instead of words from a given list.

4.2 Primary Model

Quantitative Results:
After building, evaluating and tuning the primary model, the results can be seen in Figure 6. From the graphs, we notice a very different trend as compared to the baseline model. Just visually, the model takes far longer to train and has a far smoother path. The model is well fit as there are no signs of deflection in validation loss. The model's training and validation accuracies both rise gradually and only show signs of stagnation towards the 50^{th} epoch. The model manages to reach around 85% accuracy in training and 75% accuracy in validation. Compared to the baseline, these results outperform our original expectations.

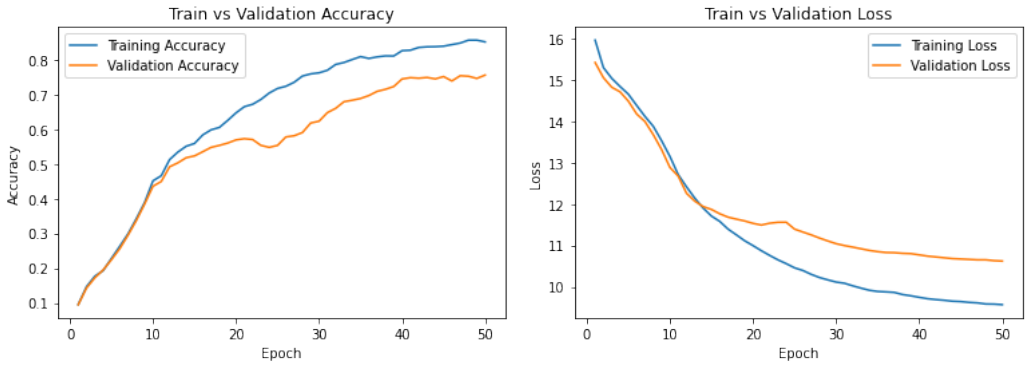

Figure 6: *Training vs Validation Accuracy and Loss for Primary Model*

Qualitative Results:
The Primary model being a CRNN uses a combination of CNNs and RNNs to sequentially identify the letters and establish a string. The model's output results after the RNN are further processed to remove spaces and low probability letters. Our accuracy is further calculated based on the number of common substrings between the truth label and the predicted truth. This is done using a Sequence Matcher library in Python. The reason this is important is to understand that our model's high accuracy score does not refer to exact matches. Below, in Figure 7 is a sample of our model's predicted output against the ground truths of some images.

What we notice on a larger scale is that 100 percent matches between the ground truth and the predicted word are more prevalent in smaller words but when they face errors, they're equally likely

```
[32] test_loader = torch.utils.data.DataLoader(dataset=test_data, batch_size=256, shuffle=True)
     acc = get_accuracy(best_model, test_loader)

     Truth:  once  | Prediction:  orncEce
     Truth:  incident  | Prediction:  incicdentHt
     Truth:  rolling  | Prediction:  rolLingj
     Truth:  the  | Prediction:  the
     Truth:  and  | Prediction:  and
     Truth:  her  | Prediction:  he
     Truth:  House  | Prediction:  HAHtousS
     Truth:  pay  | Prediction:  payFP
     Truth:  the  | Prediction:  the
     Truth:  proposal  | Prediction:  proposalT
     Truth:  means  | Prediction:  meanmTP
     Truth:  may  | Prediction:  may
     Truth:  England  | Prediction:  England
     Truth:  may  | Prediction:  muyTyFP
     Truth:  strike  | Prediction:  strike
```

Figure 7: *Model Output vs Ground Truth*

irrespective of the length of the word. Longer words, in contrast, make less differences in its predictions but hardly ever have 100 percent matches.

4.3 TESTING

Our testing data is comprised of two things: the first is the 10% split from our original dataset, and the second is our own collection of test data. We collected our own testing data, by asking various students in the university to write random words on blank paper. An example of test data from the combination of both is below in Figure 8.

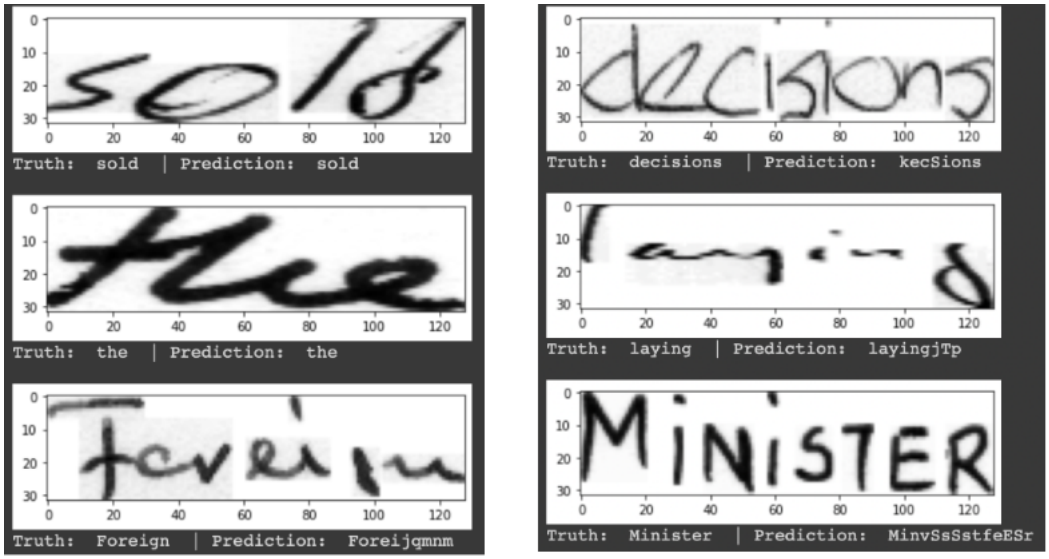

Figure 8: *Visualizing a Sample of Test Images, Truth Labels and Predictions*

Similar to our previous analysis, we see our model performs reasonably well in deciphering the written words and has a large number of 100 percent hit rates with smaller words. Surprisingly, we also see that in cases like the word ”Minister”, which is clearly written in bold, the model is unable to decipher, although we naturally see the distinctive characters. On the other hand, for words like ”laying” which is written in cursive, our model is able to detect the important features and make accurate predictions that humans often fail to.

We were able to run our test dataset through our model and got a testing accuracy of 75.76%. The model has outperformed our estimation of 65%.

4.4 Ethical considerations

We have identified two areas in which ethical concerns could arise.

4.4.1 Ethical Consideration 1: Data Collection

While collecting test data, we ensured to ask for consent to prevent a breach of privacy. Furthermore, we also understand the importance of diversity in our dataset and asked people regardless of age or sex, reducing the chance of bias affecting the model itself.

4.4.2 Ethical Consideration 2: Implications of the model

We recognize that our model could be used for plagiarism as it can parse a photo of handwritten text and convert it accurately to digital text.

Another consideration is that our model could randomly predict a text that could be offensive to certain parties. Although part of the responsibility lies with the writer, our model is the one that predicts the output.

5 Conclusions

Throughout the process of building this model, we've managed to find an appropriate dataset, pre-process according to our proposed model architecture, create a baseline model, design and train a primary CRNN model and analyze the established results. Although the team faced difficulties in finding an ideal dataset, implementing CTC-Loss and performing Hyperparameter Tuning, our final test results are better than initially expected. The model on test data, performs with a 75% accuracy, exceeding that of the baseline, even though the baseline works only with 12 distinct classes and is, hence, a classification problem. The model still has the potential to grow with further implementations of word dictionary matchings and more optimized GPU-based training. Finally, as a team, we've learned much from this project and hope to improve our model in due time.

REFERENCES

Handwritten words dataset. URL https://www.kaggle.com/datasets/nabeel965/handwritten-words-dataset.

Handwriting Recognition with ML (An In-Depth Guide), June 2022. URL https://nanonets.com/blog/handwritten-character-recognition/.

Batuhan Balci, Dan Saadati, and Dan Shiferaw. Handwritten Text Recognition using Deep Learning. pp. 8.

Field Cady. Hidden Markov Models: an Overview, February 2022. URL https://towardsdatascience.com/hidden-markov-models-an-overview-98926404da0e.

Nibin Vidyadharan. iam_handwriting_word_database. URL https://www.kaggle.com/datasets/nibinv23/iam-handwriting-word-database.