CS3244-2010-0006 - Evolving Handwritten Words, One At A Time

Authors: Chan Wen Yong, Chew BangYao Paul, Ignatius Ong Sing Ye, Ge Wai Lok, Dannica Ong, Low Hon Zheng
Email: e0415677@u.nus.edu, e0424739@u.nus.edu, e0324134@u.nus.edu, e0323018@u.nus.edu, e0202849@u.nus.edu, e0424726@u.nus.edu

Abstract

Using Deep Learning to predict handwritten texts, we can help to increase the efficiency of workers who often struggle to read illegible handwritten documents. We aim to build a model that can predict English handwritten texts word-by-word; this model can then be used by workers to aid in their daily operations. Using Convolutional Neural Networks coupled with Bidirectional Long-Short Term Memory neural networks, we are able to achieve a prediction accuracy of 78% on handwritten words, which is further boosted to 81% after passing our results through an existing vocabulary spell checker.

1 Introduction

In today's working environment, we cannot escape the fact that we have to deal with tons of paperwork every day. As such, the different kinds of handwriting on this paperwork might be a hassle to read and understand. Therefore, there is a need for a model that can accurately predict handwritten words and transcribe them into digital texts. The situation is even more severe in the field of education, where research has shown that learning by writing is more powerful than learning by typing[1]. Amid the current COVID-19 pandemic, scanned handwritten texts are the only option for many workers such as teachers as access to physical handwritten copies is often restricted due to safety reasons. As such, the demand for a model that can read handwritten texts and give accurate predictions has never been higher. This motivated our group to work on this topic of handwriting Optical Character Recognition(OCR).

In this report, we explore the effects of varying the numbers and types of hidden layers on the performance of the model and provide a hypothesis as to why some hidden layers were preferred over others. Additionally, we also investigate whether using a more complicated model necessarily leads to better performance in the context of image recognition. Lastly, we aim to study the effects of regularisation on model performance.

Finally, a section of the report will be allocated to error analysis which hopes to identify potential pitfalls of our model and to explore what is causing our model to perform worse than expected. This will provide opportunities for future work on improving our model and coming up with a more conclusive model that is capable of performing better by targeting and tackling different causes of errors that we successfully identify.

As part of our exploratory learning, we employed two models in our testing, the first was made with reference to Madeyoga's[2] previous work and using concepts learned during lectures while the other was a modification from the ResNet50 model. The codes used for both models can be found here[1][2]. The ResNet50 model was chosen due to its ability to perform with its fine-tuned model parameters. Both models had the same input of 32 x 128 images and both were made to produce and output digitalized texts which we evaluated against the image labels. We then used accuracy score metrics to evaluate the final performance of these two models.

2 Related Work

Handwritten word recognition is a combination of computer vision and sequential data processing where given an input word image, we attempt to predict the word present in the image. The sequential data processing arises from the fact that the next character to be predicted in the word is dependent on the previously predicted character. With recent developments and a better understanding of Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN), there have been breakthroughs on solving the problem of handwritten word recognition by using a combination of CNN and RNN. Since CNNs are typically used as feature extractors for image classification and recognition, and RNNs are used for sequential data, theoretically, they will be an excellent fit for handwritten word recognition. Based on past handwritten word recognition projects such as the one by Madeyoga whose model architecture is inspired by Shi et al. (2015)[3], we know that a hybrid model of CNN and RNN can be used to solve the handwritten word recognition task effectively.

However, due to rapid developments in this field, there are many different variants of RNNs for transcription. For example, under RNN alone, there are many types of RNN architectures, such as Vanilla RNN, Gated Recurrent Units (GRU), Long Short Term Memory (LSTM) among others. With so many variants to choose from, it presents a new problem where it is difficult to determine which variant of RNN would be best suited for handwritten word recognition. While CNNs do not have as many variants, the freedom to choose how many layers we want the model to have also makes it hard to determine the optimal number of layers to use for handwritten word recognition.

In this project, we aim to find the optimal combination of CNN and RNN architecture otherwise known as Convolutional Recurrent Neural Network (CRNN) that would perform well in recognising handwritten words. Building on Madeyoga's previous work on handwritten word recognition, we will be working on replacing the CNN and RNN architecture with other variants to investigate the effects of different CNN and RNN architectures on handwritten word recognition. Additionally, we will be exploring the effects of varying the number of layers in CNN as well as the impact of CNN and RNN complexity on overall word recognition performance.

3 Methods

As the community of deep learning and machine learning gets bigger, an ever-increasing number of methods are surfacing to tackle image classification problems. However, many of these methods are plagued with a variety of problems and the majority of these problems are non-trivial, but quite a few can be solved with an apt choice of data used for machine learning.

Data is a crucial aspect of machine learning, and having a good dataset, which often means that the data is clean and is a good representation of the underlying hypothesis that we would like to

[1] CRNN model: https://colab.research.google.com/drive/14J8BopUbH2P1lh-c1AF3d3VvhrkUgBAu?usp=sharing
[2] ResNet50 model: https://colab.research.google.com/drive/1Wq59kFRNE-G4rayN4GWLmg63u8_5zljS?usp=sharing

produce, is of utmost importance. Machines need good data to learn and to differentiate between classes of objects. Without a good data representation, what the machine learns will not be a good representation of the ground truth generalisation model of what we would like to predict, and the bias generated will also ruin the model.

As such, our group researched and compared different sources of data, ultimately deciding on using the IAM dataset of handwritten words over the full course of our experiment. The IAM handwriting Database 3.0 contains over 115,000 images of labelled English words from over 600 different writers.

From Kaggle, we obtained a dataset consisting of a sample of images representative of the entire IAM dataset. This dataset consists of approximately 96,454 images of handwritten words, and have up to 10,841 different words in 1,066 different forms, including 78 different characters, numerical values, English alphabet and special symbols such as comma, semicolon and full stops. These are all contributed by approximately 400 writers.

We first preprocessed the dataset by resizing the input images to a size of 32 x 128. The dataset along with their individual labels are then separated into a training set, validation set and testing set, in the ratio of 7:2:1. The training set and validation set are used during the training phase whereas the testing set is used to evaluate the final performance of the model after the training phase has been completed. Finally, we feed the data to the model.

3.1 Convolutional Recurrent Neural Network

Optical character recognition converts the handwritten scripts present in an image into digital text. In this work, we use a CRNN network as shown in the figure below. It consists of 8 CNN layers with max-pooling, followed by 2 bidirectional LSTM (BiLSTM) layers. Softmax activation is used on the last layer to output the probability of classification.

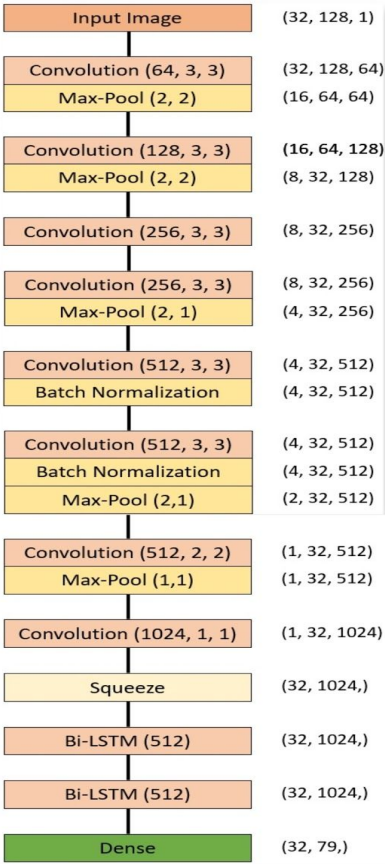


Figure 1. Model architecture of CRNN model

In general, the CRNN models that we explored have architectures similar to the one shown above. We train the model using gradient descent, minimising Connectionist Temporal Classification (CTC) loss as defined below.

$$p(Y|X) = \sum_{A \epsilon A_{x,y}} \prod_{t=1}^{T} P_t(a_t|x)$$

CTC is commonly used as a transcription layer in neural networks which converts a vector of probabilities resulting from the neural network computation and converts it into text.

3.2 ResNet50 model

As part of our learning through exploration, we also tried to use a more sophisticated trained model called the ResNet50, coupled with BiLSTM as our second network to train. ResNet50 models are rising in popularity over the years due to its capability of skipping connections in very deep neural networks. Residual connections allow the model to dynamically tune its layers during training, ultimately obtaining the most optimal number of layers for the model. A self-learning model would be more efficient in learning as we do not need to tune its parameters during the training phase.

We modelled our ResNet50 network from Priya Dwivedi[4]. Using a similar input of size 32x128, the image will then be passed through 5 different stages.

Stage 1: The 2D Convolution takes in an input of 32x128 and uses a convolution filter of 7x7. BatchNorm is also applied to the channels axis of the input. Additionally, MaxPooling was used with a (3,3) window and a (2,2) stride.
Stage 2: The convolutional block uses three sets of filters of size [64,64,256] and the 2 identity blocks use three sets of filters of size [64,64,256].
Stage 3: The convolutional block uses three sets of filters of size [128,128,512] and the 3 identity blocks use three sets of filters of size [128,128,512]
Stage 4: The convolutional block uses three sets of filters of size [256, 256, 1024]. 5 identity blocks were used here with a filter size of [256,256,1024]
Stage 5: The convolutional block uses three sets of filters of size [512, 512, 2048]. The 2 identity blocks use three sets of filters of size [512, 512, 2048]

This ResNet model is then added to a BiLSTM layer and was trained using CTC loss.

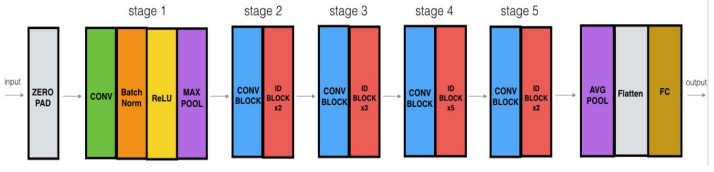


Figure 2. Model architecture of ResNet50 model
Diagram credit: Priya Dwivedi@towardsdatascience

4 Experiment

4.1 Evaluation metrics

To ensure a fair evaluation is conducted, we used the same evaluation metrics to predict the accuracy of the model throughout

the experiment. The accuracy metrics is a measure of correct predictions against all predictions made by our network, whereby the number of correct predictions is the sum of True Positives (TP) and True Negatives (TN), and the total number of predictions is the total sum of True Positives (TP), True Negatives (TN), False Positive (FP) and False Negatives (FB). The formula is shown below

$$Accuracy = \frac{Correct\ Predictions}{All\ Predictions} = \frac{TP + TN}{TP + TN + FP + FN}$$

4.2 Effects of CNN vs RNN in character recognition

	Number of parameters	Test accuracy	Lowest validation loss
Deepening CNN	19,321,581	75.2%	1.585
Deepening RNN	20,143,439	77.1%	1.471

Figure 3. Results of deepening CNN vs RNN

We observed that while keeping the number of parameters relatively similar, the deepening of RNN proved to be more useful in recognising handwritten texts. For time-series data that contains repeated patterns, the RNN is able to recognize and take advantage of the time-related context[5].

4.3 Effects of using RNN variants

	Number of layers	Test accuracy	Lowest val_loss
LSTM	1	63.8%	1.816
BiLSTM	1	73.5%	1.481
GRU	1	67.6%	1.946

Figure 4. Results of using different types of RNN variants

Figure 4 above shows how different RNN variants affected the test accuracy of the network to different extents. By analysing and comparing the test accuracy as well as the validation loss of the different RNN variants, BiLSTM seems to be the most suitable RNN model to merge with the existing CNN model to predict image sequence data. This goes along with our hypothesis that it is better to learn sequence data from both directions, as it provides the model with more time-related context and allows the model to predict more accurately.

In general, because GRUs control the flow of information without using a memory unit, they are simpler, have faster training time and perform better than LSTMs on less training data. However, as suggested by Yin[6], in experiments that take in data of longer lengths, the RNN models chosen are required to remember these longer sequences. Due to the lack of a memory unit in GRUs, LSTMs thus outperform them in tasks which require modelling long term dependencies. Therefore we chose LSTM variants over GRU for our final model.

4.4 Effects of regularisation

	Test accuracy	Lowest validation loss
Baseline model	78.1%	1.449
No dropout	75.7%	1.407
No batchnorm	72.8%	1.662

Figure 5. Results of not using regularisation techniques

Baseline model used was CRNN with 2 BiLSTM layer
Figure 5 showed that using dropout and batchnorm in the model is rather important as it prevented our model from overfitting the data. As observed, when compared with the baseline model, the model with no dropouts attained a lower test accuracy even though it has a lower validation loss. This suggests that the model overfitted the data and we will need to minimize its generalization error. This is aligned with our hypothesis that regularization techniques are certainly important as we need our model to generalise well if we want to implement this model in the real world.

Additionally, we showed that batchnorm is very vital in such CRNN models as it is capable of improving model performance by at least 6%. Removing the batchnorm layer impaired our model's performance. One of the possible reasons is that batchnorm allowed the model to converge faster and substantially reduced training epochs. It mitigates the problem of internal covariate shift, where parameter initialization and changes in the distribution of the inputs of each layer affect the learning rate of the network. In addition, batchnorm allowed each layer of the network to learn by itself more independently of other layers, hence improving test accuracy.

4.5 Results and Discussion

Type of model	Number of hidden CNN layers	Test accuracy	Lowest val_loss
CRNN (2 BiLSTM)	7	-	3.228
CRNN (1 BILSTM)	8	73.3%	1.481
CRNN (2 BILSTM)	8	78.1%	1.449
CRNN (3 BILSTM)	8	73.5%	1.692
CRNN (2 BiLSTM)	9	66.2%	2.158
ResNet50 (2 BiLSTM)	50	75.0%	1.528
CRNN (2 BiLSTM) (with spellchecker)	8	81.0%	1.449

Figure 6. Results of varying RNN layers and ResNet

Following the results of our various experimentations, we decided to employ the use of a CRNN model with 8 hidden layers and 2 BiLSTM RNN layers as our final model.

Comparing the performances of the different experiments, one other observation that we made was that using a model with complexity greater than the one mentioned above could lead to poorer results. The reason could perhaps be that the higher complexity models tend to overfit the data, which resulted in poorer performances. Therefore, our groups hypothesized that more data may be necessary if we want to use a higher complexity model (9 layers of CNN and 3 layers of BiLSTM) for this problem.

Also, it is worth mentioning that the use of a Levenshtein based spell checker on the final model improved our model performance from 78.1% to 81%.

The Levenshtein distance based spell checker makes use of the minimum edit distance to identify a word present in its vast vocabulary that is most similar to the input given. This will help to further refine the output generated by our model. The minimum edit distance refers to the number of edits required to change one word into another through insertions, deletions or substitutions. The Levenshtein distance between two strings can be calculated based on the following function in figure 7.

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Figure 7. Levenshtein Distance formula

where $1_{(ai \neq bj)}$ is the indicator function equating to 0 when $a_i = b_j$ or 1 otherwise, and $\mathrm{lev}_{a,b}(i,j)$ is the distance between the first i characters of a and the first j characters of b. i and j are 1-based indices.

As observed from figure 6, the use of a rather basic CRNN model with 2 BiLSTM layers outperformed the ResNet50 model, even

though the ResNet50 model has a significantly higher number of model parameters and theoretical performance. Our hypothesis on this is that to fully maximise the capability of a complex model like the ResNet50, we need to have a larger amount of data that provides a good representation of the underlying hypothesis that we are trying to model, which is not the case in this experiment.

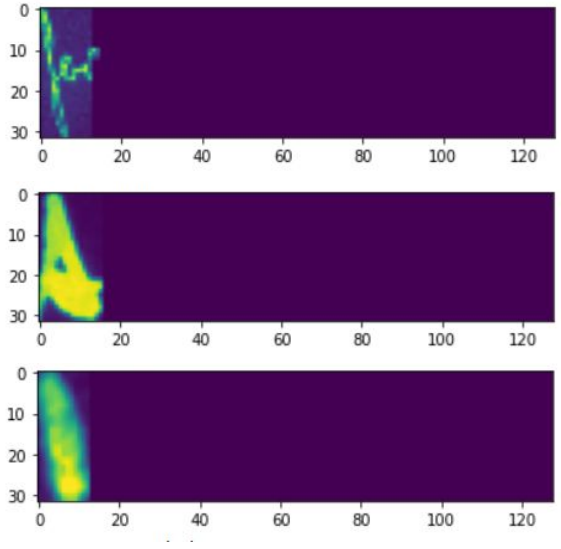
For the variants of RNN layers, we observed that using BiLSTM in lieu of a normal LSTM significantly improved our model's accuracy. We hypothesise that because BiLSTM allows information to flow in both directions compared to only one direction in LSTM, it allows our model to store and learn from information based on context from the past and the future. Through our experimentations, we also observed that having 2 layers of BiLSTM provided us with a model that returned the highest test accuracy and the lowest validation loss. This could be due to having 2 layers that give sufficient parameters while keeping a relatively smaller overfit[7].

Hence, we decided to employ the use of a CRNN model with 8 hidden layers with batchnorm and dropout and 2 BiLSTM layers as our final model.

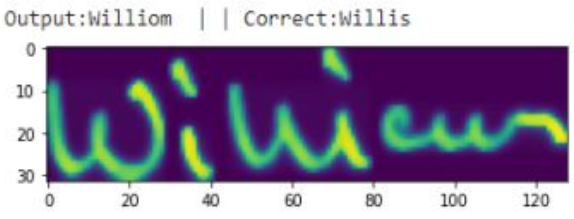
5 Data and Error Analysis

5.1 Data analysis

Overall, the data from the dataset was generally consistent except that some images were too small even after doing preprocessing. Hence, when we resized them to a standardised size, the quality of the image was compromised. Some examples of these images can be seen below.

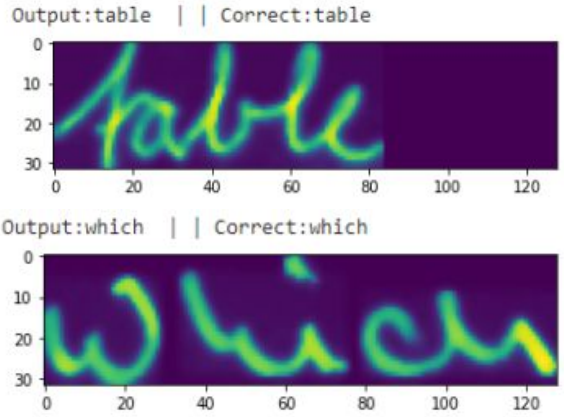


Some of the data were also wrongly labelled and were misleading as the label did not reflect the word it should be if the image were to be read by the average person. One such example can be seen below.



5.2 Error analysis

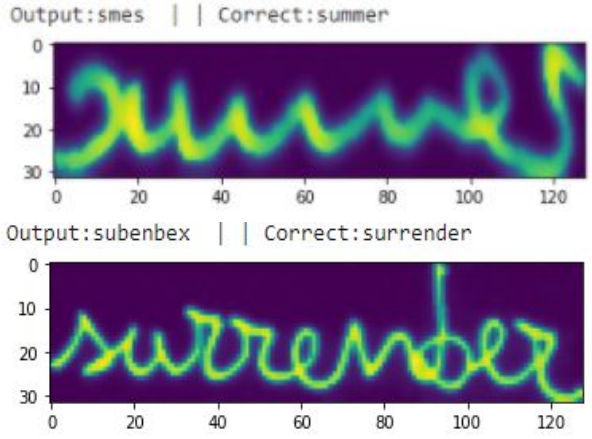
Surprisingly, our model exceeded our expectations as it was able to distinguish a handful of words that even humans struggled with the naked eye as seen below. This is a pleasant surprise as it aligns with our original motivation of recognizing bad handwriting.



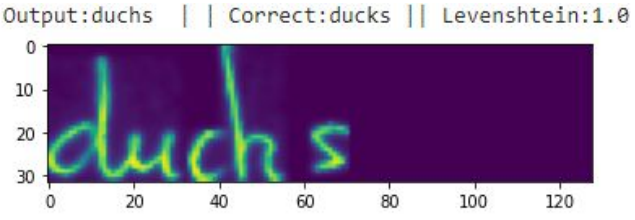
However, we observed there were some errors which our model consistently struggled with. In the following subsections, we will be taking a more in-depth analysis of these different errors which our model made.

5.2.1 Cursive handwritings

There were some instances in which our model struggled to correctly predict cursive handwriting. This could be due to the difficulty in distinguishing between certain cursive characters and non-cursive characters for example the cursive 'r' as seen in the following 2 examples.

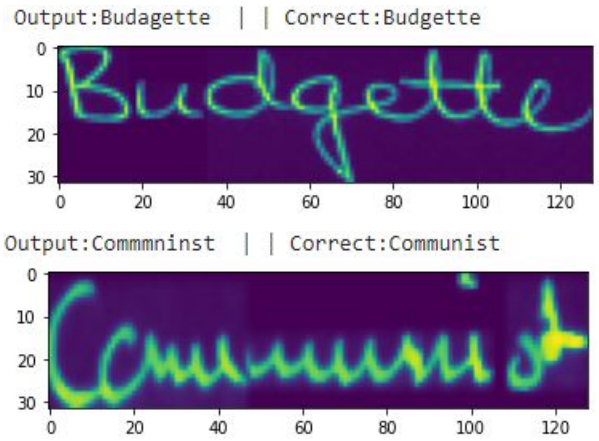


In the image below, we also observe that the model mistook the cursive 'k' for the non-cursive 'h' due to the way the cursive 'k' was written.



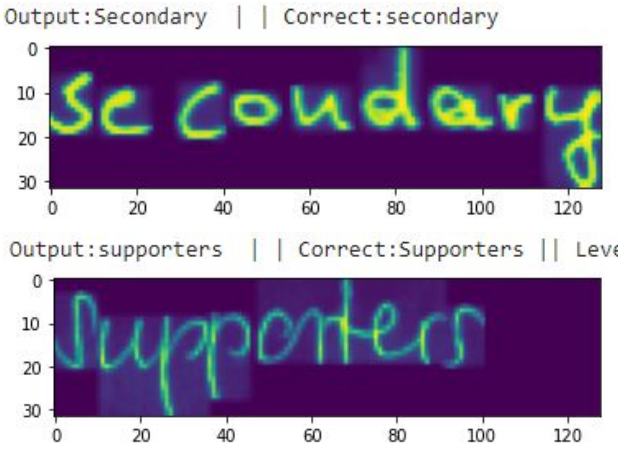
5.2.2 Separating word images accurately into characters

Our model also failed to accurately split the input images into individual characters in certain instances. Due to the difficulty and difference in separating characters in cursive words as compared to non-cursive words, it resulted in extra or fewer characters being predicted by the model in certain instances as seen in the following examples.



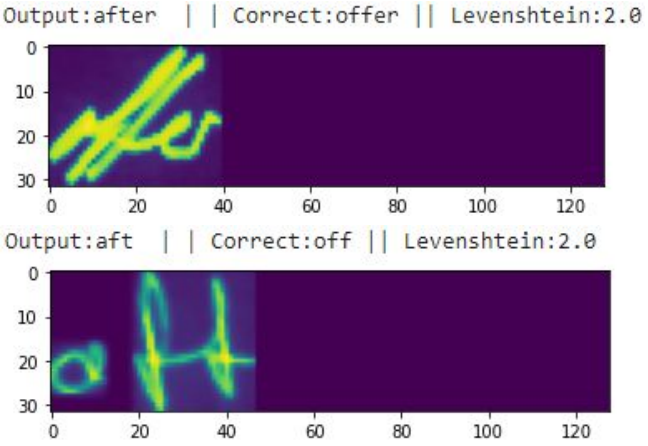
5.2.3 Similarities between upper and lower case letters

In some instances, we notice that the model struggles to distinguish the correct form of the character when the upper-cased character is similar to the lower-cased character. One clear example is presented in the image below where the letter 's' was mistakenly predicted as 'S' by the model. However, the converse is also true. In the second example, the model mistakenly predicted 'S' as 's'. Upon observing both images, it can be quite difficult to distinguish whether the 's' should be upper or lower case and we would need the context of the prior word to help us determine this. While not shown here, we have also found this to be true for other characters such as 'u', 'o', 'w', 'm' and many others. Hence, we can see that the model will potentially struggle with predicting characters whose upper and lower cases are very similar.



5.2.4 Similarities between different characters

Additionally, this tendency to mistake characters for their upper or lower case can be extended to other characters too. As seen in the examples below, our model predicted 'a' instead of 'o' in both cases. One reason for this wrongly predicted output could be due to the similarity in form between 'a' and 'o'. However, this error could also be attributed to characters of cursive handwritings being less legible at the beginning compared to the middle of the word. Regardless of the reason, we can see that our model could have some difficulty in predicting characters that look similar to other characters. With the added illegibility, it would further hamper our model's ability to correctly predict the character.



6 Conclusion

6.1 Summary

Our final model, with the highest test accuracy of 77.8%, is made of 8 convolution layers with batchnorm and dropout followed by 2 Bidirectional LSTM layers and an output layer. By attaching a spell checker to our final model, it pushed the accuracy up to 81.0%.

Through our experiments, we have shown that the model can perform decently on handwritten words that are less cursive in nature. However, more work still needs to be done to improve the model's performance on words of a cursive nature.

Additionally, with just the IAM dataset, it appears that using a less complex network would be a better choice in feature extraction over more complex ones like ResNet50. However, with a more comprehensive dataset, perhaps more complex networks may be able to outperform our chosen model due to the increase in expressive power.

6.2 Shortcomings

Even though we have fine-tuned the model to have an accuracy of as high as 81%, there are still shortcomings to this model we have built.

Our current model is only able to predict up to an accuracy of roughly 81%. For it to be reliable and used in office or work settings, the accuracy of the model has to be improved.

On top of that, from our analyses, we have discovered that in some instances, cursive and illegible handwritings resulted in inaccurate predictions by the model. Our model would definitely have to be improved in this aspect as well. Other than that, for our model to be a universal handwritten word recognition software, the model needs to be able to overcome several key issues such as correctly predicting the same characters written by different people in different settings. As discussed in the earlier section of error analysis, the model faces some difficulty in correctly predicting characters that inherently look very similar to other characters. Adding on the variable of different writing styles, it would be even

harder for the model to correctly determine which character is being written.

Furthermore, the spell checker within the model uses a vocabulary list that may not be updated regularly. With new words being added to dictionaries, we would need to find a way to regularly update the vocabulary list for the spell checker component of the model to remain relevant and accurate.

6.3 Future Works

There are definitely things we can work on in the future to improve not only on overcoming the shortcomings of the model but also to further improve on the accuracy of the model made.

These are the areas we can work on in the future:

1. Removing noisy data from the IAM dataset. It is important to remove noisy data as it will lower the risk of our CRNN model memorising the incorrectly labelled data and giving out incorrect predictions when it is put to test.
2. Separating data into cursive and non-cursive and subsequently train models specialised in each task. With the help of specialised models to predict cursive and non-cursive characters separately, it will help to improve the accuracy of our overall prediction.
3. Incorporating paragraph segmentation models. By merging and incorporating a paragraph segmentation model to our word prediction model, it will widely increase the usability of the model and also increase the efficiency on the user's end.
4. Using NLP to correct wrongly predicted words. The current spell checker implemented solely works on correcting the output to the next closest word present in its vocabulary. With the addition of NLP, the context of the sentence a word is in will also be studied to give us an even more accurate prediction of the word. This can also solve the issue of being unable to determine if a predicted character should be capitalised.
5. Ensembling different models together to further improve accuracy. By taking the average of many hypotheses, using ensembling will help reduce the variance of the final hypothesis and hence allow our model to be more easily generalised to new data. On top of that, this method will also help resolve some of the issues we encountered in our experiments where the model was unable to accurately separate words into characters and struggled in accurately predicting characters that bear a close resemblance to others.
6. We can work on introducing greater customizability to the languages the model can predict on. With a diverse variety of languages written throughout different parts of the world, any viable and competitive OCR model needs to be more flexible in the variety of languages it can predict words in. A possible direction would be to encase different character lists as well as vocabularies for the different languages. When the user specifies the language his document is in, the model would then use the character list and vocabulary tied to that language to predict the output.

REFERENCES

[1] Writing By Hand Makes Children Better At Learning, Study Says https://www.weforum.org/agenda/2020/10/handwriting-children-digital-soft-skills/

[2] Madeyoga https://github.com/madeyoga/Handwritten-Text-Recognition

[3] B. Shi, X. Bai, C. Yao. An End-to-End Trainable Neural Network for Image-based Sequence Recognition and Its Application to Scene Text Recognition. IEEE Trans. Pattern Analysis and Machine Intelligence, 2016.

[4]Priya-dwivedi/deep-learning Priya-Dwivedi- https://github.com/priya-dwivedi/Deep-Learning/blob/master/resnet_keras/Residual_Networks_yourself.ipynb

[5]Rnn Vs Cnn For Deep Learning: Let's Learn the Difference Marketing-Exxact Corporation - https://blog.exxactcorp.com/lets-learn-the-difference-between-a-deep-learning-cnn-and-rnn/

[6]W. Yin, K.Kann, M. Yum H.Schutze. Comparative Study of CNN and RNN for Natural Language Processing. arXiv:1702.01923

[7]A.Zeyer, P.Doetsch, P.Voigtlaender, R.Schluter, H.Ney. A Comprehensive Study of Deep Bidirectional LSTM RNNs for Acoustic Modeling in Speech Recognition. 2016. arXiv:1606.0681