Optical Character Recognition for Handwritten Images

Dhayarkar, Rishikesh
rbd291@nyu.edu

Kasireddy, Durga Prasad Reddy
dpk290@nyu.edu

Abstract

This project aims to implement a robust approach in solving the task of Optical character recognition specifically on handwritten images. We utilize various techniques in image processing and deep learning to achieve this objective. The overall objective is split into three major sub-tasks. The first one includes pre-processing the input images by performing operations such as blurring, edge detection, noise removal, erosion, dilation etc to obtain a useful version of the input images. The second sub-task includes running segmentation algorithms to generate segments of lines and words. The third sub-task includes running a deep learning model to predict a text version of input image. Our approach is well suited for generating word-by-word predictions. Our approach is works well for both scanned images of text as well as photos of text documents.

Github Link -
https://github.com/RishikeshDhayarkar/CS-GY-6643-Computer-Vision-Final-Project

1 Introduction

Optical character recognition(OCR) is one of the most important computer vision tasks which has attracted numerous work and challenges on building effective and robust OCR systems. OCR is broadly classified into two classes. The first type is for converting images of typed computer text to a more convenient text format. The second one is for converting images of handwritten text to text format. Some popular applications of OCR include - Data entry for business documents, e.g. Cheque, passport, invoice, bank statement and receipt, Automatic number plate recognition, Passport recognition and information extraction systems, Automatic grading of handwritten tests, Archiving historical handwritten information etc.

While we have been using OCR for a wide variety of applications, modern methods still have some difficulty in recognizing handwritten information. Handwritten information is extremely diverse in nature (e.g. cursive writing, different forms of the same character etc). People have different styles of writing the same piece of information and this poses significant challenges when it comes to converting them to text format.

Novel techniques in deep learning have made a significant positive impact on almost all applications in the field of computer vision and this applies to OCR as well. The problem of OCR for computer typed text has been solved with great accuracy by modern systems but the same cannot be said about hand written images. In our work we specifically focus on OCR for handwritten text images which is also called as Handwritten Character recognition(HCR).

2 Approach

Our approach is broadly divided into three stages - Preprocessing input images, Segmentation of lines and words, and image to text conversion using a deep learning model.

2.1 Preprocessing input images

Our approach can handle both scanned images as well as photographs of text documents obtained from a camera. These inputs have different characteristics and need individual image processing techniques to convert the input image into a usable form.

2.1.1 Preprocessing scanned images

There are a series of steps we follow to preprocess the image before sending it to the deep learning model. First, we convert the input image into grayscale and then use Gaussian filter with a standard deviation of 1.2 to remove noise from the image. We noticed that edge detection using derivative filter in X and Y axes didn't add much value since the images have enough contrast already and individual objects in the image do not take up much area. The higher the sigma value, the more the blur is. We went ahead with a sigma = 1.2 We did not perform any transformation on these images because they were already scanned copies.

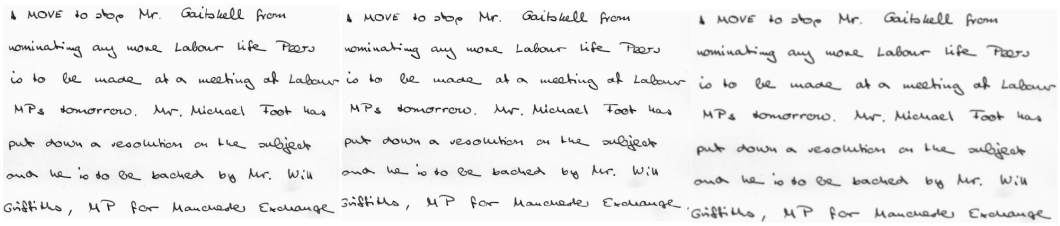

Figure 1: (a) Original image (b) Gaussian sigma = 0.2 (c) Gaussian sigma = 1.2

The size of each image is 3542*2479 which is too large to process with the limited computation power we have. Hence, we reduced the image to 1200*839 maintaining the same height to width ratio. This is an optimization problem where we need to choose image size such that it didn't compromise the sentence segmentation or word segmentation or deep learning part and yet be processed in a timely manner. To solve this optimization problem, we performed a binary search algorithm to figure out the image size suitable and the 1200*839 worked well among many options explored.

Gamma correction is a simple but effective process to increase the luminance of the image. It takes each pixel intensity and replace with that value raised to gamma. At the end we normalize the values to stay in the range of [0-255]. Our image dataset although were scanned copies, did not have good luminance. So, we had to use Gamma correction with g = 0.5 to increase the luminance. The hyperparameter Gamma = 0.5 worked well for us from the experiments we conducted.

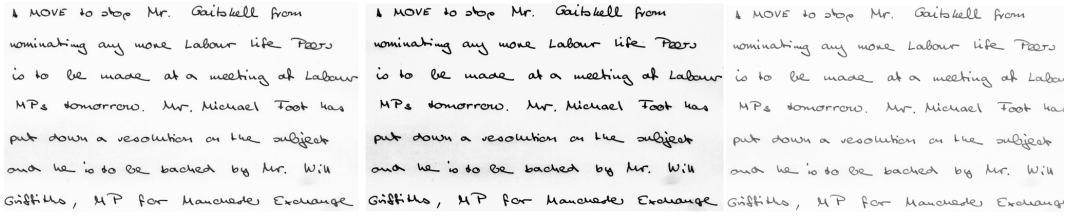

Figure 2: (a) Original image (b) Gamma = 0.5 (c) Gamma = 2

We use a laplacian filter to increase the sharpness of the image. As we can see from the filter matrix, a laplacian operator will darken the pixels which have high value. We found that this helps our segmentation algorithm to detect all the sentences appropriately. The derivative image doesn't have sharp intensity values as we can see below, which is why we used a laplacian operator and then applied a threshold to convert into black and white image.

The laplacian operator is as follows $\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$

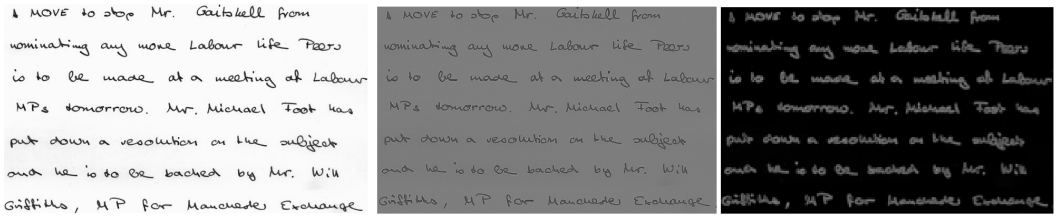

Figure 3: (a) Original image (b) Laplacian (c) Derivative

2.1.2 Preprocessing photographs of text documents

Images scanned using a camera often require an additional step of perspective transformation and warping to project the input image onto a new viewing plane that is free from the background as well as free from the angle of inclination of the camera. Preprocessing of these images involves similar preprocessing steps that are used for scanned images in addition to perspective transformation. The key to an accurate transformation is generating the optimal transformation matrix. Optimal transformation matrix can be obtained by choosing good points within the input image. In our case the optimal transformation points are the corners of the page. We follow a series of steps to automatically obtain the corners of the page.

Input images from modern cameras have a very high resolution. So, to facilitate efficient computation the first step is to resize the image to a smaller image. After resizing we perform denoising on this resized image to eliminate noisy pixels[Figure 4:(a)]. We perform operations such as edge detection, closing, Hough transforms in further steps. These operations work well when the input image is binary. Naturally the next step is perform thresholding operation to generate a binary image. For further processing we need to eliminate the handwritten content from the input images. We use closing operation to achieve this. Closing is a two step process, Dilation followed by Erosion. It is useful in closing small holes inside the foreground objects, or small black points or small black structures on the page[Figure 4:(b)]. Edge detection operation is then performed on

this closed image to generate the boundary of the page[Figure 4:(c)].

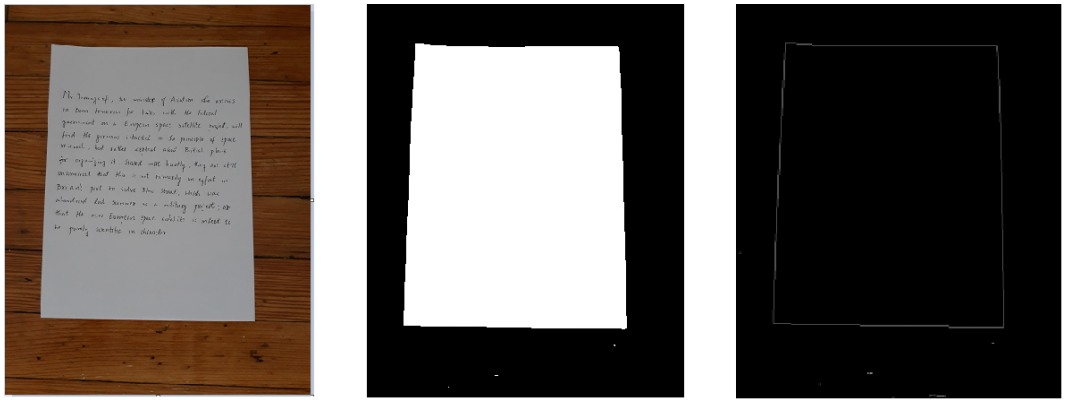

Figure 4: (a) Original image (b) Image after closing operation (c) Edge detection on closed image

The next step is to generate Hough lines along the edges of the page. Hough line function requires three parameters - rho accuracy, theta accuracy, and minimum vote needed to be considered as a line. After some experimentation we found that rho of 2, theta of 360 degrees and a minimum vote of 100 delivered accurate Hough lines[Figure 5:(a)]. Once these lines are generated, the next step is to find the points of intersection of these Hough lines. This is because points of Hough line intersection gives the corners of the page. Not all intersections are corners of the image. Since, most pages are rectangular the angle of edge intersection is approximately 90 degrees. To generate only a small set of potential corner points we first calculate angles between all possible pairs of Hough lines and discard the ones that have an intersection angle less than 80 degrees or more than 100 degrees. The pairs of lines that exceed these criteria are most likely not corner points[Figure 5:(b)]. Once we have a small set of potential corner points, we perform k-means clustering to classify the points into 4 clusters, each cluster representing one corner of the page. The final page corners are the 4 cluster centroids[Figure 2:(c)]. We use these 4 cluster centroids to perform perspective transformation on the denoised input image[Figure 6:(b)]. The transformed image is thresholded to obtain a binary image[Figure 6:(c)]. The output obtained at the end of this process usually includes very thin lines of black background which we remove manually.

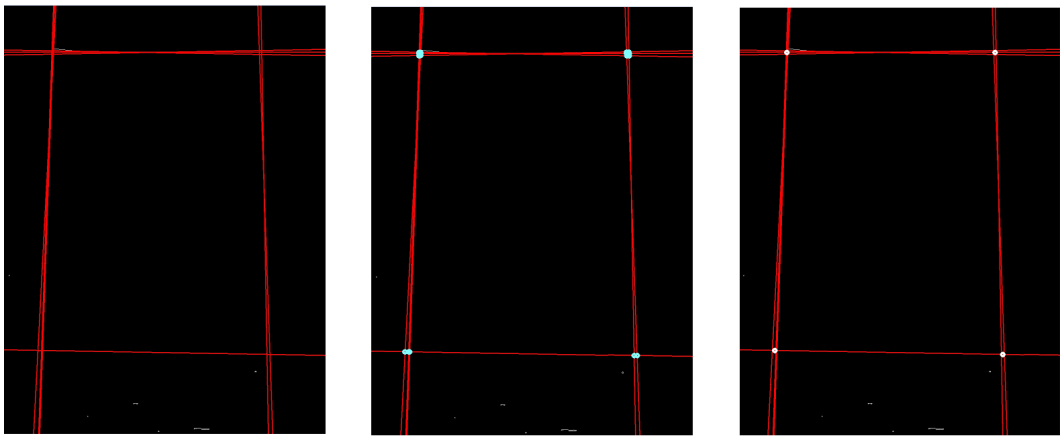

Figure 5: (a) Hough lines (b) Hough lines intersection points (c) Grouped intersection points

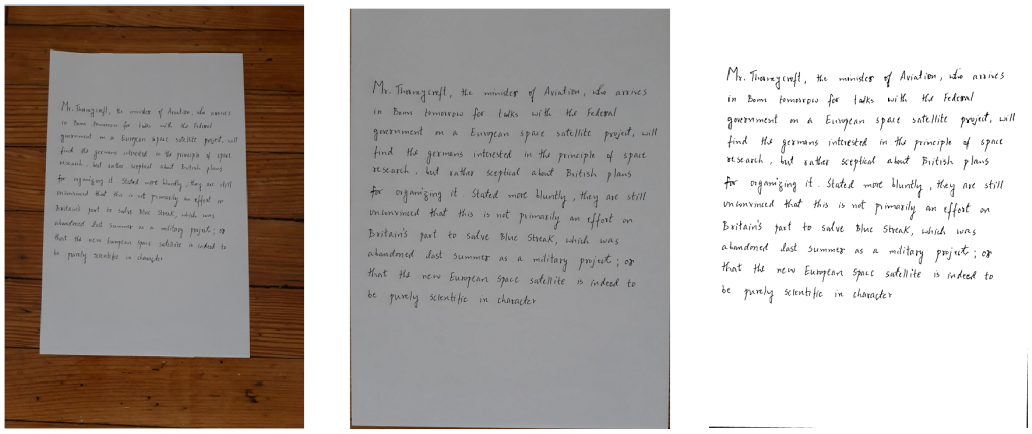

Figure 6: (a) Original Image (b) After perspective transformation (c) After perspective transformation (thresholded)

2.2 Sentence and word segmentation

We use a threshold function to convert the image into black and white. This is done to improve the results of dilation and segmentation pipeline.

In dilation, we take a pixel value and replace it with the maximum value in the neighborhood. We use the cv2.dilate function with one iteration to achieve this result. When we dilate the image, the pixel coordinates where the text was present will become white and rest all will become black as we can see in the attached images.

Once we get the dilated image, we pick coordinates of bounding boxes around these individual dilated areas and overlay these boxes on our original image to acheive the output. These boxes are individual lines and for each box, we store it in a separate png file to later send it into word segmentation pipeline which converts these lines into individual words.

Segmentation performed on the input images from a camera

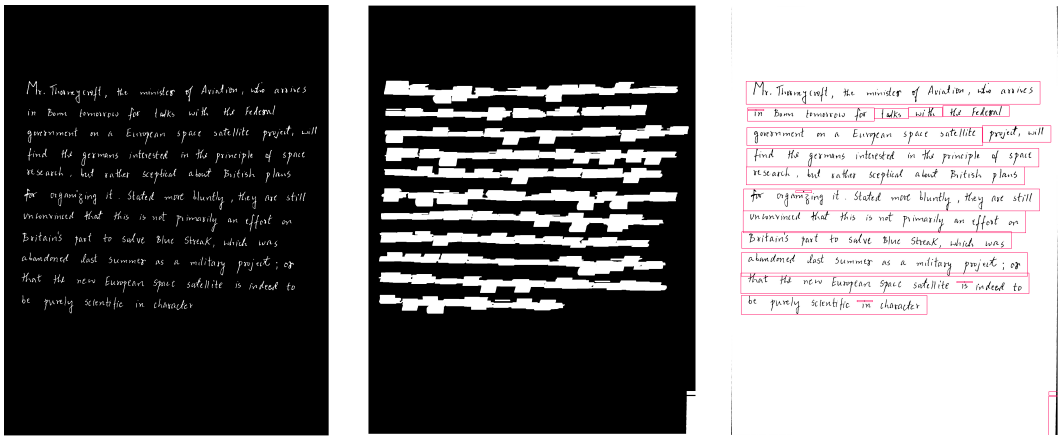

Figure 7: (a) Inverse thresholded image (b) After dilation (c) Segmented lines

In the word segmentation part of the pipeline, each input image consists of a single sentence consisting of individual words. Here we use connected components to find out which characters are close enough to be segmented as words and which are not. The way we build these components is we take the thresholded sentence segmentation image and find contours in this image. We pass these contours in a loop to check if there are any contours whose area is too small (punctuations) and combine those small contours into a single contour. If it is large enough, we then find the center, height, width or the bounding rectangle. The bounding boxes around these segmented words will be able to cover the small details like the dot on top of "i" or "j". To handle these, we chose a slightly bigger bounding box area so that they are covered.

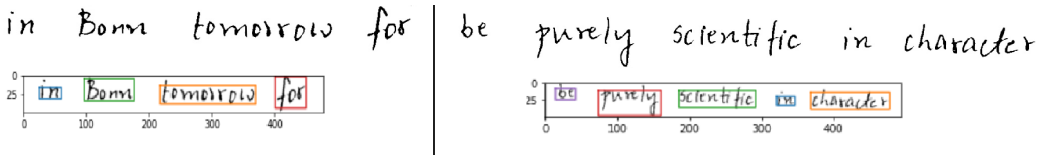

Figure 8: (a) Word level segmentation example one (b) Example two

Segmentation performed on the scanned images

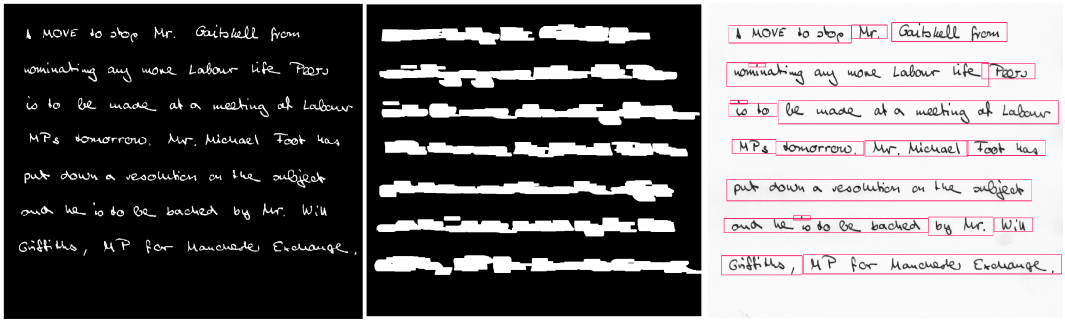

Figure 9: (a) Inverse thresholded image (b) After dilation (c) Segmented lines

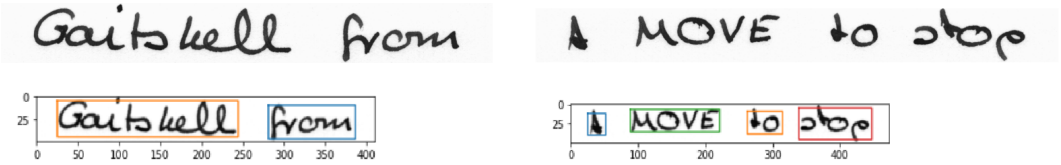

Figure 10: (a) Inverse thresholded image (b) After dilation (c) Segmented lines

2.3 Prediction using a Deep Learning model

Once the input image is segmented into words, the next step is pass these segmented word snippets through a deep learning model to generate a text equivalent. We fine-tuned a pre-trained deep learning model that is designed for the process of OCR on handwritten images[Figure 11]. The network includes a stack of convolutional NN (CNN) layers, recurrent NN (RNN) layers and a final Connectionist Temporal Classification (CTC) layer.

The input image is fed into the CNN layers. These layers are trained to extract relevant features from the image. The output of the CNN part is a downsized version of the input with all relavent features captured. While the image height is downsized in each layer, feature maps (channels) are added, so that the output feature map (or sequence) has a size of 32×256.

The final 32x256 dimensional output is fed to an RNN. The type of RNN we use is an LSTM. The feature sequence contains 256 features per time-step, the RNN propagates relevant information through this sequence. The RNN output sequence is mapped to a matrix of size 32×80. The IAM data-set consists of 79 different characters, further one additional character is needed for the CTC operation (CTC blank label), therefore there are 80 entries for each of the 32 time-steps.

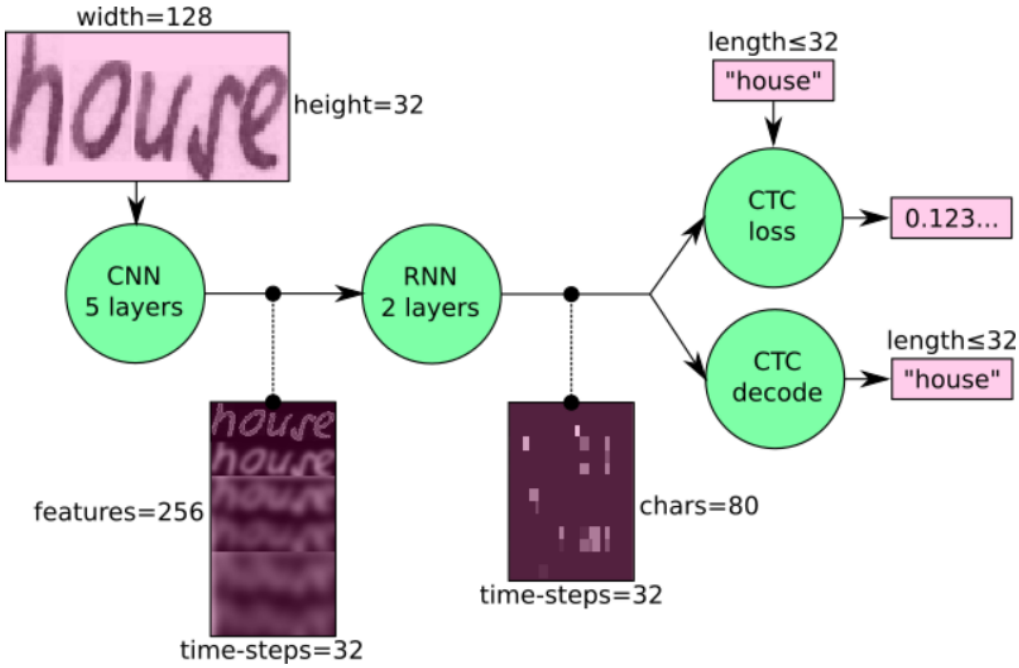

Figure 11: Deep learning model used to generate text

The CTC loss takes RNN output matrix and the ground truth text to generate the loss value in every forward pass during training. During testing the network performs a forward pass and decodes the input image into a final string output[Figure 9]. While testing, the CTC is only given the matrix and it decodes it into the final text. Both the ground truth text and the recognized text can be at most 32 characters long.

We mainly refer to this repository to for the deep learning model architecture and code - Github Link - `https://github.com/githubharald/SimpleHTR`

The images below are predictions generated by this deep learning model on the segmented words.

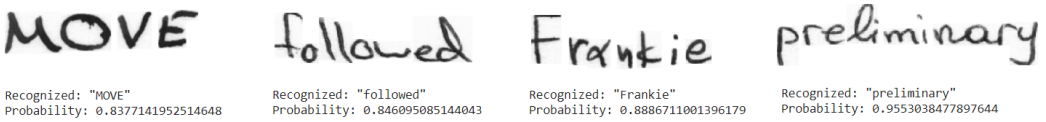

Figure 12: Predictions made by the model on the segmented words

3 Experiments

3.1 Data

Our approach is robust to image outputs from a scanner as well as a camera. We used the IAM dataset which includes a Handwriting Database that contains forms of handwritten English text which can be used to train and test handwritten text recognizers. The database contains forms of unconstrained handwritten text, which were scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels. In addition to these scanned images we were able to generate new handwritten input images captured using a phone camera.

IAM dataset contains scanned copies of the entire page along with segmented words that are present in all the pages. The pre-trained deep learning model mentioned in the previous section is trained on the IAM dataset to perform word level OCR. We augmented this dataset with our own segmented images obtained via a camera. This augmented dataset was then used to fine tune the above referenced deep learning model.

Link to the IAM dataset - `https://fki.tic.heia-fr.ch/databases/iam-handwriting-database`

3.2 Predictive model

The format of input to the deep learning model is a tuple of segmented image of a word and it's corresponding label. While our segmentation approach worked well, we were not able to use our segmented images for training the model from scratch. This is because we did not have labels for the the segmented words. For this reason we had to resort to the approach of using a pre-trained model. We were able to fine tune this model by augmenting the IAM dataset with segmented images obtained from a camera. We were able to generate about 1000 input-output tuples by manually labelling the segmented camera images. The fine tuned model gave a word accuracy of 74.26% and Character error rate of 10.58%.

3.3 Code and Infrastructure

All of our code is written in Python. We used Jupyter notebooks for interactive experimentation. Most of image processing related tasks were handled using functions from OpenCV library. Deep learning related experiments were performed using Tensorflow.

Experiments were performed on our local machines. Google Colab Pro was used for all experiments. The GPUs included in this infrastructure are T4 and P100 with 16GB of memory. The CPUs come with 25GB of RAM and 250GB of disk space.

4 Conclusion and Future Work

In this report we have discussed image processing and deep learning aspects for HCR task in detail. We provide definitive reasons for the design and model choices that we make in our experimentation. We also show and analyze the results at every step of our HCR pipeline. These are some advantages and disadvantages of our system and we hope to improve our current approach by working on some of the disadvantages in the future.

Advantages:

- Our approach can handle inputs in the form scanned images or in the form of photographs from a camera.
- Our approach is robust to various styles of handwriting. In our work we specifically talk about working with hand written images, but with minor variations our approach should work well even on typed text images.

Disadvantages:

- For camera input our approach needs the background(content other than the page) to be dark. Minor issues might arise when it comes to finding the optimal transformation points if this is not followed.
- The segmentation function needs some improvement. In our experimentation we manually set constraints to make sure very small segments(dot over letters such as 'i' or 'j') and very large segments(segment generated due to a thin long dark potions of the background) get removed.
- The accuracy of deep learning model can be improved by using a more sophisticated architecture.

Even though our system is currently functional, it is not perfect. From a software engineering perspective, most of the components in our pipeline are decoupled, more work can be done to integrate everything under one package. From a design perspective, high quality segmentation algorithms and complex deep learning models would significantly boost the performance.