Handwritten to Computer Written

A text recognition implementation

Justin Lin Sin Cho
Computer Vision 4220U
Ontario Tech University
Oshawa, ON, Canada
justin.linsincho@ontariotechu.net

ABSTRACT

Image recognition in computers has a growing market in the world with applications such as Google translate, security and self-driving cars. The desire to exchange information has changed throughout the years starting from cave paintings, to handwritten notes in quill and pen, to letters written with typewriters to today's use of the keyboard and computer. In the current day where technology is constantly evolving and is prevalent in our every day, image recognition is the connecting bridge between these primitive notions of information exchange to the modern understanding of a computer. Text recognition is a subcategory of image recognition where handwritten or computer-generated text is interpreted visually and converted from picture to computer strings.

CCS CONCEPTS

• Machine learning→ models; model fitting; model predictions; • Computer vision→Image detection; Image recognition; Image preprocessing

KEYWORDS

Datasets, neural networks, Keras model, contours

ACM Reference format:

Justin Lin Sin Cho, 2020. Handwritten to Computer Written: A text recognition implementation. In *Proceedings of ACM Woodstock conference (WOODSTOCK'18). ACM, New York, NY, USA, 2 pages.* https://doi.org/10.1145/1234567890

1 INTRODUCTION

To display an understanding of the course and its content, I have created an application which takes from certain parts of the Computer Vision course and uses machine learning to detect and recognize handwritten text. This application will be broken down into steps and each step will be further analyzed in order to get a clear understanding of how the application works and what is necessary in order to improve on this program. The application is written in python and can be found in the final project main folder in the form of a Jupyter notebook. As I researched methods to implement the function of text recognition, I learned many new prebuilt functions that can easily be implemented to create text recognition. I explored these different options however opted for creating my own application with the use of OpenCV and Tensorflow.

2 PROGRAM PIPELINE

The text recognition application can be broken down into 5 different steps.

1. Image processing
2. Feature detection & cropping
3. Loading the dataset
4. Designing a machine learning model & fitting the data
5. Making predictions

2.1 Image processing

Image processing is the use of processing an image prior to using it. This can include, converting the image to grayscale or highlighting certain colours, applying blurring effects or several different filters depending on what features are required of the image. The image processing techniques that are used in the application are first to render the image in greyscale. All three colour channels are reduced to black and white intensities. This is useful for this scenario as I use detection of shapes in order to determine a character's identity rather than its colours. Furthermore, the dataset that I use is purely in black and white rendering colours meaningless to the machine learning model.

The following step taken to preprocess the image in order to give the model an easier time interpreting data is utilizing the OpenCV threshold() function. Thresholding in terms of image preprocessing is used to create a higher contrast between gradient colours which can help separate an image foreground from the background. The specific

type of thresholding that I used was binary thresholding. The effect of this method is to be able to convert a gradient image into two distinct colours. This method was chosen because a this creates strong edges and can be used to better detect characters from the background image.

The final step of the image preprocessing is to apply a bilateral filter. The formula for bilateral filter is found below.

$$BF[I]_p = \frac{1}{W_p} \sum_{q \epsilon S} G_s(||p - q||) G_S(|I_P - I_q|) I_q$$

This filter provides noise-reduction while also preserving edges which is essential for the model prediction. A noisy image can confuse image detection and produce unexpected results. By smoothing out the noise, the model can better interpret the intended text with more relevant data.

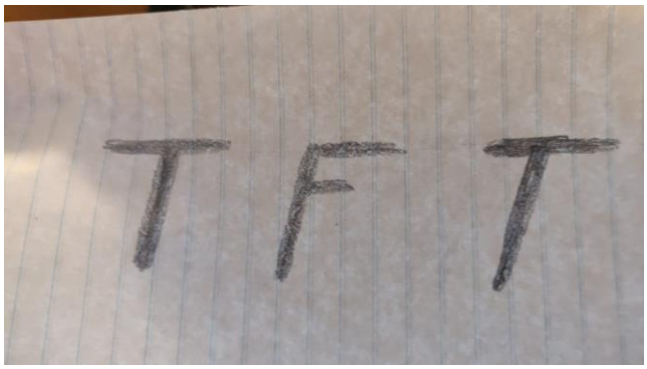

Figure 1: The letters TFT handwritten by myself on lined paper and captured using my phone.

Figure 2: Using the preprocessing methods mentioned in section 2.1 to provide the output image.

The image shown in figure 1 is a good example of a slightly noise picture. The aspects of figure 1 that are undesirable are the lines on the paper, the graininess of the picture quality, the slightly yellow tinge from the yellow lights and the uneven lighting. These aspects can easily disrupt the accuracy of the read so it is imperative that the image is preprocessed. Figure 2 shows the fully preprocessed image and the undesirable aspects have been cleanly removed.

2.2 Image Contouring & Cropping

The following step is simple with the help of OpenCV's built in library. What is necessary is being able to extract the points of value from this picture which in this case are the characters. OpenCV contains the findContours() function which can easily find the contours of images and isolate them. For this case, "CHAIN_APPROX_SIMPLE", a parameter of findContours() has been chosen which means it will be detecting contours based on corner points rather than the entire solid line of drawings. This can help reduce the complexity of the program. Once the contours have been identified, we will visit each contour in a "for in" style loop. Contours whose width and height do not match a sufficient size will be ignored as noise. We will then pad the contour and send this cropped image to our model for predictions.

2.3 Loading the dataset

In order to train the model, we require a lot of handwritten data to feed to the model. I selected from a handwritten text dataset from NIST. https://www.nist.gov/srd/nist-special-database-19. The dataset contains thousands of samples per lowercase and uppercase letter as well as numbers. To simplify my project, I restricted the learning to uppercase letters and numbers due to system limitations. Each folder was renamed to the character's ASCII number. This was used so that letters could also be interpreted to an equivalent integer. I loaded each picture individually, resized it from (128,128) to (64,64) and normalized each picture by dividing it by 255. The number 0 corresponds to 48 and Z to 90 and so each picture was assigned a value of their ASCII number minus 48 due to the integer range 0-47 being unused which helped to reduce the number of possible outputs. The dataset is read in order of letters so it must be shuffled to ensure that the order does not matter.

2.4 Designing a machine learning model

To construct the machine learning model, I used the Keras library from Tensorflow as it was used in Machine Learning 4050U and I had some pre-existing understanding of this system. The model is composed of several Conv2D, Dropout and Dense layers as well as Input, Reshape and Flatten layers. The input and reshape specifies the dimensions that the picture will be. The Conv2D is a machine learning algorithm which means 2D convolution which is commonly used for image detection due to its ability to detect patterns. The Dropout layer is used to prevent overfitting and sets some input units to 0. The Dense layer is a densely connected neural network layer which takes inputs and provides an output. The loss function I selected for this model is the SPARSE_CATEGORICAL_CROSSENTROPY. This loss function is extremely useful when the output of the data is a single integer value.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape (Reshape)            (None, 64, 64, 1)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 61, 61, 64)        1088
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 64)        65600
_________________________________________________________________
dropout (Dropout)            (None, 29, 29, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 26, 26, 128)       131200
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 128)       262272
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 9, 9, 256)         524544
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 3, 3, 256)         1048832
_________________________________________________________________
flatten (Flatten)            (None, 2304)              0
_________________________________________________________________
dropout_2 (Dropout)          (None, 2304)              0
_________________________________________________________________
dense (Dense)                (None, 43)                99115
_________________________________________________________________
dense_1 (Dense)              (None, 43)                1892
=================================================================
Total params: 2,134,543
Trainable params: 2,134,543
Non-trainable params: 0
```

Figure 3: Model Summary showing the layer breakdown and parameters.

```
In [4]: history = model.fit(x_train, y_train, epochs=5, validation_split=0.1).history

        Train on 4860 samples, validate on 540 samples
        Epoch 1/5
        4860/4860 [==============================] - 165s 34ms/sample - loss: 3.6922 - acc: 0.0265 - val_loss: 3.5750 - va
        l_acc: 0.0333
        Epoch 2/5
        4860/4860 [==============================] - 163s 34ms/sample - loss: 2.7057 - acc: 0.2471 - val_loss: 1.4655 - va
        l_acc: 0.5833
        Epoch 3/5
        4860/4860 [==============================] - 161s 33ms/sample - loss: 1.2786 - acc: 0.6360 - val_loss: 0.7708 - va
        l_acc: 0.7759
        Epoch 4/5
        4860/4860 [==============================] - 163s 34ms/sample - loss: 0.7760 - acc: 0.7689 - val_loss: 0.5891 - va
        l_acc: 0.8352
        Epoch 5/5
        4860/4860 [==============================] - 167s 34ms/sample - loss: 0.5870 - acc: 0.8167 - val_loss: 0.5077 - va
        l_acc: 0.8426
```

Figure 4: Fitting the training data to our model.

When fitting the data to the model, it is important not to overfit which means the model is getting overly accustomed to the training data. This can happen when the epochs, a pass over the entire dataset, are too high and can be detected by using a validation split. This validation split value of 0.1 means that 10% of the training data will be used to cross reference and test as we fit. Once the validation accuracy plateaus, this can be an indication that the model is no longer actually learning but rather overfitting to the dataset. This model fitting contained many parameters and took 819 seconds to complete. By reviewing figure 4, we can see that during the first pass, the accuracy is extremely low, being only 2.65% however with each passing epoch, the accuracy significantly rises and by the fifth and final epoch, the accuracy is at 81.67%.

2.5 Making predictions

The final step of the application is to finally make a prediction. The cropped image from step 2.2 is fed to the model to make a prediction. The value returned is the ASCII equivalent integer minus 48. This prediction is then displayed onto the original image in character form.

3 Experiments

The application ran smoothly on words that I drew myself in the paint.net application.

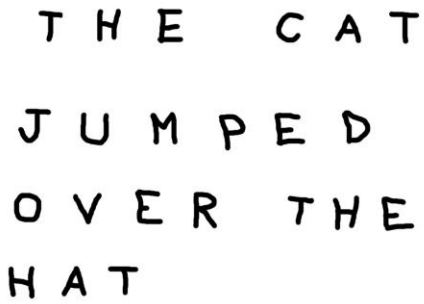

Figure 5: Image drawn by mouse on the paint.net program.

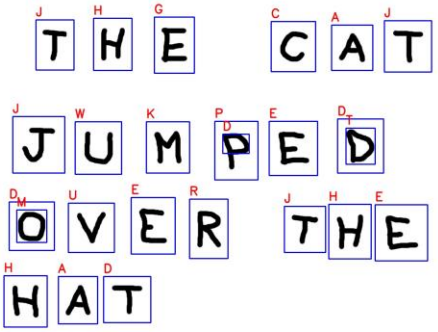

Figure 6: Analysis and prediction of figure 5 fed to the application.

As seen in figure 6, the application was successful in correctly predicting 13 characters. Each cropped image is bound in a blue rectangle to show the cropped image that was sent to the machine learning model with the prediction made by the application on top of this box. Certain predictions can be visually understood such as confusing the letter V for U and incorrectly guessing many of the Ts for Js. The image contouring also detected the inside of certain letters to be letters on their own for example the P was correctly detected as a P and the white space inside the P was again detected as an D. This could be removed by eliminating any completely overlapping letters.

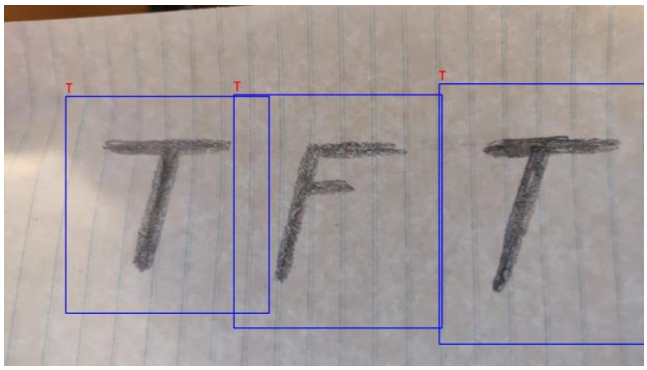

Figure 7: Application analysis on pencil handwritten letters

The application can correctly predict with some accuracy, a picture of handwritten text that is clearly drawn with bolded letters. This was my main goal in working on this project and I was extremely excited to see results.

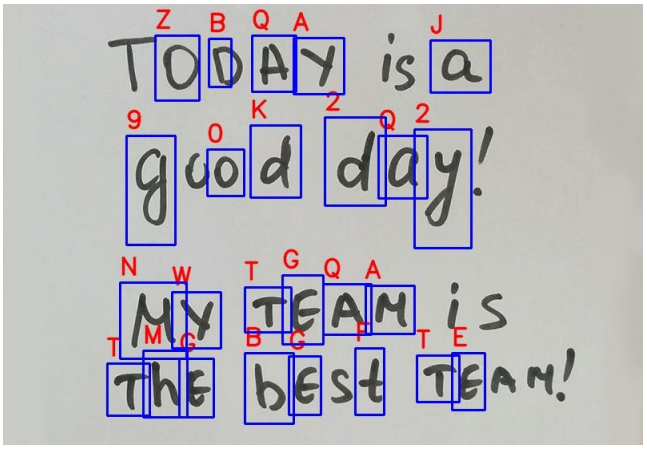

Figure 8: Prediction of my application on handwritten text via abtosoftware. (https://ocr-demo.abtosoftware.com/uploads/handwritten3.jpg)

A huge limitation that this program has is the fact that to reduce dataset size, lowercase letters were eliminated. Any lower-case letter that the program encounters will only be an inaccurate prediction based on the shape that most closely resembles it. Figure 8 shows the major flaws of the program as many characters were not contoured and this shows a much higher rate of inaccuracy when the program is presented with a slightly less clear handwriting.

4 Project challenges and limitations

The original intention of this project was to be able to be able to take a video and in real time be able to detect and recognize words and numbers and unfortunately due to many reasons, the application had to be severely scaled down. One major issue in executing this program was system limitations.

The dataset that I had contained thousands of characters for each letter and number however when testing it, I limited the training data to only read 150 of each character. When the training data was increased, the Jupyter Notebook kernel would die due to exceeding maximum allowed memory. This limitation may be because it is being run in a virtual machine and only allowed 4 gigabytes of ram. If the model had been allowed to use a greater training data size, it would surely be able to produce greater results. Similarly, the dataset images were of size 128x128 and they had to be scaled down to 64x64 in order to reduce the total number of parameters. This model is only trained on capital letters and numbers to reduce the output by 26 by removing lower case letters. These scaling decisions are at the cost of model accuracy but are necessary to ensure the application ran

smoothly on my system. Using these settings, my computer was being pushed to its limits and reached heat temperatures of 85 degrees Celcius while my computer is set to automatically shut down at 90 degrees in order to minimize damage to the system. In order to achieve better results in the future, the use of a virtual machine should be avoided in order to allocate as many resources as possible to executing the program.

Another limitation of the machine learning model is to have to make a prediction on individual characters rather than words and sentences. This could be implemented by grouping close letters together and merging the predictions however this was not possible due to time limitations working on this project alone. The original intention was to predict text in videos and this could be done by applying the model contouring, cropping and predictions on each frame of the image however I am certain this would not work smoothly in a real time video. Applying this idea to a video file however would not be too difficult as most of the application run time is spent on loading the dataset and training model.

5 Powerful text recognition tools

While researching text recognition, two powerful tools stood out among any others. The first tool is an Efficient and Accurate Scene Text detector (EAST) which as its name implies, is used for text detection and was designed by 7 individuals working at Megvii Technology Inc in Beijing, China. This application uses a neural network model which is fed information and trained to be able to detect text. Features in a picture are analyzed to determine if this image feature is indeed text or can be disregarded. The application has no issue with any rotated text and even works extremely well in very complex pictures. Two limitations that EAST has are extremely long lines of text possibly spanning the entire image as well as completely vertical text as they reported that only a very small portion of their dataset is comprised of vertical text.

The other library that really stands out in text recognition is Tesseract which is an Optical Character Recognition engine (OCR). Tesseract is available on multiple different platforms and is easy to implement in python using PyTesseract. Tesseract is capable of accurately recognizing text and the latest implementation of Tesseract 4 uses a Long Short Term Memory neural network (LSMT) to compute the text recognition. Tesseract was originally open sourced by HP in 2005 and is now being developed by Google since 2006. Online tutorials and example code can be found for both EAST and PyTesseract which can be used to quickly implement text recognition however the current application being presented relies only on OpenCV and Tensorflow.

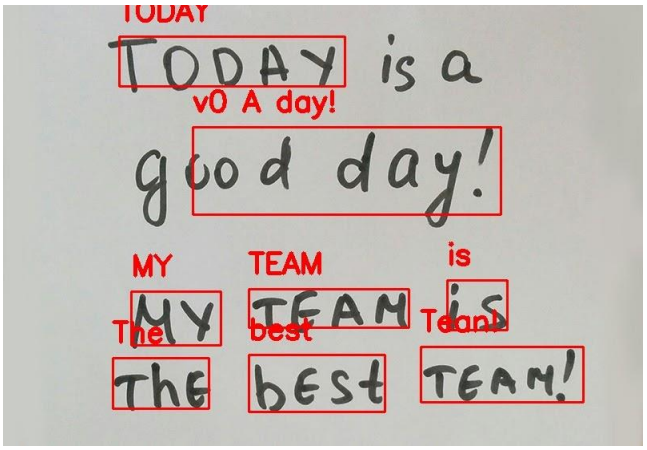

Figure 9: Using an implementation of EAST and PyTesseract online to make same analysis as figure 8

As seen from figure 9, the example implementation of EAST and PyTesseract is not perfect however the results are still very accurate. This example code can be found at https://www.pyimagesearch.com/2017/07/10/using-tesseract-ocr-python/. To use this example, a pb file of EAST must be downloaded and specified. This pb file contains a pretrained Tensorflow model so that a model does not have to be trained at every execution of the program.

6 Improvements

Just as society and technology are always advancing, small applications and research projects can also make changes. This application is far from perfect and many things can be improved in order to get a more polished program. Many limitations are mentioned in section 4 however much can be improved upon. A more accurate and robust text recognition system for noisy images can make a huge difference. Currently the images in the dataset are all upright however in real applications, images are often rotated, warped or even wrapped around physical objects. This can lead to much more difficult text for the computer to interpret. The current application only supports single images currently however feeding a video and breaking down the video frame by frame to provide video text recognition is a possibility as well as real time text recognition in live video feeds. Different fonts can also be evaluated in the dataset to provide a wider range that the

model can evaluate as well as different languages to internationalize the application.

7 Applications of text recognition

Improvements to a text recognition application is meaningful especially in an advancing society where the demand for this type of technology is quickly growing. Text recognition is already implemented today and can involve license plate recognition for traffic violations caught on camera or tracking toll routes. Other implementations include the use of google translate for OCR and real time translation when pointing a camera at text in English or other languages.

OCR and its current applications are extremely interesting however there are still improvements that can be made to seamlessly integrate it into everyday use. Using a scanner and scanning pages to have it automatically interpret an image of text into actual text is another form of OCR that we may take for granted.

While my program is specializing in handwritten text, it can still have numerous uses once its polished such as writing on a tablet in using a stylus and having it automatically convert to computer text though I'm sure that has already been implemented. Another use could be to visually analyze an individual's handwriting and determine who wrote some handwritten text based on this machine learning algorithm. The possible uses for OCR are vast and have yet to be fully explored.

REFERENCES

[1] X. Zhou *et al*., "EAST: An Efficient and Accurate Scene Text Detector," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Honolulu, HI, 2017, pp. 2642-2651. doi: 10.1109/CVPR.2017.283

[2] R. Smith, "An Overview of the Tesseract OCR Engine," *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Parana, 2007, pp. 629-633. doi: 10.1109/ICDAR.2007.4376991

[3] Antonius Herusutopo et al. "RECOGNITION DESIGN OF LICENSE PLATE AND CAR TYPE USING TESSERACT OCR AND EmguCV." *CommIT Journal* 6.2 (2012): 76–84. Web.

[4] Spellman R. Developing Best Practices for Machine Translation Using Google Translate and OCR Terminal. *Journal of Interlibrary Loan, Document Delivery & Electronic Reserves*. 2011;21(3):141-147. doi:10.1080/1072303X.2011.585570.

[5] Zhou, Winyu et al. "EAST: An Efficient and Accurate Scene Text Detector." 2017. https://arxiv.org/pdf/1704.03155.pdf