UiO : Department of Informatics
University of Oslo

UNIK4690 Project

Text Recognition

Akhsarbek Gozoev - akhsarbg
Sadegh Hoseinpoor - sadeghh
Key Lung Wong - keylw

Report

Contents

1 Introduction

This report is divided into 5 sections. In Section **??**, we introduce the Report and how the report is organized. Section **??** is where we describe the project and where we present our initial thoughts and approach on how solve our problem. In Section **??** we review our approach and present how we solved the problem, methods we used and changes which had to be performed. In Section **??** we present thoughts on the project and what could be done differently, difficulties that we faced and, a step by step tutorial on how to run our software. Section **??** includes third party material we have used.

refrence and recog-nition should maybe be di-vided?

2 Project description

The purpose of the software is to recognize text from any surface with uneven lighting. Hence this falls under the "Optical character recognition" (OCR) problem

As OCRs are still a challenging task even for companies like Google, ref. reader to Googles OCR translator application on smartphones; "Translate", drawbacks such as; difficulty finding all the text on the photo because of lighting, noise etc., therefore we will have to limit our software significantly.

2.1 Initial limitations

- English alphabet + numbers [0-9]
- Homogeneous background
- Skew free text
- Computer printed text

2.2 Project components

The group have come to the conclusion that the OCR software has 3 main components to it.

1. *Text segmentation*
    - Finding text on an image and returning the text segments
2. *Preprocessing*
    - Do preprocessing on the segmented text; rotation, symbol segmentation, etc.. (preprocessing from its definition, should be done first, however because of simplification we assume we manage to segment out text first.)
3. *Classification*
    - Classification of the symbols

Additionally there is one more very important component for this OCR software to work, ***labeled data***. Even though one might not need to code for this part, a good pool of labeled data is needed to be able to classify symbols. More on this later.
(4. *Data classification - Gathering labeled data to train a classification algorithm*)

2.3 Component description

This section we will describe our thoughts on how we plan to solve each component, in the form of algorithms, APIs, and datasets. Note that this is our initial thoughts, not necessary the solution we will end up implementing. Additionally, as we want to test proof-of-concept while we are coding, we will be making several simplifications. Relevant simplifications will be described further under each component.

2.3.1 Text Segmentation

Description
We where uncertain on how to do this part when we initial started the project. We ended up looking for a lot of different ways to do this part. We ended up trying 3 different approaches with mix result.

1. Simple image analysis technics, using Otsu thresholding and Morphology.
2. Stroke Width Transform(SWT) to detect text in natural images.
3. OpenCv implementation of Scene Text Detection.

Approach 1: Simple Image Analysis Technics
We was inspired by the this online blog Source[**_finding_????**]. We simplified the original approach where we only look at an image with text. The approach is then simply do an Otsu Thresholding and then morphological Closing to connect component. After that we used OpenCv FindContours method to find the contour of the different text regions. FindContours in OpenCv uses the algorithm proposed by Suzuki, S. and Abe, K. 1985[**suzuki_topological_????**].

Approach 2: Stroke Width Transform
We also found Stroke Width Transform to do Text Segmentation[**epshtein_stroke_2010**]. Similar to OpenCv Scene text detection, it looks for text in natural images. The approach of the Algorithm is to calculate the different stroke length of the different elements in the image. Based on the stroke length, evaluate if it is a text element or not. The concept was relative simple to grasp. The implementation, in the other hand, was quite difficult to implement. We will look at the full implementation later in.

Approach 3: OpenCv Scene Text Detection
OpenCV have it own Text Scene Detection. The approach of this algorithm is to detect text in scene using Classifier and Components Tree, propose by Lukás Neumann & Jiri Matas [**neumann_real-time_2012**].

2.3.2 Preprocessing

Description
Definition of preprocessing; the act of readying the data for further use, in our case for classification.

The specification of how the format of the image should be is not in our hands, as we will use datasets from third parties. We have to make the format and type of our input the same as the datasets. Other than the specifications we also need to take into account the spatial characteristics of the data, as we want the chance of the data correctly classified as high as possible. Such as background-foreground intensity, lighting, rotation, location of the character, etc..

Limitation - proof-of-concept
We will here limit our input data to only include grey-level images with rotation. As we progress further into the project we will include more aspects of a realistic image.

Challenges

- Rotated text
  - Hough transform is a valid solution to this problem. Finding the angle of the lines and then rotate the text segments accordingly.
- Line segmentation
  - Projection histograms allows us to find where the lines start and end, both vertically and horizontally.
- Character segmentation
  - We can use projection histogram here as well, however one line at a time. This way we can find out where each symbol starts and end.

2.3.3 Classification

Description

At this point because of available knowledge and the interest in Convolutional Neural network (CNN), we ended up trying to solve this part both with a CNN, and a Multilayer Perceptron (MLP or Deep neural network (DNN)). Illustration of each of the architectures are available, CNN Figure **??**, and the MLP network Figure **??**.

To avoid inventing the wheel again, we will use the TensorFlow API, the low level API will suffice for the MLP, and the high level API we will try for the CNN.

Deep Neural Network

Multilayer perceptron neural networks are relatively straight forward to code, however the challenging part is to decide on good hyper-parameters and to not overfit our network.

Research has shown that the choices of parameters can have huge effects on the error rate through empirical testing. As empirical testing has shown that some combinations of parameters are better than others, we will also use the same method to find decent values on several of the hyper-parameters. More on this below, where a short description of the hyper-parameters follow.

One obvious disadvantage we might face by using MLP is that slight spatial change on where in the image the characters are located, might lead to characters classified differently. This is because there is no spatial connections on a MLP.

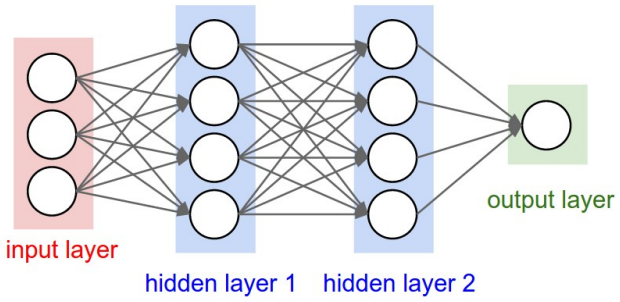


Figure 1: MLP Neural network Source

Hyper-parameters

- Number of hidden layers
  - Layers decide how well the software can define the decision borders. Hence increase in layers can have a positive effect, there are also cons with the amount of layers. The more layers, the greater the computational power needed to train the system. We will be using the empirical method to decide how many layers we need
- Number of nodes in each hidden layer
  - Nodes in each hidden layer has the same effect as the number of hidden layers, hence the same applies for this hyper-parameter.
- Activation functions
  - The activation function decides which combination of nodes, with their signals, are allowed to propagate through the network. Here we will be using the *rectified linear unit* (RELU) activation function. This is an activation function that allows propagation if the signal is positive, otherwise it will forward a zero. The reason for choosing this activation function is because this function handles the *vanishing gradient problem* better than sigmoid and a tanh activation functions. More on vanishing gradient problem under "optimization function".
- Loss function
  - The loss function describes how far off the predicted class of the character is from the real class. In our case since we have multiple classes and we are going to use *softmax regression* as the output layer, we also will be using the *cross-entropy loss function.*
- Optimization function
  - The backpropogation will train the weights by Gradient Decent Optimization. However as training with several thousand examples, and then optimizing the weights and run the training process, is too costly resource wise, we will have to implement the *mini-batch gradient decent optimizations.* Same principle as gradient decent optimization, but this way we will find the gradient decent for each batch. As long as these batches are randomly chosen, and the sizes are large enough, (we will be using 100 as batch size), these will represent the entire dataset well enough.
- Learning rate
  - Learning rate is a scalar that decides how large the steps towards the gradient minimum will be, for the weights. Choosing too small of a learning-rate we might risk not reaching the bottom of the graph, we also might get stuck in a local minimum. Choosing too large of a learning rate we might risk never settle down on a minimum.

    For the learning rate we will be using the empirical method too.

- Initialization of the weights and biases
  - Initialization of the weights also seems to be of importance, researchers have found out. This is obvious, as for example setting all the weights to zero, would of course lead to a network with very few active nodes.

    We will be using the initialization of zeros for the biases, not any apparent reason. Based on our research, it seems people have gotten decent results when using this initialization. For the weights we will be using a gaussian distribution, mean=0, standard deviation=1. Again this is also something that we have read should be a good initialization for the weights, no other reason.

- Number of epochs
  - Number of epochs are only relevant when we have a small number of dataset. When we have a small dataset we might want to run the software on the same dataset several times. This might result in overfitting the software to the dataset, therefore it is really important to be careful of the number of epochs, in cases with small datasets.

Convolutional Neural Network

Convolutional neural networks are especially good for image classification, because they take local spatial connections into account when they classify. This way it doesn't matter where in the image our object/character is it will be able to recognize it, same yields for rotation, as the CNN classifies based on local spatial connections it doesn't matter if the object is rotated. Hence the classification would be even more robust compared to the MLP.

The basic idea of a CNN is somewhat understood, but the algorithms and how to implement it is still not 100% grasped. Hence because of lack of knowledge, trying to solve the classification problem with a CNN will mostly be done by researching and trying to solve it as it unfolds.

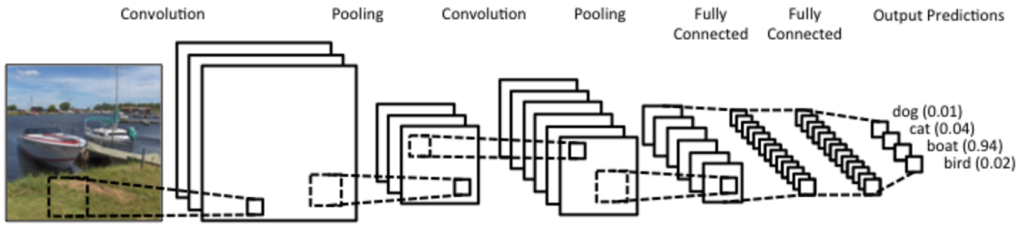


Figure 2: Convolutional Neural network Source

CNN - Basics

- Convolutional layer
  - This layer consists of several filters/kernels that are convolved over the image and based on how many filters one uses, the same amount of feature images

are made. The purpose of the filters are to grasp some spatial characteristics of the image, and make it available for further processing. The filters here correspond to the weights in the MLP, these are first initialized at random (or with a smart initialization method), then these will be updated as the network is trained. Some features that the filters might find could be; vertical or horizontal lines, they could also find more complex features like faces or animals at later layers.

In a CNN we can have arbitrary many convolutional layer and arbitrary many filters at each layer. However just like a regular MLP, increasing the number of layers and or filters; can allow the network to fit the training data arbitrarily well, unfortunately at the cost of processing time.

- Pooling
  - The pooling layer allows us to downsize the data. This makes it possible to start with data which are relatively big, and as the data propagates through the network we downsize it. As for the convolution layer the pooling layer can be used arbitrarily many times in a network.
- Fully connected layers (FC)
  - This layer works basically the same way as the hidden layers in an MLP. The pixels of the image is sent as data for each node and they propagate through these layers (usually 2 layers of FC are used) just as they would in an MLP.

2.3.4 Labeled Data

Description
Labeled data is needed because our classifiers need to be trained to understand the difference between the characters. This is usually done by training a classifier with a set of training data, labels are needed in our case, since it is a supervised machine learning algorithm we want to use. As the training data is used to train the software, we will need data to test our software as well, hence the need for test data. The test data is used to get a measure of what the error rate of our software is, based on the results we can then tune the hyper-parameters to get a better/smaller error rate. Lastly we will need validation dataset. This is an independent dataset that the software is not familiar with. The accuracy of the software on the validation set will then be a measure of how good the software can classify the characters.

Limitation - proof-of-concept
As we have limited us to the English alphabet and numbers ranging from [0-9], we will need labeled data for each of these 36 characters; training, test and validation sets. As the concept of classifying only numbers vs all 36 characters does not differ that much, we will first see if we can solve the OCR problem with just numbers. Therefore we only need a dataset containing numbers at first. Thereafter we will search for a dataset containing all the characters we need.

Dataset

MNIST

This is a dataset containing handwritten numbers [0-9]. It has a training set of 60.000 examples and a test set of 10.000 examples. (ref. reader to http://yann.lecun.com/exdb/mnist/).

3 Project review

3.1 Text segmentation

As mention in section **??** we tried 3 approach on Text Segmentation. We ended with our first approach since it was both simplest and gave most preferred result. We started simple with approach one, the result gave us confident to try out different approaches. We came across STW(Stroke Width Transform. It looks interesting and at the same time want to look at the possibility to detect text in natural images. We was finish some parts of the algorithm, but was not able to finish. The of the reason is because we instead focus on other parts of the project, also we realize that OpenCv have its own Text Detection.

3.1.1 Approach 1: Simple Image Analysis Technics

This approach is inspired by different sources, one of them is by this online blog Source[**_finding_????**]. We did a few modifications since we found it The approach here is simple:

1. **Find Edges/Outliners of the image.** Initial idea is to use Canny, but we found Morphological Gradient to perform better. It indicates the contrast of the edges, so we can get better differences in some natural images.(so long the text and background is close to black and white)

2. **Otsu Thresholding.** We need our image to be a binary image. We simply use OpenCV Otsu algorithm to achieve it.

3. **Morphological Closing.** Since we want line segments we used Morphological Closing with large horizontal filter to merge as many horizontal letter together as possible.

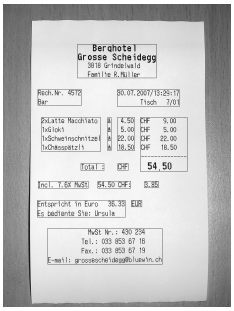
4. **Extract Regions** OpenCv FindContours was used to find the different text regions. We then exclude any regions that is smaller than a selected threshold. The different region are return as coordinate of the different rectangular boxes.

3.1.2 Approach 2: Stroke Width Transform

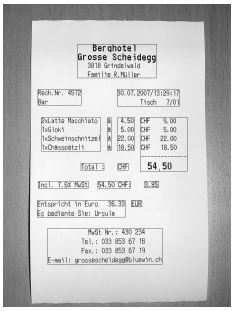
The second approach we tried Stroke Width Transform to do Text Segmentation, it was original propose by Epstein et al 2010 [**epshtein_stroke_2010**]. We was not able to finish this, but think we should mention it since we spend some time on it. The steps of Stroke Width Transform is as followed:

1. **Edge Detection and edge orientation(Done)** We need to have Edge image and orientation of the gradient image. Canny and Sobel was used in the original paper and other sources. This is simple since OpenCv have both Canny and Sobel implemented.

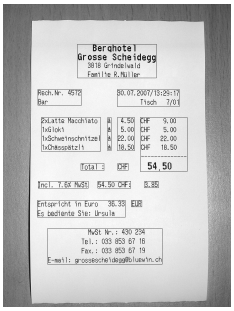
2. **Stroke Width Transform(Done)** Here we had to do more. We have to find a line from a starting point and the angle. We was able to implement this part, but was some uncertainties. It only work on black text with white background. That is because the orientation(Sobel filtering) are dependent on it. The paper talk about doing a second pass with inverse image, but we decided to ignore it, in order to come farther in the algorithm.



(a) Text Segmentation with Approach 1 - Morphology and FindContour



(b) Text Segmentation with Approach 2



(c) text

Figure 3: Text Segmentation with Approach 3

3. **Find Connected Component(Done)** In this point we are to connect component with the same Stride length together. We used One component at a time algorithm to find all the different components. We was able to finish it.

4. **Exude noise and find letters(not Done)** Since Stroke Width Transform tend to make a lot of noise. The obvious one is making single lines. This part are suppose to exclude this noise and at the same time exclude anything that is not a letter. The theory is, since letter and text all usually have the same stroke width, we can use this information do estimate what is letter and what is not. We was not able to finish this part.

5. **Find lines(not Done)** Was not able to get to this part, but ideal it will combine letters to a single line or words.

3.1.3 Approach 3: OpenCv Scene Text Detection

Since we decided to abandon Approach 2 and found out that OpenCv has a Scene Text Detection module, we though about testing it out. The problem was we found it out relative late and have a somewhat working Approach 1 already. Given that it did not give good initial result we decided to go back to approach 1, and optimize it more instead.

3.1.4 Conclusion

We ended up using the simplest approach, it is both easy to implement and faster that the other approaches. For our purpose, simple text segmentations, it gave us the best result as well. Figure **??** show different result of the different approaches.

3.2 Preprocessing

3.2.1 Find rotation

Images are not necessary rotated so that the text is vertical, and rotated correctly up side down. What we need to do is to find the rotation of the text and rotate it accordingly to be able to find lines and symbols with our "line and symbol finder approach". Bellow a review of our approaches follows.

Appraoch 1: Hough Transform

Hough transform is a well known algorithm to find lines, its approach is to see if it can alligne some threshold of pixels on one straight line.

We tried this approach, however we would end up with several lines on a text segment, both along the vertical and horizontal direction of the text. However the lines were not restricted to just these directions, we sometimes got lines where there where no apperant reason for it to be. Because of these uncertainties we tried another approach, minAreaRect method from OpenCV, see below.

Appraoch 2: OpenCV minAreaRect()
This approach is completly diffrent from the Hough Transform. Whereas Hough Transform tries to find lines, this approach uses convex hull to find the convex hull of the text segments, and then rotating calipers to find the minimum Area rectangle.

One important note here is that this method only works if the images sent as input is binary. Because we want the convex hull algorithm to find the convex hull of the text segment.

This method is really good at finding the angles, however it does not solve the problem of rotating the text correctly, Figure **??** illustrates this problem. It will only allow us to rotate it along one of the text segments edges. To rotate the text correctly from the result of cv.minAreaRect we used an original approach, see bellow "CNN - find correct rotation".

Figure 4: cv.minAreaRect cannot differentiate between 0°and 180°, and 90°and 270°

Appraoch 3: CNN - find correct rotation

3.2.2 Find line

3.2.3 Find Symbol/Letter Segmentation

Letter segmentation was similar to Text Segmentation. we added a few steps and removed the morphology part. The additional step was to fill holes in the image, example like 8 and O can give multiple wrong contour. cv2.floodFill was used to solve this problem.

1. Do threshold.
2. Inverse the image since we work on black text on white background.
3. Fill any holes, cv2.floodFills
4. FindContours