

UNIK4690 Project

Text Recognition

Akhsarbek Gozoev - akhsarbg

Sadegh Hoseinpoor - sadeghh

Key Lung Wong - keylw

Oppgave Rapport



Contents

1	Introduction	2
2	Project description	2
2.1	Initial limitations	2
2.2	Project components	2
2.3	Component description	3
2.3.1	Text Segmentation	3
2.3.2	Preprocessing	3
2.3.3	Classification	4
2.3.4	Labeled Data	8
3	Project review	8
3.1	Text segmentation	8
3.2	Preprocessing	8
3.3	Classification	8
4	Project conclusion	8
5	Recognition	8

1 Introduction

Skriv noe om hvordan reporten er satt sammen, slik at leser er forberedt på utformingen av rapporten. Hva rapporten skal handle om osv

.

.

.

.

.

2 Project description

The purpose of the software is to recognise 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; “Transalte”, 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
- 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 were uncertain on how to do this part when we initially 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 mixed results.

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

Approach 1: Simple Image Analysis Techniques

We were inspired by this online blog [Source\[2\]](#). We simplified the original approach where we only look at an image with text. The approach is then simply to do an Otsu Thresholding and then morphological Closing to connect components. 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[4].

Approach 2: OpenCV Scene Text Detection

OpenCV has its own Text Scene Detection. The approach of this algorithm is to detect text in scene using Classifier and Components Tree, proposed by Lukáš Neumann & Jiri Matas [3].

Approach 3: Stroke Width Transform

We also found Stroke Width Transform to do Text Segmentation[1]. 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 relatively simple to grasp. The implementation, on the other hand, was quite difficult to implement. We will look at the full implementation later on.

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-forground intensity, lighting, rotation, location of the character, etc..

Limitation - proof-of-concept

We will here limit our input data to only include greylevel 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 2, and the MLP network Figure 1.

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 hyperparameters 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 hyperparameters. More on this below, where a short description of the hyperparameters 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.

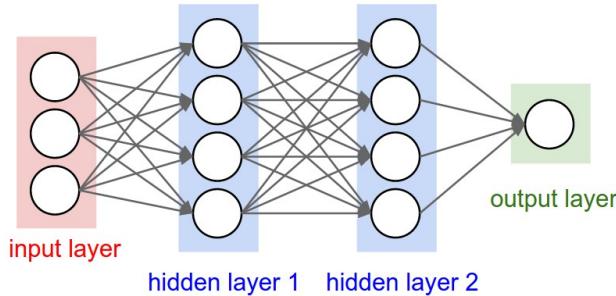


Figure 1: MLP Neural network [Source](#)

Hyperparameters

- 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 hyperparameter.
- 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 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 backpropagation 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 recognise 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.

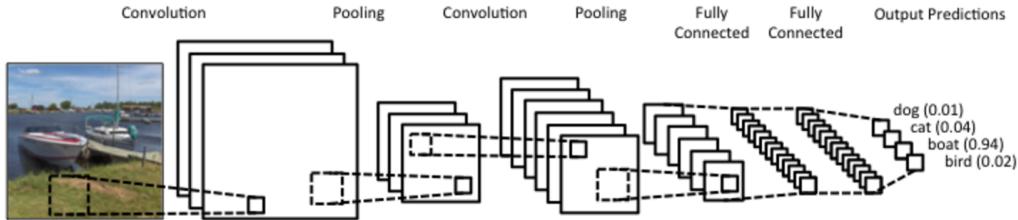


Figure 2: Convolutional Neural network [Source](#)

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.
- 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 convol 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 are 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.

Limitation - proof-of-concept

Description

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 hyperparameters 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

3.2 Preprocessing

3.3 Classification

4 Project conclusion

5 Recognition

Report

Week 1

19.04.18

- Feedback on project proposal
- Overview of project
 - simplification
 - binary image → numbers → straight text → Classify
- init; github - atom
- first test of character Segmentation

- Character Segmentation - Projection Histograms - OpenCV
 - By projecting the histogram of the binary image on the Y-axis, we can find where the sentences/lines of text appears. Following, a projection histogram on the X-axis can discover where the characters appear.

9 2 7 0 4 5 3 1 8 6

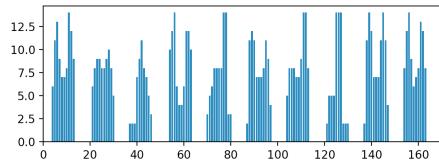


Figure 3: [0-9] segmented with projection histogram

- Classification - Perceptron neural network - TensorFlow
 - MNIST dataset - Dataset consisting of several thousand handwritten labeled numbers
 - * Numbers ranging from [0-9]
 - * Images are 28x28pixels
 - Hyperparameter tuning
 - * Activation function
 - * Number of hidden layers
 - * Nodes in hidden layers
 - * Cost function
 - * Optimization function
 - * Learning rate
 - Theoretic accuracy of the network with 2 hidden layers 98%
 - * Measured accuracy 97%

```
4690-p2018|Sadegh(master)$ p3 src/find_symbol.py
Model restored
Extracted text: 9220453189
```

Figure 4: First output with classification. input see Figure 3

- Rotation of text
 - Hough transform
 - `cv2.minAreaRect()`
- How to distinguish between upside-down, and verticle vs horisontal text segments
 - Classify in all 4 rotations, and choose the classification with highest avrage confidence
- Classification - Perceptron neural network - Error
 - Error rate too high, test-set accuracy 97%, validation set accuracy < 50%
 - CNN - TensorFlow Estimator API
 - * Challenging documentation; load/save models
 - Dataset - FNIST - Group contribution
 - * Dataset including several fonts
 - * English alphabet, and numbers [0-9]

- ——
— ——
— ——
- ——