Assignment 5 - TDT4173

Vegard Hassel - vegarhas@stud.ntnu.no, Piraveen Perinparajen - piraveep@stud.ntnu.no

Introduction

Task a

We're using *os* to access the filesystem and *random* to generate random numbers such that training images can be chosen at random. Additionally, *numpy* is used to handle arrays. *sklearn,* and *skimages* are used for machine learning algorithms. *matplotlib* is used to plot images and characters. *PIL* is used to read and process the images.

The underlying functionality is divided into several smaller sub-problems which are defined inside various methods. The functionality is abstracted using a main function called *run* making it easier to run while still maintaining customizability. For instance, the code is structured in such a way that the different parts of the task can be turned on/off using True/False statements. Feature engineering is on by default. This can be adjusted by changing SCALING or HOG variables. Classifiers (SVM and neural networks) and character detection can be used separately or in conjunction.

Feature Engineering

Task b

We've taken advantage of feature scaling and histogram of oriented gradient (HOG) for feature engineering.

Scaling is done using *scikit-learns* built-in Scaling function inside the preprocessing class. The purpose of this is to avoid cases where a feature may have a broad range of values which is likely to give an unfair advantage to one feature. Normalizing the data using scaling helps process the data such that each feature has a fair chance of being considered as an important feature [1].

The library *skimage* provides a built in HOG method. This is a feature engineering method which throws away feature vectors which are not useful, while keeping the good features. HOG works by dividing the image into small regions called cells and then creating a histogram of gradient directions. In the end, gradients are concatenated to create one big HOG feature vector which represents the whole image [2]. Using HOG before training increased our accuracy by 5 %.

Task c

We would have liked to create some great algorithms for cropping the images and removing incorrect or misleading features. The main reason to why we didn't do it was time limitations combined with complexity. We do not have any experience with such algorithms, and would have had to use a lot of experimentations and studying of the feature data in order to create a sufficiently good cropping.

We did also consider using PCA, but due to the fact that our two preprocessing methods provided sufficiently good results, we chose to not implement PCA this time around. Processing time was not an issue, so substantial dimensionality reduction was deemed to not be necessary.

Character Classification

Task d

By looking at the data, both K-NN and Decision trees were deemed to be inferior to other methods. The main reason for this is their problems with handling complex data and complex classification [3] [4]. The dataset seemed quite complex, and we felt like SVMs and neural networks were better fits. They are effective in high-dimensional spaces and when working with complex features, which is often the case when working in the domain of image classification.

Task e

SVM (SVC)
SVM is well suited to handle large feature spaces because the complexity does not depend on the dimensionality of the feature space. It works by trying to find the closest points between classes, called the “support vectors”. Once the closest points have been found it uses this information to draw a line which best separates the classes. [5] SVM was chosen because of its ability to handle a large set of features and to handle situations with many labels. For our case an extension of SMVs were used, namely Support Vector Classifiers.

Deep Neural Networks [NN]
Neural networks are built up using at least one input layer and one output layer. Each layer has a given number of neurons. Neurons are connected to other neurons using a weight which tells something about the significance of that particular connection. A deep neural network is a term used to describe a neural network with more than one hidden layer. As the data flows through each layer the abstraction layer gets higher and the deep neural network is able to extract more patterns from the data. [6] We decided to use NN because they’re suitable for classifying images and because they’re suitable to learn the complex features of our dataset.

Task f

We mainly use test accuracy to measure the performance of the models. This is done by counting the total number of correctly classified images, and dividing it by the total number of images attempted classified. This gives a percentage value for the accuracy. Test accuracy gives an estimation for how well-trained the classifier is and how well it is able to generalize. We also looked at training accuracy, which is expected to be a lot higher than test accuracy. Too high training accuracy, as for our NN, may be a sign of overfitting. This could have been mitigated by using a validation set to determine when the classifier is optimally trained and when overfitting starts to occur.

Both models performed reasonably well, getting a test accuracy of around 80%. This is far from a perfect score, but considering the dataset consists of 26 classes, 80% accuracy is quite impressive. Implementing more features and algorithms into the system can raise the accuracy even higher, and we'll say that both methods have great potential for improvement, given the appropriate time and knowledge.

SVM (SVC) gave slightly superior results compared to NN, since the preferable measure is test accuracy.

The appendix outlays outputs from running both SVM and NN to classify images. There are also outputs from predicting the labels of single images using SVM.

Most of the single images we sent through the predictor gave high accuracy and correct classification. Some did however not, mainly when they closely resembled other characters. Two examples can be seen in the appendix.

Task g

We would have liked to implement a deeper, more complex and custom built neural network. This is however something we felt was outside the scope of this project, and would therefore have to be done at a later point in time. In such a network we could have customized the various layers, activation functions and other similar patterns. A proper way of doing this would be by using Tensorflow directly, and not any of its high-level implementation.

We would also have liked to experiment with different custom ensembles and their input parameters. The results may not outperform our chosen methods, but it would provide great and valuable experience for us. We didn't implement ensembles due to the time and effort required to construct, tune and compare them. It is however something we may consider doing in a future project.

Character Detection

Task h

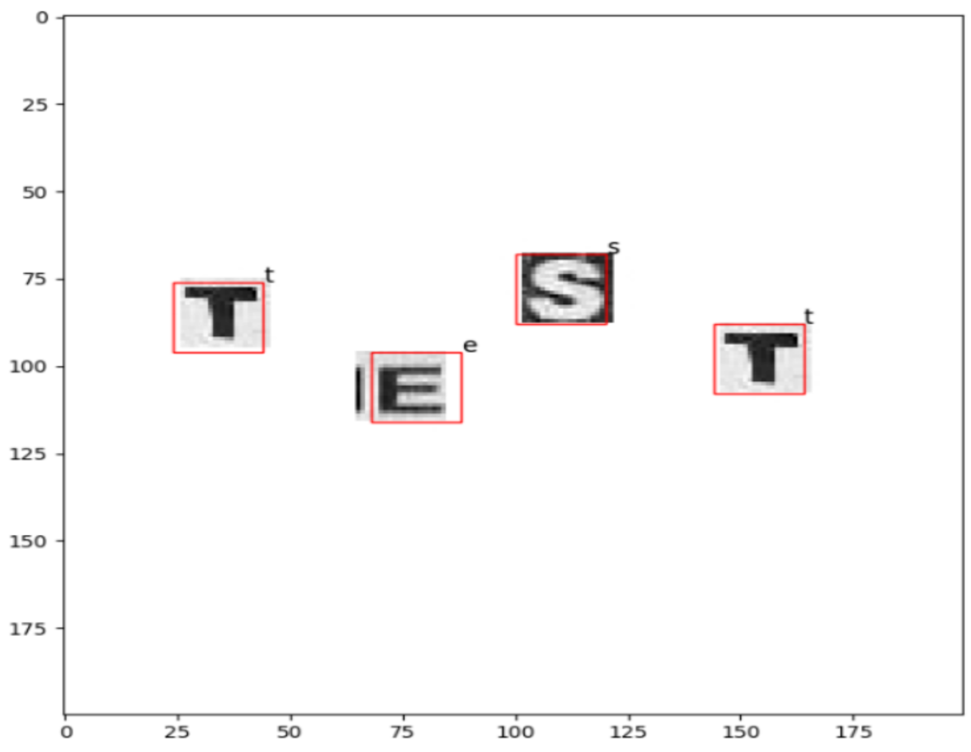


Fig 1: Detection for detection-1.jpg using SVM

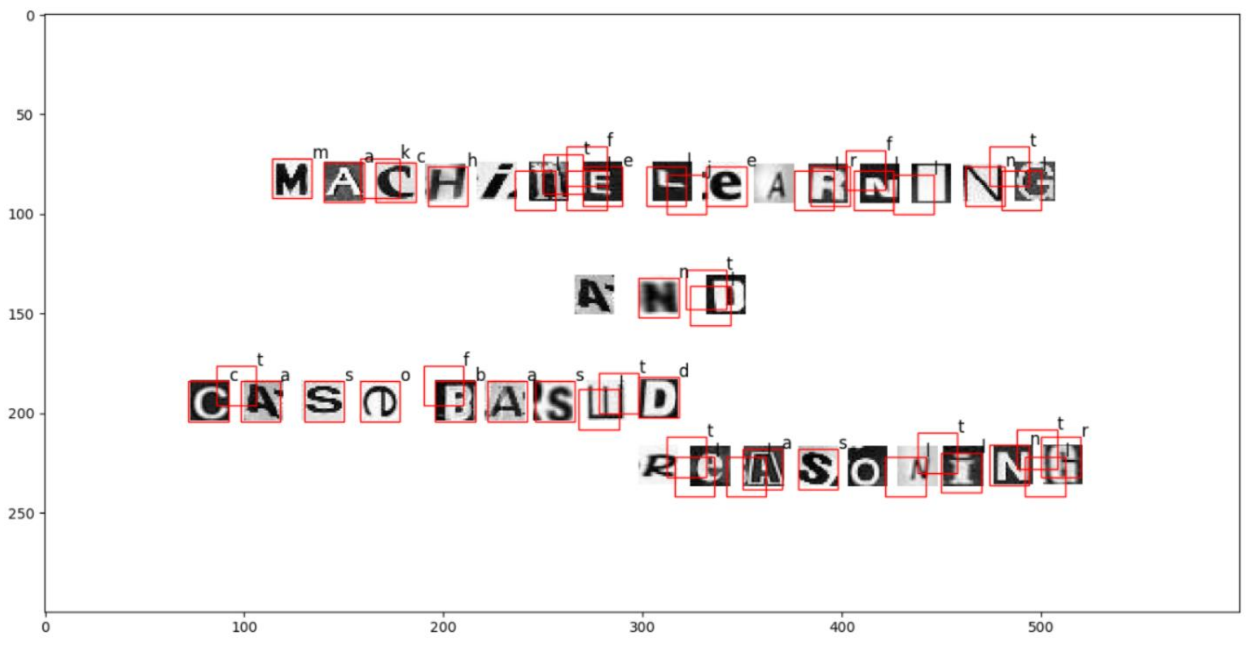


Fig 2: Detection for detection-2.jpg using SVM

Task i

Our understanding is that it performs reasonably good. It has no problem doing a correct detection when the characters are normal-sized and with a clear contrast between the character itself and the background, regardless of whether the background is black or white, which can be seen in figure 1. However, when the characters are shaped abnormally, the letters are thinner, rotated or the image is blurry, the system seems to struggle. We can see that despite these challenges, the system still manages to detect most of the letters in figure 2. The system does however have problems with misclassifying some characters, not finding every character and finding characters that are non-existent.

One example of a misclassification is the letter e in “case”. It is rotated a lot, and does therefore carry some resemblance to an o. Due to the limited size of the training data, and limited variations in rotations, the classifier is unable to make a correct prediction.

Another issue with the task is that it detects characters that does not exist. This can happen when the sliding window has found a straight line of black in a white area and therefore classifies it as a certain i. This happens if a character has black background and has a lot of white around the 20x20 pixels making up the character image. This situation can also occur with t’s, as can be seen in every word in figure 2. The red box is slightly elevated and between two characters, making it into a white t with black background.

Overall the task of character detection can be deemed challenging, and we are therefore considering our solution to be reasonably good.

Task j

In order to filter out several instances of the same detected character we implemented a method named *refine_chars.* This method looked at all windows which detected a character with certainty above a given threshold. For all windows that were within a specified distance from each other, it found the best prediction and deleted the others. This limited the number of classified character and made it so that they were not unreasonably close. By implementing this feature, we could have a lower threshold, so that more possible characters could be detected without too many appearing at the same place. This allowed the detector to find characters that were hard to classify, such as characters with different features than those trained upon. An example of this is the n in “and” from figure 2. It is very noisy and has unclear edges.

There are a lot of improvements that could be made to the detector. One in particular can be considered substantial in scaled-up cases, cropping. The detection-images provided for this project had a lot of white along the edges. Cropping away all this could reduce the image substantially, and therefore improve processing time. In our case, time was not too big of an issue, so we chose to not implement this feature.

Another option for improving the performance of the detector is to find and implement better preprocessing. This can for example be done by attempting to make all the characters black on white background, which would reduce the number of non-existing characters found between backgrounds. Another option is to implement problem specific rules, such as being more cautious if classifying the window as i or t. A third option for this exact problem is to try and find windows of 20x20 that has no rows or columns of perfect white, since practically all perfectly white pixels in this specific problem belong to the background between images.

Conclusion

Task k

The weakest component of our system is probably the character detection. This component is able to quite precisely detect characters that are far apart or not too difficult to separate. When the characters get too close, it detects characters consisting of parts of the original characters, as can be seen from our results. The character detection has substantial room for improvement, as discussed in task j.

The strongest component is likely to be the character classification. Both the SVM and the neural network give reasonably good results, although they could be improved further, both by tuning parameters and by expanding the training set. Compared to the other components, the character classification provides the best results compared to an ideal version of itself.

Feature engineering could have been improved quite a lot by applying more complex methods. It is therefore not the worst nor the best component of our system.

Task l

The project mostly went quite well. We managed to use available libraries to create an OCR-system with reasonable performance. The implementation went well, and we managed to find suitable parameters for all models. Further, our implementation had decent running time, and we therefore managed to do some testing and tuning of parameters.

One of the biggest hurdles we faced were tuning HOG to have good input parameters. This took some research and testing, but we eventually understood their intended meaning and from that managed to provide reasonable input values. Performance wise, character detection was the most challenging part. We had to tune several parameters and apply some additional restrictions in order to get a reasonably good result.

All in all, we had a positive experience with the project, and got a lot of knowledge and experience from creating the OCR-system.

References

[1] Scikit-learn developers (BSD License) (2017) 4.3. Preprocessing Data[Online]. Available: http://scikit-learn.org/stable/modules/preprocessing.html

[2] Scikit-learn developers (BSD License) (2017) Histogram of Oriented Gradients [Online]. Available: http://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html

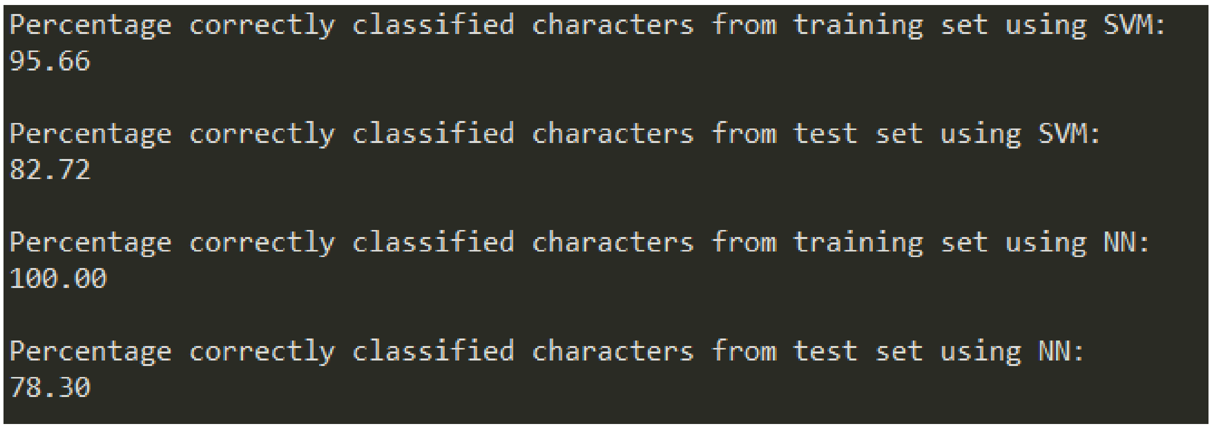
[3] Scikit-learn developers (BSD License) (2017) 1.6. Nearest Neighbors [Online]. Available: http://scikit-learn.org/stable/modules/neighbors.html

[4] Scikit-learn developers (BSD License) (2017) 1.10. Decision Trees [Online]. Available: http://scikit-learn.org/stable/modules/tree.html

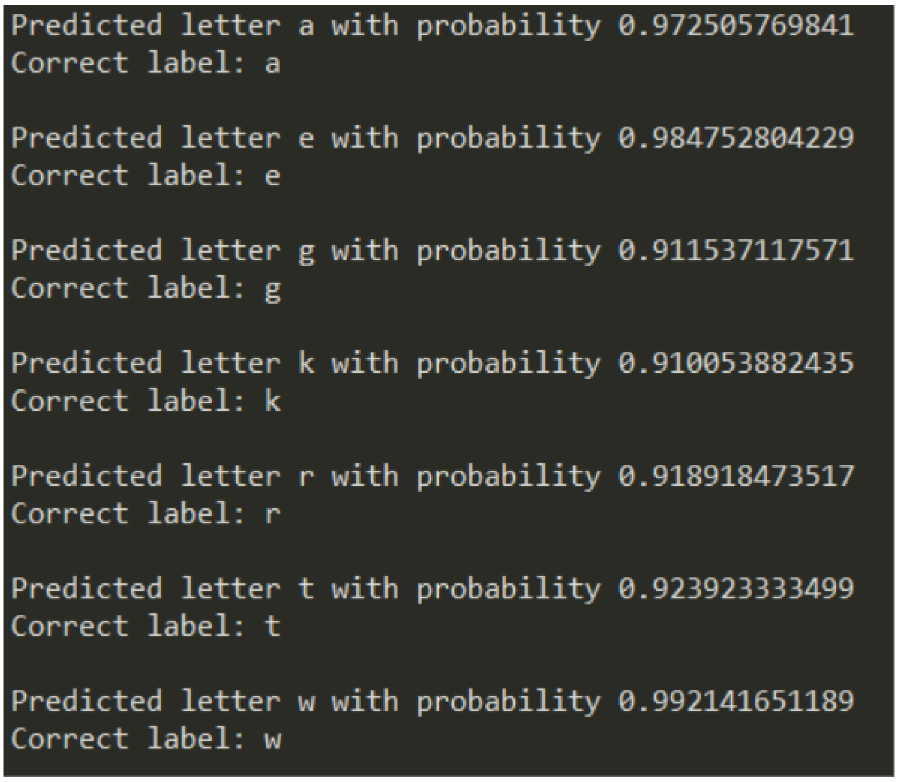
[5] Scikit-learn developers (BSD License) (2017) 1.4. Support Vector Machines [Online]. Available: http://scikit-learn.org/stable/modules/svm.html

[6] Scikit-learn developers (BSD License) (2017) 1.16. Neural Network models [Online]. Available: http://scikit-learn.org/stable/modules/neural_networks_supervised.html

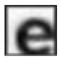
Appendix

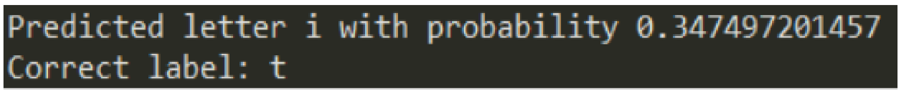


```
Percentage correctly classified characters from training set using SVM:
95.66

Percentage correctly classified characters from test set using SVM:
82.72

Percentage correctly classified characters from training set using NN:
100.00

Percentage correctly classified characters from test set using NN:
78.30
```

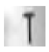


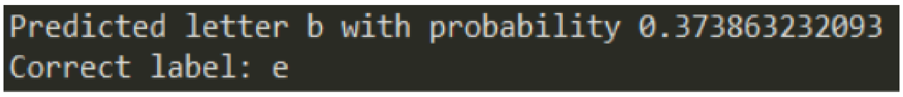
```
Predicted letter a with probability 0.972505769841
Correct label: a

Predicted letter e with probability 0.984752804229
Correct label: e

Predicted letter g with probability 0.911537117571
Correct label: g

Predicted letter k with probability 0.910053882435
Correct label: k

Predicted letter r with probability 0.918918473517
Correct label: r

Predicted letter t with probability 0.923923333499
Correct label: t

Predicted letter w with probability 0.992141651189
Correct label: w
```





```
Predicted letter i with probability 0.347497201457
Correct label: t
```





```
Predicted letter b with probability 0.373863232093
Correct label: e
```