Assignment 5: Optical Character Recognition

Even Wanvik (249359), Kristian Henriksen (501783)
TDT4173 Machine Learning and Case-based Reasoning

Objective: Gain experience with how to set up a pipeline for a non-trivial machine learning problem.

November 15, 2019

Optical Character Recognition (OCR) is the process of transforming images of characters into digital text. It is an old field of research in computer science and has been until recent years been a electro-mechanical task. New and effective methods in Machine Learning has emerged to renew the cumbersome process. This paper concerns the procedure of implementing two different machine learning methods, namely a Support Vector Machine (SVM) and a Convolutional Neural Network (CNN), to perform OCR with the Chars74k-lite dataset. Different feature extraction techniques will be applied to pre-process the data. The two methods will be evaluated and compared based upon their ability to classify and detect characters.

Keywords: OCR, SVM, CNN

1 INTRODUCTION

The ability to recognize characters retrieved from an optical scanner requires a reliable system that can tolerate shifting lighting conditions, motion blur, and twisted characters. An OCR system can be described using three main components: *feature engineering*, *classification* and *detection*.

The process of extracting useful features is critical in machine learning problems. Raw data gathered by a sensor often includes an abundance of information in addition to noise. Therefore, we often see that the first and maybe most time-consuming part of the process is to analyze the data and find the features that yield the best classification. Our dataset has 400 features (or pixels) per image, which provides an adequate number of possible configurations. For instance, if the model is to converge within a reasonable time window, the number of features needs to be reduced by normalization in a process known as feature scaling. The rest of the techniques are covered in Section 2.

With an appropriate number of features available, the task of assigning a class to each image will be less complicated. The dataset has 26 labels ($a - z$) that must be recognized inside a $20x20$ window of pixel intensities. To facilitate solving the multi-class problem, we first transform the alphabetical letters to numerical labels from 0 to 25. Then, we apply a Support Vector Machine and Convolutional Neural Network to perform the classification. The CNN is implemented using the Keras library, while the SVM is developed using the Scikit-learn package.

Finally, we can perform object detection capable of localizing and characterizing letters within a more or less ordinary image. Our algorithm detects a letter within a given window with a technique called *sliding window*. In short, the sliding window sweeps the full image with a smaller grid than the original image while it tries to classify letters within the frame. If the model's probability of prediction correct is higher than a set confidence threshold, the algorithm decides that a letter is localized at that position.

The system requires the packages and libraries that are listed below. **To run the code, type** `python3 cnn.py` **or** `python3 svm.py`.

- Python3
- Numpy
- OpenCV
- Scikit-learn
- Pillow
- TensorFlow
- Keras

2 FEATURE ENGINEERING

This section will describe the steps taken to pre-process the data and extract relevant features for our models. That includes which techniques that were chosen, how they work and why they were chosen. To conclude, a brief explanation of alternative techniques will be provided.

2.1 SVM

Before feeding the data into the Scikit-learn SVM algorithm, the data was flattened, changed into oriented gradient cells through HOG (Histogram of Oriented Gradient) and normalized (0-255 to 0-1 float). The HOG feature descriptor is used to differentiate between features through encoding interesting information into a numerical fingerprint made up of histogram gradients of some sort [1]. There are a lot of different feature descriptors, for instance, a SIFT descriptor, which computes oriented first order gradients, and the HOG descriptor, which on the other hand only computes a simple histogram of oriented gradients as the name implies. We chose to apply the latter, as a former is unnecessarily complex for our images and is better at describing the importance of a point.

The before and after transforming the images to a HOG representation is shown in Figure 1 and 2. This feature extraction step proved to be a huge improvement for the accuracy of the model. Without transforming the images to HOG, it was 19,34%. When transformed with the "default" HOG parameters (4 orientations, 2x2 cells and 1x1 block), it reached 72,39%, and

after tweaking the parameters (8 orientations, 4x4 cells and 1x1 block) it achieved 84,23%.

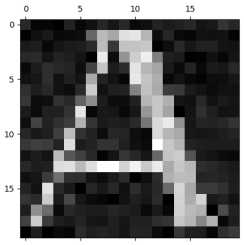

Figure 1: *Image before HOG.*

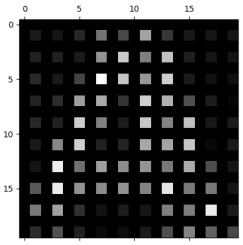

Figure 2: *Image after HOG.*

2.2 CNN

Pre-processing for convolutional networks is executed in layers. Prior to the classification process, the data is reshaped with several techniques including *convolutional, pooling* and *dropout* layers.

A convolutional layer systematically applies learned filters to the input images in order to create feature maps that summarize the presence of those features in the input [2]. The approach is proven to be very effective, and stacking them in deep models allows the layers to learn abstract features like shapes and objects. This is very intuitive for our data, as recognizing line segments and curves are directly applicable for characters. However, the convolutional layers record the position of the features in the input with high precision. This means that even small distortions of the feature in an input image will result in a totally different feature map. Such changes are prone to happen when the input image is noised, shifted or rotated.

To counteract the limitations of the convolutional layers, a subsequent Maximum pooling layer is added. Pooling helps to make the representation become approximately invariant to small translations of the input [2]. The layer will summarize the most activated presence of a feature to down-sample the feature map, like in Figure 3. This way, the small movements detected by the convolutional layers will result in a pooled feature map with the feature located in the same place.

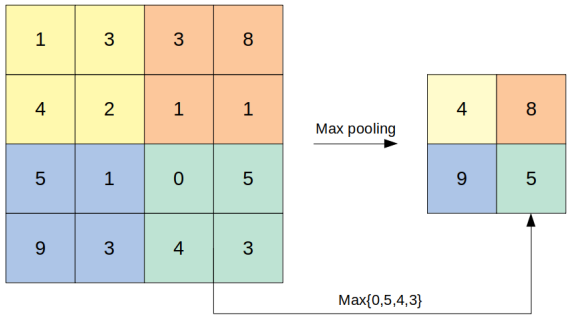

Figure 3: *Max Pooling process.*

For further processing, a dropout layer is included. The dropout technique randomly selects neurons that will be neglected for the duration of the current iteration. It is believed that the network this way will learn multiple independent internal representations [3]. Practically, the layer will drop out randomly selected features. The effect of dropout layers is a reduced chance of *overfitting* as the network becomes less sensitive to specific weights of neurons.

2.3 ALTERNATIVE TECHNIQUES

Examining the images of the letters, they seem to be greyscale images where the contour of the letters fits snuggly inside the image frame. Hence, cropping were not a necessary measure, as the letters fit perfectly inside the frame.

The noise layers that can be applied with Keras seems very interesting for this task. Adding noise to a model with a small training dataset can reduce the chance of overfitting. Chars74k-lite has 7112 images in total, yielding 273 images per character which is prior to splitting them in training and test sets. Augmenting the dataset with noised images could prove to make the model more accurate. We did try to supplement the existing dataset with augmented data using a data generator tool from Keras, however it seemed to directly transform the data and not simply append it to our dataset. After completing the full machine learning project did we find a way to extend our dataset with augmented images, which it is not used in our code.

However, a tool from Keras called Data Generators were instead applied, which has the same purpose: adding flipped, rotated and rescaled images.

For the SVM we tried to convert to greyscale and apply a blur to remove possible "salt and pepper" noise (misplaced black and white dots), but it only seemed to decrease the accuracy by a few tenths.

Another method that were considered for the SVM was a Principal Component Analysis (PCA). PCA reduces the number of dimensions by projecting them onto a lower-dimensional sub-space [4]. For instance it would be possible to produce some kind of projection of the a-z letters onto a 3d plane, by finding the three variables that contains most of the information about the image. It would be cool to maybe try to plot them on a 3d graph, but we deemed it unnecessary.

3 CHARACTER CLASSIFICATION

3.1 Initial thoughts

OCR involves interpreting images and inferring characters. With this in mind, Artificial Neural Networks (ANN's) became a natural choice the task. ANN's excel at performing object recognition tasks. Their popularity have risen dramatically in recent years, partially due to the high accuracy they achieve in tasks that involve computer vision. Especially effective for problems like OCR is the Convolutional Neural Networks (CNN's). The intuition of using a CNN aroused from its very nature: they preserve the spatial relationship between pixels in an image, and use this to extract surprisingly abstract features in their classification process. The networks are also very modular as a large variety of layers can be applied.

To compare with the CNN, we wanted a *simpler* and a less complex method that would provide adequate classification accuracy. We chose to use an SVM, which is widely recognized in classification problems, so there should not be too large of a gap between the models when classifying the letters. A CNN is a non-linear classifier because of its use of non-linear activation functions, whereas an SVM is a linear classifier because it uses kernels to project features onto a hyperplane where it is possible to separate the classes linearly.

3.2 Choice of classifiers

3.2.1 SVM

The Scikit implementation of a multiclass SVM uses a "one-vs-one" approach and might look cumbersome when it comes to computational time [1]. However, compared to something like k-NN, which must evaluate the dataset every time, we will have a trained model that can be applied easily. The main reason for choosing an SVM was because we wanted to compare it with the CNN algorithm. Before implementation, we hypothesized that the CNN would outperform the SVM, which would not necessarily be true for OCR.

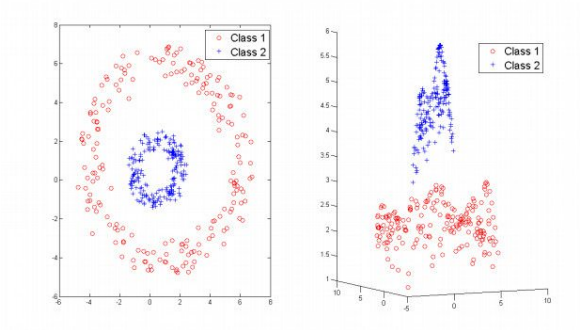

Figure 4: *The SVM finds a hyperplane (r.h.s.) that makes the classes linearly seperable.*

SVM is a supervised machine learning algorithm that can be employed for both classification and regression purposes. Basically, SVM finds a hyperplane that divides the dataset (Figure 4). Support vectors are the points nearest to the hyperplane. The hyperplane may be altered if any support vectors are removed from the dataset.

SVM uses different kernels to find the hyperplane. Although the model has auto-tuning, it still relies on an expert to tune the parameters to gain those extra accuracy points. In this experiment it was an easy choice to use the Radial Basis Function (RBF) kernel, also known as a Gaussian Kernel. The RBF kernel has two parameters γ (*gamma*) and 'C'. γ is the inverse of the radius of influence of the support vectors and C trades off correct classification of training examples against maximization of the decision function's margin (hyperplane) [4]. Support vectors are the data points samples selected by the model closest to the separation boundary between the classes on the hyperplane axis.

To find a optimal parameter selection for this kernel, we cross-validated different combinations that were pipelined into the "GridSearchCV" method provided by sklearn. A grid search performs hyper parameter tuning in order to find the best possible values for a given model. This can make a significant difference for a SVM model. The grid searched through the following parameters:

```
parameters = {
    'clf__gamma': (0.1,0.25,0.5,1,1.5),
    'clf__C':     (1,1.5,2,2.5,3,2.5),
}
```

and found the best parameter combination for the RBF kernel to be $C = 3$ and $\gamma = 1$.

3.2.2 CNN

A convolutional neural network is a class of deep neural networks that specializes in feature extraction and processing data with a grid-like topology, e.g. image data. A variety of filters are used to reveal characteristics of the input. In the first layers these characteristics can be lines or circles, while in deeper layers they can form abstract objects like faces. Compared to conventional deep neural networks, CNN's store fewer parameters which reduce the memory requirements and improve the statistical efficiency.

In its essence, a CNN has three types of layers: convolutional layers, pooling layers and dense layers [5]. An example showing these layers is depicted in Figure 5. The convolutional layers are compromised of filters and feature maps. Their task is to extract features from the input. A pooling layer follow a sequence of one or more convolutional layers, and are intended to consolidate the features learned in the features maps. In other words, it is meant to compress or generalize feature representations and thus reduce the chance of overfitting the model. Lastly, the dense layers are included at the end of the network, after feature extraction and consolidation, to create final non-linear combinations of features and make predictions.

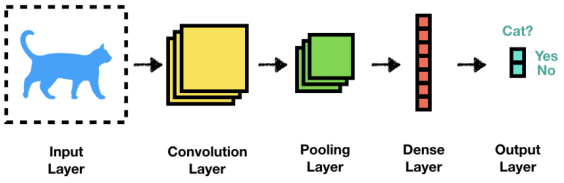

Figure 5: *Representation of the layered structure in a convolutional neural network.*

3.3 Evaluation of the classifiers

Performance of the CNN is measured when the trained model is evaluated with the test samples. In Keras, there is a function called `evaluate()` that outputs the test loss and test accuracy. The SVM accuracy was obtained by creating a function that cross-validates the predictions of the test-set with their real value. The common performance measure for the two models will therefore be the accuracy.

Figure 6 and 7 displays some samples from the classification process with SVM and CNN, respectively. Above each image,

both the true label and the prediction is given. Looking at nine random predictions made by the models, which has an accuracy of about 80-90%, at least one of the predictions should be a wrong classification. The prediction in the upper left corner of figure 6 is an example of a faulty prediction. If we were to speculate, it might be that the dataset contains more lower-case *t*'s rather than upper-cases (*T*), and the opposite is true for an *i*. On the other side, the image is very blurry, which might have been preventable by supplementing the training data with blurred copies of the original data. Another bad prediction can be seen by the prediction in the upper right and lower left corner of the CNN predictions in figure 7. The lower left "i" was predicted as a "l", which is understandable given that bottom of the next letter in the image will trick the CNN into thinking that these pixels are more likely connected.

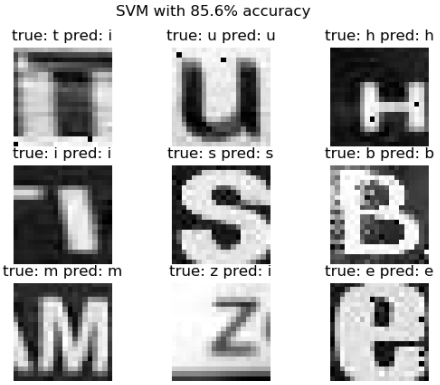

Figure 6: *Samples from the classification process with SVM.*

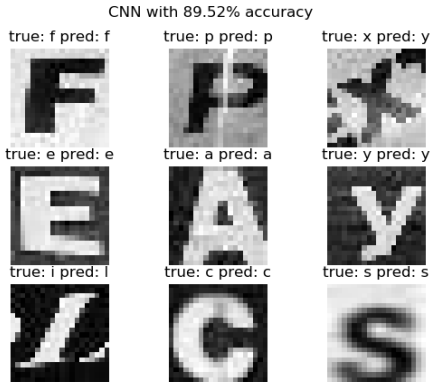

Figure 7: *Samples from the classification process with CNN.*

As it can be seen from the samples above, both models achieved a good accuracy. However, the overall highest accuracy was as expected accomplished by CNN with ~89%, as to ~85% for the SVM.

3.4 Alternative classifiers

It would be interesting to test the k-Nearest Neighbour (k-NN) algorithm for character classification. It would be fun to know the accuracy of such a straightforward and humble algorithm. The k-NN algorithm would probably work best if we performed some of the same pre-processing steps as for the SVM, such as converting to gray-scale (potentially binary) and extracting the HOG.

Another method we could have implemented for this project is ensemble methods, which use various learning algorithms to obtain a better prediction than the models would get individually. Aggregating a diverse set of models into one would make it more robust and able to handle new data much better. We could have, for instance, combined the result of both of our methods to produce a more favorable result.

4 CHARACTER DETECTION

For the last endeavor of this project, we are to detect and predict letters in an image with the help of object detection or object localization. We were able to do so with the help of our classifiers and a technique called *sliding window*. The idea behind a sliding window is to divide a larger image into smaller windows of fixed size and apply an image classifier to determine if a target is in the picture - in our case a letter. We implemented a simple adjustable sliding window for each of the algorithms, which returns an image with green squares around a *valid* classification while printing the letter.

4.1 Detection

The parameters for the following two detection images were kept the same: the step size was kept at 4, i.e. it moves 4 pixels per window, and the window size was 20x20.

4.1.1 detection-1.jpg

Both methods were capable of detecting all letters, but apparently struggled with the 'S' . The sliding windows whose classification had a confidence level higher than the probability threshold set in the algorithm are shown as green squares in Figure 8. To be able to classify the 'S', the confidence boundary of both algorithms were reduced to ~88%, which is why a second 'E' is classified (if you look closely).

Figure 8: *"T-E-S-T"*

4.1.2 detection-2.jpg

The second image (figure 9) was predicted by the CNN model. The performance was not satisfactory at all and there are a few things we could have done differently, which will be discussed in the evaluation of this section. There seemed to be a problem with the confidence of our Keras CNN models, as it was almost always above 99.9% sure about its classifications, which made it hard to decide on a confidence threshold and

partly explains all the redundant squares. The model had 65% accuracy in the classifications made in the image.

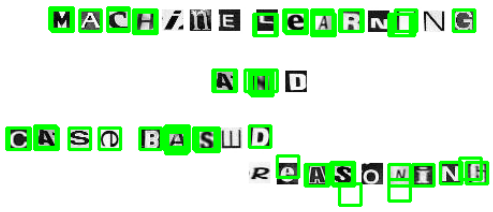

Figure 9: *"MACHINE LEARNING AND CASE BASED REASONING"*

4.2 Evaluation and potential improvements

Our detection algorithms performed well on the first image and not so good on the second, but it has a lot of potential. The first giveaway is that the required confidence level that were used could be lower in order to detect the more obscure letters. We could have reduced it if we implemented a restriction to the re-classification of the same letters in windows close to each other. A simple votation or average confidence between of the neighboring windows was considered, as well as removing outliers such as all-white images. An average of neighbors would also reduce the possibility of a faulty classification.

Another improvement we could have tried is to use different representations of the image and the sliding windows. Characters could have different orientations, and different angles and projections of the image/window should be evaluated and consider as possible classifications.

Every possible scale of the image could also be considered. One possible way to do this is to use "*Image Pyramids*", which is a multi-scale representation of an image. This would help our sliding window by making it able to classify characters that require smaller scales.

In the second detection image (figure 9) there was a lot more obscure characters and some with different orientations. As we mentioned, this could have been accounted for if we had extended our dataset with augmented representations of the existing samples.

5 CONCLUSION

With the existence of open source and an abundance of machine learning tools, you do not need a degree in computer science to perform machine learning. You should, however, know what type of model to apply to a given problem, and the features are more likely to make the model more capable of discriminating between different classes. This project introduced us to a lot of new models and methods of performing machine learning, which has led to in-depth investigations of documentation and tutorials in order to produce the most successful algorithm. When using existing and well-tested libraries like Keras and Sklearn, a large portion of the effort is to read and understand other's work. For instance, it was helpful to have a basic understanding of how a neural network works, but the process of implementing a CNN in this assignment gave a more profound understanding of how different parts of a neural network can be tweaked to alter the result, based on the area of application.

Much time was used trying to improve the accuracy of the classifiers in Section 3. With little knowledge of how the training process works, tendencies towards brute force had little effect on the results. Even though we trained and tested multiple generations of models with different hyperparameters, it seemed like most of the model's performance depended on correct preprocessing of the data. Applying and knowing what feature's and how it is fed to the ML model will be of much higher importance the next time we are to set up a pipeline for a machine learning problem.

REFERENCES

[1] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: Experiences from the scikit-learn project", in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[2] J. Brownlee, *A Gentle Introduction to Pooling Layers for Convolutional Neural Networks*, 2019. [Online]. Available: https://machinelearningmastery.com/pooling-layers-for-convolutional-neural-networks/ (visited on 11/09/2019).

[3] ——, *Dropout Regularization in Deep Learning Models With Keras*, 2016. [Online]. Available: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/ (visited on 11/09/2019).

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python", *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[5] J. Brownlee, *Crash Course in Convolutional Neural Networks for Machine Learning*, 2019. [Online]. Available: https://machinelearningmastery.com/crash-course-convolutional-neural-networks/ (visited on 11/11/2019).