Sigurd Nybakk Vang, sigurnv
Svein Ole M. Sevle, sosevle
Vemund Fredriksen, vemundfr

Assignment 5

Introduction

a)

The system is delivered as a Jupyter Notebook file (ex_5.ipynb), as well as a python file (utils.py). To load relevant data, and run relevant code, one simply must run the cells of the notebook in a sequential manner. What happens in each cell is made clear using comments as well as showing image results.

Data loading and pre-processing is done in the utils.py file, called from the notebook. It utilises the following libraries: PIL to load images, numpy for vector and matrix representation, matplotlib for visualization as well as skimage for pre-processing. The notebook uses the following libraries: numpy, matplotlib, utils (the one created by us) as well as sklearn for training machine learning models.

(To run the project the dataset folder must be filled with the dataset)

Feature engineering

b)

We tried utilising quite a few different feature engineering techniques. The ones that we kept in the solution (found in utils.py) are the following, which all improved the accuracy of our classifiers. Firstly, to increase the size of the dataset, all images are copied, their colours inverted, and then added to the dataset. This process doubles the size of the dataset. Secondly, the pixel values of all images are normalized. Normalization is especially important when dealing with domains with multiple parameters. If some of the parameters have numerical values that har bigger than another by the factors of 10 or 100 or even more, then normalizing all the parameters to fit within a fixed range helps the learning process because the parameters with the big numerical values wont be served more attention than the others in the start. Normalizing the pixel values to fit between 0 and 1 in this domain might help against the vanishing/exploding gradients problem. This is because if the aggregated values from the pixel values gets very big, the weights must be very small for the network to produce values between 0 and 1 in the last layer. Since CNN was decided against we decided to turn the images into HOG (Histogram of Gradients) – diagrams. The idea is that the HOGs will let the ANN get more information about the shapes in the image rather than just raw pixel values. The same argument goes for the KNN version. Finally, we flatten the images into one dimensional array, so that they are easier for our classifiers to work on.

c)

There were some techniques that we tried but decided to not keep in our final solution. To increase the quality of images, we initially removed noise from them. This, however, led to a decrease in accuracy for our classifiers. We hypothesise that this is since noise in images creates more variance in the training set. By removing noise, images are more similar, and

will expose the models to less variance. Noise also puts an intensive for the network to learn the true patterns of each character since it learns to ignore small invariances that in reality does not have an impact on the definition of the character. Additionally, we tried performing edge detection on the dataset. This both using various techniques, such as canny edge detection and sobel filters. Using these techniques had the effect of decreasing the efficiency of our classifiers.

Character classification

d)

Having looked at the dataset, we noted a few properties in it. It consisted of letters of different kinds. Some seemingly being handwritten, others digital. Characters typically have little to no rotation and are rather centred in images. The dataset does not differentiate between lower and uppercase letters.

These properties made the group believe that neural networks in general, and convolutional neural networks (CNN) specifically, would be suitable for the task. CNN uses convolutional layers, which are learned filters, to detect features in images, that are used for classification. Regular artificial neural networks (ANNs), achieve the same, by interconnecting all pixels in images with neurons of hidden layers, with learned weights. Typically, CNNs are characterized as being the most suitable machine learning method for image classification[i].

e)

We decided, however, against using CNNs. This was mainly due to not having a great deal of experience in developing them. We did, however, decide to implement an ANN. As previously mentioned, the feature engineering included turning the raw pixel values into HOGs that can be seen as a kind of feature extraction like convolutional layers would do. We also decided to use a KNN classifier. KNN is a lazy learner algorithm that given a training set, and an instance, will find the K instances of the dataset most closely resembling the given instance. Classification occurs when finding the majority class amongst the K neighbours. The reason we felt KNN could be a reasonable classifier to use is twofold. Firstly, the spatial structure of images of given characters are quite similar, meaning that classifying an image by the most similar example images is meaningful. Additionally, as the images in our dataset are same sized vectors consisting of floats, the dataset is quite suited for using the similarity measures often used in KNN, such as Euclidian distance or Manhattan distance.

f)

Both our models were trained using the sklearn library. We evaluated them in the following way. Using methods included in the sklearn library we split our dataset into a training set, and a test set. The split was 80% training set and 20% test set, where the images were shuffled around randomly before applying the split. The models were trained on the training set. After training they were given the test set to classify. The model's mean accuracy over the test set was used as the model's accuracy measure. The KNN classifier had an accuracy of around 75% whilst the ANN had an accuracy of around 89%.

Sigurd Nybakk Vang, sigurnv
Svein Ole M. Sevle, sosevle
Vemund Fredriksen, vemundfr

In [55]:
```
#HERE ARE SOME EXAMPLES OF THE SYSTEM PERFORMING CLASSIFICATION
#I choose at random five to classify. index 5, 1000, 5678, 7890, 1356
labels = [5, 1000, 5678, 7890, 1356]
for l in labels:
    predicted_label = nn.predict(np.array([X[l]]))
    predicted_label = get_label(predicted_label[0])
    actual_label = get_label(y[l])
    print("actual letter:", actual_label, "predicted letter:", predicted_label )
```

```
actual letter: a predicted letter: a
actual letter: a predicted letter: a
actual letter: i predicted letter: i
actual letter: n predicted letter: n
actual letter: a predicted letter: a
```

g)

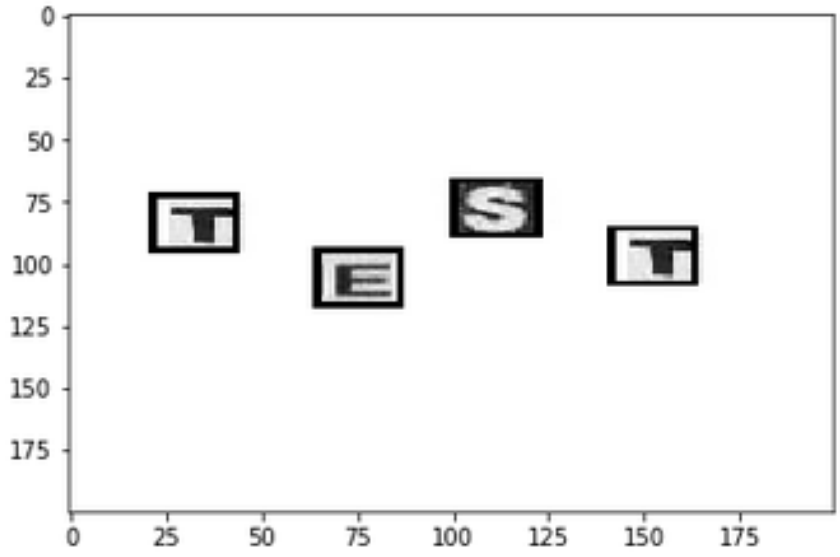
As mentioned in task e, we wanted to implement CNNs, but decided against it. As of now the HOG process is done in the classifying process rather than on the image itself which leads to bad performance in detection which will be seen in the character detection problem.

Another thing that was never tried is to create Histograms of Histograms. The way HOGs are done now results in just as many gradients as there was pixels. We could have researched creating histograms that was made up of other nearby histograms by clustering them. This could potentially let the ANN and KNN see even more of the overall context of each shape, and additionally it would make the width of the first layer in the network smaller. It was not researched however, so this is just a mere suspicion.

Character Detection

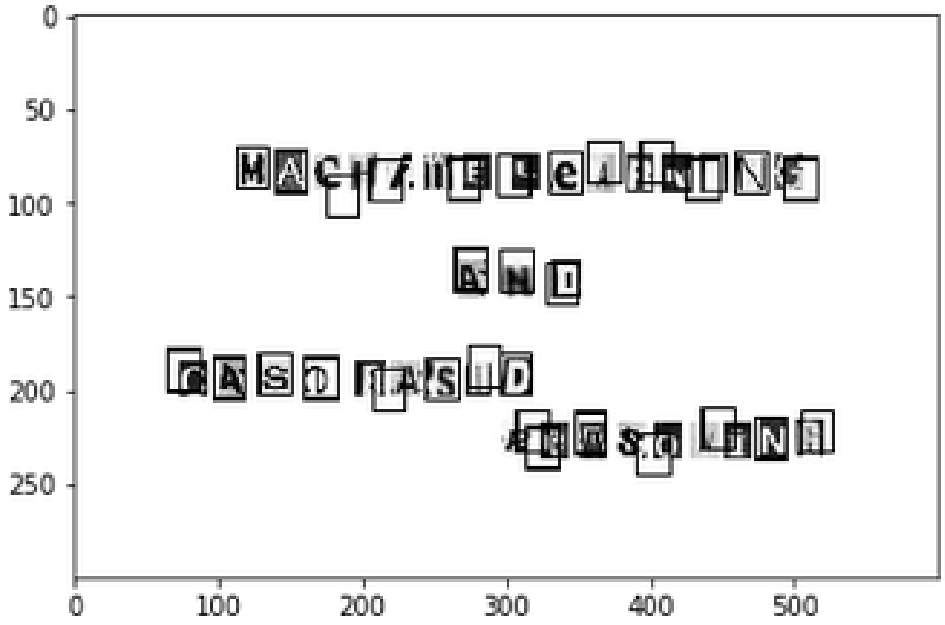
h)

detection-1.jpg:



detection-2.jpg:

Sigurd Nybakk Vang, sigurnv
Svein Ole M. Sevle, sosevle
Vemund Fredriksen, vemundfr



i)

On the first image, detection-1.jpg, it looks like it performs good. For some of the letters the detection box is not optimally placed, but the letter is contained inside the box and therefore it is good enough.

On the second image, detection-2.jpg, it performs good for many letters, but there are some letters where the bounding box is not centred around the letters. There is also some letter that is not detected, but most of the letters are detected and has a bounding box around it fits ok.

j)

The detection algorithm utilizes a sliding window approach. In order to get decent results from this technique an implementation of non-maximum-suppression was needed to select the bounding box that best fit the characters. To elaborate; when sliding the window past a character, the classifier might recognize the same character multiple times. Instead of having 10-20 bounding boxes on each character, we want 1 bounding box, the one that has the highest confidence score. The non-maximum-suppression works by checking a detection against the previously detected objects and if the new detection overlaps with any previous the confidences are compared. The detection with the highest confidence gets selected and the other gets discarded.

Conclusion

k) We think the strongest component of the OCR system was the feature engineering component. We think this was the strongest component because when we modified this component then both the accuracy of the ANN improved much.

We think the weakest component of the OCR system was the character detection, given that there is plenty of information in the image and the letters are close together. We think this was the weakest component because faulty detections of letter could propagate because of the non-maximum-suppression implementation. We believe that improving this part of the solution would make the biggest difference in regards of performance.

l)

All in all, the project went quite good, however the object detection part could have been better. We tested different method and discovered which method had a positive effect on the result and which did not and then remove it if it had a bad effect on the result.

We learned how important it is to look at documentation for libraries that we used. We also gained first hand experience with feature detection methods and the object detection method sliding window.

[i] https://medium.com/zylapp/review-of-deep-learning-algorithms-for-image-classification-5fdbca4a05e2