INM460: Computer Vision

# Facial and Optical Character recognition

*Vaida Gulbinskaite*

---

## 1. About the project

In this project I have created a function that identifies faces of our classmates, using feature extraction methods and assigns them with the label (which, in most cases, was the number classmates were holding). This was achieved using 3 types of classifiers (Support Vector Machine (SVM), Feed Forward Neural Network (NN) and AlexNet) and 2 types of feature extraction techniques (HOG and SURF). The types of feature extraction techniques were used in combination with SVM and NN (e.g. NN+HOG, NN+SURF), AlexNet did not need a separate feature Extraction to be performed. I have also created a function that identifies digits from pictures (.jpg format) and videos (.mov format). For this I have created an SVM classifier with HOG features. Both functions were created using Matlab R2018a.

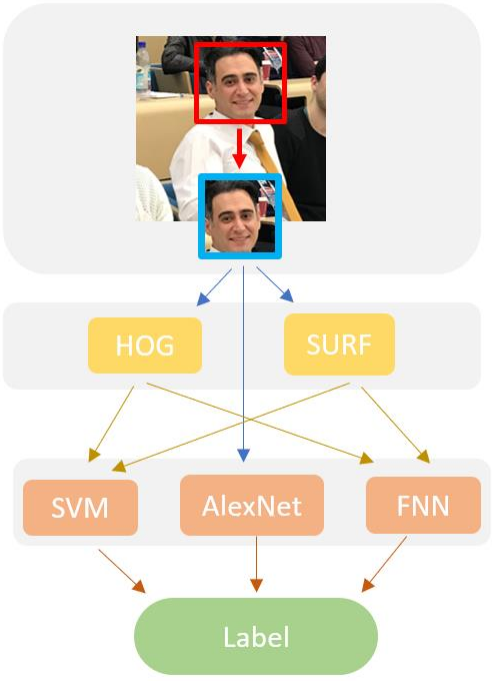


***Figure 1:*** *Example face recognition workflow (top to bottom): Extracting face from an unseen picture, extracting features from the detected face, using the extracted features to predict the label, showing the predicted label.*

## 2. How to run functions

### 2.1. RecogniseFace

This function takes in 3 arguments: 1. Image path; 2. Feature type; 3. Classifier name (figure 2). It returns a N x 3 matrix that contains predicted label, x coordinate and y coordinate for every face, detected in the picture. It returns a label of 0 if faces, found in the picture, did not match any faces, used to train the classifiers.

| Image | Feature | Classifier |
|---|---|---|
| Image path | SURF | SVM |
| Image path | HOG | SVM |
| Image path | SURF | NN |
| Image path | HOG | NN |
| Image path | None | AlexNet |

***Figure 2:*** *All possible combinations of function calls, presented in a table*

Before running, please keep the following in the same folder: SVMHog.mat, HOGNNet.mat, SVM_SURF.mat, SURF_Net.mat, AlexNet.mat, RecogniseFace.m.

### 2.2. detectNum

This function takes file path as an argument and returns a matrix N x 1 that contains each number, seen in the picture (individual numbers are recognised, which means if a person is holding 108, the function should return 1,0,8). This function works for images (.jpg) and videos (.mov). This function uses trained SVM classifier with HOG features. Please keep the following files in the same folder: SVMHOG_OCR.mat, detectNum.m.

# 3. Face recognition

### 3.1. Data collection

Undergraduate and Postgraduate Computer vision students were photographed and filmed in order to create a database. Students were asked to express a series of emotions (e.g. sadness, happiness) as well as photographed in different angles so the dataset is more varied. To assign the labels, students were asked to hold a number. There were 53 students who participated, which in turn created 53 classes for the classifier to recognise. Some participants held the same number by mistake, therefore their labels have been re-assigned (e.g. Student labelled 161 held a number 61)

### 3.2. Face Extraction

FaceDB.m was created in order to extract faces from each picture of the student. Viola - Jones cascade object detector (function: CascadeObjectDetector) was used detect faces from each picture. This algorithm detects people's faces, facial features (mouth, nose, eyes) and upper body. Using this algorithm with default settings (threshold f of 4) found quite a few false positives as buttons, shadows, or random clothing patterns were recognised as faces. For that reason, I have increased the threshold to 8 which helped to eliminate some of the false positives. I have recognised that student 14 was wearing clothing with faces on it. To avoid these being extracted as faces, I have defined a minimum square around the face size of 50x50 pixels.

After faces were detected, its coordinates were used to create a bounding box, which then was used to crop each face from the picture. Each cropped face was re-sized to 227x227 (the format, needed for AlexNet) pixel image and saved in a new file directory. Pictures of faces were then manually sorted into folders, named correspondingly with their assigned labels. Any non-faces were deleted.

Lastly, I have realised that some classes (e.g.class 63) did not have enough images for accurate classification. For that reason, I have tried extracting faces from videos (VideoDB.m), however the code was very slow to process a vast number of .mov files and CascadeObjectDetector [1] was not accurate; as instead of faces it was detecting numbers and background images. I have ultimately decided not to use face detection from videos and instead manually augmented existing pictures by rotating them left, right, and upside down. In total, 1,647 images were generated and used for the classification.

## 3.3. Feature extraction

### 3.3.1. Histogram of Oriented Gradients (HOG) features

Defined in 2005 by Dalal and Tiggs [2], HOG features are widely used in image processing and computer vision, HOG features count the occurrence of gradient (change in image intensity or colour) orientation in localized portions of an image to identify objects in pictures [2]. To calculate HOG features image is divided into radial or rectangle cells. The histogram channels are spread evenly, over either signed or unsigned angles. Each cell is evaluated to create a local one - dimensional histogram of gradient directions x number of pixels in a cell [3]. The gradient strengths then must be locally normalized by grouping cells together into larger, connected blocks. Hog descriptor then becomes a vector of components of the normalized histograms from all the blocks, created in the previous step. There is a possibility that blocks can overlap, which means that each cell can contribute to the final descriptor more than once [1].

***Figure 3:*** *Example of HOG features, presented over the original image.*

For this project, function called extractHOGFeatures [4] was used. I have decided to keep the default parameters, as these were proven to perform best in Dalal and Tiggs paper [2].

### 3.3.2. Speeded Up Robust Features (SURF)

Created in 2008, SURF a novel feature extraction technique, used in object recognition, image registration, classification and even 3D reconstruction. The concept was inspired by another feature extraction technique called SIFT (Scale – Invariant Feature Transformation), however is a faster and a more robust feature extraction technique. It includes scale, translation, lighting, contrast and rotation invariance [6].

SURF uses integral images to speed up the feature extraction process (as these types of images allow fast computation of box type convolution). The detector is based on Hessian Matrix, which is

used for both scale and location, due to its exceptional performance and accuracy. Unlike SIFT, SURF uses Box filter to approximate Laplacian of Gaussian. Some big advantages of this kind of approximation is that, in combination with integral images, convolution with box filter can be calculated swiftly. It can also be done in parallel for different scales [7].

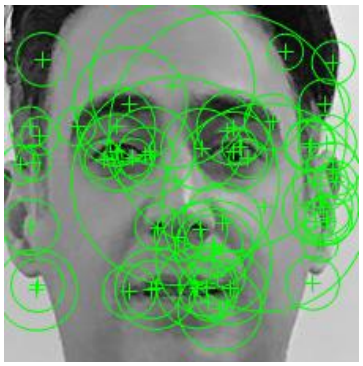

***Figure 4:*** *Example of SURF features, presented over the original image.*

In this project, bagOfFeatures function with default parameters was used to extract SURF features from an image [8]. Inspired by the Bag of words concept in document classification, bag of features create a dictionary of features, extracted using SURF feature detection. The function returns a bag of features object, with a default size of 500 objects per input picture.

## 3.4. Classification

### 3.4.1. Training classifiers

There were coloured images used in the training of the classifiers using HOG features and no feature extraction (AlexNet). Classifiers that used SURF features had images converted to grayscale automatically, using bagOfFeatures function. Each classifier was trained using 70% of the face dataset and tested using the remaining 30% on the image dataset. Unlike SVM, AlexNet and NN used testing, training and validation data.

### 3.4.2. Support Vector Machine (SVM)

SVM aims to separate the data by constructing a hyperplane (or a number of hyperplanes) in an infinite - dimensional space (example shown in figure 4). Traditionally, SVM is a two - class classifier. However, it can be adapted to predict multiple classes as it takes a 'one versus rest' or 'one versus one' approach. One versus one approach was implemented in this paper, which creates multiple classifiers (one classifier per each class pair) to evaluate classes one by one (e.g. 1 vs 2, 1 vs 3, 1 vs 4… etc) and selects the class, that is selected by the most classifiers. While it might sound counter intuitive to build that many classifiers (in our case - 1,378!) it may actually reduce training time spent since the training set for each classifier is much smaller. [9]

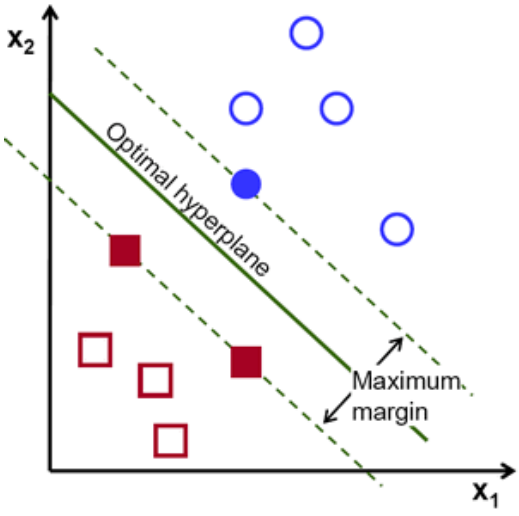


***Figure 4:*** *Example of Support Vector Machine classification [12]*

In this project, two built - in functions were used: 1. trainImageCategoryClassifier [10] - dealt with SURF features; 2. fitcecoc [11] - dealt with HOG features. Both classifiers showed acceptable accuracy when tested however, SVM with HOG features slightly out-performed SVM with SURF features (SVM with HOG features = 94.3% SVM with SURF features = 89.2%).

### 3.4.3. Feed – Forward Neural Network (NN)

An artificial neural network with a linear working pattern is constructed of 3 types of layers: 1. Input layer; 2. Hidden layer; 3. Output layer. It uses series of weights, biases and activation functions (in our case - sigmoid) and a combination of learning parameters (e.g. learning rate) to process the input and predict its class. The number of hidden layers as well as the number of neurons in these layers can be defined, depending on the complexity of the task.

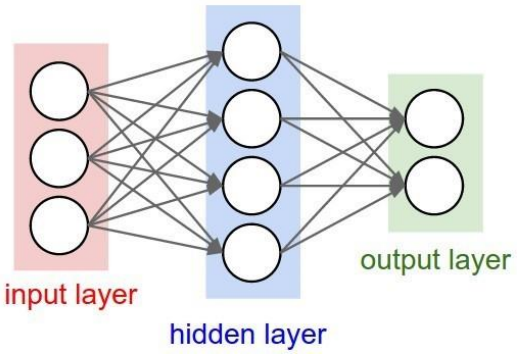


***Figure 5:*** *Example Feed Forward Neural Network Structure [13]*

In this project, Neural Networks toolbox was used in order to implement the feed forward neural network [14]. This particular neural network structure used with both SURF and HOG features had 1 hidden layer (Figure 6). I have tried to experiment with the number of neurons in the hidden layer, by originally setting it to 30, which resulted in poor accuracy (due to overfitting), then increased it gradually to see what number of neurons yields the best performance. I have decided to stick to 70 neurons. The network used Scaled conjugate Gradient Backpropagation in order to adjust the default weights with every iteration.

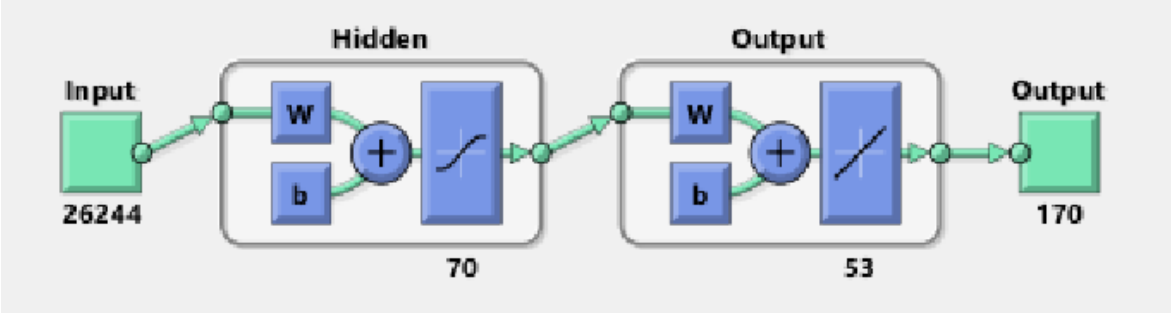


***Figure 6:*** *The structure of the trained Neural Network*

Both trained networks showed a very small Mean Square Error, with network, using HOG features slightly outperforming the one trained with SURF features (Figure 7 and Figure 8).

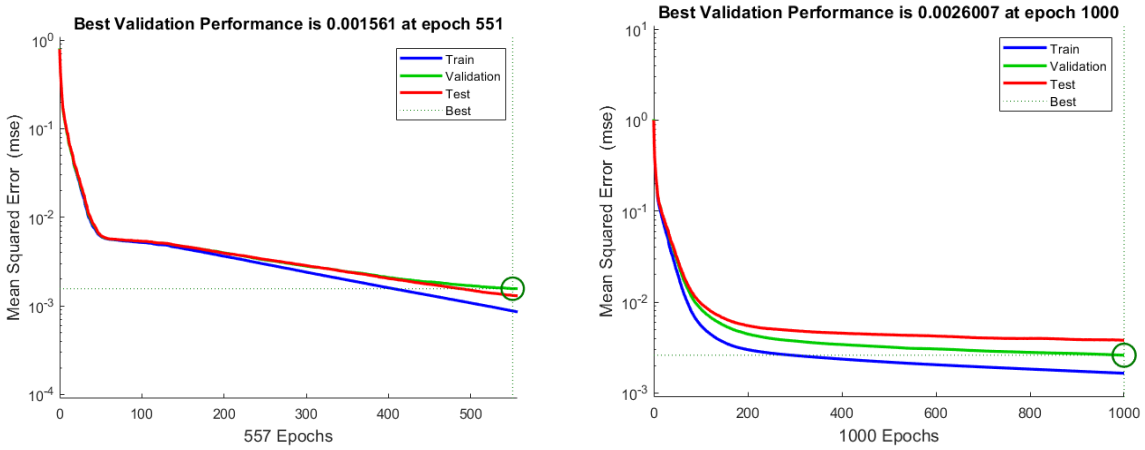


***Figure 7 & 8:*** *Mean Squared Error calculations throughout the number of epochs (right: network, trained with HOG features; left: network, trained with SURF features)*

### 3.4.4. AlexNet

As a part of the paper we were asked to use at least one Convolutional Neural Network (CNN). CNN is a deep learning algorithm, used in image classification. They are biologically inspired (as the connectivity between neurons is designed to emulate that of human /animal visual cortex) and consist of an input layer, a series of hidden layers (convolutional layer, pooling layer and filter layers) it uses Softmax classifier in order to identify the labels for the output layer.

I chose to use AlexNet as it was easy to implement in Matlab. AlexNet is a pre-trained CNN using ImageNet - an image dataset with over a million labelled high-resolution Images, that belong to over a thousand classes. All categories were labelled by humans in the Amazon's Mechanical Turk program [15]. Its structure is unique as it is using Relu activation function instead of Tanh which improves its speed without sacrificing its accuracy; it used dropout layers in order to deal with overfitting; uses overlap pooling to reduce its network size [15] (Figure 9). It takes an input of 227 x 227 RGB imaged and does not require any additional feature extractions (as image processing is built in the structure of the network - convolution layer).

Although AlexNet is already pre-trained with images, classification for other, unseen classes is possible via Transfer learning by replacing the last 3 layers of the AlexNet with a fully connected layer, a Softmax layer, and a classification output layer [16]. Fine tuning an already pre-trained network is usually much faster and easier than training a classifier from scratch.

A function called trainNetwork [16] was used in order to implement the AlexNet in this paper. The replaced layers had an initial learning rate of 0.0001, minimum epoch size of 6 and was optimised using stochastic gradient descent with momentum (SGDM). It showed the best performance out of all the created classifiers as its accuracy on validation data was 100%.

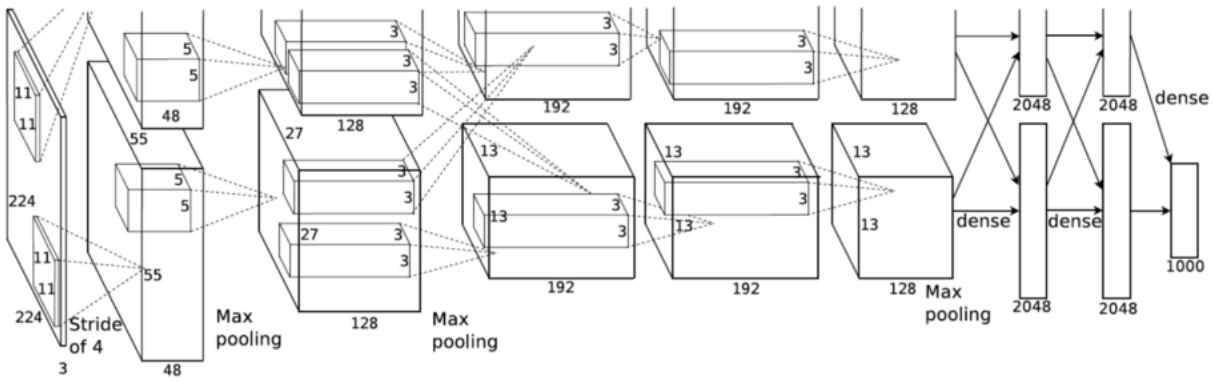


***Figure 9:*** *AlexNet architecture [15]*

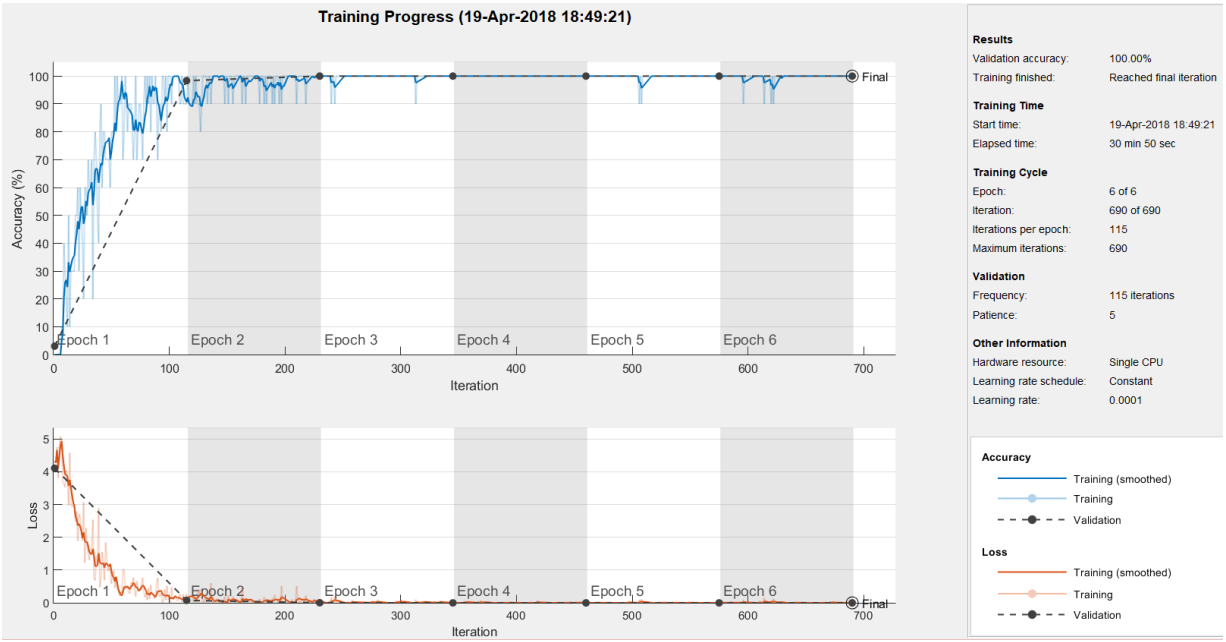


***Figure 10:*** *AlexNet accuracy (top graph) and loss (bottom graph)*

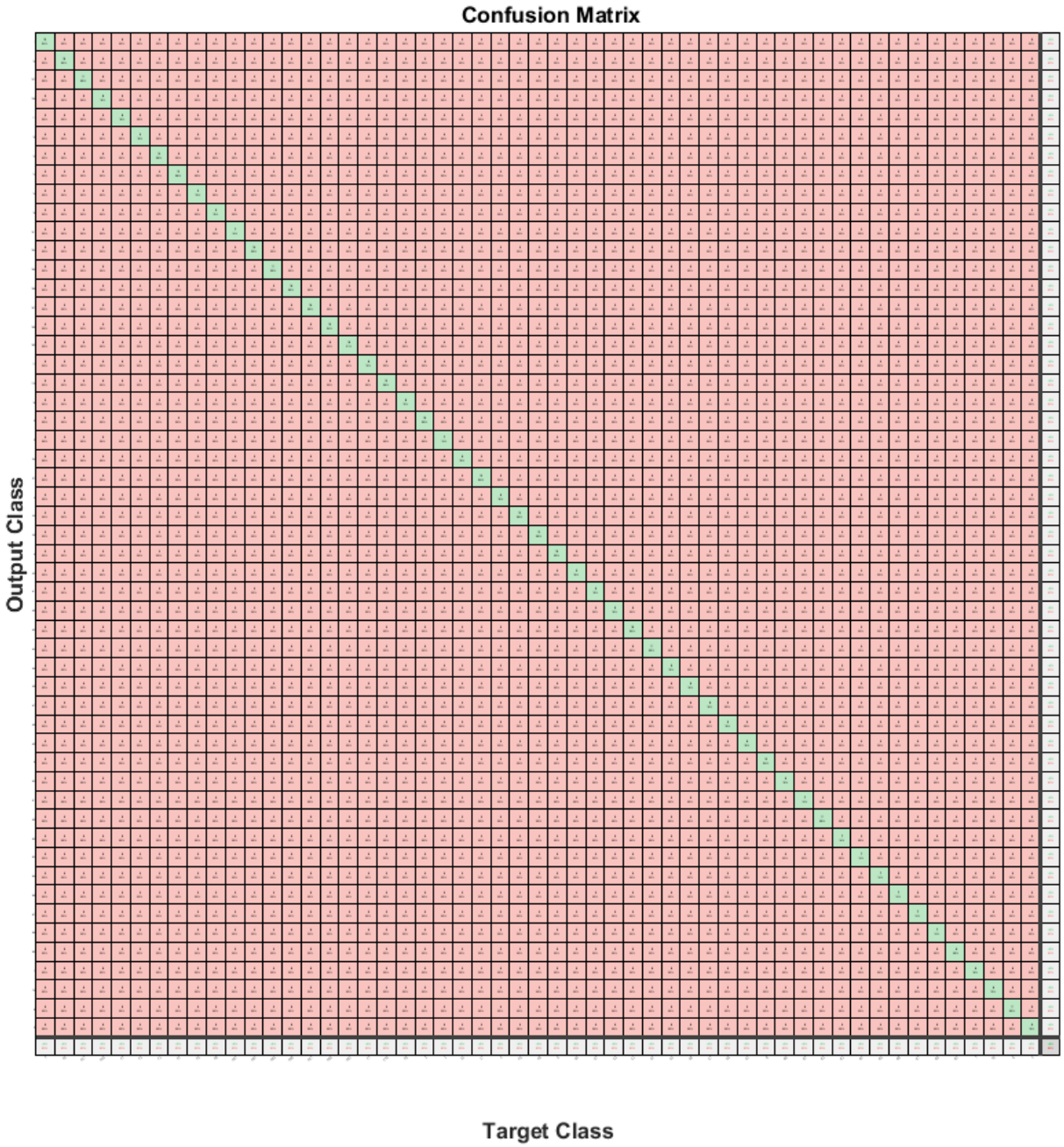


***Figure 11:*** *AlexNet confusion matrix illustrates that target class and output class matches 100% therefore giving an average accuracy of 100%*

# 4. Optical Character recognition (OCR)

## 4.1. Digit Database and classifier

A database of 1,000 images was used in order to train the classifier. The images were obtained from Matlab's in - software database of synthetic digits. There were 100 images for each digit, which were extracted from Matlab's database, resized to 227x227 pixel images and extracted to a new file. The extracted images were then manually sorted into files, labelled corresponding to their class. I have tried augmenting the pictures in order to create more samples, however my attempt was unsuccessful as trained classifiers started confusing 6 with 9, 8 with 0 and 1 with 7.

As learned from face detection, I have decided to try using SVM with HOG features and AlexNet. I have later decided to discard AlexNet and stick with SVM trained with HOG features. It identified numbers with 100% accuracy in the training set. Aside from training classifiers.

## 4.2. Digit extraction and classification

Firstly, I have created a mask by converting the image to gray scale, then cleaning up the image in order to remove any noise and merge some of the smaller regions (figure 13).



***Figure 12&13:*** *left: example of input image; right: example of the mask, created for the input image*

The mask was then used in identification of the MSER regions and properties. These were used in order to identify regions that may contain digits and create bounding boxes around them (please note, this process allows multiple numbers to be extracted from the picture).

After some investigation I have noticed that a lot of the identified digits were false and further filtering was needed before images of digits were sent to the classifier. I have decided to ignore all the extracted images under 25 pixels of size. Although this helped me to eliminate some falsely identified digits, there were still some false digits left to filter out. I have then proceeded to clean up the remaining images by increasing visibility of the characters, cleaning up any noise and reversing black and white colours (figure 14).

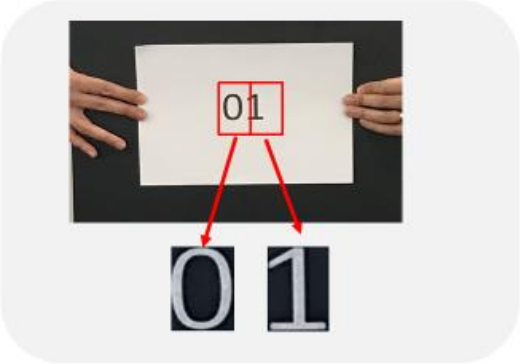


***Figure 14:*** *Example of extracted and cleaned digits, before they are passed for classification*

After cleaning the images I have decided to analyse the MSER regions further and noticed the following pattern: most of the real digits in the images that I have tested had a perimeter of over 100, mean intensity of 0 and convex area of over 1000. I have also calculated the intensity and realized that all of the true numbers had an intensity larger than 150. I have decoded to re-defined the extraction process as follows: 1. Find MSER region; 2. Crop only the regions that have a perimeter of over 100, intensity of 0 and convex area of at least 1000; 3. Clean up any digits that are larger than 25 pixels; 4. Store the digits in a matrix. 5. Pass all digits, with intensity of at least 150 to be classified. Once the non - digits were filtered out, the remaining digits are resized to 227x227 pixels, sent to have their HOG features extracted and labels assigned, using SVM classifier.

## 4.3. Application to video files

The detectNum.m function works on both images and videos. I have done this by creating three possible paths for this function: 1. If the last 3 characters in the file name are .jpg, file is processed as a picture; 2. If the last 3 characters in the file name .mov, file is processed as a video; 3. Anything else - display an error message asking to submit files in either .jpg or .mov format.

Although the image extraction and classification processes are very similar for both image and video paths of the function, a few adjustments needed to be made in order to pass video files. Every $5^{th}$ video frame was extracted from the video and passed through the digit extraction and classification. All of the numbers, found and identified in each frame are stored in a matrix (with each frame, matrix is updated and not overwritten). This created a few possible problems: what if the same set of digits appear in multiple frames of the video?; what if the selected frame is so

blurry that the wrong number was identified? To overcome this, I have decided to display unique, most frequently appearing numbers as an output to this function (if video file is passed as an input).

### 4.4. Failed attempts at digit recognition

I have originally tried using the built-in ocr function, however, even when numbers were cropped put of the picture.  Multiple numbers in one picture were recognized (e.g. if 9 was passed, it sometimes was recognized as 0 and 1).

# 5. Discussion and reflection

In this project I have created two functions: one for facial recognition and another one for optical character recognition. While training classifiers for face recognition I have learned that the best HOG features in combination with a NN or SVM classifier perform better than classifiers with SURF feature extraction technique. A more recent technique, AlexNet with a perfect performance of 100% accuracy, outperformed both SVM and NN classifiers, regardless of the feature extraction technique assigned to them.

Although face recognition task went smoothly, I had a few hiccups as I was trying to understand how the label matrix should look like for NN classifiers. Once I have learned this and built a classifier I realized that labels were wrong (as images were assigned to classes that do not exist due to an error in creating a label matrix). This could have been avoided, however I am glad that I have spotted this error with enough time to fix it.

While working on optical character recognition, I have learned the importance of understanding how to reduce noise in images. I have started working on this task in February, before I have started on face recognition task, and yet it took me weeks of trying different approaches, experimenting with image cleaning techniques, experimenting with classifiers and other functions that could possibly identify digits to come up with the solution, which works (although not perfectly).

Given more time, I would like to introduce SIFT as another feature extraction technique and, when used in training of classifiers, compare its performance and training time to classifiers, trained with SURF features. I would like to experiment more with feed forward neural network, by changing other parameters (learning rate, number of hidden layers, number of neurons in hidden layers, momentum) and implement a grid search in order to find the best possible parameters. I would also spend more time in learning how to clean up noisy images in a more robust way.

# 6. References


[1] Uk.mathworks.com. (2018). *Detect objects using the Viola-Jones algorithm - MATLAB- MathWorks United Kingdom.* [online] Available at: https://uk.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html [Accessed 11 Apr. 2018].

[2] Dalal, N. and B. Triggs. "*Histograms of Oriented Gradients for Human Detection*", IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1 (June 2005), pp. 886–893.

[3] Tsai G. *Histogram of oriented gradients*. University of Michigan. 2010; 1(1):1–17. http://web.eecs.umich.edu/~silvio/teaching/EECS598_2010/slides/09_28_Grace.pdf.

[4] Uk.mathworks.com. *Extract histogram of oriented gradients (HOG) features - MATLAB* extractHOGFeatures- MathWorks United Kingdom. Available at: https://uk.mathworks.com/help/vision/ref/extracthogfeatures.html [Accessed 15 Apr. 2018].

[5] Uk.mathworks.com.. *Computer Vision System Toolbox.* [online] Available at: https://uk.mathworks.com/products/computer-vision.html?requestedDomain= [Accessed 1Apr. 2018].

[6] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool, "*Speeded Up Robust Features*", ETH Zurich, Katholieke Universiteit Leuven

[7] Mordvintsev, A. (2013). *Introduction to SURF (Speeded-Up Robust Features) — OpenCV-Python Tutorials 1 documentation*. [online] Available at: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.html [Accessed 12 Apr. 2018].

[8] Uk.mathworks.com.. *Bag of visual words object - MATLAB- MathWorks United Kingdom.* [online] Available at: https://uk.mathworks.com/help/vision/ref/bagoffeatures-class.html [Accessed 10 Apr. 2018].

[9] Nlp.stanford.edu. (2018). *Multiclass SVMs.* [online] Available at: https://nlp.stanford.edu/IR-book/html/htmledition/multiclass-svms-1.html [Accessed 15 Apr. 2018].

[10] Uk.mathworks.com. *Train an image category classifier - MATLAB trainImageCategoryClassifier- MathWorks United Kingdom*. [online] Available at: https://uk.mathworks.com/help/vision/ref/trainimagecategoryclassifier.html [Accessed 18 Apr. 2018].

[11] Uk.mathworks.com..*Fit multiclass models for support vector machines or otherclassifiers - MATLAB fitcecoc- MathWorks United Kingdom*. [online] Available at: https://uk.mathworks.com/help/stats/fitcecoc.html [Accessed 18 Apr. 2018].

[12] Docs.opencv.org. *Introduction to Support Vector Machines — OpenCV 2.4.13.6 documentation*. [online] Available at: https://docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html [Accessed 21 Apr. 2018].

[13] Sukhadeve, A. (2017). *Understanding Neural Network: A beginner's guide*. [online] Available at: https://www.datasciencecentral.com/profiles/blogs/understanding-neural-network-a-beginner-s-guide [Accessed 19 Apr. 2018].

[14] Mathworks.com. (2018). *Apps - Neural Networks Toolbox for MATLAB & Simulink*. [online] Available at: https://www.mathworks.com/products/neural-network/apps.html [Accessed 09 Apr. 2018].

[15] Gao, H., (2017). *A Walk-through of AlexNet - Medium*. [online] Available at: https://medium.com/@smallfishbigsea/a-walk-through-of-alexnet-6cbd137a5637 [Accessed 17 Apr. 2018].

[16] Uk.mathworks.com. *Pretrained AlexNet convolutional neural network - MATLAB alexnet- MathWorks United Kingdom*. [online] Available at: https://uk.mathworks.com/help/nnet/ref/alexnet.html [Accessed 11 Apr. 2018].

[17] Uk.mathworks.com. *Recognize Text Using Optical Character Recognition (OCR)- MATLAB & Simulink- MathWorks United Kingdom.* [online] Available at: https://uk.mathworks.com/help/vision/examples/recognize-text-using-optical-character-recognition-ocr.html [Accessed 21 Apr. 2018].