Computer Vision Coursework Project Report

Ashley Mason
City, University of London

July 4, 2018

Contents

1 Introduction

Computer vision is a scientific discipline that studies how computers can efficiently perceive, process, and understand visual data such as images and video.

This is a project report for a piece of coursework consisting of two computer vision tasks. Task 1, described in more detail in section 2, involves creating an optical character recognition (OCR) function. Task 2, described in more detail in section 3, involves creating a facial recognition function.

The data-set for this project consists of digital photographs and videos taken in class. These photos were taken with Apple's Live Photo feature which records a video a couple of seconds long either side of the photo[3]. The data-set consists of two types of Live Photos. The first type are individual 'mugshot' photos taken of approximately 50 students who were holding an ostensibly unique ID number up on a piece of paper. The second type are group images of the class. The data-set also contains some group videos which were not captured from Live Photo. The majority of the photos and videos captured were released to us, with a few held back with which to test our functions.

The software I will be using for this coursework is Matlab 2018a [4]. Much of my Matlab code was altered from that of the Matlab example documentation [4]. If taken (wholly or in part) from an external source, it is referenced as such at the top of that Matlab file. Otherwise, the code was my own creation.

2 OCR Task

The objective of this task is to develop a function which reads an image or video file of a person holding up a two digit ID number, and returns the correct ID number as an output.

The logical approach for this task seems to be to split it into two parts. The first part would be identifying the region in the image/video in which the white piece of paper and ID number are positioned. The second part would be correctly classifying the ID on the paper.

To evaluate the performance of my final function, I withheld the 'mugshot' photos as a testing data-set and used the 'mugshot' videos to generate a training data-set. Apple's Live Photo works by recording video for 1.5 seconds before and 1.5 seconds after the static photo capture. This means that the static photo corresponding to each Live Photo is the central frame in the video. To avoid crossover between my training and testing data-set, I therefore chose to collect every 14th frame in each video as this skipped the central frame in each Live Photo's video.

2.1 Initial Approaches

To tackle part 1 of this task, there are a number of possible approaches. One approach is to use feature detection to detect the four border lines of the white paper and use these as a boundary for the ID number. Another approach is to use feature detection to detect the corners of the paper and use these as a boundary for the ID number.

After some initial experiments with these methods, I found that they had differing levels of success across the training data. When using a feature detec-

tion approach a lot more features than those desired are usually found, and the problem becomes differentiating between all features and those which accurately describe the desired region. For example, I used Matlab's detectHarrisFeatures() function (which implements the Harris–Stephens corner detection algorithm [2]) to detect corners in the training data. I then selected a number of the strongest corners and created a bounding box around them. Intuitively, one might expect the corners of the paper to be among the strongest in each 'mugshot' image, given it was white paper held against a black background. However, I found that this was not consistently the case due to a number of variables which were inconsistent across the data-set, such as lighting, background objects and clothing patterns.

Figure 1: An example of mixed success utilising a feature detection method.

An attempt I made to solve this problem was to use a distance metric to constrain corner selection to those closer to the centre of the image. The idea behind this was that, over the training data, the majority of the students were holding their ID in the centre of the image. However, this method resulted in too many missed ID regions due to some students holding their ID astray or alternative camera angles.

Part 2 of this task is a little simpler. Once the IDs have been extracted from the image, it becomes a classification problem based on pixel values of an image. As such, a sensible approach would be to make use of a Convolutional Neural Network (CNN), a model which has been shown to outperform other models in such situations [5].

2.2 Final Implementation

For part 1, I trained an image cascade classifier to classify regions of the image which contained ID numbers. This was a binary classification problem: does this section of the image contain an ID number, yes or no.

The data which I used for this task came from the Apple live-photo videos. For each unique student-ID pair, I read the video into Matlab and extracted a number or images from frames periodically throughout the videos, as mentioned above. I then loaded these images into the Matlab image labeller application and manually selected the regions of interest (ROIs) which contained ID numbers.

Once these ROIs were labelled, I used them as positive examples to train a cascade object classifier.

Along with positive examples, negative examples (pictures where two digit ID numbers are absent) are also necessary to train the classifier. To obtain these, I returned to the location where the photos were taken and collected scene images from which to generate negative examples. The cascade classifier is trained by extracting features from the positive examples and comparing them to those found in the negative examples. The features present in the positive images but not in the negative ones are then searched for in a step search across the target image to identify the regions containing an ID.

The features used in this case were histogram of oriented gradient (HOG) features. After training with these scene images as generators of negative examples, then testing on the 'mugshot' images, I found that clothing patterns were causing a lot of false positives. To improve performance, I cropped images of problem clothing to enhance the negative examples generated, Figure 2.

Figure 2: An example of false positives due to clothing, eradicated by using clothing samples as negative generators.

A benefit of the cascade object classifier approach is that it is no longer necessary to constrain the search space to a concentrated central region, allowing a more robust search and greater generalisability.

After solving the problem of clothing patterns, my cascade object classifier was still producing a few false positives. One problem was a section of the background which contained a plug socket kept producing a false positive. Whenever I tinkered with the parameters to train the cascade classifier (i.e. the number of cascade stages, the false alarm rate, the positive and negative image sets), I found that any solution to picking up the plug actually caused a greater problem of some sort elsewhere. Given that the plug is in the bottom left corner, close to the edge of each image in which it appears, I decided to perform a small crop in my OCR function. I reduced the width and height by 20%, 10% on each side, to leave the central part of the image untouched. This area was enough to allow for variation in where the IDs were being held but large enough to eliminate the plug problem.

Another problem was that my test data contained images of different dimensions. Experimentally, I found that test images of size approximately 450x600 pixels resulted in the least false positives. The images in the test set were either 3032x4032, 2044x2044 or 1536x3000, so in my OCR function I included a re-sizing of the image, conditioned on the area of the original image.

For part 2 of the OCR task, I extracted the ID regions found in part one and used these as a data-set to train a CNN to classify the images into one of the 53 classes. Rather than training a CNN from scratch, I decided to use Alexnet [1] [5] as a starting point and retrain specifically for my purpose. I will refer to my retrained Alexnet as twoDigitAlexnet. The benefit of this that it is a considerably less computationally expensive, saving time and computing power, both in raw run time and in time spent fine tuning hyper-parameters to achieve the desired results.

When training twoDigitAlexnet, I split my data into 70% training data and 30% validation data, and used a learning rate of $\alpha = 0.0001$ over 6 epochs. TwoDigitAlexnet achieved an accuracy of 96.75% in classifying the two digit ID numbers, Figure 3.

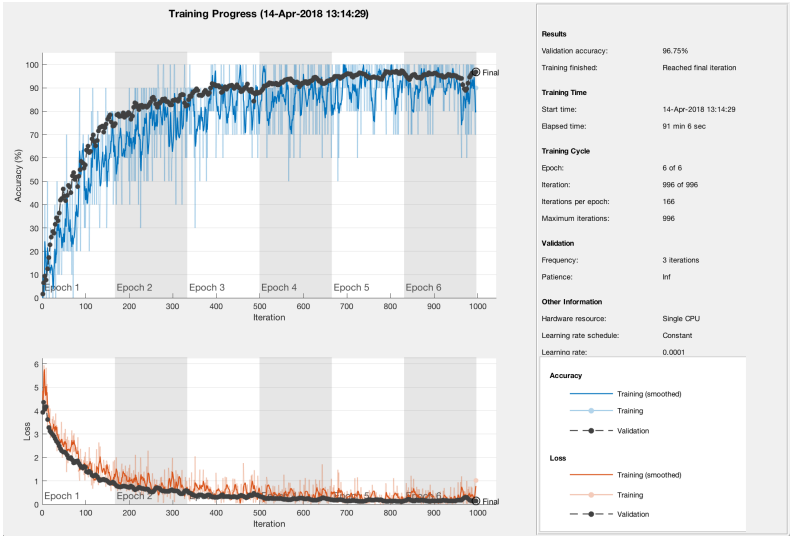

Figure 3: Screen-shot of Matlab output after training twoDigitAlexnet.

[1] Alexnet is a CNN trained on over a million images to classify images into 1000 object classes which rose to fame for its high performance in the ImageNet Challenge[].

The following block diagram details the key algorithmic components to complete the OCR recognition task.

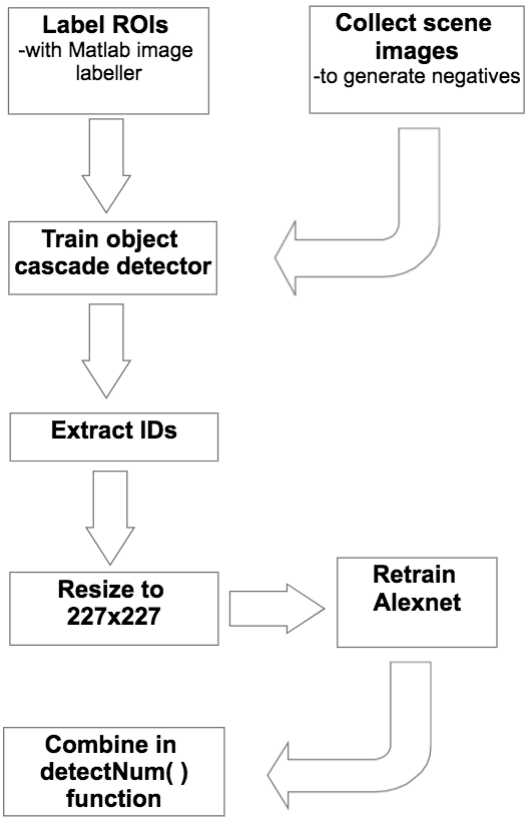

Figure 4: Block diagram of key algorithmic components for OCR task.

2.3 Performance

After training the cascade object classifier and combining it with twoDigitAlexnet in a Matlab function, I wrote a script to test its accuracy on my test data. Figure 5 shows the form that the test data was stored in and a sample of the script is detailed in the Matlab code following.

testOCR.mat - *Matlab code to test OCR function accuracy.*

```
% start counts
correct_count = 0;
count = 0;

% test images for ID = 060
for i = 1:7
    if exist(sprintf('testIMAGES/IMG_%04d.JPG',i),'file')==0
        continue;
    end
    answer = detectNum(sprintf('testIMAGES/IMG_%04d.JPG',i));
    if answer == '060'
        correct_count = correct_count+1;
    end
    count = count + 1;
end

% test images for ID = 061
for i = 8:13
    if exist(sprintf('testIMAGES/IMG_%04d.JPG',i),'file')==0
        continue;
    end
    answer = detectNum(sprintf('testIMAGES/IMG_%04d.JPG',i));
    if answer == '061'
        correct_count = correct_count+1;
    end
    count = count + 1;
end
... % and so on until
... % the end of the
... % training images

% Evaluate accuracy of detectNum function
accuracy = correct_count/(count)
```

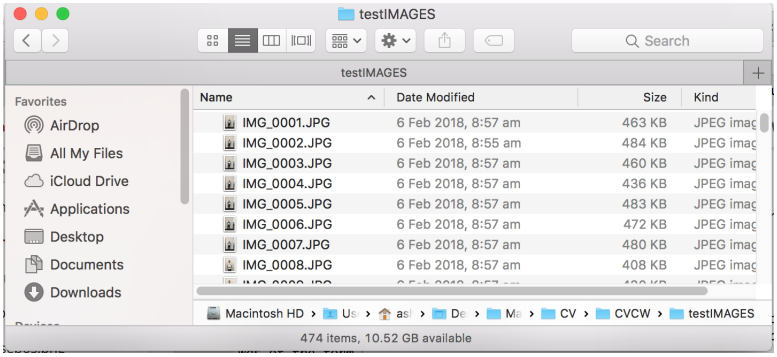

Figure 5: Screen-shot of Matlab output after training twoDigitAlexnet.

After testing my detectNum() function, I found it correctly identified 460/473 ID numbers in images, resulting in an overall accuracy of 97.25%. On the occasions when the function failed, it was due to one of three reasons. It failed due to incorrectly reading the ID number on 10/470 occasions. This failure took two forms, twoDigitAlexnet either incorrectly misidentified a six as a zero, or a one as a four figure 6. It failed due to reading a false positive on 2/470 occasions, figure 7. It failed on 1/470 occasions due to not picking up the ID number in the image, figure 8.

My detectNum() function achieved an accuracy of 100% when tested on the 'mugshot' videos. The design of the detectNum() function for video input is such that it repeats the OCR on frames 10, 20, 30, 40 and the final frame, with the output being the mode output from each repetition. The fact that my test data came from every 14th frame in these videos and that 14 does not divide 10, 20, 30, 40 or the number of frames means that this evaluation of accuracy is a valid one; there is no cross over between training and testing data.

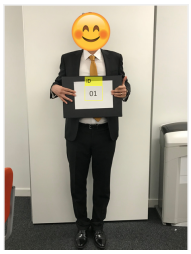

Figure 6: An example of an incorrect classification of the ID number.

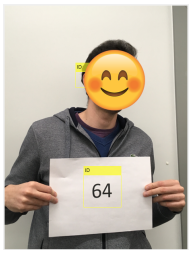

Figure 7: An example of an error due to false positive.

Figure 8: An example of an error due to false negative.

3 Facial Recognition Task

The object of this task is to develop a function which takes a loaded image as an input and returns a matrix detailing the positions of any faces in the image along with the ID number for that face. The function should use three models to perform this classification, with two feature types for each model where relevant.

Again, this task seems to be split into two parts. The first part requires the faces in the image to be detected. The second part requires that each detected face is then passed into a classifier and the result returned.

3.1 Feature Extraction

Feature extraction is a way of finding generalised patterns or features from images without having to compare each pixel value. For example, a common form of feature extraction in computer vision is edge or corner detection. Images are stored as high dimensional objects, containing many thousands of pixels each of which containing one to three dimensions depending on colour-space. This means that feature extraction is an essential form of dimensionality reduction for computer vision with current hardware limitations. Feature extraction also improves human understanding of computer vision as more general patterns are more recognisable to humans than pixel values. The features I will be using in my models are histogram of oriented gradients (HOG) and speeded-up robust features (SURF).

HOG features are generated by counting occurrences of gradient orientation in localized portions of an image. SURF (features) are designed to be robust against a variety of image transformations and distortions.

After extracting SURF and HOG features and then clustering them, each image could be represented by a 500 dimensional clustered feature vector; a vast reduction in dimensionality.

3.2 Models

I will be using three models to classify the faces found in my image. The classification models I will be using are a Random Forest (RF), a Support Vector Machine (SVM) and a Convolutional Neural Network (CNN).

3.2.1 Random Forest

An RF is an ensemble method for classification (in this case) which operates by constructing a multitude of decision trees, sending an example down each of the trees, and outputting the class which received the most 'votes' from the forest [1].

In the context of this task, each image will be represented by a 500-dimensional clustered SURF or HOG feature vector. Each node in a tree will contain a decision based on one of these features. The training data is split into the "in bag" and "out of bag" portions. During training the "in bag" data sent through each tree and each leaf is assigned the class from which the maximum number of examples reached that leaf. The "out of bag" portion is then passed through each complete tree in the forest to obtain a validation accuracy.

3.2.2 Support Vector Machine

An SVM is a supervised learning model which performs a binary classification by constructing a hyper-plane between two sets of points corresponding to the feature vectors of the observations in the training data-set. SVM can be extended to a multi-classification model by performing a number of "one vs. many" classifications. SVM have been shown to perform well but numerous "one vs. many" classifications increases computational cost and it can become unfeasible for some tasks.

In the context of this task, each image will be represented by a 500-dimensional vector of clustered SURF or HOG features, previously extracted. The SVM will find a hyper-plane which separates the set of points corresponding to images of a given person to the remaining points from the training data. This is performed for each person to train the model. When an image is input into the trained SVM, its feature vector is calculated and the model classifies it according to the section of the 500-dimensional space it occupies, bounded by the hyper-planes found in training.

3.2.3 Convolutional Neural Network

CNNs are a form of feed forward artificial neural networks containing convolution layers. CNNs are a classifier which take pixel values as inputs rather than depending on feature extraction techniques. As with the OCR task, I will be using Alexnet [5] as a starting point and retraining for this purpose. Alexnet takes a concatenated vector representing an image of size 227x227 pixels in RGB colour space, so all my input images had to be re-sized to 227x227 before training.

3.3 Implementation

For the first part of this task, it is necessary to extract the faces from both the 'mugshot' images and the group images. To do this I attempted to train my own cascade object detector following the same steps as outlined above in section 2.2. However, after several iterations of the process, I found this to be less accurate than the Matlab 'vision.CascadeObjectDetector' function, which utilises the Viola-Jones algorithm [6].

As a set of training images, I used the 'mugshot' training set from above and augmented it with faces extracted from every 5th frame of all but one of the group videos taken with Apple Live Photo. As a test data-set, I used my previous test set and augmented it with every 5th frame taken from the final group video taken with Apple Live Photo. This ensured no crossover between training and testing data-sets.

In order to extract the faces from my testing data, I wrote a script which can be seen below.

ExtractFaces.m - *extracts and writes faces from imds.*

```
1 % Create image datastore object
2 imds = imageDatastore('moreGroup','IncludeSubfolders',true,'
      LabelSource','foldernames');
3
4 % Iterate over all images in imds
```

```
for i = 1:length(imds.Files);
    [img, fileinfo] = readimage(imds,i);
    % Extract file path
    filename = strcat(extractBefore(fileinfo.Filename,'.jpg'));
    % Choose Viola-Jones detector
    detector = vision.CascadeObjectDetector;
    % Find bounding boxes
    bboxes = detector(img);
    % If only 1 found
    if size(bboxes,1) == 1
        img = imcrop(img,bboxes(1,:)); % Crop
        img = imresize(img,[227 227]); % Re-size
        imwrite(img, [filename '_face.jpg']); % Write
    else % Otherwise iterate over each bbox found
        for j = 1:size(bboxes,1)
            img = imcrop(img,bboxes(j,:)); % Crop
            img = imresize(img,[227 227]); % Re-size
            imwrite(img, [filename int2str(i) int2str(j) '_face.jpg
                ']); % Write
        end
    end
end
```

After doing this, the extracted files were in one folder and needed sorting into appropriately labelled subfolders. Unfortunately, I was not creative enough to think of a way to automate this rather arduous and very lengthy process. However, this was a necessary evil to progress to the next stage of the task, so 7 hours later, I had successfully sorted the 5941 images into 54 folders.

The following block diagram details the key algorithmic components to complete the facial recognition task.

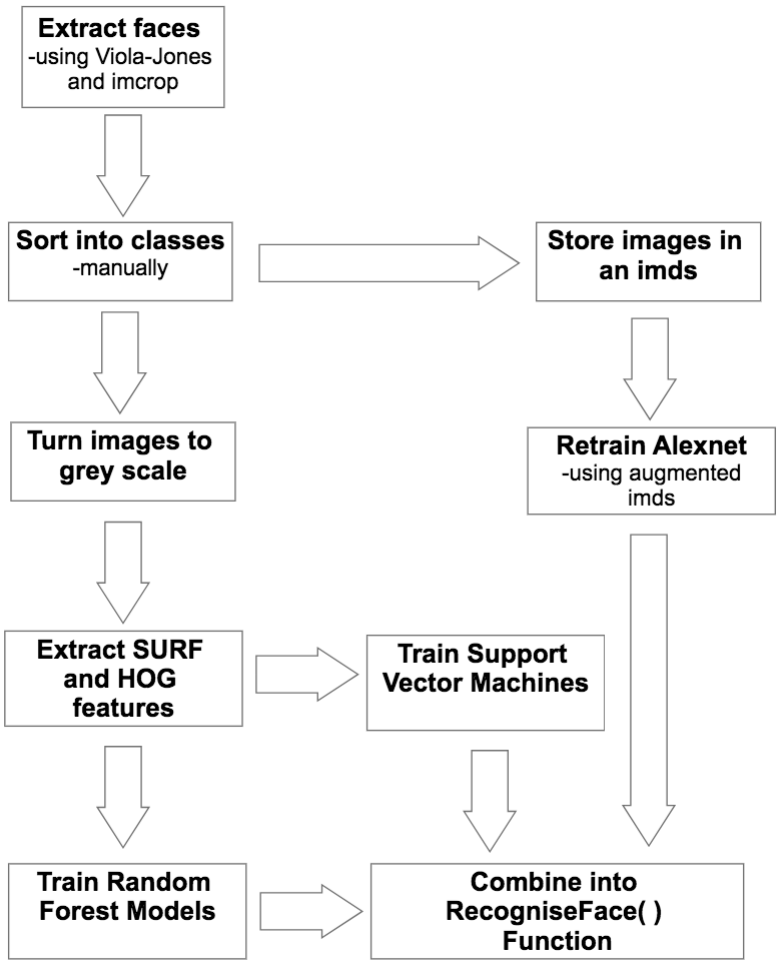

Figure 9: Block diagram of key algorithmic components for facial recognition task.

3.4 Analysis

After a few iterations, tweaking hyper-parameters to try and improve performance, the final hyper-parameters used in my RecogniseFace() function are detailed below.

SURF (features) - number of clusters: 500, clustered on: strongest 80% of features, block width: [32 64 96 128].
HOG features - block size: 5x5, number of bins: 9, cell size: 25x25.
RF with HOG - 30% holdout, number of features to sample: 10.
RF with SURF - 30% holdout, number of features to sample: 10.
SVM with HOG - Kernel function: linear, 70% training, 30% validation.
SVM with SURF - Kernel function: linear, 70% training, 30% validation.
CNN - initial learning rate: 0.0001, mini batch size: 10, number of epochs: 6, 70% training, 30% validation.

The results for my models are detailed in the table below for the hyper-parameters detailed above. To evaluate the testing accuracy, I tested the model and all the 'mugshot' images in my testing data set.

Model	**Validation Accuracy**	**Testing Accuracy**
RF with HOG	96.69%	91.75%
RF with SURF	94.05%	75.05%
SVM with HOG	97.14%	93.02%
SVM with SURF	78.13%	75.05%
CNN	95.13%	79.07%

My best performing model was an SVM using HOG features. Figure 13 displays a confusion matrix for this model's performance on the validation data. Unfortunately confusion matrices of this size can be hard to read as there are so many classes, so I shall also include some examples of when the function worked and when it failed.

Below are three examples of when the SVM with HOG features failed. Figure 10 shows the student with ID 007 being incorrectly identified as a member of the 'Unknown' class. In this instance I believe the failure is due to the face detector failing to correctly identify the whole face due to the angle of the students head. Figure 11 contains the similarly tilted head of student 045 being misidentified as 007. Whilst student 045 may look like James Bond, this is not the reason behind the functions classifying him as 007 but rather an error due to an obscured view of the face. Figure 12 shows an example of a false negative due to the reflection in the glass window. The function manages to correctly identify student 009, but also classifies a section of window as student 054.

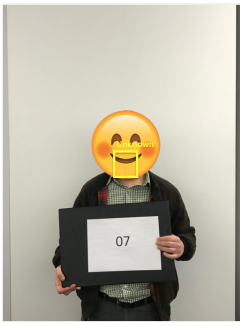

Figure 10: Confusion Matrix for SVM with HOG on validation set.

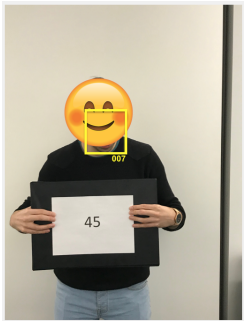

Figure 11: Confusion Matrix for SVM with HOG on validation set.

Figure 12: Confusion Matrix for SVM with HOG on validation set.

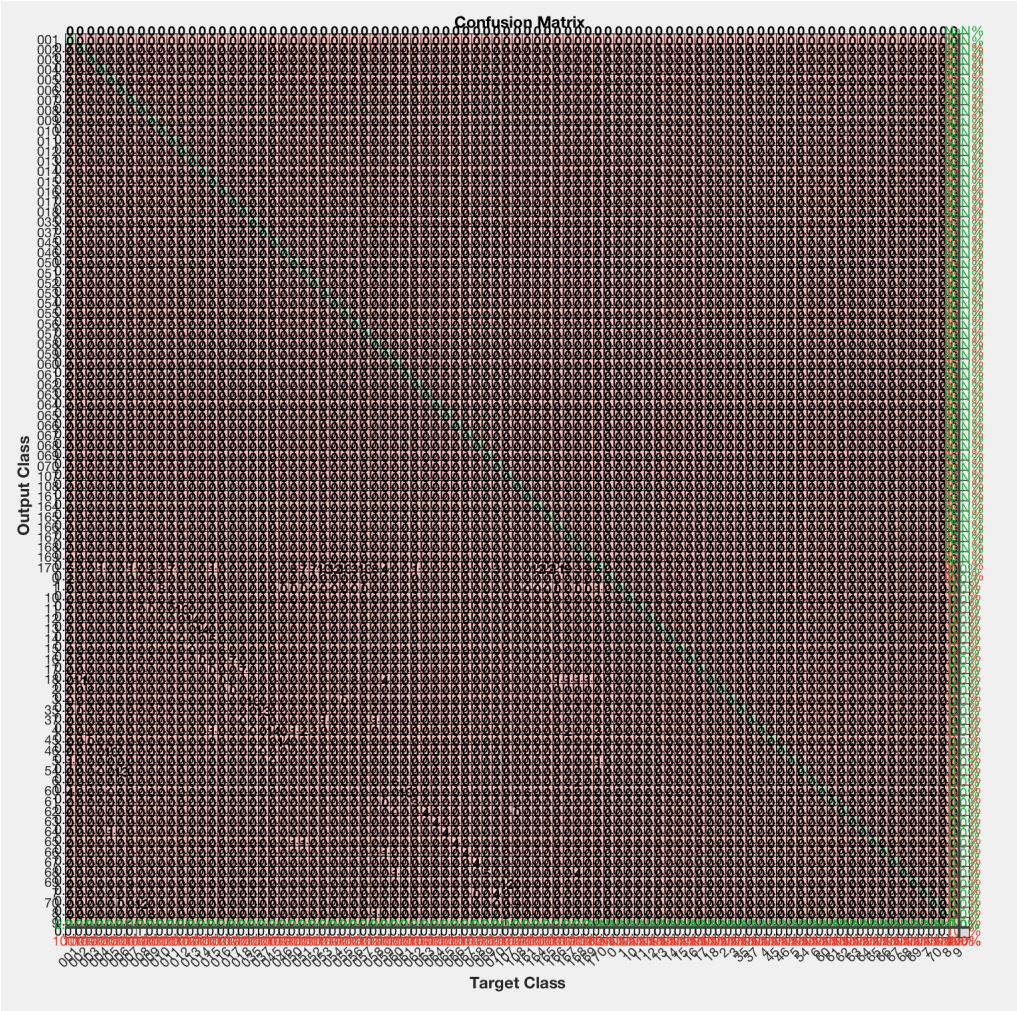

Figure 13: Confusion Matrix for SVM with HOG on validation set.

4 Reflections

I found this project to be very interesting, albeit quite frustrating at times. As my first computer vision project, it serves as a salient lesson as to what is needed for success in computer vision.

I found certain aspects can limit the success of a computer vision project. Computational expense is so great that an abundance of computational power, time to run code and, above all, patience is needed. With the limited hardware resources available to me, I believe I did well and achieved acceptable levels of accuracy in each task.

In the OCR task, I achieved an accuracy of 97.25% on my test images and 100% on my test videos. Whilst this level of accuracy is certainly a great achievement, it is possible that my model would not maintain this accuracy on new data. This is because of the way in which I trained my cascade object

detector. To improve it's accuracy, I used images from the scene of the test images and cropped images of students clothing I knew would be in the test images. I have doubts that my function would perform as well on images taken at another location, or with students with new, different clothing patterns.

A major limitation to improving the results of the facial recognition task was an insufficient amount of data. As image data is so large in nature, this was an unavoidable problem given hardware and time constraints. My retrained Alexnet CNN performed poorly compared with the other methods implemented, particularly given their reputation for performing so well in computer vision tasks. I think this is because my model was likely overfit due to an abundance of training images coming from group images when students faces were lower quality due to distance from the camera.

A note on using my RecogniseFace() function: my function, RecogniseFace(image, featureType, classifierName), takes two additional arguments to an image. For proper usage of this function, accepted inputs for featureType will be 'SURF' or 'HOG' and accepted inputs for classifierName will be 'CNN', 'RF' or 'SVM'. Whilst the 'CNN' does not rely on either SURF or HOG features, an input must be provided for featureType.

References

[1] Leo Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[2] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Citeseer, 1988.

[3] Apple Inc. Apple live photos. `https://support.apple.com/en-gb/HT207310`. Online; accessed 15 April 2018.

[4] The Mathworks Inc. Matlab. `https://www.mathworks.com/products/matlab.html`. Online; accessed 15 April 2018.

[5] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[6] Paul Viola and Michael J Jones. Robust real-time face detection. *International journal of computer vision*, 57(2):137–154, 2004.