Computer Vision INM460

Face Recognition and Optical Character Recognition Coursework Report

By Lesley Dwyer

1 Introduction

The aim of this coursework was to use computer vision techniques to build two programs. First, a face recognition program to identify our classmates in an unseen image using a known database of images. Second, an Optical Character Recognition (OCR) program to identify a number in an unseen image or video. In the following sections, I describe my approaches used, results and analysis of these approaches and a discussion of strengths, weaknesses and ideas for future work.

2 Approaches

For this project, I used a combination of approaches to implement the final solution. I began by preparing the image training database. Next, I built five models using three different classifiers and two different feature types to classify the face images. Then, I build a function which identifies a person from an unseen image. I also built a second function which identifies the number a person is holding in an unseen image or video. Finally, I tested the face recognition function on unseen images and the number recognition function on unseen images and videos. Below are detailed steps on how each stage was achieved.

2.1 Image Preparation

We were provided the original images and videos of our classmates taken by four cameras. These consisted of 289 images and 289 videos of individual people and 19 images and 18 videos of a group of people. In the individual images, the person is holding up a piece of paper with a number on it. The file formats provided were .jpg, .jpeg and .heic for the images and .mov and .mp4 for the videos.

I started preparing images by creating a numbered folder (1-81 with some gaps) for each person. I moved all original images and videos of the person with that number into the corresponding folder manually. Matlab does not support the .heic format, so these needed to be converted to a supported image format. I converted these to .jpg using the iMazing application. This application is installed locally, and the data does not get uploaded into the cloud [1]. For the rest of the image processing and model building, I used Matlab version R2018b. I extracted 10 images from each video using Matlab and saved them back to original folders. I used face detection code provided to us in lectures and available from Mathworks, which uses the Viola-Jones algorithm [10] to detect faces in each image. I then cropped images of the faces only, resized the image to 64 by 64 pixels and saved them to a new folder structure with the same labels. I then removed bad images manually, including non-faces and a few blurry faces where people were moving in videos. After removing bad images, this left fewer than I expected for some people. So, I extracted additional images from videos for 13 people who had fewer than 40 good images. This resulted in a training database of 3,524 images for 69 people. I chose not to include faces from the group images in the training database, as it would have involved a time-consuming task of manually identifying each person from each cropped face image and adding them back into the labelled folders.

2.2 Face Recognition

For the face recognition task, I used three different classifiers: a Convolutional Neural Network (CNN), a Support Vector Machine (SVM) and a Multi-layer Perceptron (MLP). For the SVM and MLP classifiers, I also used two different feature types: Histogram of Oriented Gradient Features (HOG) and Speeded Up Robust Features (SURF). This resulted in five separate models to classify the faces.

Features are extracted from images in computer vision problems, so they can be sent to classifiers instead of training the classifiers on the images themselves. HOG and SURF features are commonly used in computer vision, as they are scale-invariant. Another technique used is the bag of features model, which creates a vocabulary of "visual words" or features from images. A histogram of feature occurrences can then be computed for each image to create a feature vector used to train a classifier [5]. In two of the models, I used the bag of features approach to extract SURF features.

For each model, I began by reducing the number of images for each person to the minimum number, so the models were trained on a consistent number of images for each person. Otherwise, an unbalanced dataset can lead to biased results if some people have more images that others. The minimum number was 40, so this reduced the training database to 2,760 images. I also split the database into 80% training data and 20% validation data to leave some images for testing. In addition, I kept 20 images (18 individual and two group) completely separate from the models to use when testing the final face recognition function.

2.2.1 Overall Face Recognition Solution

Below are the different components of the face recognition solution. The first flow diagram shows the image preparation and model training. The second diagram shows how the RecogniseFace function is used to classify a face(s) on an unseen image.

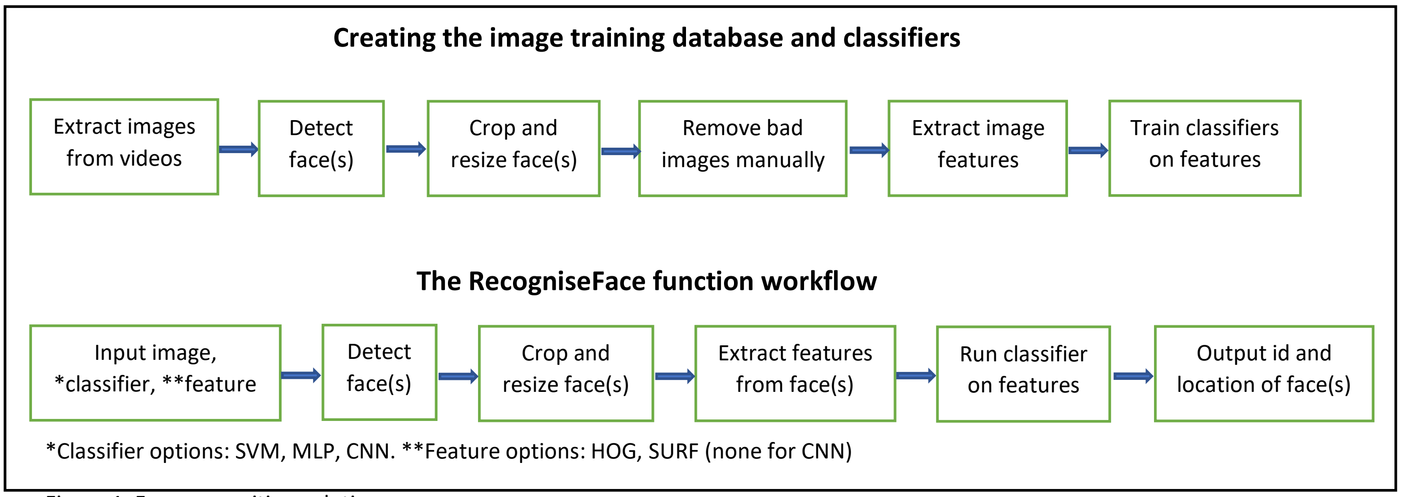

Creating the image training database and classifiers
Extract images from videos
Detect face(s)
Crop and resize face(s)
Remove bad images manually
Extract image features
Train classifiers on features
The RecogniseFace function workflow
Input image, *classifier, **feature
Detect face(s)
Crop and resize face(s)
Extract features from face(s)
Run classifier on features
Output id and location of face(s)
*Classifier options: SVM, MLP, CNN. **Feature options: HOG, SURF (none for CNN)

Figure 1. Face recognition solution.

In the next few sections, I describe the models that were built, the RecogniseFace function and unsuccessful approaches.

2.2.2 CNN

For the CNN, I chose to use AlexNet, a pretrained CNN, and implement transfer learning for this model. I installed AlexNet from the Mathworks file exchange [8] and modified an example provided by Mathworks to re-train the last three layers of the network using my training data [9]. The performance method used for CNN was single-fold and classification accuracy.

2.2.3 SVM with SURF

For the SVM model using SURF features, I began by adapting code provided to us in lab five [3]. I extracted features using a bag of features for all images to create a vocabulary of 500 features. These features were then used with the trainImageCategoryClassifier function to train the SVM classifier. The performance method used for SVM with SURF was single-fold and classification accuracy. I did not perform any hyperparameter tuning on this model, as the validation accuracy was very high.

2.2.4 SVM with HOG

For the SVM model using HOG features, I began by adapting code provided to us in lab six [4]. I extracted 1,764 HOG features for each image and used them to train the SVM classifier. The performance method used for SVM with HOG was k-fold cross-validation and classification accuracy. I did not perform any hyperparameter tuning on this model, as the validation accuracy was very high.

2.2.5 MLP with SURF

For the MLP model using SURF features, I began by adapting code provided to us in lab five [3] and code generated from the Neural Network toolbox. For this model only, I converted the images to grayscale when early testing produced poor results. I thought this may have improved the accuracy, as SURF features are not robust to changes in lighting [6], and converting to grayscale can potentially help. However, this was not the root cause of the accuracy issue, and therefore I did not notice any improvement. Further details on this are discussed in section 2.2.8.

I extracted features using a bag of features for all images to create a vocabulary of 500 features. These features were then used to train the network. I used one-hot encoded labels with a softmax output layer.

I tried different values for the hyperparameters learning rate, number of hidden neurons and training function. The combination of 10 hidden neurons, a learning rate of 0.1 and the Levenberg-Marquart backpropagation training function produced the best performance. The performance method used when training the MLP with SURF was single-fold and mean-squared error, although classification accuracy was also calculated.

2.2.6 MLP with HOG

For the MLP model using HOG features, I began by adapting code provided to us in lab six [4] and code generated from the Neural Network toolbox. I extracted 1,764 HOG features for each image and used them to train the MLP classifier. I used one-hot encoded labels with a softmax output layer. The scaled conjugate gradient backpropagation function was used to train the network. I tried different values for the hyperparameters learning rate and number of hidden neurons. The combination of 10 hidden neurons and a learning rate of 0.01 produced the best performance. The performance method used when training the MLP with SURF was single-fold and mean-squared error, although classification accuracy was also calculated.

2.2.7 RecogniseFace Function

Once all the models were built, I built a function called RecogniseFace. This function takes as input one image, a feature type and a classifier name. It then uses the input feature type and classifier name values to call the correct model, extract features from the input image using the feature type and predict the face from the input image. Initial results were producing additional non-faces as faces, but increasing the detection threshold parameter in the face detection resolved this. The function documentation mentioned that increasing this parameter value can reduce false positives [10].

I tested the RecogniseFace function manually running each model on 20 unseen images, two of which were group images. In order to test the group images, I built short script to create a plot, shown in Figure 2, of each person and their correct label. I built another script to annotate the predicted class onto the group image. Using both of these along with Excel, I was able to calculate the group accuracy. There were four people in the group images who I was not able to identify due to occlusion. There were also three people who were not in the individual images, and therefore were not used to train the models. I chose to remove these individuals from my group accuracy calculations.

[*Figure with face images removed*]
Figure 2. All faces and their labels.

After the testing, I discovered there were four individual images (shown in Figure 3) for which faces were not identified correctly by any model. This pointed to a problem with the face detection.

[*Figure with face images removed*]
Figure 3. Unsuccessful RecogniseFace examples. No face identified, faces identified incorrectly on a finger and an arm, and an electrical outlet identified as a face.

After trying some different values, I found that setting the minimum size of the faces to 75 by 75 pixels within the face detection function allowed the face detector to ignore smaller non-face objects it was detecting as faces, but it was still large enough to detect all the faces in the group images. This increased the individual image accuracy by 16% on average leaving only one image where the face was not detected correctly. This change did decrease the group accuracy by 3% on average. Even though it detected all the faces as before, it had more trouble identifying the smaller faces, i.e. people in the back rows.

The **RecogniseFace** function can be run in the Matlab command window. It requires the following as input arguments:

- **Image**: This must be entered in the format 'IMG_name.jpg'.
- **classifierName:** Valid values for this are SVM, CNN, or MLP.
- **featureType:** Valid values for this are HOG, SVM, or '' (CNN does not require a feature type, so this can be entered as '').

The function can be called like this:

- **RecogniseFace(**'IMG_name.jpg', 'SVM', 'HOG')

2.2.8 Unsuccessful Face Recognition Approaches

There were some early approaches that did not work. The MLP with HOG using classes as original labels produced accuracy of 83%, but when I tested on unseen images using the RecogniseFace function, it produced 0% accuracy. After I reformatted the labels to one-hot encoding and used a different function, patternnet, to build the network, this produced a much better test accuracy on unseen images via the function.

For the MLP with SURF, I ran into the same issue as above with the labels, which was fixed by implementing one-hot encoding. I also initially tried extracting the features in the RecogniseFace function by creating a new bag of features for the test image. This resulted in 0% accuracy for the test images even when the model itself performed well. It also produced different results for the same test image when it was tested repeatedly with the function. I changed this to load and use the bag of features created from the training images, i.e. the same one used to train the model, and this fixed the accuracy issue in the function.

For the CNN model, the early accuracy when testing though the RecogniseFace were very low at 27%. This improved greatly when I augmented the test image in the RecogniseFace function, just as I had done when training the images in the CNN model.

2.3 OCR

For the OCR task, I used a combination of Maximally Stable Extremal Regions (MSER) and Matlab's built-in OCR function. I adapted code from Mathworks to build this [7]. MSER is a method used to identify connected components, which can be used for text detection. OCR is a method that can then be applied to regions of text to identify the characters [2].

2.3.1 detectNum Function

I built a function called detectNum, and within that function I used MSER to identify candidate regions of text. After that, I used geometric properties to identify regions to remove and removed them. Next, I removed more non-text regions using stroke width variation. Here I changed the stroke width threshold from 0.4 to 0.55 which improved the accuracy on sample images. Once the non-text regions had been removed, I used OCR to identify the number in the region. This worked best when only returning text with character confidences greater than 0.75, limiting the character set to 0-9 and setting the TextLayout property to 'Line'. For videos, I extract the 10th frame of the video and then processed that image as the other images. Once the detectNum function was built, I tested it on 10 images and 10 videos.

The **detectNum** function can be run in the Matlab command window. It requires the following as input arguments:

- **Image or Video**: This must be entered in the format 'IMG_name.jpg' or 'IMG_name.mov'.

The function can be called like this:

- **detectNum(**'IMG_name.jpg')

2.3.2 Unsuccessful OCR Approaches

I did try a few approaches that did not work well. First, I tried the Matlab OCR function on its own, but it only detected text in 20% of sample images. When it did detect something, it was a lot of characters and not the actual number. I also tried using a blob analyser to find the sheets of paper, but that only found parts of the paper in 20-40% of sample images, depending on the value of the MaximumCount parameter. Next, I tried MSER with a region of interest (ROI) parameter and OCR at the end which was successful in detected digits in 90% of sample images, but the OCR function could not recognise the number in most of them.

3 Results and Analysis

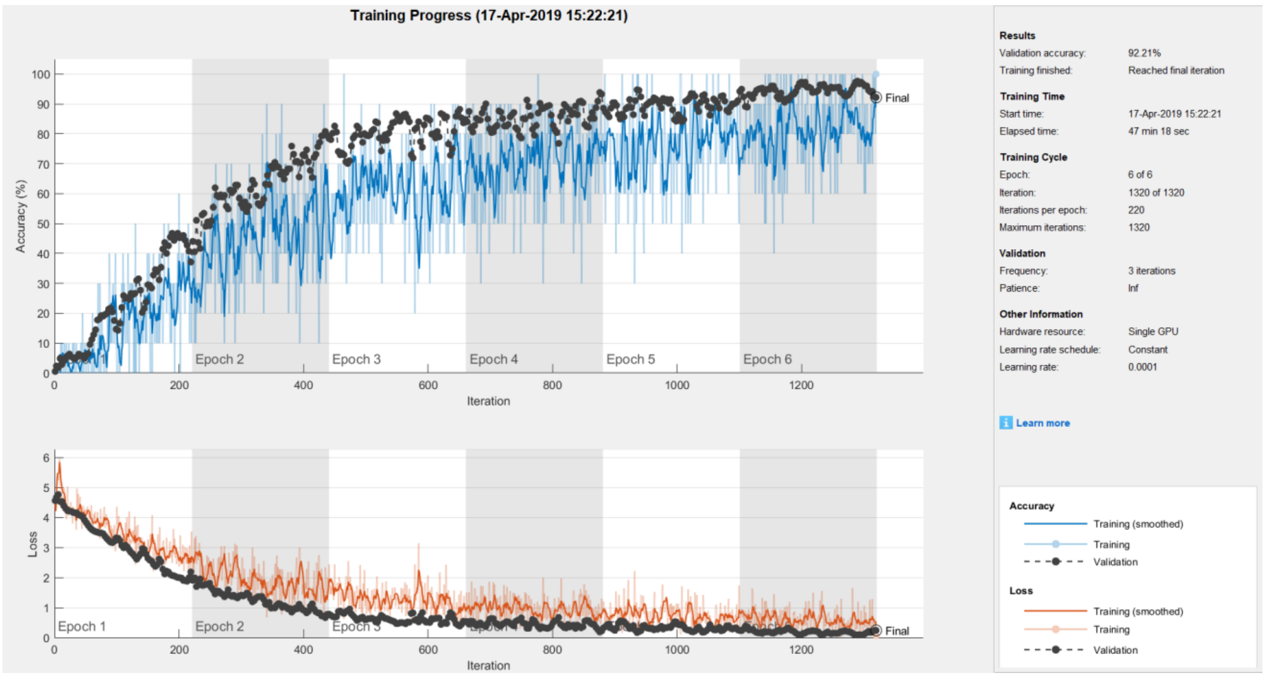
In this section, I present the results of the face recognition and number detection solutions. This includes the results of each model with some examples of correctly and incorrectly identified faces. I also present a table showing the accuracy for each model and a confusion matrix for the best model. I then show the results of the OCR solution. Lastly, I explain scenarios when the solutions perform well and not well.

3.1 CNN Results

The accuracy was high at 89% for training and 92% for validation, and when testing five images it predicted four correctly. I trained the model using a Nvidia GPU, which took 47 minutes and 18 seconds to complete. Figure 4 shows examples of correct and incorrect classification from the CNN. It gets four out of five correct.

[*Figure with face images removed*]
Figure 4. Predicted faces using the CNN model. Face 80 is predicted incorrectly.

Figure 5 shows a graph of the accuracy improving during the training of the model.



Results	
Validation accuracy:	92.21%
Training finished:	Reached final iteration
Training Time	
Start time:	17-Apr-2019 15:22:21
Elapsed time:	47 min 18 sec
Training Cycle	
Epoch:	6 of 6
Iteration:	1320 of 1320
Iterations per epoch:	220
Maximum iterations:	1320
Validation	
Frequency:	3 iterations
Patience:	Inf
Other Information	
Hardware resource:	Single GPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Figure 5. Training progress of the CNN model.

However, when testing this through the RecogniseFace function, the accuracy was only 72% for individual images. It was much worse for group images, at 13%. I believe that resizing the images to 64 by 64 pixels during image preparation and then resizing them for AlexNet to 227 by 227 pixels could have impacted the test accuracy for the CNN.

3.2 SVM with SURF Results

The SVM with SURF features performed much better with 100% accuracy for training and 99% accuracy for validation. It was also much faster to train at three minutes and 30 seconds. When testing this model through the RecogniseFace function, the accuracy was also better at 94% for individual images. Unfortunately, it was still poor for group images, at 22%.

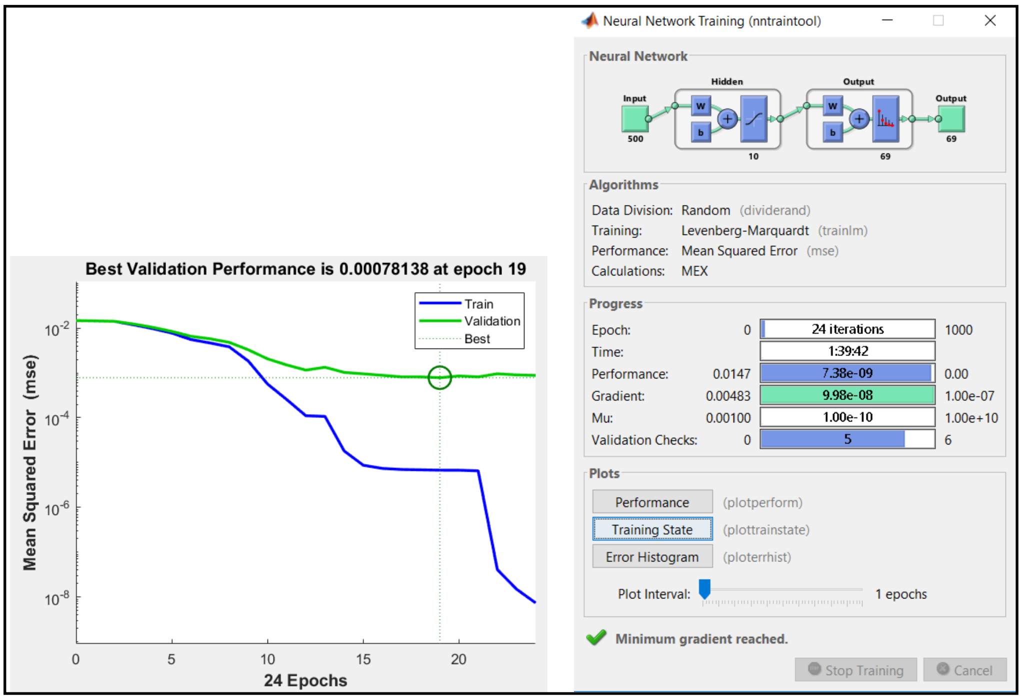
3.3 SVM with HOG Results

The SVM with HOG features performed best of all models with 100% accuracy for both training and validation. When testing 10 images, it predicted all of them correctly. It took 12 minutes and 24 seconds to train. When testing this model through the RecogniseFace function, the accuracy was tied with the SVM SURF model at 94% for individual images. However, it performed much better than all other models on the group images, at 51%. Figure 6 shows examples of correct classification from the SVM with HOG classifier. It is able to match correctly when the person is turned, when they have their eyes closed and when the image is blurry.

[*Figure with face images removed*]
Figure 6. Predicted faces using SVM with HOG model. All faces are predicted correctly.

3.4 MLP with SURF Results

The MLP with SURF features performed very well with 99% accuracy for training and validation. When testing five images, it predicted all of them correctly. It was the longest to train at 99 minutes and 42 seconds. This was likely because the training function used was not supported for GPUs, and therefore it ran on the CPU. When testing this model through the RecogniseFace function, the accuracy was only 83% for individual images. It was the worst performing model on the group images, with only 8% accuracy. Figure 7 shows the performance and the training of the classifier.



Neural Network Training (nntraintool)

Neural Network

Algorithms

Data Division:	Random (dividerand)
Training:	Levenberg-Marquardt (trainlm)
Performance:	Mean Squared Error (mse)
Calculations:	MEX

Progress

Epoch:	0	24 iterations	1000
Time:		1:39:42	
Performance:	0.0147	7.38e-09	0.00
Gradient:	0.00483	9.98e-08	1.00e-07
Mu:	0.00100	1.00e-10	1.00e+10
Validation Checks:	0	5	6

Plots

Performance (plotperform)
Training State (plottrainstate)
Error Histogram (ploterrhist)

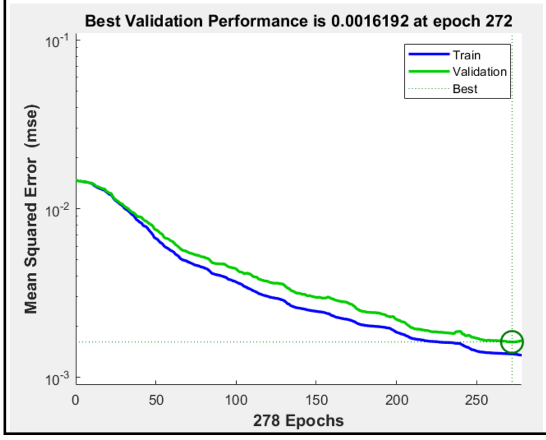
Plot Interval: 1 epochs

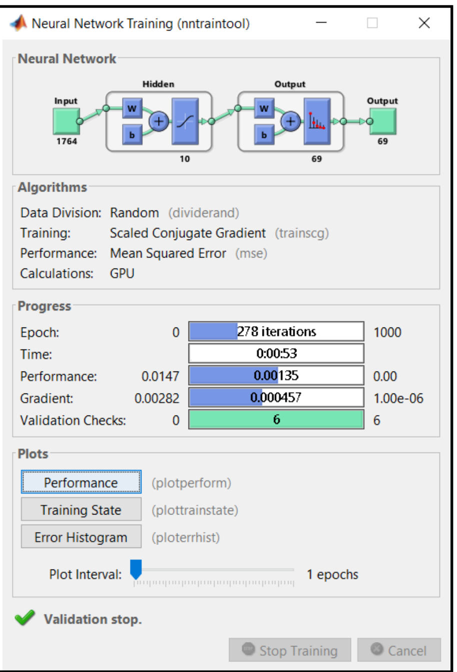
✓ Minimum gradient reached.

Figure 7. Performance and training of the MLP with SURF network. Performance method used during training was mean-squared error, but classification accuracy was also calculated.

3.5 MLP with HOG Results

The MLP with HOG features performed well with 85% accuracy for training and validation. It was the quickest to train at only 53 seconds with the training done on a Nvidia GPU. When testing this model through the RecogniseFace function, the accuracy was only 67% for individual images. It was the second worst performing model on the group images, with only 12% accuracy. Figure 8 shows the performance and the training of the classifier.





Neural Network

Algorithms

Data Division: Random (dividerand)
Training: Scaled Conjugate Gradient (trainscg)
Performance: Mean Squared Error (mse)
Calculations: GPU

Progress

Epoch:	0	278 iterations	1000
Time:		0:00:53	
Performance:	0.0147	0.00135	0.00
Gradient:	0.00282	0.000457	1.00e-06
Validation Checks:	0	6	6

Plots

Performance (plotperform)
Training State (plottrainstate)
Error Histogram (ploterrhist)
Plot Interval: 1 epochs

Validation stop.

Figure 8. Performance and training of the MLP with HOG network. Performance method used during training was mean-squared error, but classification accuracy was also calculated.

3.6 Overall Face Recognition Results

The table below shows the training time and accuracy for all models. The last two columns give the accuracy for these models when they are called through the RecogniseFace function.

Model	**Training Time**	**Train. Acc.**	**Val. Acc.**	**Train. Loss**	**Val. Loss**	**Performance method**	**Model Test image acc.**	**RecogniseFace function test image acc. (individual)**	**RecogniseFace function test image acc. (group)**
SVM with HOG	12 min 24 sec	100%	100%	n/a	0.09%	K-fold cross-validation, Classification accuracy	100% (10 images)	**94%** (18 images)	**51%** (2 images)
SVM with SURF	3 min 30 sec	100%	99%	n/a	n/a	Single-fold, Classification accuracy	n/a	94% (18 images)	22% (2 images)
MLP with SURF	99 min 42 sec	99%	99%	0%	0.08%	Single-fold, Mean-squared error	100% (5 images)	83% (18 images)	8% (2 images)
CNN	47 mins 18 sec	89%	92%	n/a	n/a	Single-fold, Classification accuracy	80% (5 images)	72% (18 images)	13% (2 images)
MLP with HOG	53 sec	85%	85%	0.14%	0.16%	Single-fold, Mean-squared error	n/a	67% (18 images)	12% (2 images)

Table 1. All model results.

The best performing model was the SVM with HOG, especially on the group images, although SVM with SURF did just as well on the individual images. Below is the confusion matrix for the SVM with HOG model showing 100% validation accuracy.

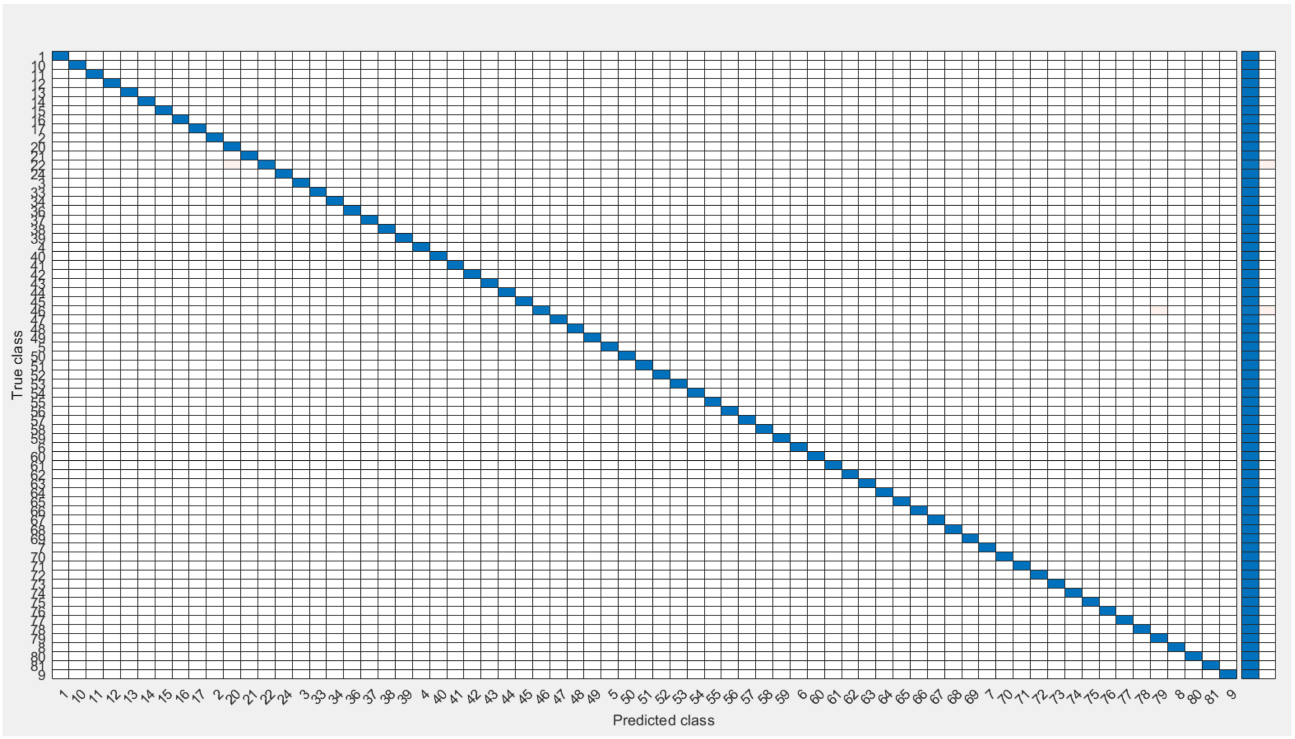


Figure 9. Confusion Matrix for SVM with HOG.

3.7 OCR Results

Below are the test results from testing the final detectNum function using 10 images and 10 videos. The videos did not perform as well as the images, and this was likely due to some images being extracted from videos where the person was moving.

File type	**Performance method**	**detectNum function test image accuracy**
Images	Classification accuracy	**70%** (10 images)
Videos	Classification accuracy	60% (10 videos)

Table 2. OCR results.

3.8 Successful and Unsuccessful Scenarios

With the face recognition solution, the final approaches implemented are successful when testing on individual images for all models, with accuracy ranging from 85-100%. The success is reduced some when the models are combined with the RecogniseFace function, with accuracy ranging from 67-94%. In one image of individuals, the function failed to detect a face, so it failed for all models.

Both MLP models were worse than the other models at identifying the correct face when the person in the test image was turned. The SVM with HOG model showed some success with group images, but it was still only 51%. All the other models performed very poorly on the group images with between 8-22% accuracy.

With the number detection solution, the final approaches implemented are more successful when testing on images with an accuracy of 70%. The videos were less successful with only a 60% accuracy.

4 Discussion

Overall, the solution built during this project was successful in both face recognition and number detection. The RecogniseFace function was able to correctly identify 94% of individual images using the best model. The OCR program also performed well predicting 70% of the numbers held in images.

The SVM with HOG performed much better on the group images than the other models. One theory is that even though it was not trained on the group images, it was more immune to pixel differences from the different lighting and camera used for those images. This is because it uses features which are based on gradients computed from the original images [5]. SURF features do not do as well with lighting and viewpoint changes, which were both exhibited in the group images when compared to the individual images [6].

The biggest weakness in the solution is face recognition in group images for models that perform well on individual images. Also, while all models performed worse during testing of the function compared to model validation testing, the CNN had the biggest difference between the two. Another weakness is number detection in videos, which is not as good as detection in original images.

To expand on this work, it would be interesting to try additional options to see if anything can improve the results, especially in the weak areas mentioned above. For face recognition, although it would be time-consuming, one thing that could make the biggest difference for group accuracy is adding faces from the group photos to the training database. The group images were taken in different lighting and with a different camera from the individual images. Both of these factors can impact the features used to train the models which could explain the poor group results. I would also like to try converting all images to grayscale during pre-processing before running them through the models to see if this improves accuracy. Additional hyperparameters, such as activation function or adding momentum, could be tuned for the MLPs. The MLP with HOG model may see improvements with a different training function, such as the one used for the MLP with SURF. For the CNN, I would not start by resizing the images to 64 by 64 pixels (and then again to 227 by 227 pixels for AlexNet), as I believe this may have impacted the accuracy. Another experiment to try would be including more images overall in the training database. In order to do this, there would need to have been more photos taken at the start, or more images could be extracted from individual videos.

For OCR, I would try to improve accuracy for frames from videos by either applying filters to sharpen the images or extracting more frames per video and identifying the one with the highest OCR character confidence.

5 References

[1] iMazing (2019) *iMazing*. Available at: https://imazing.com (Accessed: 28 January 2019).

[2] Islam, M., Mondal, C., Azam, M. and Islam, A. (2016) 'Text detection and recognition using enhanced MSER detection and a novel OCR technique', *2016 5th International Conference on Informatics, Electronics and Vision (ICIEV),* Dhaka, 13-14 May, 2016, pp. 15-20. doi: 10.1109/ICIEV.2016.7760054 (Accessed: 23 April 2019).

[3] Jalali, S. (2019) 'Lab 5 Instructions'. *INM460: Computer Vision*. Available at: https://moodle.city.ac.uk/mod/resource/view.php?id=1124769 (Accessed: 18 February 2019).

[4] Jalali, S. (2019) 'Lab 6 Instructions'. *INM460: Computer Vision*. Available at: https://moodle.city.ac.uk/mod/resource/view.php?id=1124779 (Accessed: 4 March 2019).

[5] Jalali, S. (2019) 'Lecture 5: Introduction to machine learning for image classification (Supervised Learning)' [PowerPoint Presentation]. *INM460: Computer Vision*. Available at: https://moodle.city.ac.uk/mod/resource/view.php?id=1259020 (Accessed: 18 February 2019).

[6] Jalali, S. (2019) 'Lecture 6: Unsupervised Learning, Feature Extraction' [PowerPoint Presentation]. *INM460: Computer Vision*. Available at: https://moodle.city.ac.uk/mod/resource/view.php?id=1259020 (Accessed: 18 February 2019)

[7] Mathworks (2019) *Automatically Detect and Recognize Text in Natural Images*. Available at: https://uk.mathworks.com/help/vision/examples/automatically-detect-and-recognize-text-in-natural-images.html (Accessed: 31 March 2019).

[8] Mathworks (2019) *Deep Learning Toolbox Model for AlexNet Network.* Available at: https://uk.mathworks.com/matlabcentral/fileexchange/59133-deep-learning-toolbox-model-for-alexnet-network (Accessed: 14 April 2019).

[9] Mathworks (2019) *Transfer Learning Using AlexNet.* Available at: https://uk.mathworks.com/help/deeplearning/examples/transfer-learning-using-alexnet.html (Accessed: 14 April 2019).

[10] Mathworks (2019) *vision.CascadeObjectDetector*. Available at: https://uk.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html (Accessed: 13 March 2019).