Computer Vision Coursework Report

Miguel Bravo

Project Overview

This project is about developing a Computer Vision system which uses machine learning models to detect and identify individuals in images. The models are trained on a dataset of individual and group images of students on the Computer Vision module. Using these models the system must try to identify students in new images, while also flagging up unknown individuals not included in the original dataset. This involves implementing an end-to-end pipeline covering a range of key steps from pre-processing training images and training models, to deploying these models on new test images.

In this report I will explain my implementation of this system, focusing on the following key aspects: 1) Pipeline implementation and key methods used; 2) Quantitative analysis of key modelling results; and 3) Critical evaluation of results and suggestions for future work.

Implementation Overview

The overall system implementation pipeline involves the following key steps: 1) Pre-processing of images in the original dataset in preparation for model training; 2) Training and testing of models; 3) Deployment of end-to-end face recognition system.

These steps are explained in more detail below. The system has been entirely implemented in the programming language Python using publicly available libraries as well as my own utils package with custom methods and functions for this implementation. The code, execution instructions and model outputs are included in my coursework submission.

Image Pre-processing

This step involves pre-processing the individual and group images of students ahead of model training. First the individual images were manually labelled. Then faces were extracted from both the individual and group images with a pre-trained MTCNN model [1]. The group faces were then automatically labelled with an InceptionResnetV1 model pre-trained on CASIA-Webface [1] to extract face embeddings, which were then fed into a K-Means clustering algorithm to add labels (with labels then manually verified). In total 1599 faces were extracted and labelled and were saved in dedicated class-specific folders for model training. The code for this step is found in *extract_faces.ipynb*.

Model Training

This stage involved training and testing the classifiers to recognise students' faces. First the labelled faces were read in and split randomly 64%/16%/20% into training, validation, and test sets. The split was applied consistently across individual/group faces and classes to ensure the models would be trained, validated and tested on examples from each class

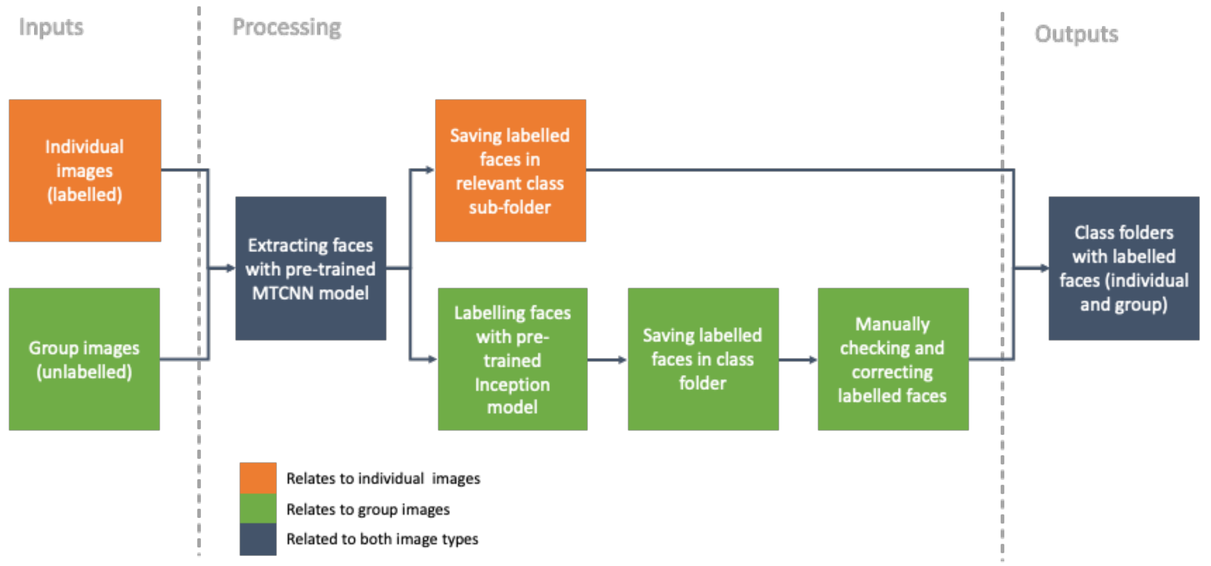
Inputs
Processing
Outputs
Individual images (labelled)
Group images (unlabelled)
Extracting faces with pre-trained MTCNN model
Saving labelled faces in relevant class sub-folder
Labelling faces with pre-trained Inception model
Saving labelled faces in class folder
Manually checking and correcting labelled faces
Class folders with labelled faces (individual and group)
Relates to individual images
Relates to group images
Related to both image types

Figure 1: Diagram illustrating end-to-end image pre-processing approach

and image type. The training set was then augmented with random horizontal flipping, gaussian blurring (std. 0-3), rotation (+/-15 degrees), and brightness adjustment, with some differences to the application of these transformations per model (explained later). The data then underwent feature extraction using either PCA or SURF, before being used for model training, optimisation and testing. The models considered were CNN, SVM and MLP with each model trained separately per feature extractor, except CNN which does not require a feature extractor. Finally, the trained models were applied on the test set to gauge their real-world accuracy (reported later on in the report) and were then saved along with the feature extractors for later use in the system deployment step. The code for this step is found in *train_models.ipynb*.

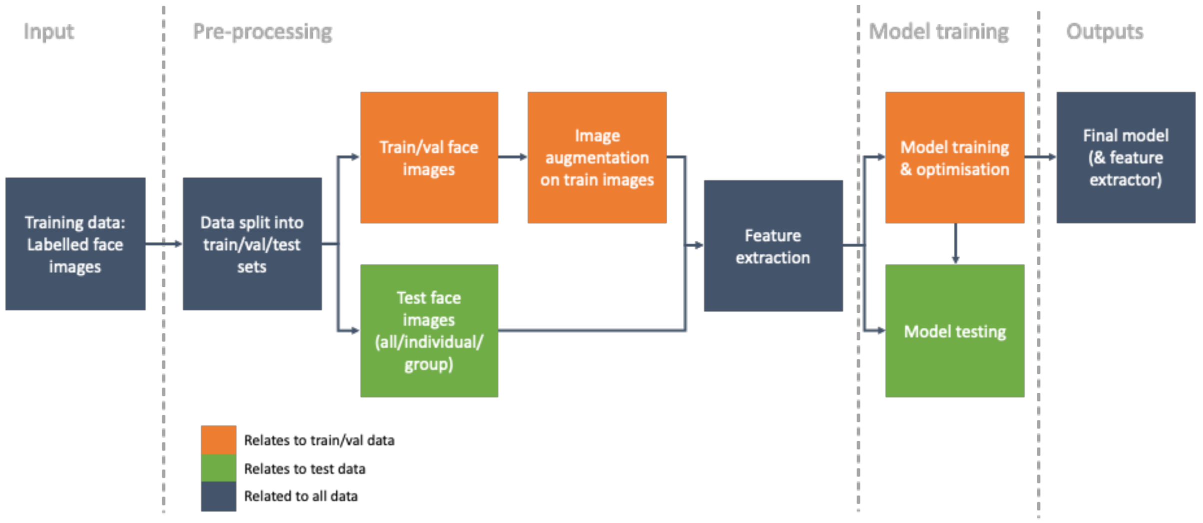
Input
Pre-processing
Model training
Outputs
Training data: Labelled face images
Data split into train/val/test sets
Train/val face images
Image augmentation on train images
Test face images (all/individual/group)
Feature extraction
Model training & optimisation
Model testing
Final model (& feature extractor)
Relates to train/val data
Relates to test data
Related to all data

Figure 2: Diagram illustrating overall model training procedure

In what follows I explain the theoretical and implementation details of each of the feature extractor and ML classifier methods used in this stage.

PCA Feature Extraction

Principal Component Analysis (PCA) is an unsupervised dimensionality reduction algorithm, which can also be used for other tasks such as feature extraction. Informally, it projects the points in a dataset along new orthogonal principal components explaining the original variance in the data. By zeroing out the least important components, this results in a lower-dimensionality projection of the data preserving most of the original information [2].

When PCA is applied on images, the most important principal components capture different key aspects of the image, such as the angle of lighting, and specific shapes. By discarding the least important components, the dimensionality of the image is reduced while retaining the key features, leading to efficient classification [2]. Note PCA feature extraction requires images to be converted to grayscale first.

SURF (Bag of Visual Words) Feature Extraction

Speeded-Up Robust Features (SURF) is a method for detecting key points and their descriptors from images, which can then be used for tasks such as image stitching and classification. To detect key points, it uses an integer approximation of the Hessian blob detector. Key points are chosen where the determinant of the Hessian is maximised. Descriptors are then calculated based on the sum of the Haar wavelet responses extracted across square sub-regions around the key point [3].

To use SURF features for image classification, once the key point descriptors have been extracted these are then used to train a k-means algorithm mapping each key point in the image to one of k visual words (i.e. clusters). These visual words are then counted up in a histogram to give a k-dimensional 'visual bag of word' vector representation of the image, which is then used for classification [3]. Note this feature extraction method requires images to be converted to grayscale first.

CNN (AlexNet)

AlexNet is the Convolutional Neural Network (CNN) architecture used in this project, and was proposed by Alex Krizhevsky et al., for the ImageNet database consisting of millions of images and 1000 classes [4]. It famously won the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), paving the way for the rise of CNNs in Computer Vision research [5]. This was the best performing model implemented in this project, with a test set accuracy of 85.4%. Figure 3 shows a block diagram of its key algorithmic components, sourced from learnopencv.com [6].

The architecture consists of 8 layers. The first 5 are convolutional and the remaining 3 are fully-connected. The convolutional layers are connected via filters (shared weights) convolved around the input and passing the result to the next layer. This allows for more efficient extraction of high-level features from the image with fewer weights since the filters enforce a restricted receptive field input to the next layer's units with shared weights between these units. The fully-connected layers consist of all units in the layer being connected to the previous layer. These layers act as the classifier part of the model, taking the features from the convolutional block as input and returning a 1000-dimensional vector which is then fed through a 1000-way softmax to produce a final distribution over the 1000 classes in the data. The class with the highest probability is then the predicted class.

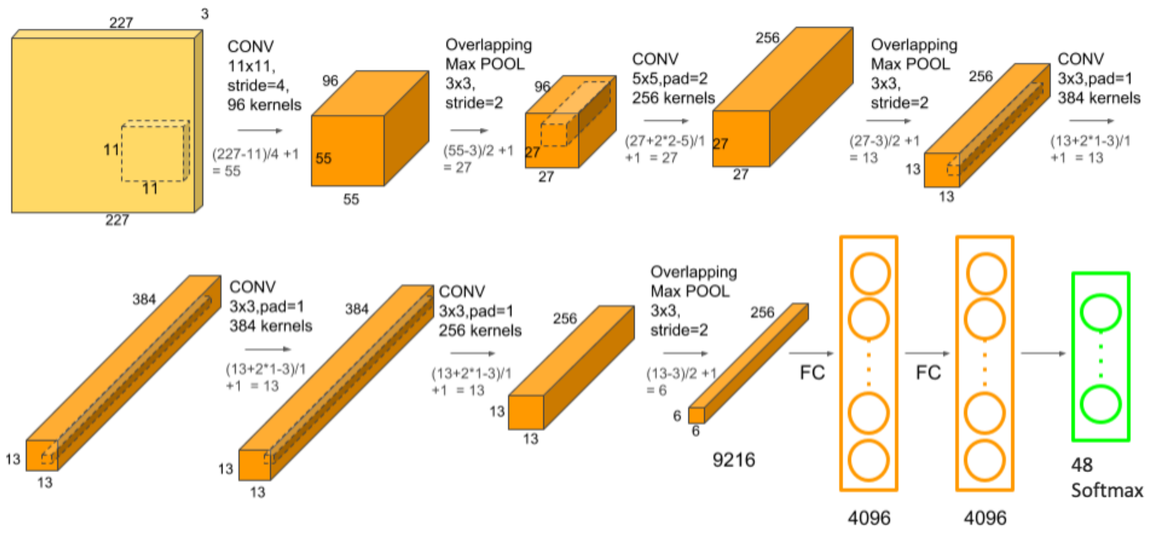
227
3
227
11
11
CONV 11x11, stride=4, 96 kernels
(227-11)/4 +1 = 55
96
55
55
Overlapping Max POOL 3x3, stride=2
(55-3)/2 +1 = 27
96
27
27
CONV 5x5,pad=2 256 kernels
(27+2*2-5)/1 +1 = 27
256
27
27
Overlapping Max POOL 3x3, stride=2
(27-3)/2 +1 = 13
256
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 384 kernels
(13+2*1-3)/1 +1 = 13
384
13
13
CONV 3x3,pad=1 256 kernels
(13+2*1-3)/1 +1 = 13
256
13
13
Overlapping Max POOL 3x3, stride=2
(13-3)/2 +1 = 6
256
6
6
9216
FC
4096
FC
4096
48 Softmax

Figure 3: Block diagram of the AlexNet architecture consisting of 5 convolutional layers and 3 fully connected layers.

The AlexNet model used in this project came pre-trained on ImageNet [7], and was then finetuned for this implementation; i.e. the output layer was changed to 48 units to reflect the 48 classes in this dataset, and then re-trained on these data to optimise the weights for this task. The training used 32-example mini-batch updates over 40 epochs, with the data augmented randomly at each epoch (with transformations as described previously).

SVM

Support Vector Machines is a linear classifier which, informally, attempts to fit a linear decision boundary between datapoints from different classes such that the margin between the line and the closest point from each class is maximised. Importantly SVMs can be combined with kernels which allows the classifier to operate on the data implicitly in higher dimensional spaces, allowing it to fit arbitrarily complex non-linear decision boundaries, and thereby enhancing its expressive power [2].

The SVM for this project was trained with an RBF kernel and with a single full batch update over the entire training set, augmented ten times using the above augmentation routine – bringing the full training set size to 6120 augmented examples. Two separate versions of the model were trained for PCA and SURF features, using grid-search cross-validation to find the best version of these models. For PCA-SVM, the hyperparameters tested were C (SVM regularisation parameter), and the number of PCA principal components. For SURF-SVM the hyperparameters were C, the hessian matrix threshold for SURF key point detection, and the number of clusters for K-means visual word mapping.

MLP

The Multi-Layer Perceptron (MLP) is a neural network model with an input layer, one or more hidden layers, and an output layer. Each layer consists of multiple computing units which are fully connected to units in the previous layer. Each unit's output is a linear transformation of inputs from connected units with a final non-linear transformation. For classification, an input vector is fed into the model to the input layer resulting in the output layer returning an output vector reflecting the different possible classes. The element with the highest value is then the predicted class. The inclusion of at least one hidden layer allows the MLP to approximate any decision boundary, making it a powerful method for classification.

The MLP for this project was trained with a single hidden layer with number of units set as the average between the number of input units and output units, based on the heuristic by Jeff Heaton [8]. The model was trained using 32-example mini-batch updates over 100 full epochs of the training data, with the data augmented randomly at each epoch. As per SVM, two versions of the MLP model were trained for PCA and SURF features respectively, although no grid-search cross-validation was performed to optimise the models.

System Deployment

This stage involved using the trained models and feature extractors (from the previous step) to develop an end-to-end face recognition system implemented in a function called RecogniseFace; as stated in the coursework brief. This function takes a new test image as input, along with the feature extractor and classifier type to use, and returns the same image edited to show the detected faces and predicted IDs, along with a separate matrix showing this in tabular form (also including the location of each detected face in the image).

Figure 4 shows the different components contained within the function in order to support this functionality. First the test image undergoes face extraction using the same pre-trained MTCNN model as before which also captures the location and bounding box co-ordinates of each face [1]. Faces are then passed through the selected feature extractor before being fed into the selected classifier to obtain the predicted ID for each face. These inferred details are all then used to construct the matrix and edited image which are rendered as the function output. The function also has one additional parameter called CreativeMode which if switched on, renders the detected faces in the output image in an artistic style. The code to run RecogniseFace is included in *RecogniseFace.ipynb*. Figure 5 shows example outputs of the function as described above.

Unsuccessful Approaches

Over the course of the project, I initially attempted alternative methods to the ones outlined in my methodology above, which ultimately proved unsuccessful yet highly informative and so are worth mentioning below.

Firstly, I initially trained the models on individual images and tested them on group images. This resulted in very poor real-world performance as faces in the individual versus group images turned out to be very different. This prompted me to include group images in the training set as well, which solved the problem.

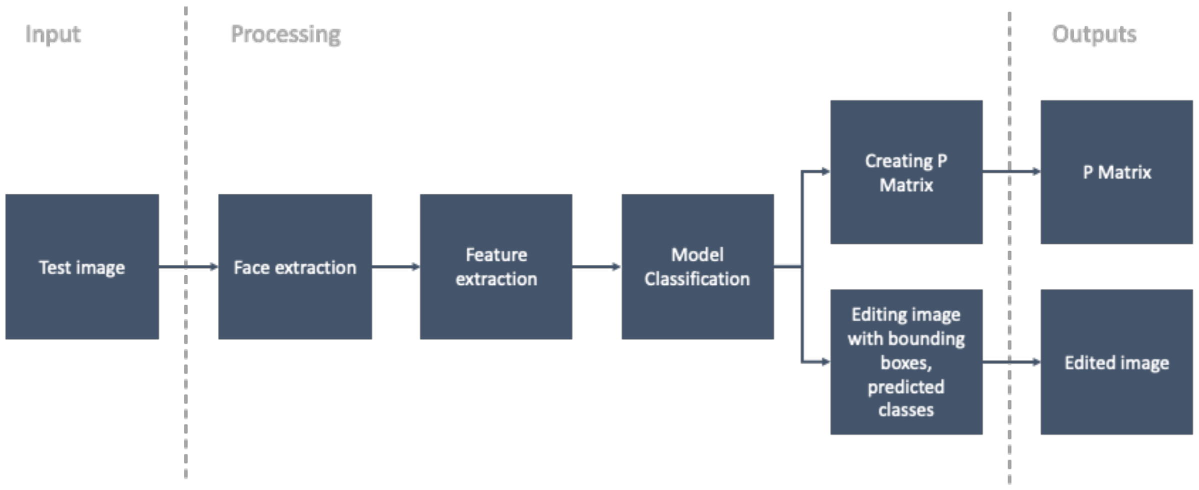

Figure 4: Diagram illustrating end-to-end deployment of the face recognition system, implemented in the 'RecogniseFace' function

Secondly, I initially included extracted video frames in the training set, but noticed these did not improve model performance as they were very similar to their corresponding images produced from the same recordings, so I ended up not using these at all.

Finally, I initially applied a very aggressive augmentation strategy on the training data to improve the robustness of the models, but found this had the opposite effect as the training images became too different from 'real world' images. As a result I simplified down the augmentation transformations significantly.

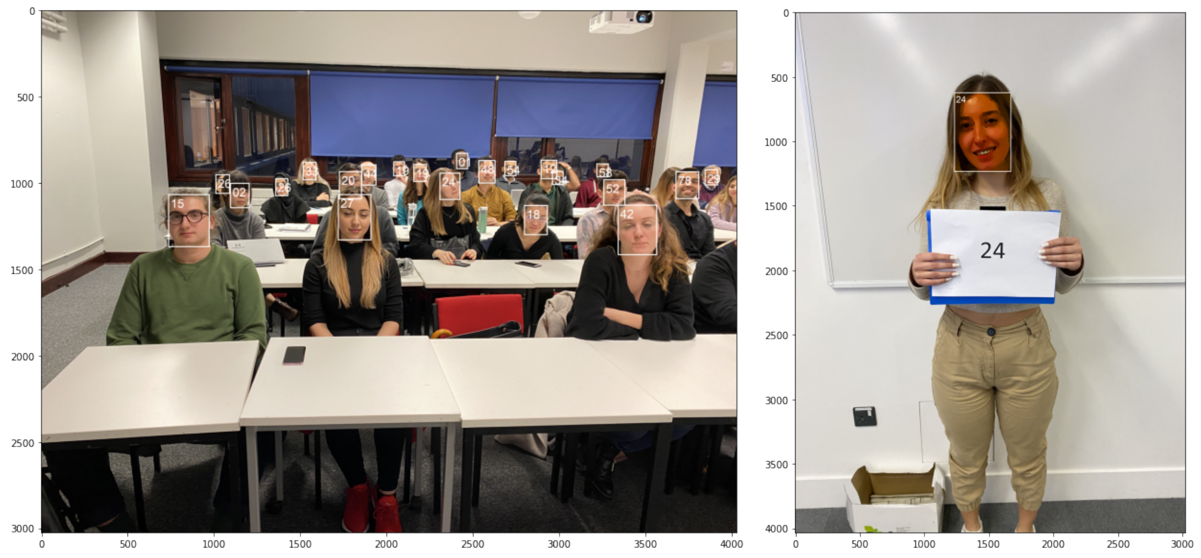

Figure 5: Example outputs of RecogniseFace using CNN classifier. Left – group image output showing detected faces and predicted ID's in the image. Right – individual image output showing detected face and predicted ID, with CreativeMode switched on.

Analysis of Models

Model performance overview

Figure 6 compares the final test set accuracy of each feature extractor/classifier combination, further sub-divided by accuracy on individual vs group images. The best performing model is AlexNet by a significant margin, consistent with the research literature over the last decade. AlexNet's performance is consistent across individual and group images also, with performance on the former being slightly higher – likely due to the higher quality resolution and detail on faces in individual images which gives the model more information to go on when making its prediction.

Feature Extractor + Model	Test Set Accuracy (overall)	Test Set Accuracy (individual images)	Test Set Accuracy (group images)
AlexNet (CNN)	85.4%	87.6%	84.1%
PCA-MLP	77.2%	76.3%	77.6%
PCA-SVM	76.8%	76.3%	77.1%
SURF-MLP	70.8%	76.3%	67.6%
SURF-SVM	65.5%	73.2%	61.2%

Figure 6: Table comparing the test set accuracy of each feature extractor and model combination, with accuracy further subdivided by individual versus group images to see how each model fared on the different image types.

This trend can also be observed in SURF-related methods, but in a more pronounced way, and may again be due to the same reason cited above, though for SURF this clearly has more of an impact. Interestingly, for PCA-related methods the opposite trend is observed. This could be due to the dimensionality reduction of PCA which may result in more information loss of key features on the higher resolution faces from individual images compared to group images, leading to the slightly lower accuracy on the former.

A couple more key points to note: firstly, MLP performs better than SVM regardless of feature extractor, which may be due to differences in the training data augmentation. MLP was trained on more highly augmented data with each training epoch undergoing a different random augmentation over 100 epochs, whereas for SVM the training data was augmented only 10 times before training. Secondly, PCA-related methods perform better than SURF, but it is unclear why this is case and should be explored further in future work.

Best model analysis

In this section I go into more detail on the best-performing model AlexNet, analysing its behaviour during training and testing, and also exploring cases where the model has misclassified examples to understand why this may have happened.

Observing Figure 7, it is noteworthy how quickly the model converged to its terminal accuracy – it rapidly achieved +80% validation accuracy after 10 epochs and then saw little gains thereafter over the remaining 30 epochs. This is likely due to the fact the model was pre-trained on the ImageNet dataset [4], which meant its initial parameters were already highly optimised for image classification. Also notice the significant gap between the terminal training vs validation accuracy, which suggests the model has scope for further performance gains through reducing its variance, for instance, by adding more training data or increasing regularisation [9] – a potential focus for future work.

In Figure 8, we see a more detailed picture of the model's performance on the test set, broken down by class. The confusion matrix shows the broadly high levels of accuracy across classes, while also highlighting specific classes seeing higher misclassification rates, notably 07, 13, and 28. Cross-referencing against the graph on the right reveals the number of misclassifications across these classes: 3, 2, and 4 respectively.

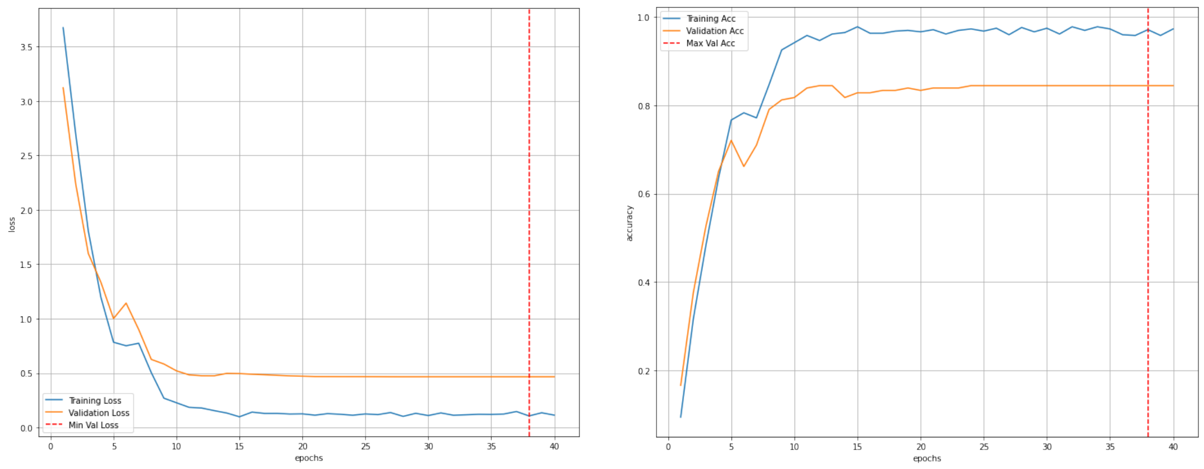

Figure 7: AlexNet training graphs. Left - train/val loss over training epochs. Right - train/val accuracy over training epochs.

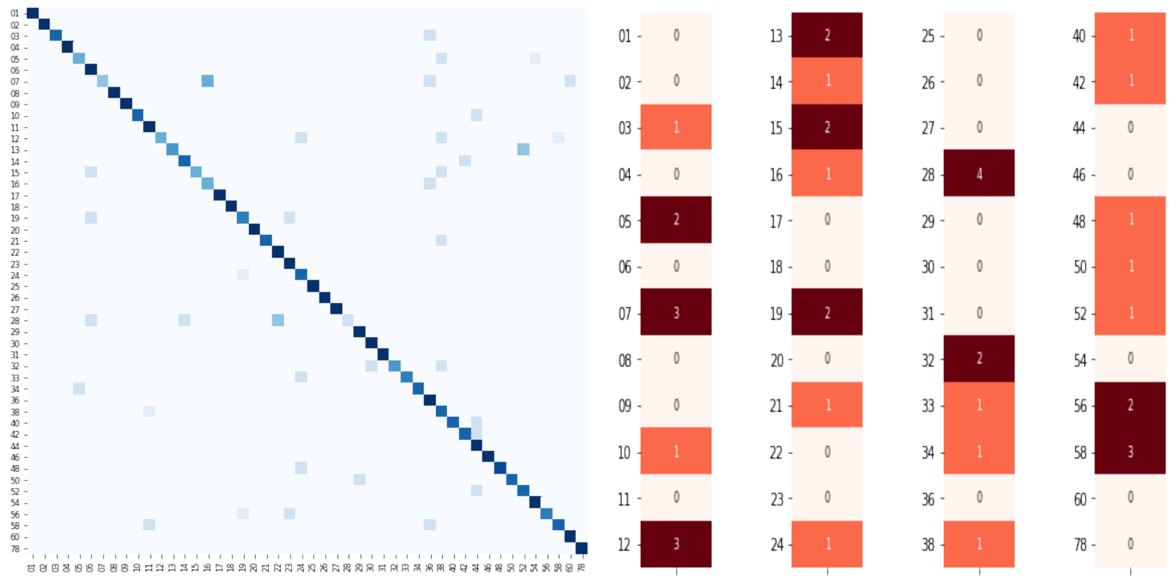

Figure 8: Graphical outputs showing performance of AlexNet on the test set. Left - Confusion matrix showing actual labels versus model predictions (normalised). Right - Graph showing the number of misclassifications per class

In Figure 9, the top row shows the test set examples from classes 07 (left) and 28 (right) and the model's predicted classes for these examples. For incorrect predictions (denoted by red font), the subsequent rows show training set examples from these 'wrong' classes for comparison purposes, to understand why the misclassification may have occurred.

Examining the examples for 07, there is a strong resemblance between 07 and the other individuals the model incorrectly classed 07 as (60, 36, and 16). All four individuals have similar skin colour and facial hair, explaining why the model may have got confused. Looking at the examples for 28, the resemblance with the 'wrong' individuals (22, 14, and 06) is less clear. While 06 does resemble 28, 22 is of the opposite sex and 14 is of a different ethnicity. This indicates the model did not properly learn the characteristics of person 28 – a topic for further research.

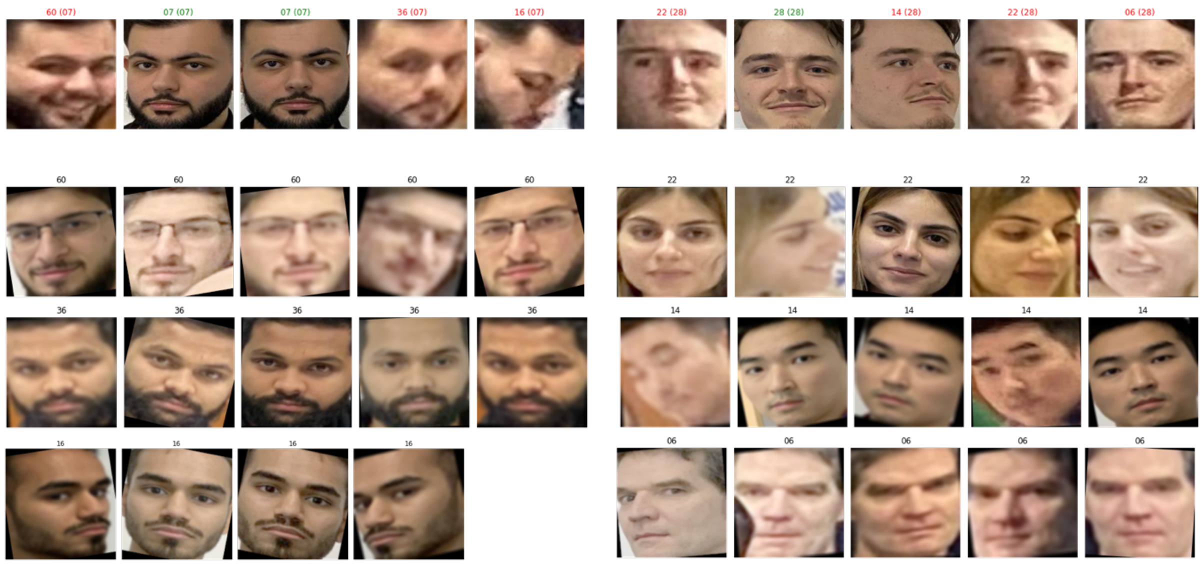

Figure 9: Top row – Test set examples from classes 07 (top left) and 28 (top right), along with the model's predictions. Subsequent rows – training set examples from the 'wrong' classes corresponding to incorrect predictions (for comparison).

Project Evaluation

Strengths

One major strength of my work was the relatively strong performance achieved across all 5 models in the limited time available. The weakest model, SURF-SVM, still achieved 65.5% accuracy, significantly better than random guessing across all 48 classes which would result in 2% accuracy rate (assuming equal probability of guessing each class). Another aspect I was pleased with was the successful deployment of the RecogniseFace function with all combination of methods working correctly including the Creative Mode to render faces artistically. The final strength worth mentioning is the complexity of the code to implement some of the methods not available out of the box in python such as SURF feature extraction which involved writing my own custom methods.

Weaknesses

The main weakness was not accounting for unknown faces in test set images; i.e. faces which do not belong to any of the 48 classes the models were trained to predict. As a result, when such faces show up, the models attempt to classify these faces in vain instead of flagging them as unknown. To address this, I initially tried to set confidence thresholds for each models' predictions below which the classification should be flagged as unknown.

However, this method failed as model's would often return a low score even for correct predictions, resulting in false positive unknowns. This should be a key area for future work.

Another major improvement would be to come up with a more creative augmentation routine on the training images, to further boost models' performance. From examining the CNN's training statistics, it shows clear room for improvement due to the large gap between training and validation accuracy. A common way to plug this gap would be to add more training data (i.e. perform better image augmentation). However, care must be taken to find the 'sweet spot' as overly aggressive augmentation can be detrimental to performance (as observed in my initial failed approaches).

Future Work

In terms of future work, some suggestions have been offered earlier on in the report. Notably, the main questions I would like to focus on for further research would be:

1. Why did PCA feature extraction result in better performance than SURF for both SVM and MLP?
2. Why did AlexNet fail to correctly classify some of the students (e.g. 28) even though there were plenty of examples of these students to learn from in the training data?
3. Would creating separate versions of the classifiers for individual images versus group images result in better performance of the overall face detection system?
4. What is the best way to account for unknown individuals in images, without this resulting in false positive unknowns?

References

[1] T. Esler, "facenet-pytorch," 2019. [Online]. Available: https://github.com/timesler/facenet-pytorch. [Accessed 17 04 2020].

[2] J. VanderPlas, Python Data Science Handbook, Sebastopol: O'Reilly Media, Inc., 2017.

[3] C. Woodruff, "Feature Detection and Matching," 2013. [Online]. Available: https://courses.cs.washington.edu/courses/cse576/13sp/projects/project1/artifacts/woodrc/index.htm. [Accessed 17 April 2020].

[4] Stanford Vision Lab, "image-net," 2016. [Online]. Available: http://image-net.org/about-overview. [Accessed 17 04 2020].

[5] A. Krizhevsky, I. Sutskever and G. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012.

[6] S. Nayak, "Learn OpenCV," 13 June 2018. [Online]. Available: https://www.learnopencv.com/understanding-alexnet/. [Accessed 17 April 2020].

[7] PyTorch, "TORCHVISION.MODELS," 2019. [Online]. Available: https://pytorch.org/docs/stable/torchvision/models.html. [Accessed 17 April 2020].

[8] J. Heaton, "Heaton Research - The Number of Hidden Layers," 01 06 2017. [Online]. Available: https://www.heatonresearch.com/2017/06/01/hidden-layers.html. [Accessed 17 04 2020].

[9] K. Weinberger, "Model Selection," Cornell University, 2018. [Online]. Available: http://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote11.html. [Accessed 17 04 2020].