Computer Vision – Face Recognition and Object Detection with OCR

John McCabe
School of Mathematics, Computer Science & Engineering

Contents

1. Project Overview

The high-level objective of the project was to write two functions. The first function required, was for face detection where the faces were to be drawn from a corpus made up of the MSc Computer Vision class. The testing images fall into three categories, a single person, a group photo containing multiple people and an image without anyone in the corpus. The function prototype is defined as:

P = RecogniseFace(I, featureType, classifierName)

Where *featureType* is a feature extraction technique and *classifierName* is a classification method that can be coupled with each feature extraction method. Additionally, a Convolutional Neural Net (CNN) is to be accepted as a classifier.

The second phase of the project concerns object detection and specifically identifying the number situated in the middle of a white card as held by a member of the class.

The function prototype is defined as:
P = detectNum (filename)

Where filename can be the path to an image or video.

Both phases require a pipeline that involves data preparation, preprocessing, training and evaluation. The detail is laid out in the following sections as is the full description of the final approaches.

2. Data Preparation

Before starting on either phase of the project it is necessary to produce a structured dataset of images of the class. As there are only a small number of images per person, it is necessary to extract stills from the videos to supplement the images.

The pre-processing threw up some interesting challenges when dealing with images and videos specific to having been produced from smart phones.

The image and video formats are in proprietary formats having been collected using both Android and Apple devices. This highlighted the challenges with compatibility across hardware. Apple collected images were not supported on a Windows machine, requiring an unsupported codec to extract the stills.

In order to produce the training data, it was necessary to manually arrange the images in the required folder structure. Each image was manually identified and copied to its own folder, identified by the number on the card the person was holding. As will be seen later, an automated labeling was not possible as the accuracy levels were not high enough.

If there had been less structured data collection i.e. individuals not having consecutively numbered images or in some cases predefined folders it would have resulted in an expensive manual identification and cross-checking process. This provides good insight into the importance of curation in production environments as well as highlighting areas where errors can be introduced.

3. Face Recognition

3.1. Overview

This phase of the project presents a classical view of face detection where features are extracted and used to train a classifier against the state-of-the-art CNN.

CNN's are recognised as the primary solution for a wide range of machine learning problems and are established as the best performing solution for face recognition as highlighted by Taigman et al.[i] in the paper on DeepFace and as such the expectation is the CNN will get better results than using the classical techniques. There are however some words of caution highlighted by Balaban[ii] and more recently Ribeiro et al.[iii] as to whether the black box nature of CNN's are the panacea they appear to be. This project will provide a glimpse at the effectiveness of classical techniques against a CNN.

3.2. Feature extraction and classifier

3.2.1. Implementational overview

The steps required to develop face recognition system using traditional computer vision methods and the steps that are followed in the development of the *RecogniseFace* function are as follows.

a. Pre-process data to extract the region of interest (ROI). Covered in detail in section 3.1.2.
b. Split data into training and test data. In order to ensure that there are no dominant images the training is balanced so that the same number of training examples of each person is used in training.
c. Extract features, the technique is problem specific and is explored further in section 3.1.3.
d. Use features to produce visual vocabulary.
e. Train classifier on features extracted from training data. Classifiers are covered in more detail in section 3.4.
f. Optimize classifier hyperparameters via grid search and cross validation.
g. Select best performing configuration for classifier.
h. Evaluate model on test data.
i. Save best performing model for use in future predictions

Figure 1 shows the major steps and sub processes that will be followed for feature extraction and classifier face recognition.

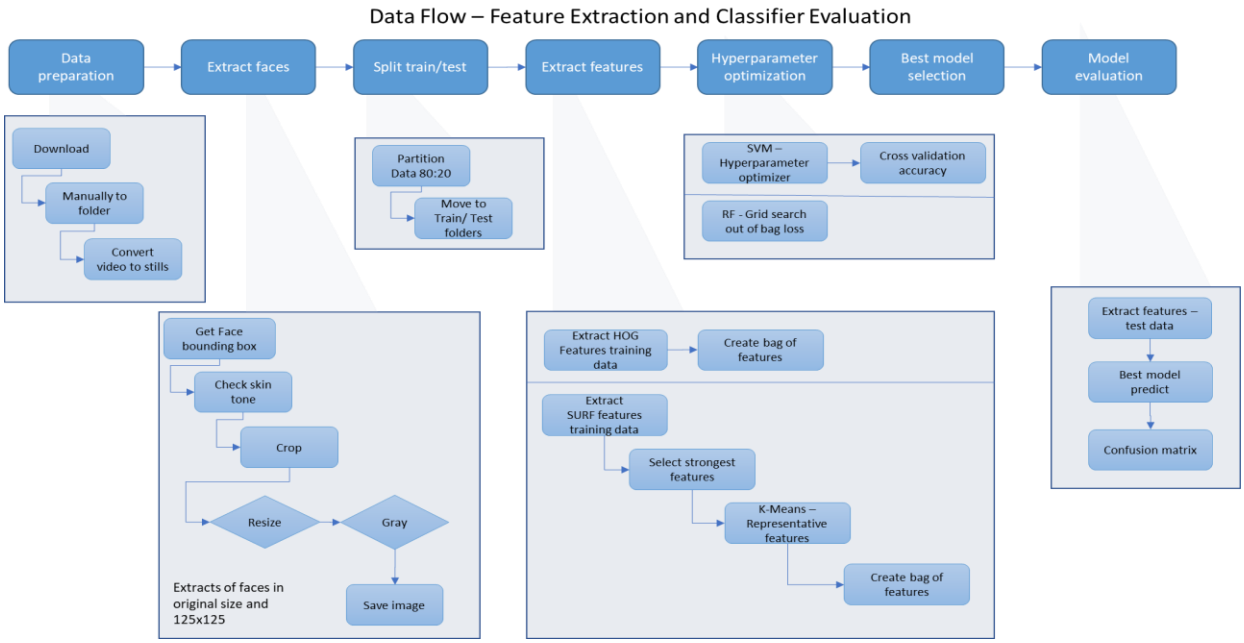

Figure 1. Data flow and subprocess of feature extraction and classifier optimization

3.2.2. Pre-processing

Regardless of the method used to classify the image whether it is feature extraction and classifier or using a CNN it is necessary to "extract" the faces from the training images. If the whole image is used, then features will be extracted or learned based on the whole image and as such will not generalize well when faces are presented out of context. This will be exaggerated in the group image setting.

The out of the box Matlab implementation of the vision.CascadeObjectDetector was used to isolate the bounding boxes of all faces within an image.

Several false positives were calculated so a a post-processing step was added based on skin tone[iv] to remove non-face regions. This proved to be successful in filtering out most but not all the false positives.

In order to ensure consistency in testing, the face extraction process was packaged up into a single function with the following prototype:

[BBoxes, aPI, howMany, sizes] = ExtractFaces(I, varargin)

Parameter values include options to save to disk, convert to gray scale and target size.

3.2.3. Feature Types

Reviewing the lecture notes HOG, SIFT and SURF would appear to be the best suited feature extraction techniques for face identification as they extract fine grained details whereas GABOR, LBP and BLOB detection such as MSER would be expected to perform poorly as they geared towards edge, fine-scale textures and areas respectively. In order to evaluate the relative effectiveness, HOG & SURF were implemented as feature types of the *RecogniseFace* function. The decision to use SURF over SIFT was primarily motivated by the claimed reduction in computational time.

3.2.3.1. Histogram of Orientated Gradients (HOG)

HOG divides the image into small overlapping blocks and establishes the dominant gradient orientation for each block. A consequence of this technique is that each image needs to be of the same size in order to get the same number of features.

A key decision was what size to rescale to. The motivation here came from the size of the extracted faces from the group photos. Plotting the number of rows by the number of columns as shown in figure 2, provides some insight into the best size to rescale to. Figure 2 shows high density around 125x125. As a result the HOG images were rescaled to 125x125, resulting in 7056 features.

Another key decision was whether to use grayscale images or RGB; here the decision was taken to go for RGB as tone is an important factor in face detection.

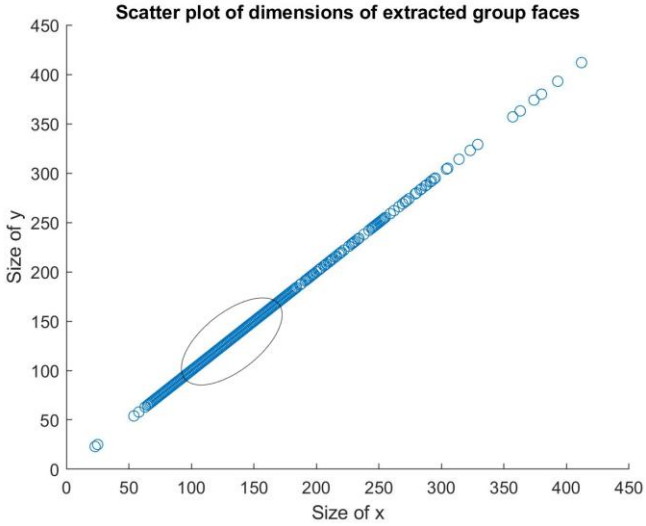

Figure 2. Scatter plot of dimensions of extracted group photo sizes.

3.2.3.2. Speeded Up Robust Features (SURF)

SIFT and SURF are more advanced feature extraction techniques, which use invariant local features. The advantage with using these techniques is the extracted features are invariant to translation, rotation and scale.

Using the out of the box MATLAB implementation of SURF extracts features from each image, selects the strongest and finally using k-means to produce the most representative features of the data.

One of the hyperparameters for the MATLAB implementation is a vocabulary size of 500, which will naturally impact the accuracy of the model so as well as the default models using vocabulary size of 1000 will also be run to explore the impact on the model accuracy by increasing number of features.

3.2.4. Classifiers

3.2.4.1. Support Vector Machines (SVM)

SVM's apply kernels to achieve maximum separation between classes. The choice of kernel function is problem dependent and as our data is high-dimensional and difficult to visualize this would need to be checked.

SVM's are vulnerable to unbalanced data and outliers so this needs to be taken into consideration before implementation. As we have organized our training data to be balanced (see section 3.1.1.), this is not a problem but consideration of the regularization term (e.g. box constraint) to handle outliers will need to be reviewed.

Taking all this into consideration, there are several hyperparameters that we would need to evaluate in order to determine the best configuration for our problem. Fortunately, MATLAB comes with a hyperparameter optimizer, which although takes a long time to run (7+ hrs) it automatically optimizes all the hyperparameters for the SVM as well as running cross-validation. This is the option taken for both the HOG and SURF models.

Table 1 shows the resulting optimized hyperparameters with the out of sample loss. Reviewing the optimizer results (appendix 1) indicates that the HOG implementation has a wide range of values for BoxConstraint and KernalScale all resulting in the optimal objective function, the implication being that the features are effectively separating the various classes. The SURF-500 features model has a poorer performance and the optimizer results point to the data being less clearly separable than the HOG implementation whereas SURF-1000 appears to be more stable and delivers a better out of sample loss.

Feature Type	Out of Sample Loss	Time Taken	Processor
HOG	2.41%	4hrs 20 mins	GPU
SURF-500	6.92%	2hrs 20 mins	CPU
SURF-1000	4.50%	2hrs	GPU

Table 1. Results of MATLAB hyperparameter optimization for each feature type

3.2.4.2. Random Forests (RF)

Random Forests are an ensemble of Decision Trees used on separate parts of the data, designed to overcome the general failings of Decision Trees, namely overfitting and sensitivity to noise. Random Forests also present a statistical outcome for classification, which is an advantage in face recognition as each prediction comes with a measure of confidence.

SVM's and CNN's are black box solutions and the results are difficult to interpret, whereas Decision Trees present interpretability and although Random Forests lose some of this interpretability it is still possible to visualize Tree's within the ensemble again an advantage when trying to determine the causes for misclassifications.

As with SVM, there are several hyperparameters that need to be configured for Random Forests to get the best results; primarily number of trees in an ensemble and minimum leaf size (cases hitting a leaf).

Table 2 shows the hyperparameters tested for each feature type and the out of bag accuracy.

Feature Type	Number of Trees	Minimum Leaf Size	Out of bag accuracy
HOG	100	5	88.7%
HOG	100	10	83.2%
HOG	300	5	92.0%
HOG	300	10	91.8%
HOG	500	5	93.2%
HOG	500	10	92.4%
SURF-500	100	5	84.2%
SURF-500	100	10	77.3%
SURF-500	300	5	87.0%
SURF-500	300	10	84.5%
SURF-500	500	5	89.3%
SURF-500	500	10	85.5%
SURF-1000	100	5	83.9%
SURF-1000	100	10	77.1%
SURF-1000	300	5	87.9%
SURF-1000	300	10	84.2%
SURF-1000	500	5	89.5%
SURF-1000	500	10	86.6%

Table 2. Results of hyperparameter grid search

The best performing model was the 500 trees, min. leaf size 5 for all feature types and are the models saved for use in the final function. Interestingly the performance of the SURF-1000 implementation is only marginally better than the SURF-500 version.

3.2.5. Final Models

After reviewing the model analysis, the following feature types and models have been implement into the *PredictFace* function.

Feature Types	Comments	Classifiers	Comments
HOG		SVM	
SURF	1000 Vocab size	Random Forests	Min leaf:5, Number Trees: 500

Table 3. Final features and classifiers implemented in the PredictFace function

3.2.6. Analysis

The final models all showed extremely high accuracy in the evaluation set (i.e. hold out data) as shown by table 4. In addition to the test set accuracy each model was evaluated against a group photo graph. The percent correctly identified in the group photo is also included.

Feature Type	Classifier	Accuracy	Accuracy Group
HOG	SVM	100%	53%
HOG	RF	97.8%	35%
SURF	SVM	98.6%	29%
SURF	RF	95.7%	14%

Table 4. Evaluation accuracy of feature types and classifiers

Digging a bit deeper into the reasons for misclassification and reviewing each combination in turn …

HOG-SVM

This combination is comfortably the best performing combination on both evaluation accuracy and performance on the group photo. As there is 100% accuracy, the confusion matrix will not offer up an insight.

Reviewing the images that are correctly classified and some examples where

What is hidden by the accuracy is the number of misclassifications that predicted the same person. Figure 3. Shows an example where this is the case.

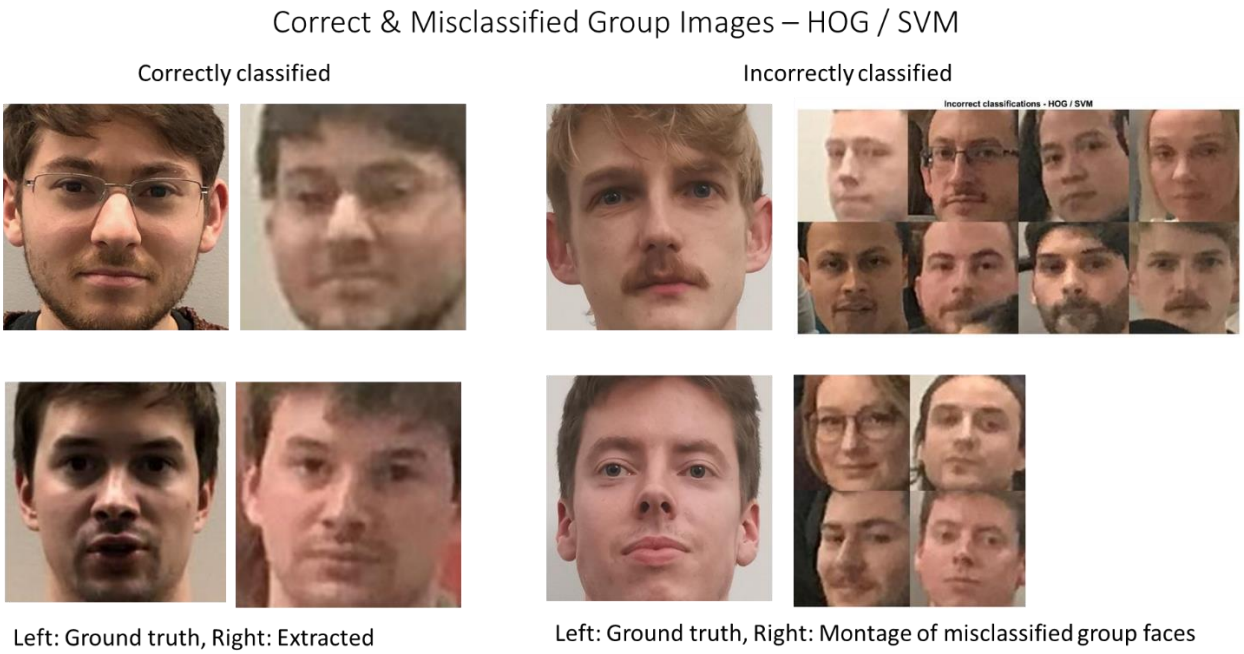

Figure 3. Example of correctly and incorrectly classified images using HOG & SVM

HOG-RF

Figure 4 presents the heatmap in place of a confusion matrix as this provides a clearer view of where the misclassifications are.

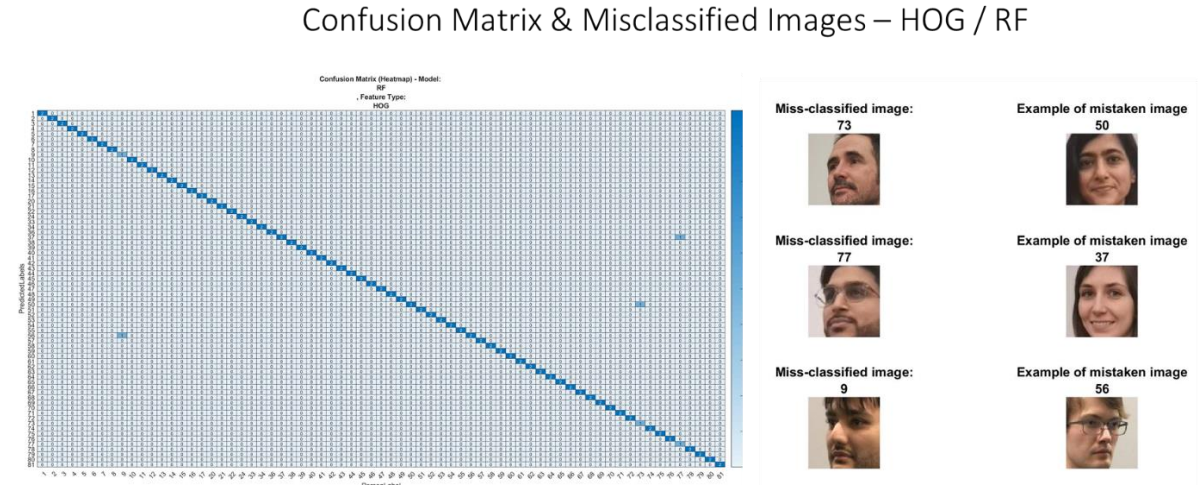

Figure 4. Heatmap representation of confusion matrix for HOG-RF and misclassified images

This shows that there was just the odd image that was misclassified, its not a single image rather, three separate ones. Reviewing the misclassified images, they are all facing the same way, though it is not clear as to the reasons behind this and could be just coincidence. Certainly, the predicted images do not obviously resemble the original image. There is considerable overlap with the correctly predicted images.

SURF-SVM

Across the board SURF is performing worse than HOG especially in the group photo test. Figure 5 shows the confusion matrix and the misclassified images. Also included is a similar montage of misclassifications to the one shown in section 3.1.6.1. This montage does potentially provide some insight; as SURF extract key features based on corners glasses would potentially cause a dominant feature that would hit accuracy. Though in this case even though the target wore glasses they were not identified correctly.

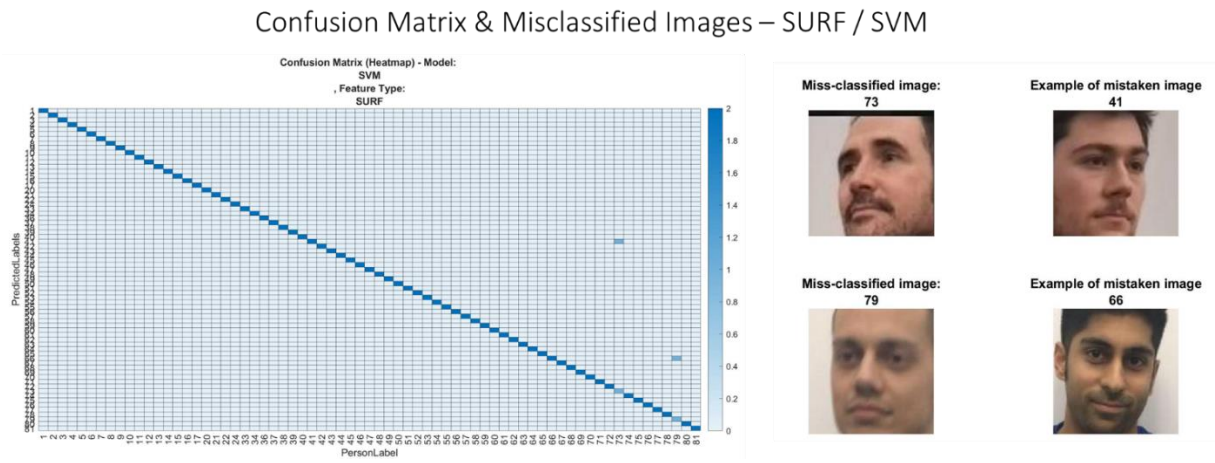

Figure 5. Heatmap representation of confusion matrix for SURF-RF and misclassified images

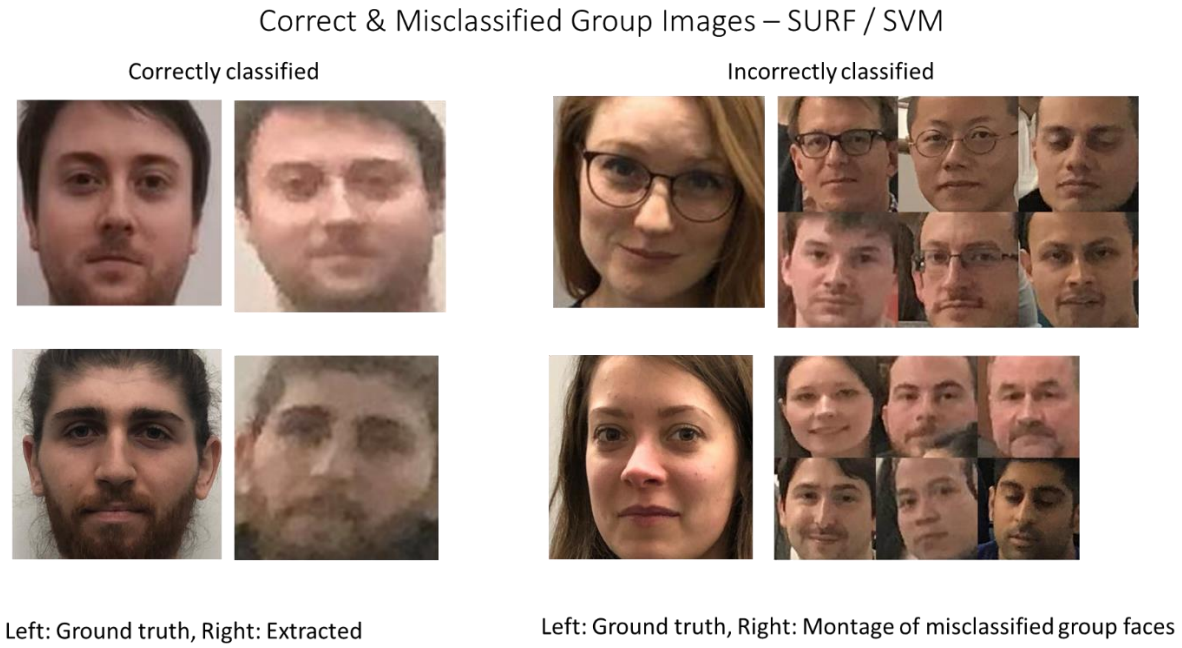

Figure 6. Example of correctly and incorrectly classified images using SURF & SVM

SURF-SVM

The performance of this combination was extremely poor, picking up only the images that were face on and with extremely obvious features.

3.3. Convolutional Neural Networks (CNN)

3.3.1. Implementational overview

There are two potential routes that can be taken in using a CNN for face recognition. The first option would be to build the CNN from scratch, which would involve designing the layers and then training the network with the data. For a CNN to be successful it requires a huge amount of training data; as we only have 10-20 training images for person this is likely to be insufficient to produce an accurate classifier.

The second option would be to leverage existing networks and use transfer learning. This considerably reduces the amount of training data required and reduces the amount of training time required and as such is the option adopted here.

CNN's differ from the classical approach in that there is no explicit feature extraction prior to model training. The CNN effectively learns the features through the various layers of the network. The input images however do need to match the input shape of the pre-trained network.

The steps required to use Transfer learning with a CNN are as follows:

a. Pre-process data to extract the region of interest (ROI).
b. Split data into training and test data.
c. Download pre-trained Neural Network.
d. Augment data, such as rotate and scale. This is required to ensure that the net doesn't memorize the images guarding against poor generalization.
e. Replace last 3 layers so they are consistent with training data.
f. Train CNN
g. Evaluate model on test data.

Figure 7 shows the data flow; this is simpler than for the legacy computer vision approaches.

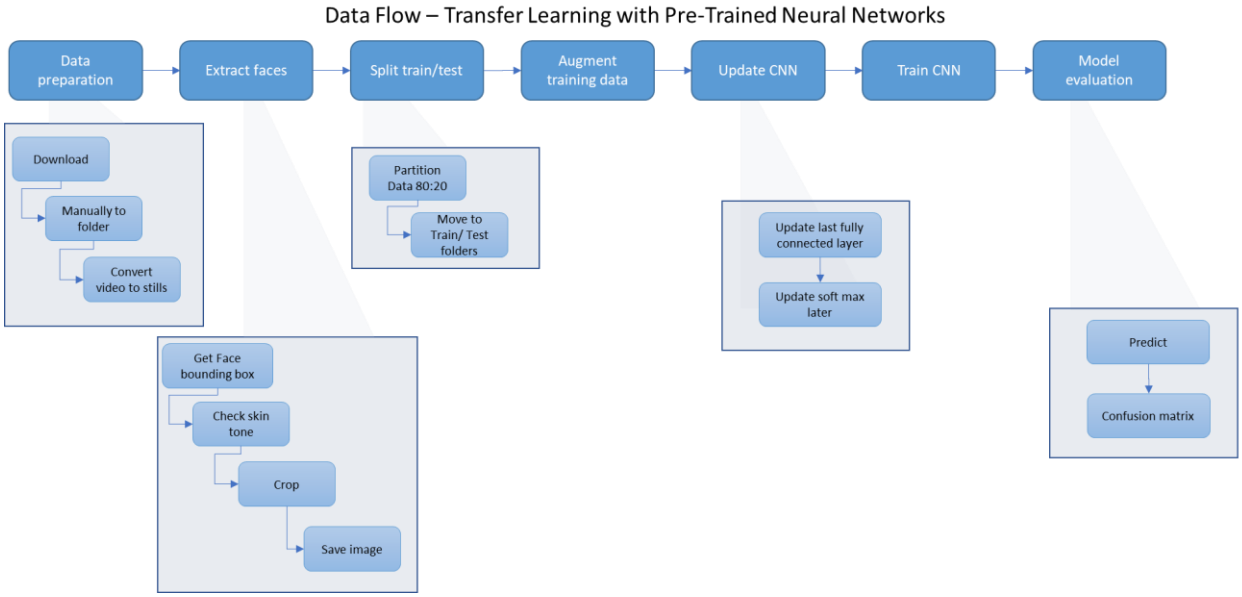

Figure 7. Data flow for Transfer Learning with Pre-trained Neural Networks

3.3.2. Transfer learning and pre-trained Neural Networks

MATLAB comes with many different pre-trained networks so the selection of the appropriate net should be determined by speed of training vs accuracy. Figure 3, as taken from the MATLAB[v] help neatly shows the pre-trained nets and the trade-off between training time and accuracy.

It is clear from figure 8 that Inception-v3 would be an ideal candidate as a pre-trained net, there is another consideration and that is ease of implementation. Outside of the networks available in MATLAB, other pre-trained networks have been trained specifically for face recognition such as vgg-face[vi] and is implemented in MATLAB via MatConvNet. After numerous failed attempts to implement (due to system incompatibilities) this was abandoned. Thus, ease of implementation became an important factor in the selection of the pre-trained net. Taking all of this into consideration AlexNet[vii] was selected as the pre-trained neural net.

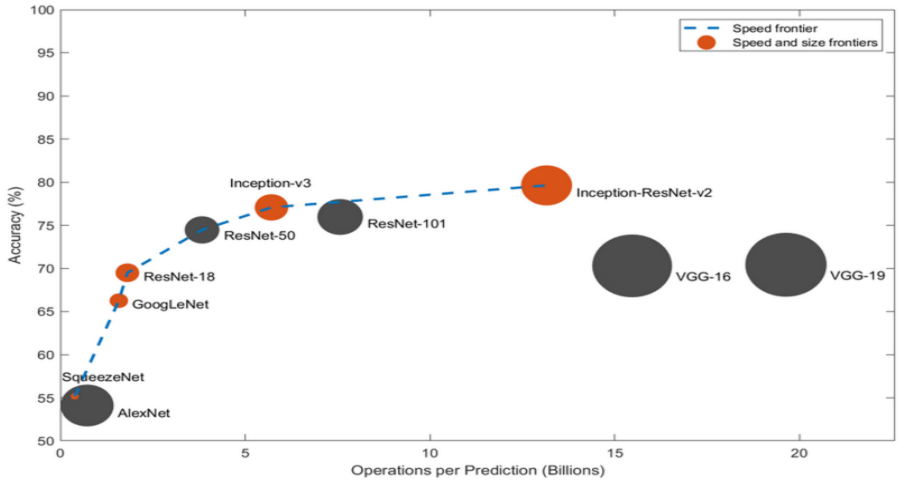

Figure 8. MATLAB documentation showing the merits of the installed pre-trained neural networks

3.3.3. Analysis

The training proved to be fast, certainly when you compared to the computational cost of the SVM optimizer. After training the CNN a validation accuracy of 99.22% was achieved, which would make it comparable to SVM models for just validation accuracy.

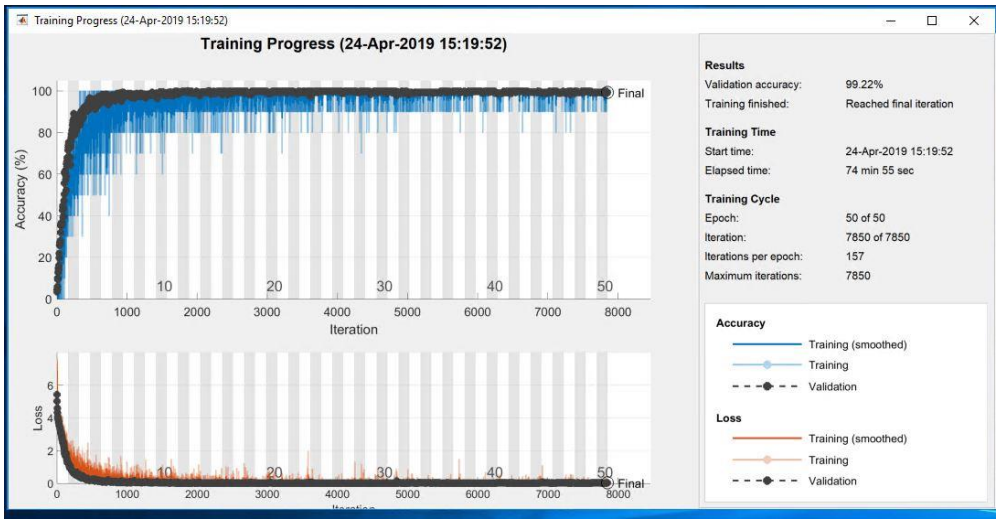

Figure 9. Graphical representation of modified AlexNet training.

Testing against the group photo however only had an accuracy of **12%,** this makes it the worst performing model of all the models.

Figure 10 shows a collection of montages of incorrectly classified images, including an example of the original target image.

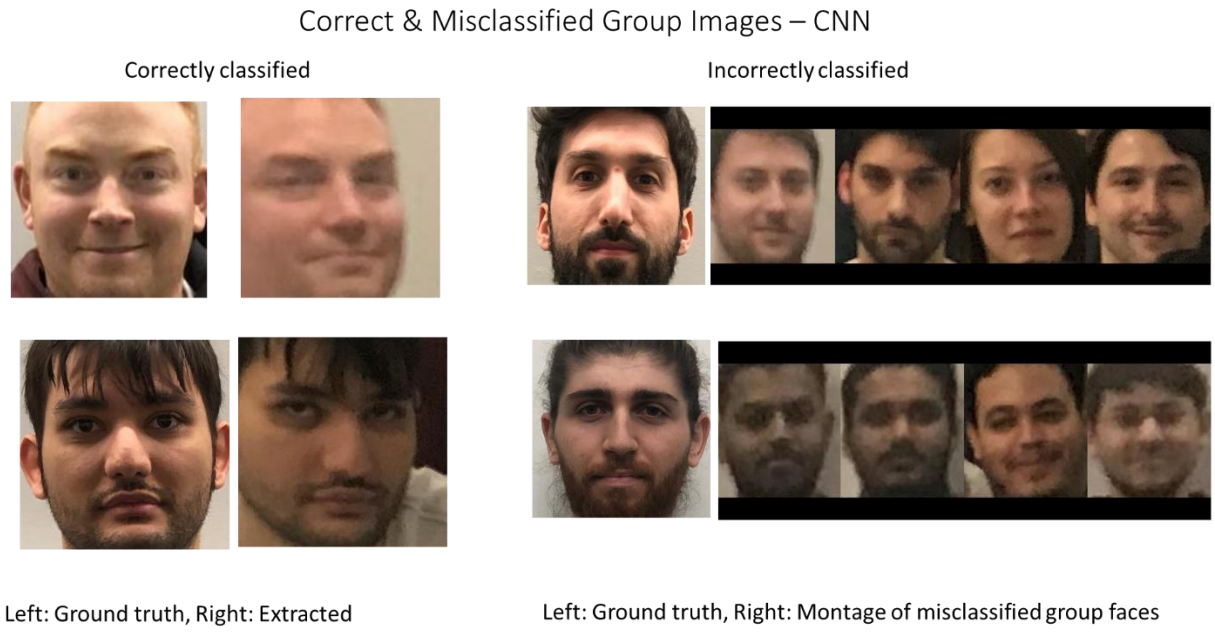

Figure 10. Example of correctly and incorrectly classified images using CNN.

It seems that anyone with a beard causes a lot of false positives, this is also true with skin tone. This montage highlights the poor quality of the extracted group images, which is likely a factor of the poor performance.

3.4. Discussion

The first area that I'd like to delve into is the relative success of the feature types and classifiers explored in this paper. It would appear that HOG features are the differentiator as its these implementations that perform best on the group photo. Considering what is happening under the hood, effectively taking the gradients of the faces rather than extracting invariant features as per SURF and indirectly CNN's it doesn't add up that HOG should perform best. There is of course one thing that happens with HOG that doesn't with the others and that is the rescaling of the images to the majority scale of the group photos. Even though SURF is meant to be scale invariant, I think this is the most likely reason why HOG is out performing all the other models. The low resolution of the resized extracted face images from the group photo is obvious in the CNN misclassification montages and also a major factor in all of the model's accuracy. Faces extracted from the front of the group photo were more likely to be classified correctly, than the low-resolution ones from the back.

The performance for all models on group photo identification would have been improved by adding extracted faces from the group photo. It was a conscious decision to exclude these images, I was keen to see just how well each technique performs in the wild. It is clear from the extremely high, test accuracy that all the models are good at memorizing images adding to the training data from the group photos would have contaminated the training data making it difficult to ascertain whether the models were just memorizing or actually learning.

With regards to the classifiers, SVM clearly outperforms Random Forests proving to be a good general-purpose classifier. From all the literature and buzz around the effectiveness of using CNN's for image recognition, it is clearly not a "point and click" solution as the results here show; though I fully accept that my implementation is compromised by not using a face-specific pre-trained model.

One area that I would have changed was the amount of training data. I supplemented the images with four additional frames from each video, this in hindsight was probably too few. This also contributed to the naive test accuracies, predicting on two test images masks the models deficiencies.

One final area that I'd like to cover is the ecosystem, I took the decision to implement in MATLAB and this had positives and negatives. It is quick to develop in and easy to debug but there are compatibility issues as experienced with MatConvNet, ultimately resulting in the implementation of an inferior CNN. This experience coupled with the enhanced offering of cloud solutions has made me seriously consider whether this would have enabled the development of a superior solution. It is worth noting the availability of face recognition API's such as Amazon Rekognition (online solution only).

4. OCR

4.1. Implementation Overview

It is important to clarify the "rules of the game" for the number identification, specifically the identification of the white card and subsequent reading of the number that will be the objective of this phase of the project. It would be a trivial task to identify the number by the features from the whole image (as per the husky and snow example in Ribeiro et al.).

The task to identify the number on the white card is in many respects the reverse of the face detection task. For the face detection, identifying the region of interest (ROI) was taken care of via out-of-the-box solutions; the main body of the work focused on classifying the faces. The biggest challenge in the *detectNumber* task is identifying the ROI as there are already text classifier built into MATLAB.

4.2. Methodology

There are different approaches that could be taken to identify the white square. These options are covered in subsequent sub-sections. The standard MATLAB Optical Character Recognition (OCR) will be used to identify the number once the card is detected.

4.2.1. Pre-processing

In order to get the best results from the methods, there are several standard transformations that can be carried out on the images. In order to effectively examine the impact of these transformations a preprocessing function was created with the following prototype:

```
[origI, newI] = preprocess(Image, resize, grayscale, sharpen, …
                                    dilate, complement, equalise, reverse, scale)
```

The impact of these transformations will be covered in the analysis.

4.2.2. Transfer learning with a pre-trained CNN

After investigating using transfer learning for CNN it was discarded as an option as the pre-requisites are an extremely large number of training data, far more than the ~700 training images available. The training time was another hurdle, with training time potentially running to days.

4.2.3. Segmentation with BLOB

BLOB analysis identifies connected areas within an image; this would appear to be ideally suited to identifying a white card. Using the code from lecture notes quickly highlighted that illumination is a major issue.

In order to get a targeted approach to the connected area, a grayscale threshold is set (effectively reducing the image to the areas above this threshold). Figure 11 shows the montages of the images produced with different thresholds.

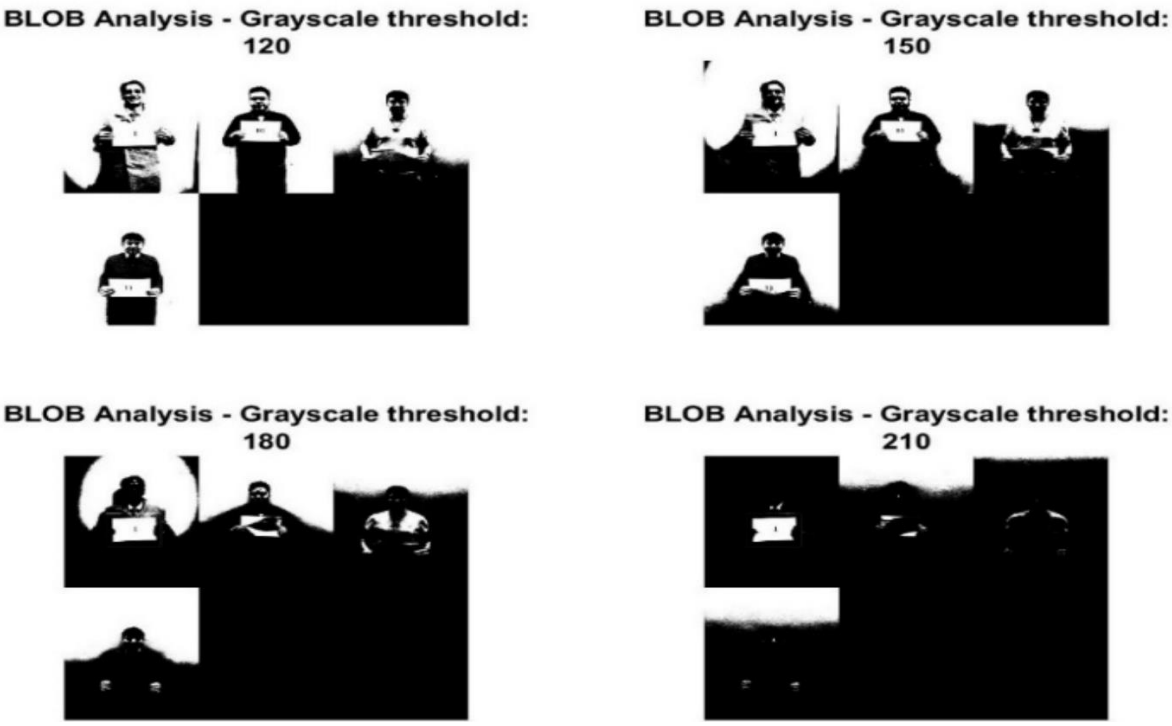

Figure 11. Examples of varying grayscale thresholds for BLOB detection.

It is clear from these montages that just using threshold isn't nuanced enough as the illumination for each image varies and as such the grayscale threshold is specific to each image and of the above images the best accuracy was less than 10% across the whole training set.

Other options explored as part of BLOB analysis, included:

- Using a tone threshold. As the white squares are whitish-gray it meant that it meant that the difference between the channels Red, Green & Blue should be narrow.
- Using statistics to isolate areas based on area size, solidity and eccentricity

All attempts to generalize the identification of the white square failed. Ultimately it is like trying to play Whac-A-Mole, knock down one problem only to cause another one. The main challenges centred around the square blending in with the background (wall and clothing) causing the BLOB to bleed out beyond the card and the bending of the card meant that there as a large variation in illumination on the card itself.

When testing a hyperparameter of the OCR function was also grid searched, that was the confidence of prediction. All results are shown in table 5, and as can be seen the results are very low. Note, due to the time taken to predict only sample of training images were evaluated.

Confidence	120	150	180	210
0.6	0%	0%	0%	0%
0.7	0%	0%	0%	0%
0.8	0%	0%	63%	63%

Table 5. Accuracies using just BLOB analysis and varying grayscale thresholds

4.2.4. Training an object detector

An alternative to using segmentation is to train an object detector. MATLAB's implementation is the trainCascadeObjectDetector function consists of multiple stages, each stage being a weak learner. Boosting (taking the weighted average of the weak learners) results in a highly accurate classifier. That said the detector is not scale or orientation invariant, so the success is dependent on sufficient training samples.

The first step is to label the data, this is done manually via the MATLAB app, Image Labeler. This is a painstaking but important task and the quality of the labeling has a big impact on the quality of the model.

In total the training set consisted of 749 images. The white square for each image was manually labeled before training the object detector.

Figure 12. shows a sample of montages of the results using the final detector. This shows that it was highly effective in identifying the white square. There are a couple of false positives but all in all it is very successful.

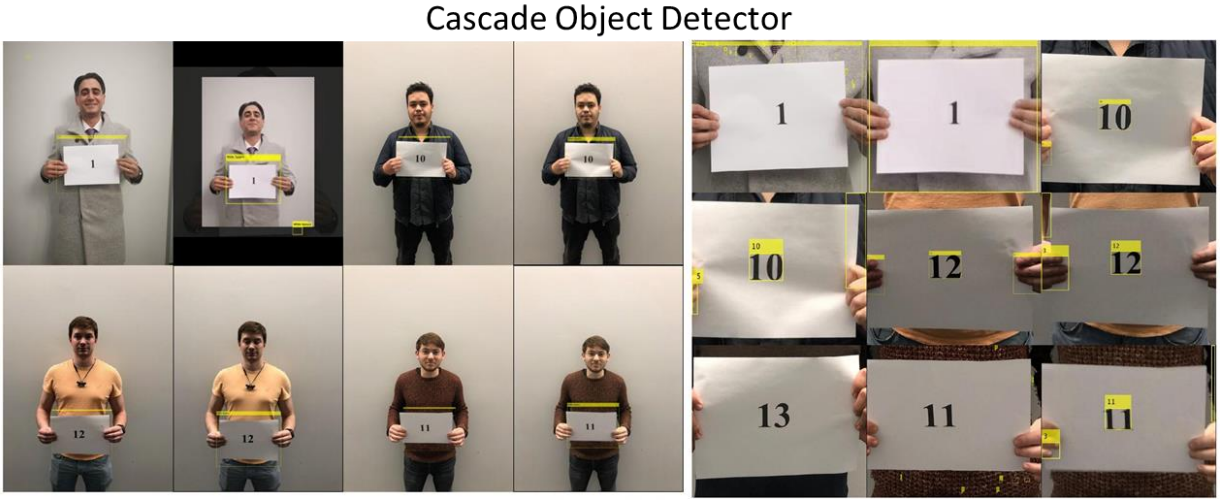

Figure 12. Object detection and subsequent OCR detection using cascade object detection

However, when applying the OCR detector, restricting to the ROI identified by the object detector, it picks up all the noise outside of the white square. Figure 13 shows the results of applying OCR to the detected object. The right side of the image highlights missed numbers and a lot of false positives. Adding a post processing step of reducing the ROI by 50% was extremely effective in eliminating this noise.

The final training set evaluation was 90% whereas the accuracy on the unseen test data set was 80%. Reviewing some of the images that were inaccurately classified it highlights vulnerabilities of the using the object classifier, sensitivities to orientation and scale. This can be seen in figure 13, which shows an example of a misclassified image and the training data.

Figure 13. Left, Images misclassified by object detection & OCR; right, training data.

Although the top image is likely to have been misclassified to blurriness, the other two are orientated towards the left and reviewing the training data there are no images similarly orientated.

4.3. Discussion Section

The problem posed seemed simple, but the development of the solution drew attention to the challenges of identifying objects: illumination; distinguishing from the backgrounds and in the case of stills extracted from videos blurring. BLOB analysis appears to be an inadequate solution whereas a cascade object detector more effective. Increasing the amount training data (potentially augmenting the training data by rotating, skewing and scaling) is likely to have addressed a lot of the scaling and orientation challenges. With regards to the blurring, adding noise to the training data would have been one way of addressing the blurring.

Code Links

Link to source code here.

Function: RecogniseFace.m
Prototype: P = RecogniseFace(I, featureType, classifierName)
Dependencies:
Models/netTransferR2018b.mat
Models/HOGRF.mat
Models/HOGSVM.mat
Models/bag1000.mat
Models/SURFRF.mat
Models/SURFSVM.mat
library/ExtractFaces.m

Input parameters:

I	m x n x 3 matrix, the result of calling imread on RGB Image
featureType	'HOG' or 'SURF' or ''
classifierName	'SVM' or 'RF' or 'CNN

Returns: P - n x 3 matrix, where:

P(n,1)	unique number of person identified
P(n,2) & P(n,3)	x,y co-ordinates of centre face region

Example:

```
I = imread(path_to_image);
P = RecogniseFace(I, 'HOG', 'SVM')
```

Function: detectNum.m
Prototype: n = detectNum(path_to_file)
Dependencies:
Models/whiteCardDetector.xml
library/ applyObjectDetector.m
library/getNumber.m

Input parameters:

filename	filename of image or video file

Name, Value parameters:

'ReturnType'	'one' - returns number on first white square found 'all' - returns numbers from all white cards found Default – 'one'

Returns:

Number	N x 1 matrix of valid integers identified from a white square if exists -1 if no white square was found -2 an invalid number was found

Example(s):

```
%Returns number from still
n=detectNum('path_to_image.jpg')
%Returns number from video
n=detectNum('path_to_image.avi')
%Returns all white card numbers found
n=detectNum('path_to_image.jpg', 'ReturnType','all')
```

Code References

"Pretrained AlexNet Convolutional Neural Network - MATLAB Alexnet - MathWorks Benelux," https://nl.mathworks.com/help/deeplearning/ref/alexnet.html#bvn44n6.
"Pretrained Deep Neural Networks - MATLAB & Simulink - MathWorks Benelux," https://nl.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html.
"Fit Multiclass Models for Support Vector Machines or Other Classifiers - MATLAB Fitcecoc - MathWorks Benelux," https://nl.mathworks.com/help/stats/fitcecoc.html.
"Detect Objects Using the Viola-Jones Algorithm - MATLAB - MathWorks Benelux," https://nl.mathworks.com/help/vision/ref/vision.cascadeobjectdetector-system-object.html.
"Train a Cascade Object Detector - MATLAB & Simulink - MathWorks Benelux," accessed April 26, 2019, https://nl.mathworks.com/help/releases/R2018b/vision/ug/train-a-cascade-object-detector.html.

References

[i] Yaniv Taigman et al., "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *2014 IEEE Conference on Computer Vision and Pattern Recognition* (2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA: IEEE, 2014), 1701–8, https://doi.org/10.1109/CVPR.2014.220.

[ii] Stephen Balaban, "Deep Learning and Face Recognition: The State of the Art," *ArXiv:1902.03524 [Cs]*, May 15, 2015, 94570B, https://doi.org/10.1117/12.2181526.

[iii] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin, "'Why Should I Trust You?': Explaining the Predictions of Any Classifier," *ArXiv:1602.04938 [Cs, Stat]*, February 16, 2016, http://arxiv.org/abs/1602.04938.

[iv] Zaher Hamid Al-Tairi et al., "Skin Segmentation Using YUV and RGB Color Spaces," *JIPS* 10 (2014): 283–99, https://doi.org/10.3745/JIPS.02.0002.

[v] "Pretrained Deep Neural Networks - MATLAB & Simulink - MathWorks Benelux," accessed April 24, 2019, https://nl.mathworks.com/help/deeplearning/ug/pretrained-convolutional-neural-networks.html.

[vi] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman, "Deep Face Recognition," in *Procedings of the British Machine Vision Conference 2015* (British Machine Vision Conference 2015, Swansea: British Machine Vision Association, 2015), 41.1-41.12, https://doi.org/10.5244/C.29.41.

[vii] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," in *Advances in Neural Information Processing Systems 25*, ed. F. Pereira et al. (Curran Associates, Inc., 2012), 1097–1105, http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.