Implementation and comparison of facial recognition methods

Aman Berry
City University, London
aman.berry@city.ac.uk

Abstract

In this paper we seek to implement a facial recognition pipeline using several different models, to test each of their benefits and drawbacks. The methods we test over are different feature extractors (SIFT and SURF), and different models (SVM, RF, CNN). We look at data augmentation methods, then build and test the entire pipeline based on this augmented dataset. We then present the results of each method on a training, validation and unseen test set.

1 Introduction

In this paper we seek to implement a series of machine learning algorithms on a provided data set of facial images in an effort to build different facial recognition pipelines. This comparison is done across different feature extractors and algorithms. State of the art in this field involves the extensive use of primarily convolutional neural networks, which we will also implement on our own data set to test their efficacy in this specific task. However, we will also look at more classical methods such as support vector machines (SVMs) and random forests used in combination with extractors such as SURF[1] and SIFT[2].

Beyond facial recognition, we will also attempt to implement a method of labelling our data set through optical character recognition (OCR) and face detection. This will be achieved through the use of state of the art model RetinaFace[3], along with Google's Tesseract[4] engine. Finally, we look to implement a creative mode which augments our images, transforming faces into cartoons.

In terms of paper structure, we first introduce the dataset and the pertinent information on collection, preparation and augmentation. We also introduce each algorithm used and associated model architectures. Furthermore, we look at the variety of feature extractors used to produce inputs for the classical machine learning algorithms. Finally, we will introduce the OCR and facial detection methods used. We will then describe our implementations of each of these methods, and assess the results and their value. Potential further work will be addressed, and our use of K-means clustering in a creative mode will be outlined.

2 Dataset description

The raw data set consists of images of 48 different students, taken from up to 8 different angles. In these images, each student holds a piece of paper containing a number (the ID to be used as class label), which we will later extract through optical character recognition. In total, there are 331 original images, meaning there is a slight imbalance in the number of images of each student, something we may have to consider later down the line. The number of original images in each class are presented in figure 1.

However, the raw data provided also contains a number of short .mp4 video files, from which we can extract frames to abate the imbalance, as well as further augment the data. The pipeline we therefore want to implement on our raw data is to sort into different folders based on the ID, and then perform different augmentations to better train our models, which we will discuss in depth in section 4.2.

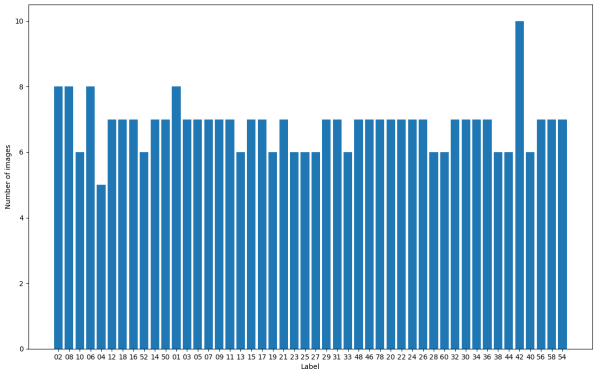

Figure 1: Number of images in each class

3 Methods

In this section, we introduce the methods used for each task in our task. These are summarised in table 1.

Task	Method(s)
Face extraction	RetinaFace with ResNet50
Feature extractors	SURF, SIFT
Models	CNN (directly on images), SVM, Random Forest
Optical character recognition	Frozen EAST detector, Tesseract

Table 1: Summary of methods used

3.1 Facial detection and recognition

We begin by detailing face detection and recognition methods. Face detection took place twice during this task: to extract faces from the individual images present in the raw data, and later in the Recognise_Face function to extract faces from group images to classify.

3.1.1 RetinaFace

We wanted to find a reliable method to extract faces in both of the described use cases. Whilst classic methods such as the Viola-Jones algorithm[5] were effective in extracting faces from the individual images, their performance faltered in group images. Thus we explored options and found that RetinaFace[3] was close to state of the art both in accuracy and speed. Furthermore, this method also allows for changing of parameters to fine-tune face detection in the different brightness, blur and size scenarios present in the test images, so we opted to include this lightweight model in our pipeline.

The RetinaFace model is a fine-tuned ResNet-50[6] model. In particular, ResNet-50 is retrained on the WIDER FACE dataset[7]. The authors of the RetinaFace model augmented their training set by labelling landmarks on faces present in WIDER FACE, and used these landmarks as an extra supervision signal in the model. Furthermore, the model also predicts on each pixel to create a 3D mesh of a face which is then projected back to the 2D image. These two methods in combination allow for very accurate predictions, especially in our images which are relatively uncomplicated.

As mentioned, using RetinaFace also allowed us to tune parameters to best fit our use cases. In particular, we set confidence and visibility thresholds to remove partially hidden or reflected faces, to allow our later models to predict on fully presented faces. This was performed manually using a number of the test images provided.

3.1.2 Feature extractors

Some models, particularly classical machine learning algorithms, cannot learn directly on image matrices effectively, and instead need to have an intermediate step where the features in the images are extracted and represented in a lower dimension. As such, we use Speeded Up Robust Features (SURF)[1] and Scale-Invariant Feature Transform (SIFT)[2] to solve this task, in the process allowing us to compare their relative effectiveness in our use case.

1. SIFT

The main aim of the SIFT (and SURF) algorithm is to create descriptors of an image's keypoints which are invariant to many transformations (scale, viewpoint, brightness and rotation). These descriptors specifically describe the area around the keypoints. They can then be used to train the classifier. The algorithm itself consists of four stages:

1.1 Scale-space extrema detection In the first step, the algorithm aims to identify potential keypoints through the use of Difference of Gaussians (DoG). Typically, these keypoints would be identified through the Laplacian of Gaussians, but this is far too computationally intensive. As a result, the DoG is used to approximate this Laplacian. The DoG is convolved with the image to find the extrema (potential keypoints) of the image.

1.2 Keypoint localisation This initial set of candidate keypoints is then refined through keypoint localisation. Here, certain types of extrema are removed for having properties not relevant to keypoints. Per the paper, the Taylor expansion of the scale space is computed to attain a more accurate location of the potential keypoint. At this stage, the intensity and curvature are calculated, and compared to set thresholds. If the intensity or curvature are too low, the keypoint candidate is discounted.

1.3 Orientation assignment As mentioned previously, one of the transformations the keypoints are invariant to is rotation. Thus in this step, the algorithm assigns an orientation to each keypoint. Gradient directions and magnitudes are calculated in regions about the keypoints, and a histogram created to represent orientation. Based on this histogram, keypoints are created with different directions but the same scale and location.

1.4 Keypoint descriptors The keypoint descriptors are then calculated based on 16×16 regions about the computed keypoints. 4×4 sub-blocks within the regions are used to compute further histograms, which are represented as vectors, i.e. the descriptors. This results in 128 feature descriptors.

2. SURF

SURF is an algorithm used to extract features from images. It was introduced as a sped up version of the SIFT algorithm, which we detailed previously. The methodology is similar to SIFT, yet some key differences arise.

In SURF, keypoints are initially detected using the Hessian matrix. Blobs are detected at locations where the Hessian's determinant is maximal. Whilst in SIFT, the scale-space is represented by the difference of Gaussians, in SURF this is approximated through the use of square filters on integral images (summed area tables). Through changing the size of box filters, different scale spaces can be found, and the use of integral images leads to far faster calculations.

Orientational assignment takes place through the use of Haar-wavelet approximations. In an area around each keypoint, 16 subregions are taken and two Haar wavelets applied for points within the subregion. Four sums of wavelet responses are taken in each subregion, in both directions and with and without signs. These sums are used as the feature descriptors, meaning $16 \times 4 = 64$ feature descriptors.

3.1.3 Algorithms

In this section we look to introduce the machine learning algorithms we used to classify faces, either training said classifiers on features extracted through methods described in section 3.1.2, or directly on images. These models are namely support vector machines, random forests and convolutional neural networks.

1. Support vector machines

Support vector machines (SVMs) are a discriminative model type which seek to compute a manifold to separate classes based on a marginal distance between the "support vectors" and the manifold. The optimal manifold maximises this geometric margin of separation between positive and negative classes. Classic SVMs are a binary classification algorithm, though they can be extended to solve multiclass classification problems.

While it relies on linear separability in its most basic form, it utilises the 'kernel trick' to deal with data which is not linearly separable. The data is mapped to a higher dimension where a optimal manifold can separate it. Everything is then mapped back to the original dimension, resulting in a non-linear optimal manifold.

In the multiclass case (akin to this task), a "1-vs-rest" or "1-vs-1" strategy can be implemented. In the former, for each class, a classifier is trained to observe whether a test case is that class or not. The most responsive classifier will determine the output class of a test image. In the latter case, a binary classifier is trained between each pair of classes. These classifiers will vote for a class at testing and a majority vote taken to decide the classifier's final output.

2. Random forests

Random forests (RFs) are an ensemble method which bring together a group of decision trees. The decision trees in the forest are trained on subsets of the data, along with the trees being split at each layer on random choices of features.

The subsets of the data are formed through sampling with replacement for n training examples. Each tree is trained on a different one of these created subsets, and the process is called bagging. These decision trees each provide a prediction, and in the classification case, a majority vote is taken to provide the random forest's outputs. This is effectively a series of weak learners ensembling through the random forest into a strong learner.

3. Convolutional neural networks

In our solution, we used a selection of convolutional neural networks. We will introduce each of these in this subsection. They are: ResNet-50 (specifically, ResNet-34 and ResNet-50) and an EAST detector, along with RetinaFace which has already been mentioned. We also used a CNN implementation of Google's Tesseract OCR engine to label our images. Due to the general simplicity of our images (clear digits and faces), we chose all of our deep learning methods based on not only accuracy, but efficiency too. Very deep architectures such as ResNet-151 would likely not provide enough of an improvement over simpler architectures in our use case to warrant their use.

Convolutional neural networks (CNNs) themselves are a class of neural networks which take advantage of convolutional layers. In these layers, the image is convolved with filters of set sizes, acting as feature extractors. CNNs learn the values of the filters during training, improving their ability to detect prominent patterns.

3.1 ResNet The ResNet class of convolutional neural networks are so named due to their use of residual blocks. These residual blocks, shown in figure 2, are implemented as a method of solving the vanishing gradient problem. Through backpropagation, repeated multiplication may result in very small gradients, saturating or deteriorating network performance. The residual blocks alleviate this issue by skipping layers, reusing activations from previous layers until weights are learned.

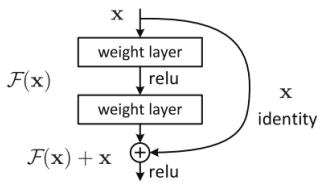

Figure 2: Residual block in ResNet models

The two ResNet models we used are ResNet-34 and ResNet-50. The numbers refer to the depth of the architectures. As described in section 3.1.1, ResNet-50 was retrained on an augmented WIDER FACE dataset to create RetinaFace. For our final facial recognition function, we found that ResNet-50's improvements were too inconsequential to be worth using computationally, so opted for the lighter weight ResNet-34 architecture. The specific architectures are provided in appendix figure 5.

3.2 EAST detector We perform text detection during the labelling of images through the use of the Efficient and Accurate Scene Text Detector (EAST)[8]. We looked for a lightweight and efficient text detector, and settled upon EAST.

The model is a convolutional neural network which aims to output a per-pixel prediction of the existence of text. The architecture consists of three sections: the feature-extracting stem, feature-merging branch and output layer. The stem can be a well known architecture, such as VGG-16[9] or PVANet[10] trained on ImageNet[11]. The branch consists of pooling and convolutional layers which merge the feature maps produced by the stem in a gradual manner. Finally, the output layer uses some further 1×1 convolutions to project the 32 channels of feature maps produced by the branch into a score map and geometry map. The score equates to the likelihood of text in the geometry (bounding box). The architecture in the PVANet case (as used in our pipeline) is presented in the appendix figure 6.

3.1.4 Tesseract

Google's Tesseract engine was used to perform optical character recognition in the image labelling phase of the task. The outputs of the EAST detection on each image were fed into the Tesseract engine. This engine uses a combination of a convolutional neural network trained on MNIST & text images and a static character classifier to identify characters in an image. By preprocessing images, Tesseract can become highly accurate, whilst also remaining very computationally efficient.

3.1.5 K-means clustering

In two separate cases, we use K-means clustering to achieve our aims. Firstly, we will use this method to create SIFT and SURF codebooks. Furthermore, we will use K-means clustering in our implementation of a creative mode, to create "cartoonified" images.

Clustering algorithms are unsupervised methods which aim to group similar data points based on the cluster centres, which are iteratively updated based on the data points within. Thus we can define K-means clustering algorithmically as follows:

Randomly initialise k clusters with centre μ_k
while μ_k not terminal **do**
  assign all data points to each cluster based on Euclidean distance (or other distance measure)
  update centre μ_k with averages of data points assigned to said cluster
  continue iteratively until no changes to $\mu_k \forall k$
end while

4 Implementation

In this section, we detail specifics of our training and testing pipelines and relevant use of the methods outlined in the previous section. Furthermore, we also include details of our creative mode implementation. We begin by outlining our steps from raw data to task completion.

The training pipeline is as follows:

1. Extract faces and label from raw data, using RetinaFace and the EAST+Tesseract combination for each task respectively
2. Augment the labelled images to better represent the faces the models will be tested on
3. Extract features where appropriate and train each of the three model types (CNN, SVM, RF)

The testing pipeline (i.e. the Recognise_Face function) is as follows:

1. Extract individual faces from group image using RetinaFace, resizing for input into models
2. Prepare extracted features for input into SVM, CNN
3. Predict on extracted faces from step 1, labelling the bounding boxes provided by RetinaFace with predicted label

4.1 Dataset creation

To prepare the dataset, we used the EAST detector to find the text labels and Tesseract to recognise the characters. Furthermore, we used RetinaFace to return the faces. RetinaFace worked perfectly out of the box due to the clarity of the faces in the pictures, requiring no adjustment of visibility or confidence thresholds. However, for EAST detection, we made some parametric adjustments. In some photographs, there were objects visible with text which the detector returned. To alleviate this problem, we added a call to our text_detector.py function to only return the largest box of text in each image, which was always the ID number.

We also slightly adjusted the default Tesseract parameters to better suit our task. Firstly, images were resized so that the length of the bounding box on the x-axis was above 100 pixels. Furthermore, we allowed Tesseract to only return numeric characters in its predictions. Together, these two adjustments resulted in almost perfect optical character recognition of the ID numbers. The few (< 10) mislabelled images were manually corrected.

The extracted faces and labels were stored in text files, and a image-label mapping created. This mapping was used to reorganise the images into train/validation sets, with each label having its own folder containing all images of that label. This operation is performed by setup_dirs.py.

4.2 Dataset augmentation

The next step we performed was to begin augmenting the dataset. Firstly, we take advantage of the included .mp4 files: saving a frame every 10 frames, we can add in further pictures. A problem arising from this, however, is that the videos are extremely short (2 seconds on average), meaning the actual frames drawn from the videos do not vary much, and are often almost identical to pre-existing images. Thus, we need to further augment the data.

Looking at the provided class images (which we will later use to test our models), we find that often the individual images are not much alike the extracted faces from the group images. So, if we can perform some distortions and resizing on our training images, we can try to align the new images to the distribution of faces present in the group photographs.

The three augmentations we perform are: resizing photos to 80×80, reducing brightness by a factor of 0.7 and adding a Gaussian blur with a 5×5 kernel. The brightness and Gaussian blur filters were chosen on a trial and error basis, and we can see their effects in figure 3. Visually, the augmented

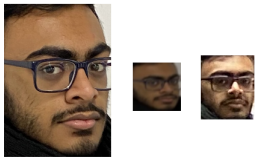

Figure 3: Original individual detected face, same image with augmentations, detected group image

image looks much more in line with the similarly orientated image extracted from a group photograph (test sample).

The result of these augmentations is that the 331 original images become around 5000. The final step in our augmentation is to add some further photographs of detected faces from some of the group images. 30 group images with accompanying videos were provided, and whilst the augmentations in the previous step aimed to mimic the images present in the group photographs, augmenting the data set with actual images drawn from a similar distribution would likely improve our model. Thus we (partially manually) label 20 group images (and by extension, the videos, from which images were taken every 10 frames as earlier), and add them to our dataset.

4.3 Feature extraction

At this stage, we need to prepare SIFT and SURF features for training images, to be fed into SVMs and RFs. For this stage, we wrote the feature_extractors.py function. OpenCV's inbuilt SURF and SIFT detectors were used to detect and compute keypoints and descriptors for each image. Based on the descriptors of each keypoint, K-means clustering was performed to create a vocabulary with size 800 (clusters). Then we create histograms through vector quantization, i.e. a bag of visual words codebook. These histograms are used as the input to our classifiers.

4.4 Model training

For SVMs and RFs, we use scikit-learn's implementations of these algorithms. Firstly, the SIFT and SURF features are scaled to have mean 0 and variance 1. Then, the models are trained and hyperparameters optimised using 5 fold cross validation. For SVMs, we perform a grid search over a range of gamma and C values, whilst also setting a radial basis function as the kernel. For RFs, the hyperparameters optimised are the number of decision trees in the forest and the max depth of the trees. We compute accuracy on the train and validation sets, while also later computing accuracy on the original test set (the group images held out). All results and the hyperparameters used are presented in the results section below.

For the CNN methods, we used the pretrained ResNet weights. We fine-tuned these weights on our own set of ~ 5000 images over 20 epochs, with a decaying learning rate beginning at 0.001 with stochastic gradient descent. We similarly tested accuracy on each of the training, validation and test datasets, and saved models for later use in real-time predictions.

5 Creative mode

The goal of the creative mode was to implement some sort of image augmentation on the final outputs. For the creative mode aspect of the project, we look at implementing a cartoonify method on each of the faces. To do so, we once again use K-means clustering. The method is very simple, yet yields great visual results.

To achieve the desired effect, we want to assign each pixel to a cluster. We will have 16 total clusters, i.e. our final "cartoonified" image will be made up of these 16 colours. These 16 colours are our cluster centres, and we initialise them randomly (alternatively, they could have been initialised to random colours from the image).

Figure 4: Original image and cartoonified image

Each pixel is then assigned to a cluster based on its Euclidean distance from the centre (the pixel values are $0 - 255$). We then update the cluster centre by averaging all the pixels that have been assigned to it, and setting the cluster centre to this new average pixel value. This is done iteratively until there is no change in the cluster centres. The effect of this can be seen in figure

The process is repeated 10 times, and the clustering which leads to the most compact clusters is chosen as the final.

6 Results

All models scored highly when predicting on the training and validation sets. However, this is to be expected as they are drawn from the same distribution, with a lot of repeat or almost repeat images present across both datasets. The high scores do indicate that our models are working correctly, but the ultimate test is of course to examine the model results on the unseen test set. All scores are presented in the table below:

Method	Chosen hyperparameters	Training accuracy	Validation accuracy	Test accuracy
SIFT + SVM	C: 10, Gamma: 0.0005 Kernel: radial basis function	0.909	0.962	0.402
SURF + SVM	C: 10, Gamma: 0.0001 Kernel: radial basis function	0.930	0.972	0.458
SIFT + RF	Max depth: 50 Number of estimators: 1000	0.898	0.957	0.379
SURF + RF	Max depth: 80 Number of estimators: 1200	0.907	0.966	0.364
CNN (ResNet)	Learning rate: 0.001, decaying by 0.1 every 7 epochs	0.962	1.000	0.628

The results of tests on the unseen images were somewhat poor, particularly for the non-deep learning methods. The best accuracy was attained by the convolutional neural network. Clearly, these models failed to generalise well to unseen images with the 5000 training images. The results do, however, clearly show a marked improvement over classical methods by convolutional neural networks. Part of this could be due to the further image augmentations we were able to add into our Pytorch pipeline. All results are available in the notebook files, and final tests using the test.py function.

In the final section, we will go over some ideas of what could be improved in future work.

7 Further work

The key area for huge improvements is almost certainly data augmentation. The most ideal scenario would be to add more images directly (by taking more pictures). Clearly, in this case, this would not be possible. Given more computational power, we could take further steps in our augmentation methods. For example, we set brightness and Gaussian blur distortions to static multipliers and kernels. If we could add to our dataset through a range of values (e.g. brightness in the range $[0.5, 1.0]$ in steps of 0.1, we might be able to improve our models.

Furthermore, adding different types of transformations such as contrast or hue changes might have benefited our model. We noted that some test images were taken with sepia filters which could have thrown off our models, but adding filtered training images would have alleviated this issue.

Adding random crops of faces into the training set would also have added to the model. We saw that sometimes RetinaFace detected half a face in the group images, but our recognition methods weren't trained to deal with this well, particularly in the case of the SVM and RF models.

Another method of improving models could be testing hyperparameters over a larger space, instead of the limited number of values we tested over (due to computational limits). There might well be different hyperparameters that would improve model accuracy. Similarly, we could have tested different architectures (e.g. increase ResNet layers or try different model such as VGG-16). From small tests, ResNet-34 and ResNet-50 were not all too different in their results, and having a lighter backbone model such as one of these allowed our prediction time to be very quick.

References

[1] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 404–417, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. 1, 3

[2] David G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, November 2004. 1, 3

[3] Jiankang Deng, Jia Guo, Zhou Yuxiang, Jinke Yu, Irene Kotsia, and Stefanos Zafeiriou. Retinaface: Single-stage dense face localisation in the wild. In *arxiv*, 2019. 1, 2

[4] Anthony Kay. Tesseract: An open-source optical character recognition engine. *Linux J.*, 2007(159):2, July 2007. 1

[5] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, 2001. 2

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 2

[7] Shuo Yang, Ping Luo, Chen Change Loy, and Xiaoou Tang. Wider face: A face detection benchmark. In *CVPR*, 2016. 2

[8] Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang. EAST: an efficient and accurate scene text detector. *CoRR*, abs/1704.03155, 2017. 5, 10

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015. 5

[10] Kye-Hyeon Kim, Yeongjae Cheon, Sanghoon Hong, Byung-Seok Roh, and Minje Park. PVANET: deep but lightweight neural networks for real-time object detection. *CoRR*, abs/1608.08021, 2016. 5

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009. 5

A Appendix

A.1 Architectures

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix} \times 3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix} \times 8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix} \times 36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix} \times 3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 5: All ResNet architectures, adapted from neurohive.io

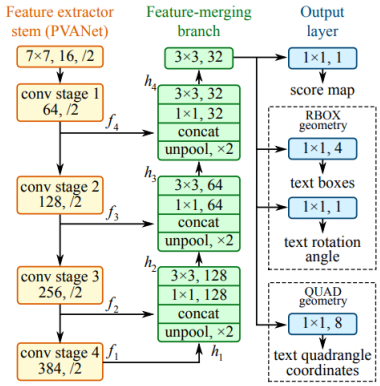

Figure 6: EAST detector architecture, adapted from [8]

A.2 Example output

Figure 7: Example of recognise_face output provided using CNN