COMPUTER VISION ASSIGNMENT

Candidate 184521

UNIVERSITY OF SUSSEX

Table of Contents

Introduction

Face landmark detection is a method used to identify different features of a face found in an image and plot them as points over the face using a previously trained model. This model has many different approaches that can be used for its training from Cascaded Regression to a Multi-Layer Perceptron classifier. In this report we will look at how a Convolutional Neural Network, CNN, can be used and how it must be configured for it to be able to function correctly. This is from pre-processing of images and data to analysing the outputs being used to better adjust training of the model. For this model we will be trying to teach it to discover the 68 feature points of a face from an image, as shown below in figure 1. For reference, a CNN is a deep neural network and functions by layers of connected 'neurons', much like an MLP classifier, assembling a hierarchal pattern of data from patterns found in the given data, this can be seen in figure 2.[2] In this case the data shall be a combination of images of faces and their 68 landmark points, figure 1, therefore patterns between the two can be found and learnt. Therefore, when an image is given to a trained model it should be able to plot the 68 points directly onto a face found in the image.

Fig1. Illustration of 1-indexed locations of the points on the face[1]

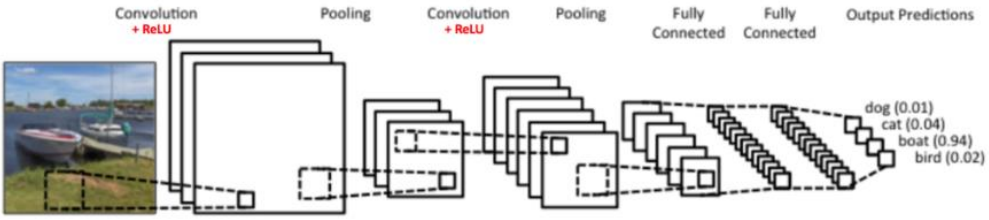

Fig 2. An example of a CNN[4]

Methodology

To begin libraries, classes and functions need to be imported that will be used to build the model before its trained to recognise facial landmarks, figure 1. For my model I decided to use the Tensor Flow library as it contains a lot of useful functions for building a CNN and allows for it to be very customisable when adding each layer individually. Therefore, from Tensor Flow I imported the classes of keras and nn as these two are the most useful when building a model. Other libraries that were used include the matplot library for plotting the output as well as CV2 for pre-processing and altering the images and numpy for storing the data. After importing the above, the supplied data must be downloaded giving the model its training data as well as any images needed for testing. This data then needs to be loaded into a numpy array where the data can be reshaped to all be the same size.

Once this is done the model can then begin building by selecting a type for the model by calling a function from the Tensor Flow keras library, for this use case the model shall be sequential meaning the chosen layers shall be called linearly. For my model, I decided to start with three convolution layers each including a ReLU activation and being followed by a pooling layer before the final output is flattened and reduced, figure 2.[4] For each convolution layer, they're given a stride size of 1 as well as a kernel size of 3 by 3 (figure 3), as this was found to get the most features from each image in testing, along with a rectifier activation function (ReLU). The only two differences between the convolutional layers are the number of filters that the layer outputs and the input shape. This is as the input shape must be adjusted after each pooling stage as the shape of the data is reduced and therefore the input shape must be adjusted. The number of filters is then changed as this controls how many times kernels are used to convolute the image to create a feature map. For my model I had filters increase in size from 32 to 256 which increases the accuracy of the plotted image points whilst not taking too long to execute. After all the convolution and pooling layers, the output is flattened and reduced giving the data in a size of a 136 list which can be expanded to a 68 by 68, 2 dimensional array. This is the sizing needed for a set of 68 coordinates that are used to plot positions on the images meaning that the model can be compiled, therefore finishing training. The model can then be used to predict landmarks of any given image by giving it an image and having it return the predicted points.[3]

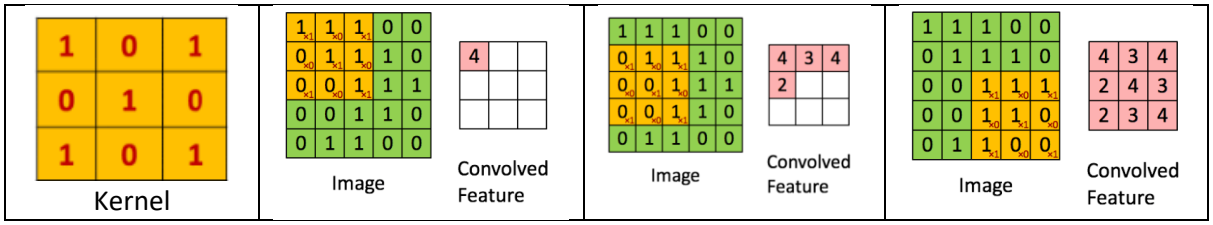

Fig 3.Demonstration of convolution[5]

After this the points and image are fed to the supplied visualise function which takes the output from the model, e.g. predicted points of the given image. These points can then be seen plotted over the image, as seen in figure 4.

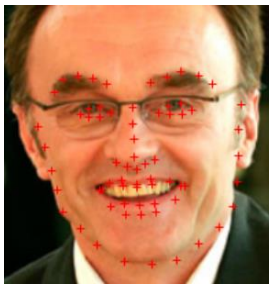

Fig 4.Example of an output

Analysis

My methodology has proven to be an effective method for face landmark detection though there is room for improvements. This is as I have been able to measure the mean Euclidean distance between predicted and ground truth points where a lower distance is better. Using this I managed to quantify the accuracy of the algorithm and improve the mean Euclidean distance from 180~ to 21, this equates to an accuracy of 91.6% as the max distance is 250. Throughout building the model different improvements were tested to try to lower this value further, one of these being adding a pre-processing method of grey scaling an image before smoothing it to increase the ease of edge detection. However, in practice this caused worse results due to some edges becoming harder to detect leading to the mean distance increasing to 60~ leading to plots like figure 5. Remnants of this can be seen in the code although the input shapes had to be altered from 4 dimensions to 3 to make it function.

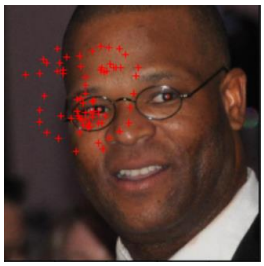

Fig5. Low contrast between face and background leads to worse results in grey scale

Fig6. Example of how the points of the mouth are irregular due to lips being

There are limitations in the created model but some of these are due to the training data, for example when detecting the mouth and lips on people when a mouth is closed the points around the mouth and nose become confused. This is likely due to a bias in the training data of smiling people meaning that the model has less information on what to do when people are not smiling, an example of this is shown in figure 6.

Another limitation of the model can be found in faces wearing sunglasses as they obscure the eyes stopping the model from plotting points around them. This can lead to misplaced points around the rest of the face not just around the eyes, figure 7. This could be improved by having these points removed or altered to be around the sunglasses in the training images and with an increased number of them in the training images. This would allow the model to know how to handle sunglasses accordingly when they appear in an image.

Fig 7. Example of how the points are irregular due sunglasses being worn

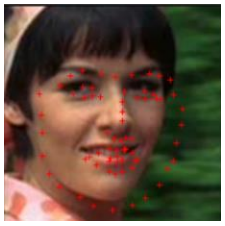

Fig 8. Example of an image with an obscured edge

The final limitation that was discovered was when the heads in the images are turned, points 10 to 17 (the right side) doesn't align properly, example in figure 6. In figure 6, the face is looking to the left of the frame causing the right side of the face to become obscured. This causes the described issue where the points can't find an edge and therefore are forced to be at their mean position. One way to fix this would be through data augmentation, where the training images are randomly sliced and rotated with their points overlaid. This would give the model more information on what to do when the parts of the face become obscured our look out of the ordinary.

Other features and functions were experimented with but they never reached a reliable state due to their high inaccuracies. These are features such as face segementaion, where the points where used to draw lines around certain areas of the face, e.g. nose, but were never accurate enough and lead to scribbles over the image. The same goes for the attempt at changing the skin colour of the found face which lead to parts of the back ground becoming coloured also and was never accurate enough to be considered a correct implementation. It should be noted that each area was marked in a different colour but not the whole area was not covered as intended.

Fig 9 Scribbles on the image from face segmentation

Conclusion

In conclusion, the face landmark detector algorithm functions well with a mean Euclidean distance of 21 and accuracy of 91.6%, meaning there's room for improvement. However, there is a bias to images such as the one shown in figure 9. These are images that show faces with smiling mouths, looking straight on and without any accessories or objects obscuring parts of the face e.g. sunglasses. However this bias could be removed with a larger and better balanced data set with some data augmentation leading to a better trained model. Other features using the landmark detection were explored but proved unreliable and lead to odd results. The accuracy of the landmark detector itself is good enough and quite reliable other than the iregularities explained above but with more training time this could be increased.

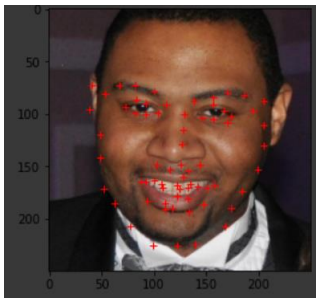

Fig 9. Example of a bias image

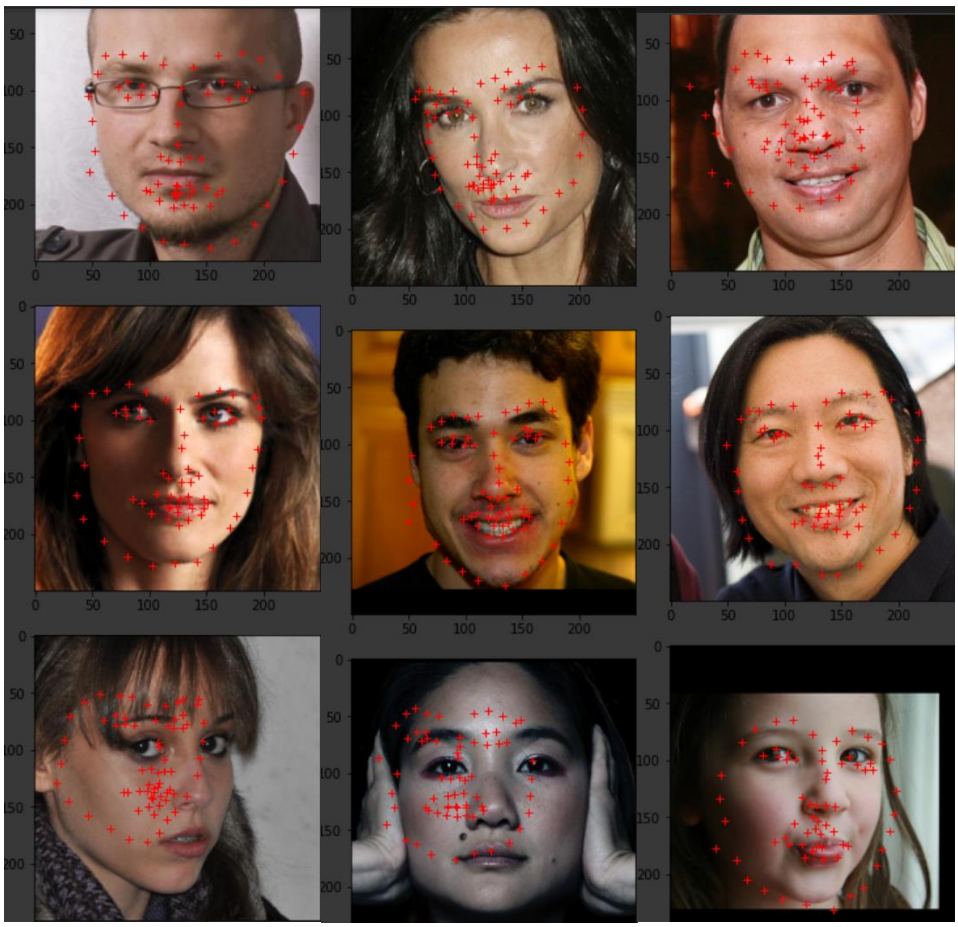

Figure 10: Example of 9 random images from the output of the model

Appendix

References

[1] https://www.researchgate.net/figure/The-68-facial-landmarks-extracted-from-a-frontal-face-view_fig3_320979643

[2] *Matusugu, Masakazu; Katsuhiko Mori; Yusuke Mitari; Yuji Kaneda (2003).* "Subject independent facial expression recognition with robust face detection using a convolutional neural network"

[3] TensorFlow documentation: https://www.tensorflow.org/

[4]CNN example: https://www.clarifai.com/technology

[5]CNN explained: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

Word Count:1477