Face Alignment Using A TensorFlow Model

Rohit Pai
Candidate Number: 198771
University of Sussex

May 28, 2020

Contents

Abstract

This report concerns the process of facial landmark detection. This goes over how I managed to create a face landmark detection model using TensorFlow and the positives and negatives of each model. Near the end, I go over what went well in the final model and compare the predictions to the ground truth points stating the euclidean distance of the images.

1 Introduction

Face alignment is a recognized problem that has been thought upon for many years. After generating the predicted points you can use them to do many face processing applications such as face alignment, emotion classification, face recognition etc.[8] Face alignment is one of the more common techniques looked into. For this, you take those points and centre the image, rotate the image such that the eyes lie on a horizontal line and also scaled so that all faces are approximately similar size[4]. There are many different ways to achieve face landmark detection, such as using cascaded regression[6], YOLO and linear regression and a CNN using TensorFlow or PyTorch which is what I am using in this report. However, I'm using TensorFlow instead of PyTorch.

2 Implementation

In this section I describe how I implemented my TensorFlow model in Python and the complications that came along with it.

2.1 Background Knowledge

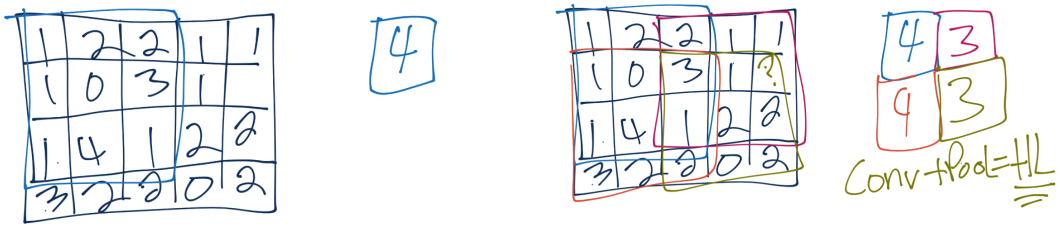
Before I could start creating my model I had to learn how to use TensorFlow and learn what each of the layers are for and the what the associated parameters are. I investigated what the most common layers are for and how I can use them in my model.

2.1.1 How Convolution Layers Work

According to Harrison[2] and Adrian[5] a convolution will learn a specified amount of filters of the specified kernel size. For example, if the kernel size is 5x5 and the number of filters is 64 then the convolution will learn 64 filters each of which is a 5x5 grid. Originally the image will be converted into pixels for the convolution to learn. It will then find the features in the grid size based on the kernel size. Those features will then be converted into a single feature on a new featuremap, this process is repeated until the end of the image.

2.1.2 How Max Pooling Works

Max pooling takes a pool window of specified size and takes the maximum feature (or value) from the window area. This is again repeated until the end of the image. See Figures 1a and 1b.



(a) Go through each window[1]

(b) After going through each window [1]

Figure 1: Max Pooling

2.1.3 How Dense Layers Work

Dense layers are fully connected between each other so they require a lot more parameters than convolution layers. The values get updated during back-propagation. A dense layer is a non-linear layer[3] that can also be represented by a matrix vector multiplication, where the matrix is the matrix of weights and the vector is the input vector, given by the previous layer. At the output, you get another vector, based on the size you specified for the layer[7].

2.2 Initial Model

My initial implementation was to use a small number of layers which yields a small number of parameters. This consisted of a couple of convolution 2D layers, max pooling 2D layers, a couple of dense layers and a few activation layers. This gave 109,704 parameters which is quite small for a

CNN. See Figure 2 for a summary of this. This model trained very quick but didn't point the face landmarks correctly. This is likely because the number of layers and parameters are relatively small. When I tried to predict I got errors saying that the output shape of the last layer was wrong. What I had to instead was to reshape the output shape down to `(1, 1, 136)` because it means the model will try to learn 136 different features per 1×1 pixel, which means I would get the required 136 positions (68 each for x and y).

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 248, 248, 64)      1792
_________________________________________________________________
activation (Activation)      (None, 248, 248, 64)      0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 124, 124, 64)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 122, 122, 128)     73856
_________________________________________________________________
activation_1 (Activation)    (None, 122, 122, 128)     0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 61, 61, 128)       0
_________________________________________________________________
dense (Dense)                (None, 61, 61, 128)       16512
_________________________________________________________________
dense_1 (Dense)              (None, 61, 61, 136)       17544
_________________________________________________________________
activation_2 (Activation)    (None, 61, 61, 136)       0
=================================================================
Total params: 109,704
Trainable params: 109,704
Non-trainable params: 0
_________________________________________________________________
```

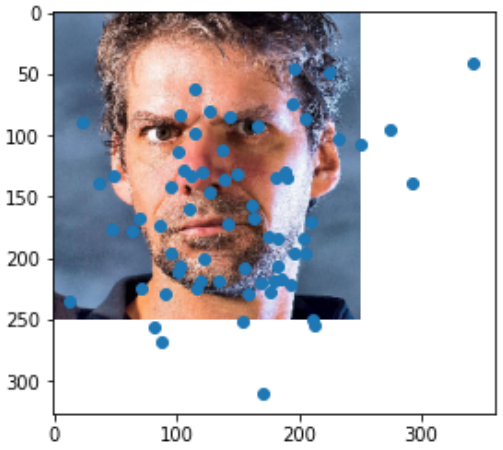
Figure 2: The out summary of the initial model.

2.3 Second Model

The initial model was not good so I decided to change up the parameters. The initial model's output shape didn't match what was needed to predict and also I hadn't flattened the layer before I used dense. The reason why you flatten the layer before dense is that flattening reduces the number of dimensions which should give a higher parameter count. See Figure 3a. Trying to run the model gave multiple issues, one being that the training time increased from 15 minutes per epoch to around 1h per epoch, which is a big difference in time. When I was training the model, the initial mean squared error was around 158141; this is quite a high loss. The model worked but it gave predictions that were not in the bounds of the image. See Figure 3b.

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_70 (Conv2D)           (None, 246, 246, 64)      4864
_________________________________________________________________
conv2d_71 (Conv2D)           (None, 242, 242, 64)      102464
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 48, 48, 64)        0
_________________________________________________________________
conv2d_72 (Conv2D)           (None, 44, 44, 64)        102464
_________________________________________________________________
conv2d_73 (Conv2D)           (None, 40, 40, 128)       204928
_________________________________________________________________
conv2d_74 (Conv2D)           (None, 36, 36, 128)       409728
_________________________________________________________________
flatten_2 (Flatten)          (None, 165888)            0
_________________________________________________________________
dense_5 (Dense)              (None, 512)               84935168
_________________________________________________________________
dense_6 (Dense)              (None, 256)               131328
_________________________________________________________________
dense_7 (Dense)              (None, 136)               34952
=================================================================
Total params: 85,925,896
Trainable params: 85,925,896
Non-trainable params: 0
_________________________________________________________________
```

(a) The output summary of the 2nd model.



(b) The predicted points of image 31 from the training images predicted using the 2nd model.

Figure 3: Second Model

2.4 Third Model

If you take a look at the second model you can see clearly that a mistake was made by choosing to flatten and dense the layers early. This is where I flatten the layer before adding dense layers. Since

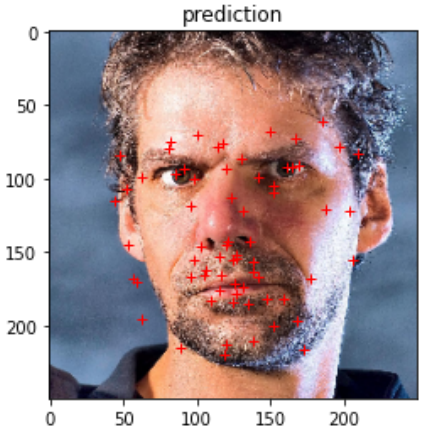
flattening multiplies the dimensions together in this case it is $36 \times 36 \times 128 = 165888$. It becomes a very large single dimension shape. So I went back to change it so that the number of parameters was small enough so that the training time was lower. See Figure 4a This time I didn't use any dense or flatten layers as I thought that wasn't the way to go forward. Doing so brought the number of parameters down significantly to 2,455,105. When predicted, the model's prediction was closer to the ground truth points making it a better model. See Figure 4b

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 201, 201, 64)	480064
activation (Activation)	(None, 201, 201, 64)	0
conv2d_1 (Conv2D)	(None, 177, 177, 32)	1280032
activation_1 (Activation)	(None, 177, 177, 32)	0
conv2d_2 (Conv2D)	(None, 153, 153, 32)	640032
conv2d_3 (Conv2D)	(None, 144, 144, 16)	51216
conv2d_4 (Conv2D)	(None, 140, 140, 8)	3208
conv2d_5 (Conv2D)	(None, 137, 137, 4)	516
activation_2 (Activation)	(None, 137, 137, 4)	0
conv2d_6 (Conv2D)	(None, 136, 136, 2)	34
activation_3 (Activation)	(None, 136, 136, 2)	0
conv2d_7 (Conv2D)	(None, 136, 136, 1)	3
activation_4 (Activation)	(None, 136, 136, 1)	0

Total params: 2,455,105
Trainable params: 2,455,105
Non-trainable params: 0

(a) The output summary of the 3rd model.



(b) The predicted points of image 31 from the training images predicted using the 3rd model.

Figure 4: Third Model

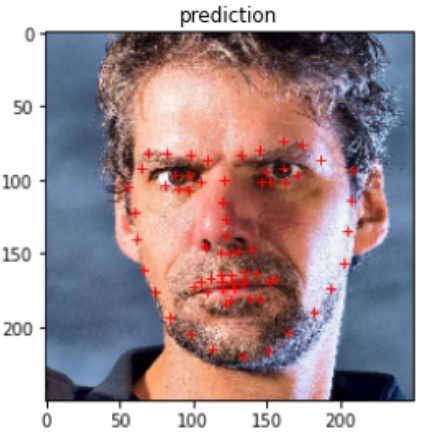
2.5 Fourth Model

From the third model, you can clearly see that it was a step in the right direction. This model had more layers but included some activation such as relu to make sure there are no negative values given. Doing this method decreased the number of parameters down to 1,197,392, which is much lower than the previous, less than half the number of parameters. When this model was training, the initial mean squared error was around 0.0023 which lower than previous models. See Figure 5a If you look at the prediction, the points have come very close to the ground truth, which is a very good sign. You can see that it isn't perfect and the points are about 4 pixels out. I calculated the euclidean distance and it came to 4.807533370959767 which is quite low and good. Overall I think this model is the best model yet and there is no need to tweak it any further because this model gives good results. See Figure 5b

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 246, 246, 64)	4864
conv2d_1 (Conv2D)	(None, 242, 242, 64)	102464
batch_normalization (BatchNo	(None, 242, 242, 64)	256
max_pooling2d (MaxPooling2D)	(None, 48, 48, 64)	0
conv2d_2 (Conv2D)	(None, 46, 46, 96)	55392
conv2d_3 (Conv2D)	(None, 44, 44, 96)	83040
batch_normalization_1 (Batch	(None, 44, 44, 96)	384
conv2d_4 (Conv2D)	(None, 42, 42, 96)	83040
conv2d_5 (Conv2D)	(None, 40, 40, 96)	83040
max_pooling2d_1 (MaxPooling2	(None, 20, 20, 96)	0
conv2d_6 (Conv2D)	(None, 18, 18, 128)	110720
conv2d_7 (Conv2D)	(None, 16, 16, 128)	147584
batch_normalization_2 (Batch	(None, 16, 16, 128)	512
conv2d_8 (Conv2D)	(None, 14, 14, 128)	147584
conv2d_9 (Conv2D)	(None, 12, 12, 128)	147584
max_pooling2d_2 (MaxPooling2	(None, 4, 4, 128)	0
conv2d_10 (Conv2D)	(None, 2, 2, 136)	156808
conv2d_11 (Conv2D)	(None, 1, 1, 136)	74120

Total params: 1,197,392
Trainable params: 1,196,816
Non-trainable params: 576

(a) The output summary of the 4th model.



(b) The predicted points of image 31 from the training images predicted on the 4th model.

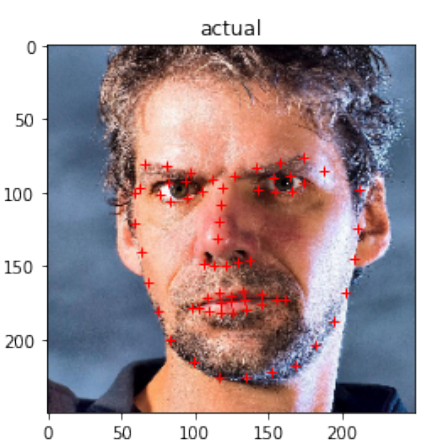
Figure 5: Fourth Model

3 Final Model Tests on Training Images

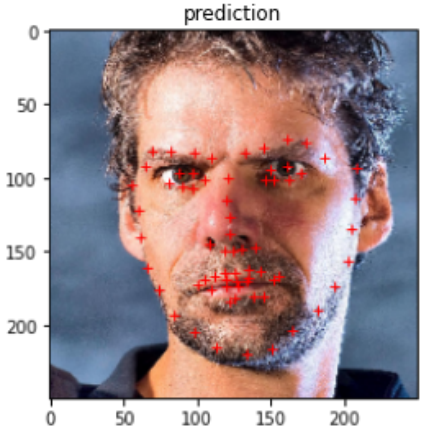
As discussed previously, the Fourth model worked very well and is the final model. To make sure that this is the final model, I did some tests to find out if there were any anomalies. I tested it on some of the training images to see if it was close to the ground truth points. Here are some examples of the tests.

3.1 Test 1

From the first test, you can see that it did a decent job all around apart from missing the right eye a little bit and the mouth. The chin area is a little higher than the ground truth but it's still very good. See Figure 6.



(a) The actual points of image 31 from the training images.

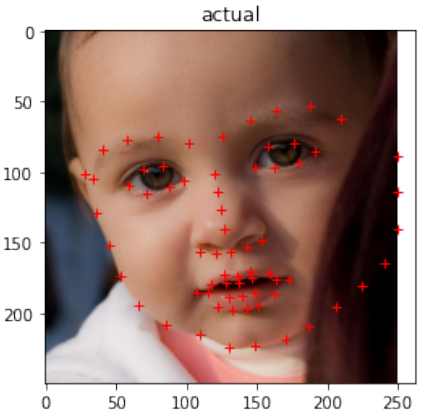


(b) The predicted points of image 31 from the training images predicted using the final model.

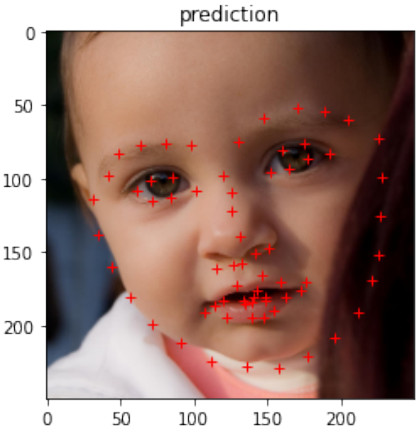
Figure 6: Test on Image 31

3.2 Test 2

From the second test, you can see it struggled to predict the left cheek properly, as it is behind an obstacle. The eyes and eyebrows are very close to the ground truths. The mouth region is jumbled up a little bit but the points are at least in the area. See Figure 7.



(a) The actual points of image 291 from the training images.

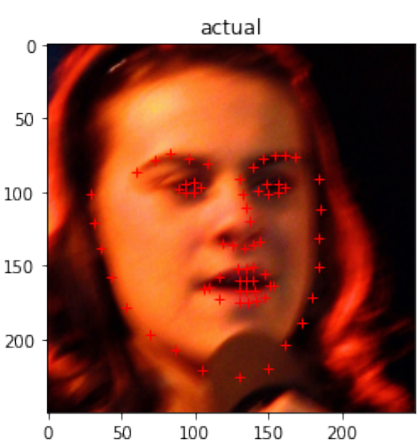


(b) The predicted points of image 291 from the training images predicted using the final model.

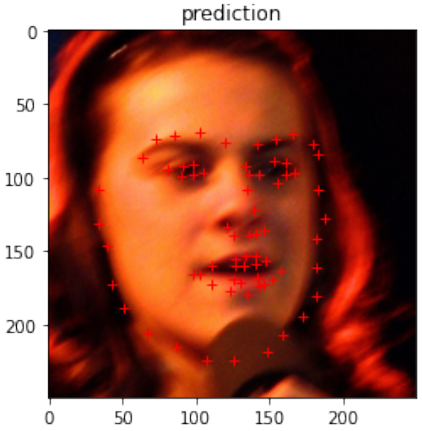
Figure 7: Test on Image 291

3.3 Test 3

From the third test, you can see it has overall done a good job, given that the image is blurred. The shape of the points is close to the actual image. The eyes and mouth however are a bit off compared to the ground truth predictions. See Figure 8.



(a) The actual points of image 371 from the training images.

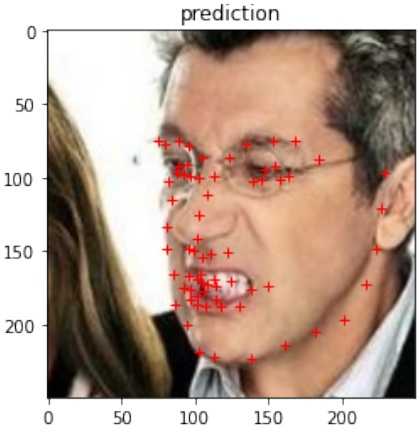


(b) The predicted points of image 371 from the training images predicted using the final model.

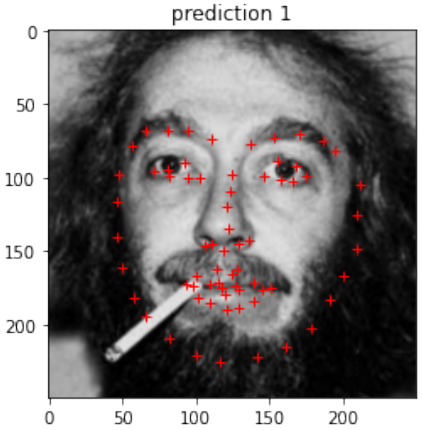
Figure 8: Test on Image 371

4 Test on Testing Images

You can clearly see that the model has done an overall good job on the testing images because the points seem to line up to where they might be on the images if there were sample points on these. As these images are test images we don't know what the ground truth points are, but I think it has done well overall. See Figure 9.



(a) The predicted points of image 30 from the testing images predicted using the final model.



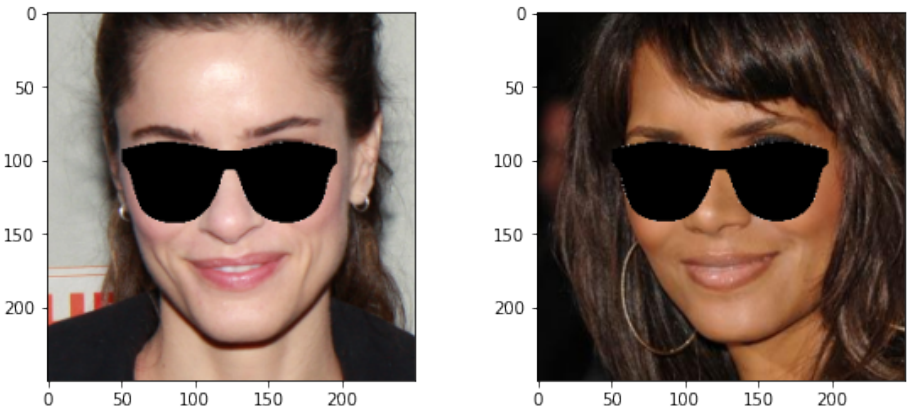
(b) The predicted points of image 97 from the testing images predicted using the final model.

Figure 9: Tests on 2 Testing Images.

5 Fun With the Images

I added some sunglasses to see whether my points are accurate enough for the face. The eye predictions were good but the glasses go further than the eyes so I had to adjust the points when drawing

the glasses. I tested it on a training image and a test image based on my predictions for both. Although the second sunglasses aren't correctly positioned. See Figure 10



(a) Sunglasses on a training image using predicted points from the model.

(b) Sunglasses on a testing image using predicted points from the model.

Figure 10: Sunglasses On a Test and Training Image.

6 Conclusion

Overall the model did some good predictions on the images. It wasn't perfect and could be improved to get a lower euclidean distance. I found that using a CNN for this particular problem was quite a good choice because it is easy to configure the model and tweak as you go along. Using the data given, a CNN can learn how to find these features on the face quite well as shown above. The model was tested on several other different images and did perform quite well on them. This goes to show that using CNN for facial landmark detection is a good method you can use if tweaked right.

References

[1] Harrison. Convolutional Neural Networks - Deep Learning basics with Python, TensorFlow and Keras p.3 - Text Version.

[2] Harrison. Convolutional Neural Networks - Deep Learning basics with Python, TensorFlow and Keras p.3 - Video Version.

[3] Assaad Moawd. Dense layers explained in a simple way.

[4] Adrian Rosebrock. Face Alignment with OpenCV and Python.

[5] Adrian Rosebrock. Keras Conv2D and Convolutional Layers.

[6] Ivor Simpson. Extra: Cascaded regression.

[7] Apurv Verma. In Keras, what is a 'dense' and a 'dropout' layer? - Quora.

[8] Y. Wu, T. Hassner, K. Kim, G. Medioni, and P. Natarajan. Facial Landmark Detection with Tweaked Convolutional Neural Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(12):3067–3074, Dec 2018.