Face alignment using deep learning

Author:
Ver Valem Paiva Willian

December 5, 2017

Contents

1 Definition

1.1 introduction

For a long time work in facial recognition has being done and one of the key points of it is face alignment as it poses its own challenge. And the main tool used for the job is OpenCV which is used with DLIB to recognize Facial landmarks.

The automatic recognition of landmarks is essential to be able to classify facial expressions, or face tracking, face animation, and even 3D face modeling.

For example to classify facial expressions it is necessary to classify Facial Action Units also known as FACS [1], which in turn needs a proper face alignment.

The problem of face alignment is among the most popular in the field of computer vision and today we have many different implementations to automatically recognize facial landmarks on images. The most known is the Active Appearance Model (AAM) [2, 3].

But today we also some good results using Deep learning to achieve the results for example the work done by Adrian Bulat on recognizing 3D facial landmarks [4] that shows remarkable results and is implemented in *Torch* a framework for **LUA**.

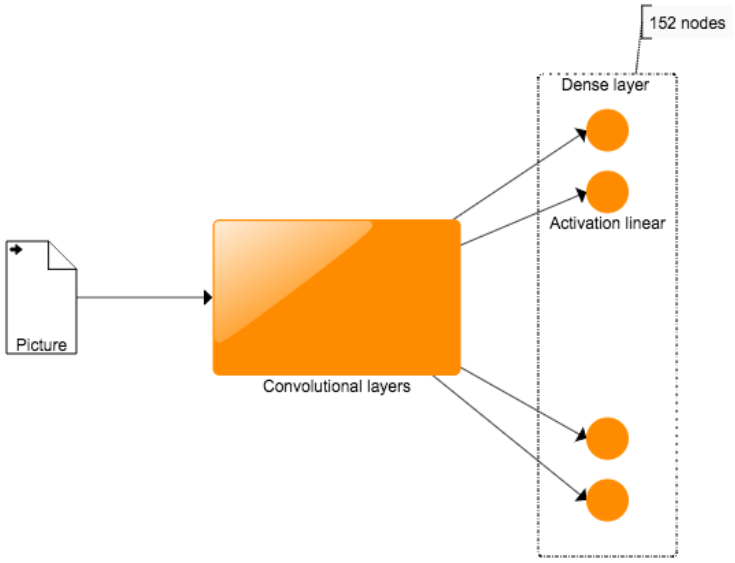
As the DLIB model has difficulty on recognizing points on faces by the side view, this project focus on this problem and does its best to tackle this problem.

1.2 Problem Statement

The problem in question take a image as input (the format and details will be discussed on the 2.1 Dataset section), and by using a regression model to calculate the position of the face landmarks on the given image.

For that the models used to tackle this problem will output 152 numbers, where each number ranges from 0 to 1 and represents the X and Y of each of the 76 landmarks.

Those models will be created using convolutional neural network as this type of model is known for its efficiency on images



The expected result is a model capable precisely recognize the set of landmarks.

1.3 Metrics

The results of this project was measured using the Mean Squared Error (MSE) this metrics was chosen as it is standard function for regression problems and it fits well problem in question.

$$\mathbf{MSE} = \frac{1}{n}\sum_{t=1}^{n}(\hat{Y}_i - Y_i)^2 \tag{1}$$

where $\hat{Y}$ is the real value and Y is the predicted value

The mean squared error was the choice here because it has a higher cost futher the point is from the real value what, and as what we are looking for is precision on the point location this kind of reinforcement is ideal for the project

2 Analysis

2.1 Dataset

In this study the MUCT Face Database [5] was used this dataset consists of pictures (resolution 480x640) taken from 276 subjects using 5 cameras in different angles and light conditions (total of 3755 images) like the image below:

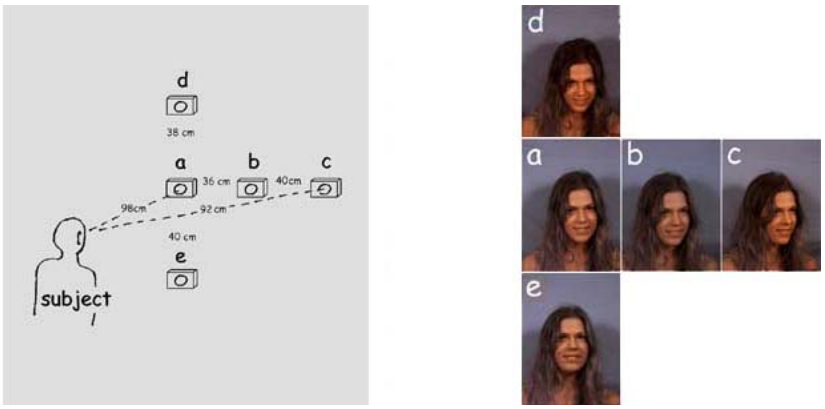
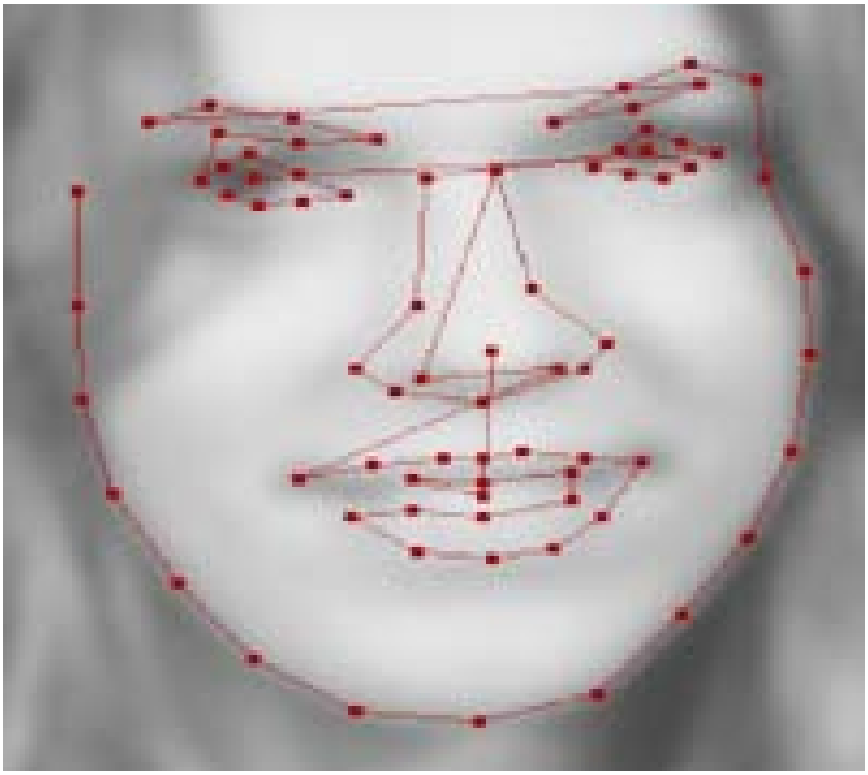


Figure 1: There is no images on the left but they can be reproduced by mirroring the right side

Each picture is manually coded with 76 facial landmark like:



The landmarks are saved in to 4 different file formats

- muct76.shape shape file [6]
- muct76.rda R data file (www.r-project.org/
- muct76.csv comma separated values
- muct76-opencv.csv comma separated values in OpenCV coords (0,0 at top left).

the coordinate system in these files is the one used by Stasm (i.e. the origin 0,0 is the center of the image, x increases as you move right, y increases as you move up). The exception is muct76-opencv.csv, where the format is the "OpenCV format" (i.e. the origin 0,0 is at the top left, x increases as you move left, y increases as you move down).

Unavailable points are marked with coordinates 0,0 (regardless of the coordinate system mentioned above). "Unavailable points" are points that are obscured by other facial features. (This refers to landmarks behind the

nose or side of the face – the position of such landmarks cannot easily be estimated by human landmarkers – in contrast, the position of landmarks behind hair or glasses was estimated by the landmarkers).

So any points with the coordinates 0,0 should be ignored. Unavailable points appear only in camera views b and c.

the subjects 247 and 248 are identical twins.

The data set is available via github or it is also a submodule on this project and by recursively initializing or cloning the data can be found in folder project/dataset that is the recommended way as the scripts depend on this folder

2.2 Split the training and testing sets

As this dataset is composed by multiple images of the same subject the division of sets for training and testing could not be done at a random way as an special attention is needed to not have the same subject on the train and testing set.

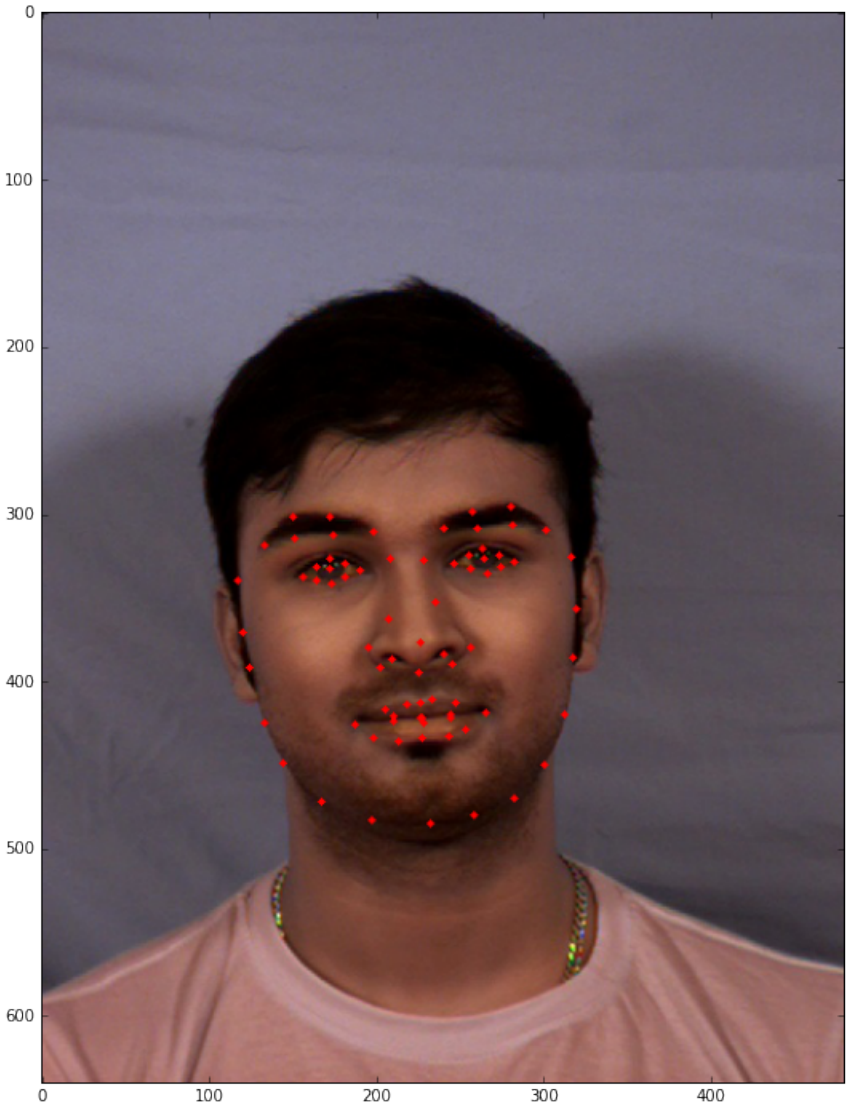
And after analyzing it was divided by taken the first 82 subjects for the testing set (249 images per camera total of 1245 around 30%)

The function "decompress$_{\text{dataset}}$" on the *DataGen.py* takes care of decompressing the data and splitting the training and testing set.

2.3 Data Analyses

lets start by looking at one image from the database those images are on the format of 640x480.

And lets plot the points on one image to have an insight:



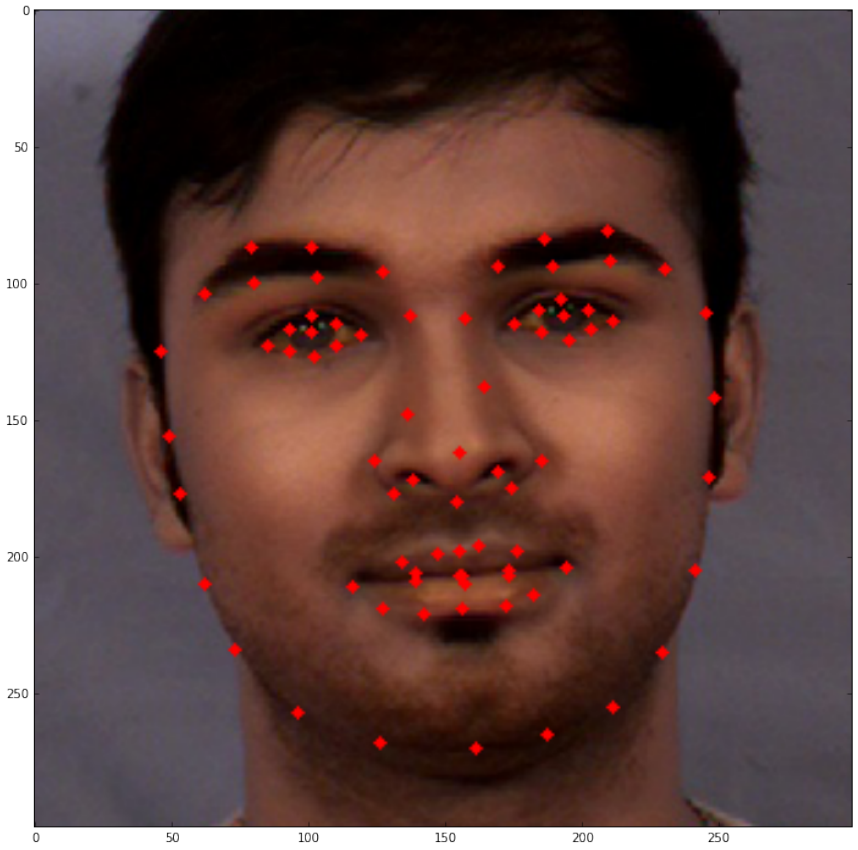
As is possible to see those images are large and have big regions with useless information like the background.

To improve that lets use DLIB's face recognition to find the face area and crop it, for that we can use the function "crop$\backslash_{\text{image}}$" from the *DataGen.py* this function takes the image and the size to crop we pass just one integer for the size as it crops the image in a square, and returns the cropped image and the bound box used to crop the image.

But as the image have being cropped we also have to move the land-

Figure 2: image cropped with the size of 299 x 299.

mark points to match the new image, for that we can use the function "replace\landmarks" which takes the bound box and the actual landmarks and return the new landmarks



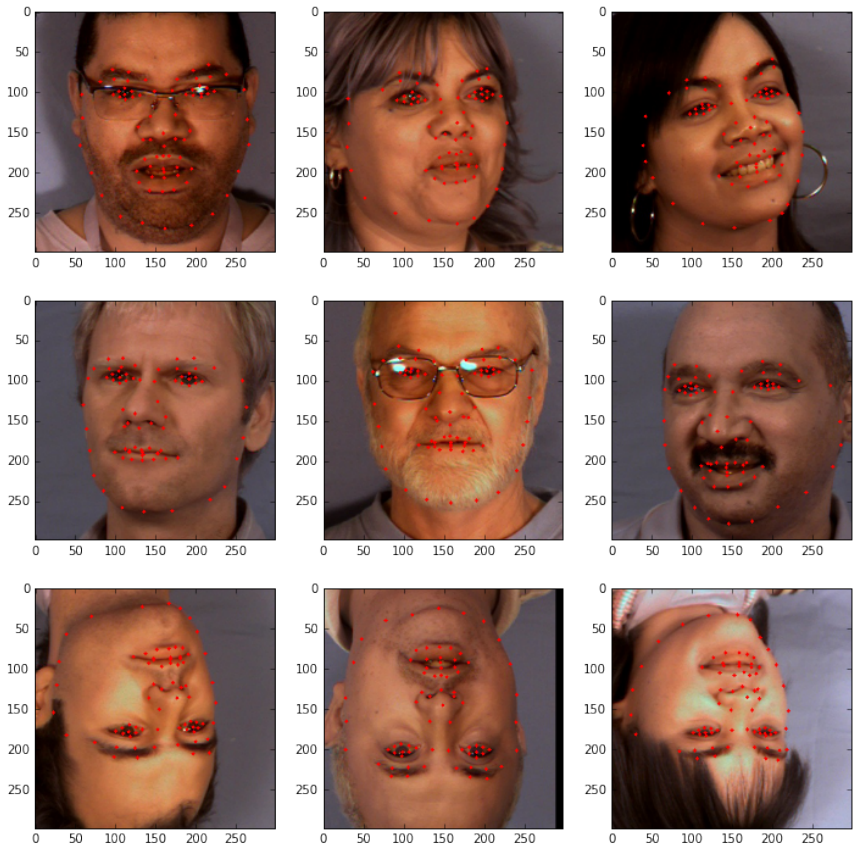
the process of cropping the image brings 2 advantages to the project:

- less resources needed to train model
- it makes possible to use images with more than one face in the future

2.4 Data Augmentation

As explained before there is no cameras on the left but we can mirror the right images, starting from that we will implement a some data augmentation like flipping every image horizontally and vertically.

To achieve that we can use the function "flip_image" also from *DataGen.py* it takes the image, the landmarks, the direction to flip, width and height. and returns the new image and landmarks. lets see the result:



By applying this to all images we finish with a total of 3723 testing images and 7497 training images

2.5 Algorithms and Techniques

To tackle this problem 2 possibilities was taken in consideration:

1. The use of a fully convolutional network (FCN) and map the points by returning the pixel of a landmarks
2. The use of a convolutional neural network (CNN) with a regression output returning the coordinates of the landmarks

While the 2 approaches are valid this project focused on the second one as the first would require more experience and knowledge to do it.

The project is composed of 2 parts:

1. The CNN layer.
2. The regression output.

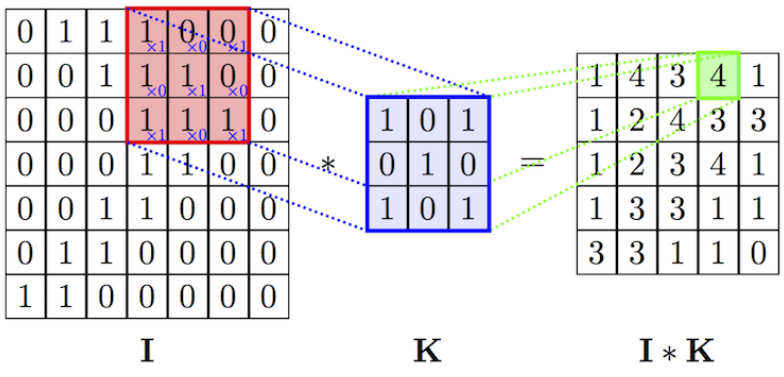
All the models have 1 thing in common the **output** all the models have a 152 neurons dense layer using linear activation, used to predict the coordinate of the landmarks on the image. In the other hand the CNN part of every model is different, here the approach was to develop a CNN capable to pass a precise information about the shape in the image to the output layer so we could have a precise landmark on the image.

2.5.1 Convolutional Neural Network

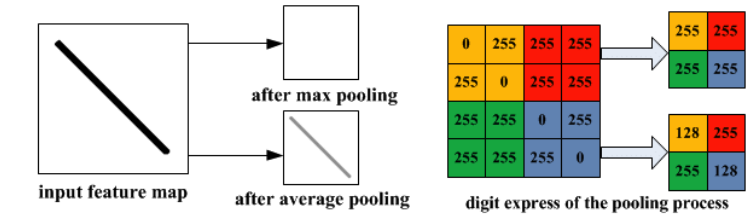
In the context of artificial intelligence and machine learning, a Convolutional Neural Network is a class of artificial neural network of the type feed-forward that have been applied with success to imaging processing and analyses.

Convolutional Neural Networks uses a variation of Multi-layer Perceptrons developed in a way to minimize the pre-processing. Those networks are composed of two kinds of neurons, the one that do the imaging processing in subparts of a image via a convolutional function, and the neurons that do the pooling.

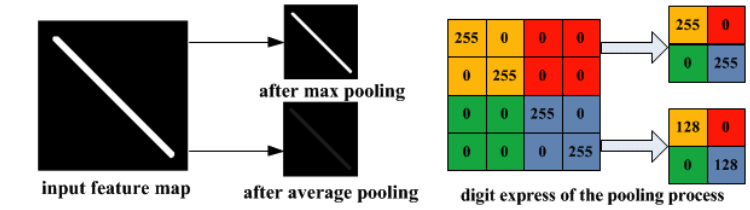
1. Convolutional layer In the convolutional layers a *filter* is applied to the image to generate the new matrix with the features of the image. For example see the image bellow:



2. Polling The pooling process can variate depending on the needs, but the idea of the polling is to reduce the matrix dimensionality and preserve the important features from it, below we can see an example of *max pooling* and *average pooling* and where it shines and where it falls short.



(a) Illustration of max pooling drawback



(b) Illustration of average pooling drawback

2.6 Benchmark

To bench mark this project a model using a CNN created from scratch was used and the model and the results of this model can be seen at the 3.2.1 **CNN from scratch** in the 3.2 **Implementation** section of this report.

3 Methodology

3.1 Data Preprocessing

Many steps of preprocessing have being taken and have being discussed on the 2.1 section being then:

- Split of the data discussed at the 2.2 **Split the training and testing sets** section
- Image cropping detailed at the 2.3 **Data Analyses** section
- Data Augmentation described on the 2.4 **Data Augmentation** section
-

3.2 Implementation

For this project many different approaches have being taken and it will be presented here the best outcome of each "kind" of approaches like:

- A CNN created from scratch
- Inception transfer learning
- ResNet50 transfer learning

All the models have being trained in a cloud server for performance wise and saved to later study.

The metrics used to observe the performance of every model was the mean square error (MSE)

The models code can be found on the file *Models.py*.

The models use a *"npz"* file that can be generated via by using the function :

```
DataGen.save_dataset(DataGen.create_dataset(size, True, True), "outputName")
```

And for the inception transfer learning it uses a bottleneck file that can be generated with the function:

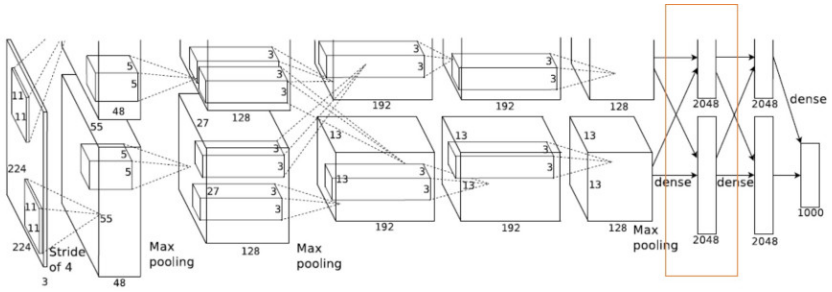
```
DataGen.inception_bottleneck(<npz file>, "outputName")
```

every model was implemented using early stopping, TensorBoard and saves the best loss weights on a *hdf5* file

About complications, this project didn't really pose a lot of problems most of them have being hardware related like lack of memory or process power, but other them that it was really straight forward: make a model, train, analyze results.

3.2.1 CNN from scratch

This CNN started with a model based on the AlexNet [7]



But we finished with a model as illustrated below:

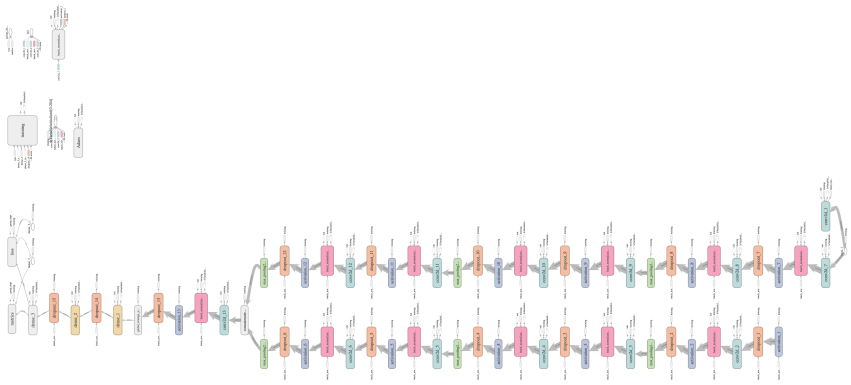
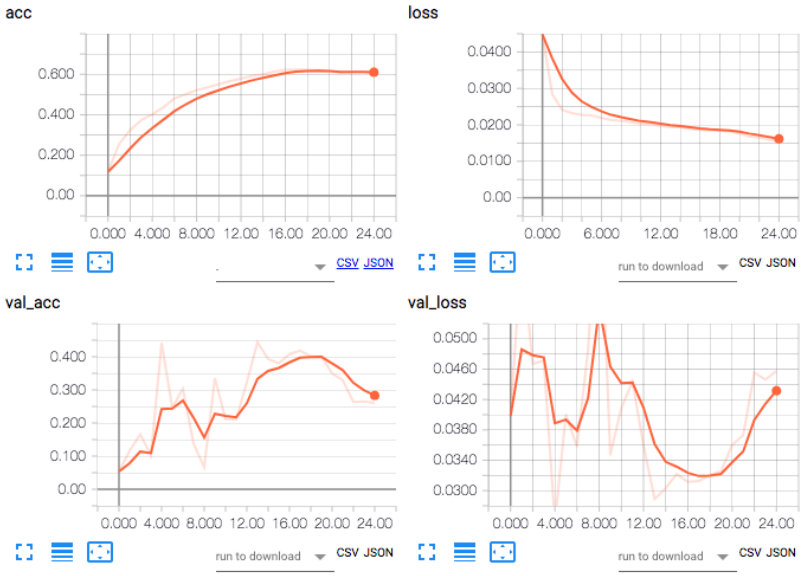


Figure 3: the full size image can be found on the report/images/mymodel.png.

here is the learning curve from this model



as we can see the learning curve for the training set was improving steadily but for the testing set it starts to get worst around the epoch 16 and the early stopping is triggered at epoch 24 as it has no improvement.

This model have 2 Convolutional paths as is visible on the plot that is because a different approach was take to extract the features, for that on convolutional path uses a small kernel so it concentrates on small details while the second one uses a bigger kernel to get features in a larger scale and by combining this process is possible to reach a good result and avoiding the model to overfeed because the network is too large.

by taking the weights of the best loss and doing an inference on a set of training images we get the following result:

It is noticible that the result is not satisfactory but it has plenty of room for improvement. but at this point of the project it started to get expensive to keep using the cloud machine to train and there was yet some other methods to try out.

3.2.2 Inception transfer learning

1. Bottleneck

   The next step was to use transfer learning, by using the inceptionV3 [8], the first approach as to create bottleneck files from the inception and train a model with those outputs.

   As the previous model many models have being made and here is the best result found by using this method was the following:

   In this model we took a different approach as just using the output of Inception wasn't given good results 3 different inputs was used the

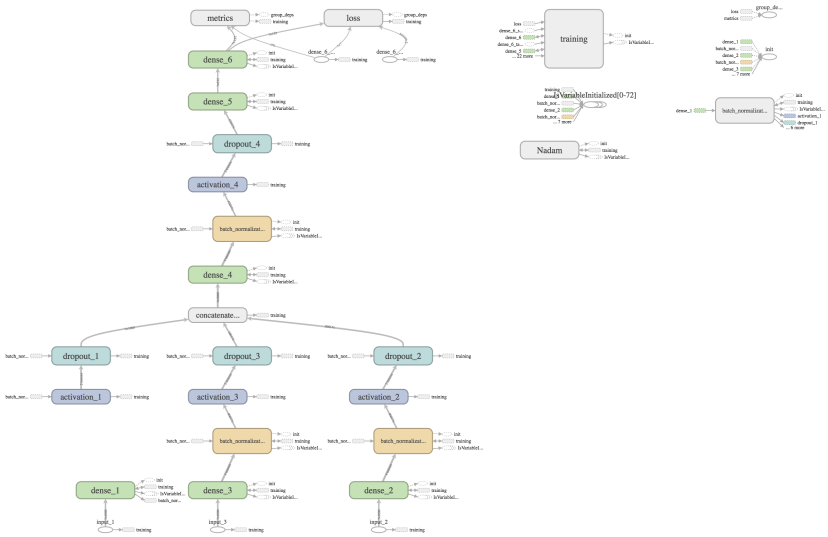
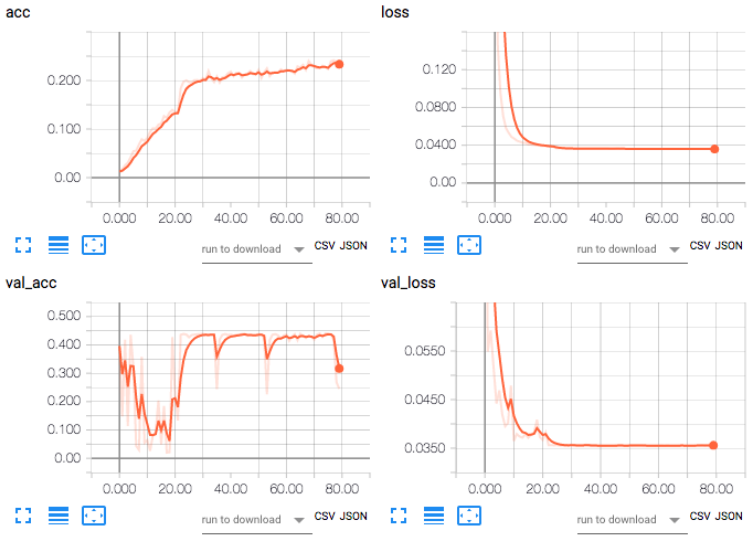


Figure 4: the full size image can be found on the report/images/btn.png.

squared and the *sine* of the inception output was also used in conjunction to the normal output



The training of this model took 80 epochs before the early stopping

was triggered while the learning curve on the test set is better them the previous model the loss is slightly bigger them the previous model:

And the results are far from good, worst then the previous model.

2. Retrain

   The next step here is to keep using the transfer learning from inception but give a bit of more room for improvement by training respectively the 2 last convolutional blocks of the inception and the 4 last ones:

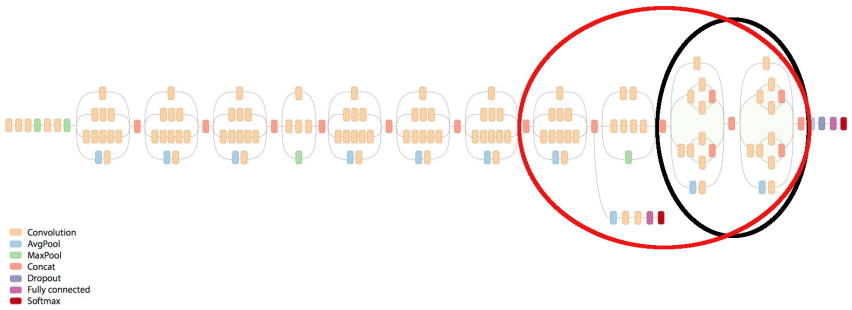
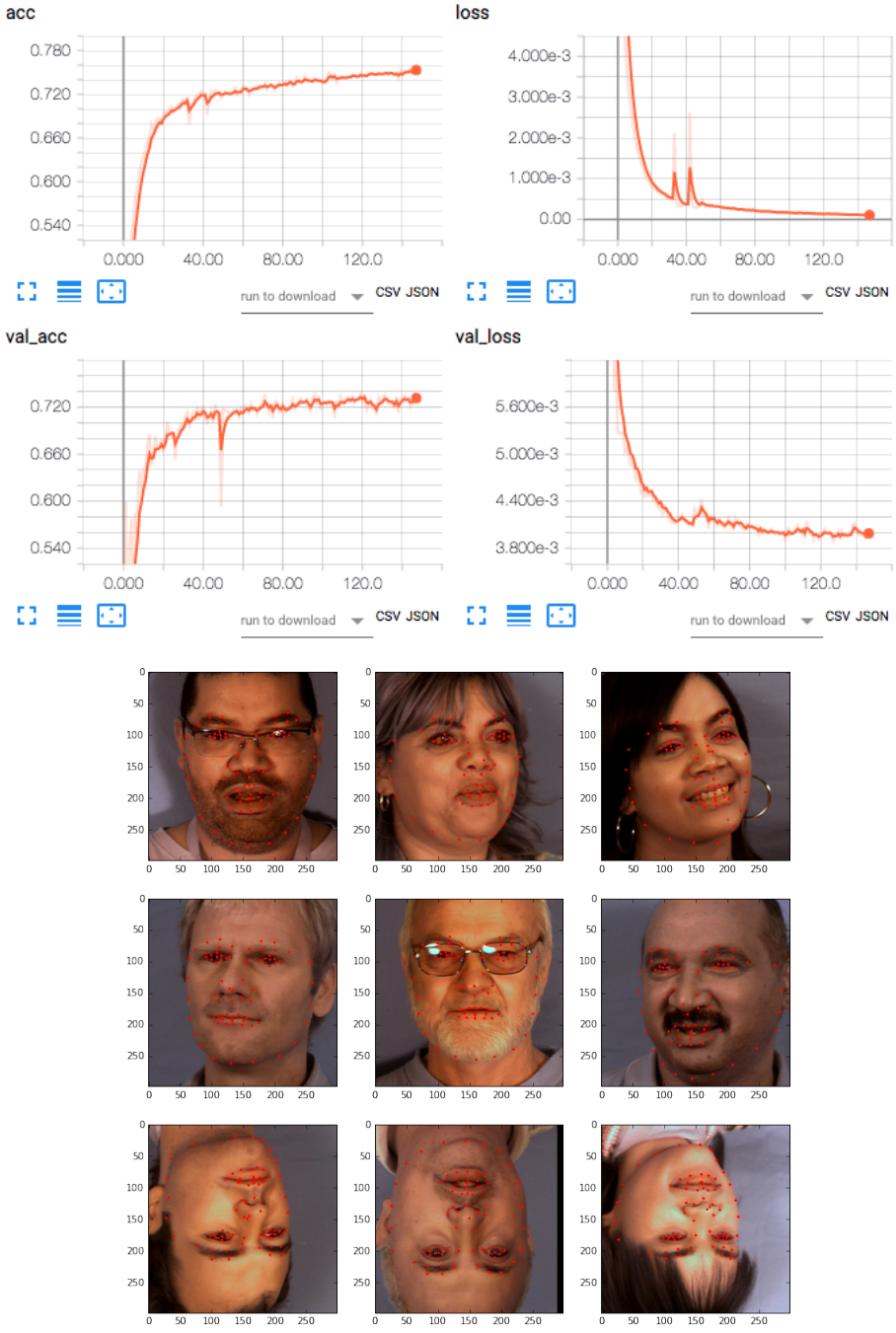
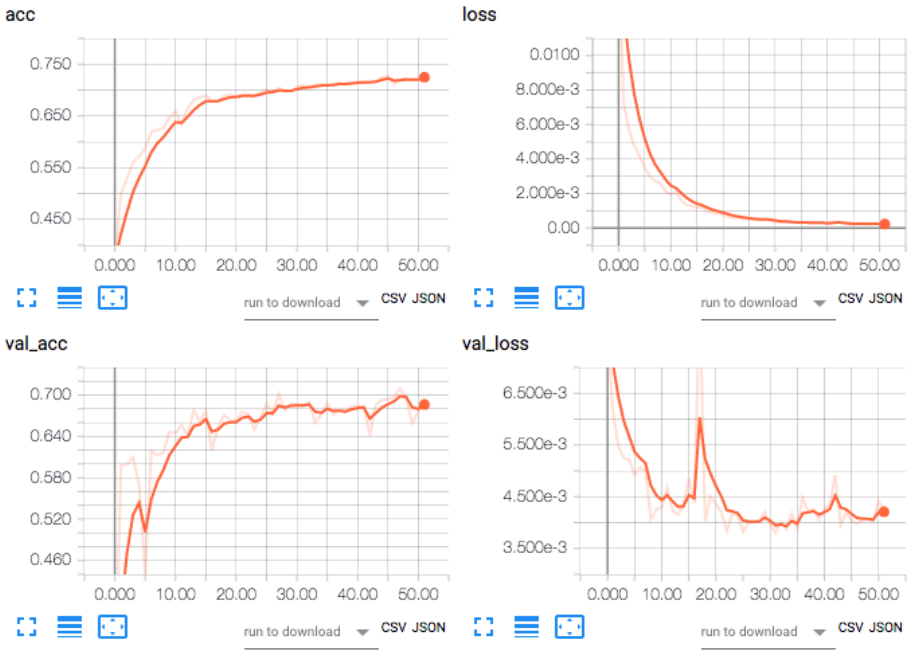


Figure 5: 4 block(red) 2 block(black).

(a) **2 blocks results**



(b) **4 blocks results**

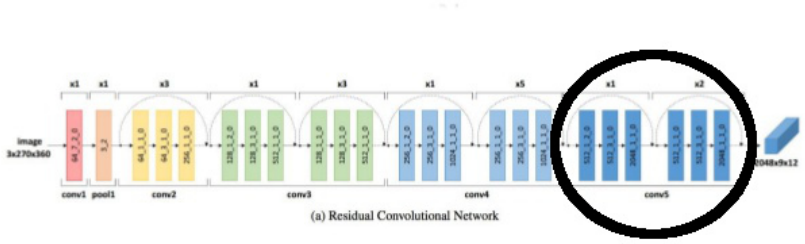


as we can see it really improved the results by using the 2 first blocks after a 120 epochs we have a steady learning curve and we get a nice prediction on the testing images, but once we increase the number of training layers by using the first 4 block, the early stopping triggers 70 epochs before and while the results look good, however, it is worst then the model with 2 blocks.

3.2.3 ResNet50 transfer learning

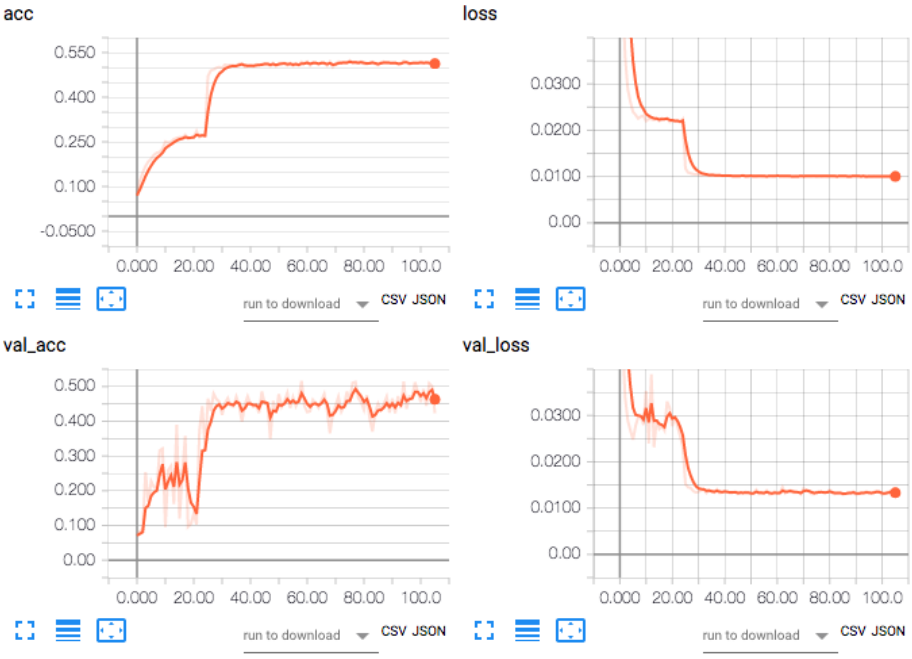
Here the same approach that as used with Inception is used with the ResNet50 [9] model, but as the ResNet50 uses images with format 224x224 and we have images with 299x299, while we could crop the images it is not a good solution as it was cutting out landmarks from some images. So the idea here is use the ResNet50 with different input.

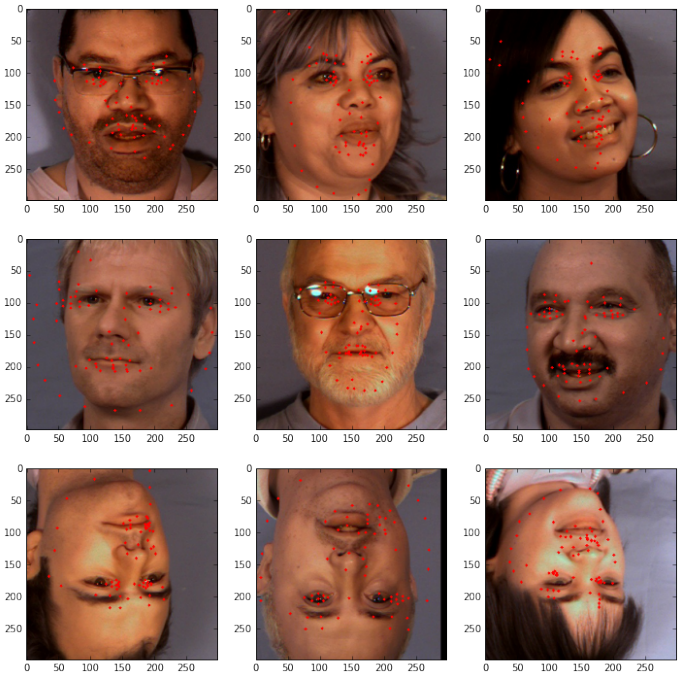
2 models was trained here one that don't train any layer from ResNet50 and another that trains 2 convolutional blocks like:



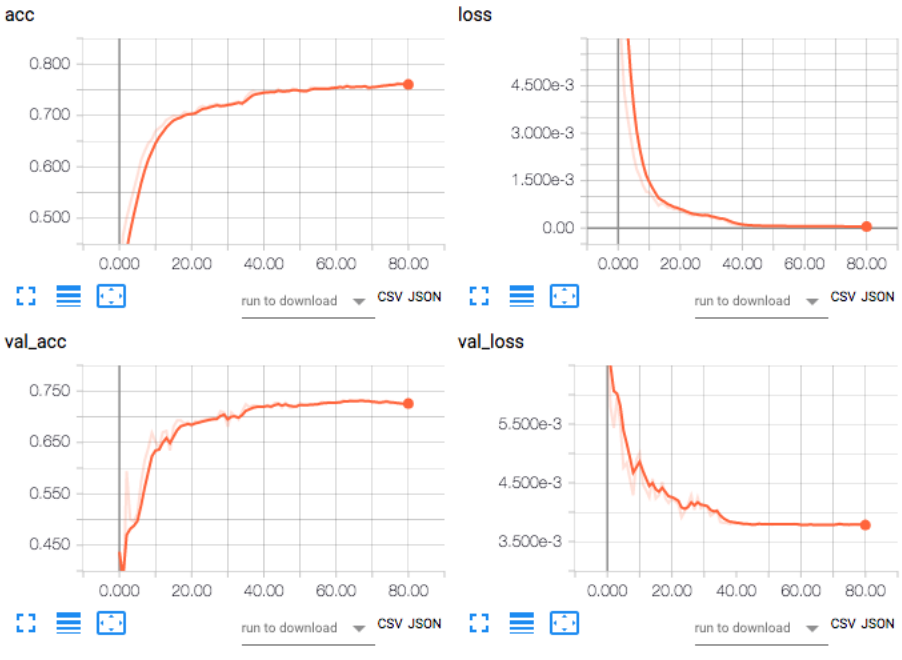
ResNet-50

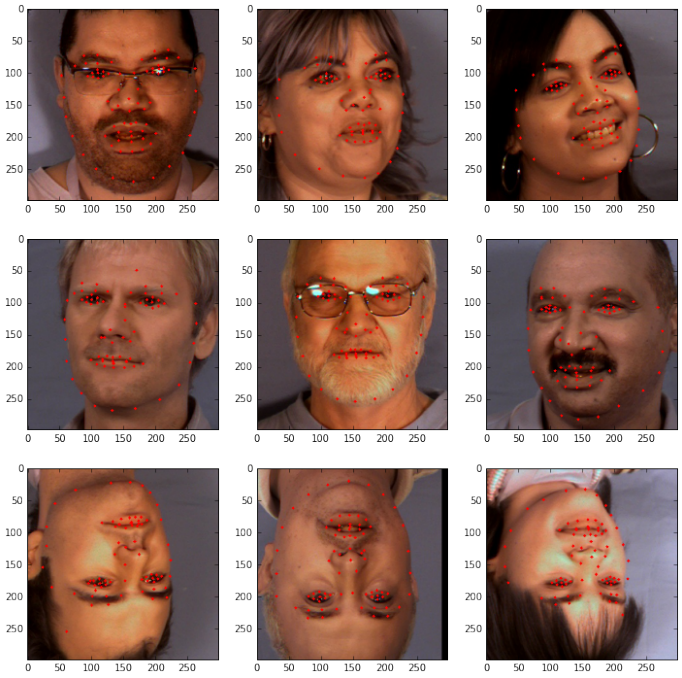
1. **0 blocks results**





2. **2 blocks results**



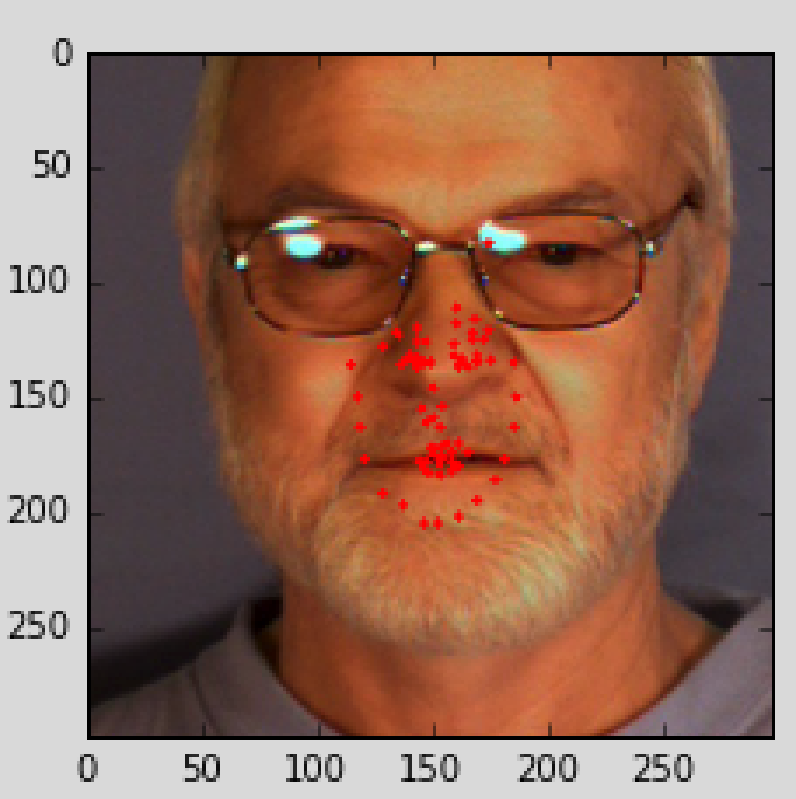


as it is noticible using just the output of ResNet50 did give better results them the inception but yet it wasn't satisfactory in the other hand when retraining 2 block it showed a really promising result

3.3 Refinement

while the results presented here are the best ones for each of the models many trials have been done the first tuning was working with the learning and the optmizers and here is what was what was used:

1. SGD all results were poor with this optimizer
2. RMSprop that was the second optimizer tried it was a bit trick as the results were showing all the points accumulating near of the center of the image like:



the approach to fix it was to increase the learning rate to 0.1 and use a

learning rate decay until 0.0001 while that give better results, further analyses showed that wasn't the best approach

3. Adam this is the optimizer used on every model with exception of the benchmark model and while the reduction of a learning rate showed improvement on some other tests it was taking too long to train and exhausting the resources for this project and we sticked with the Keras default learning rate

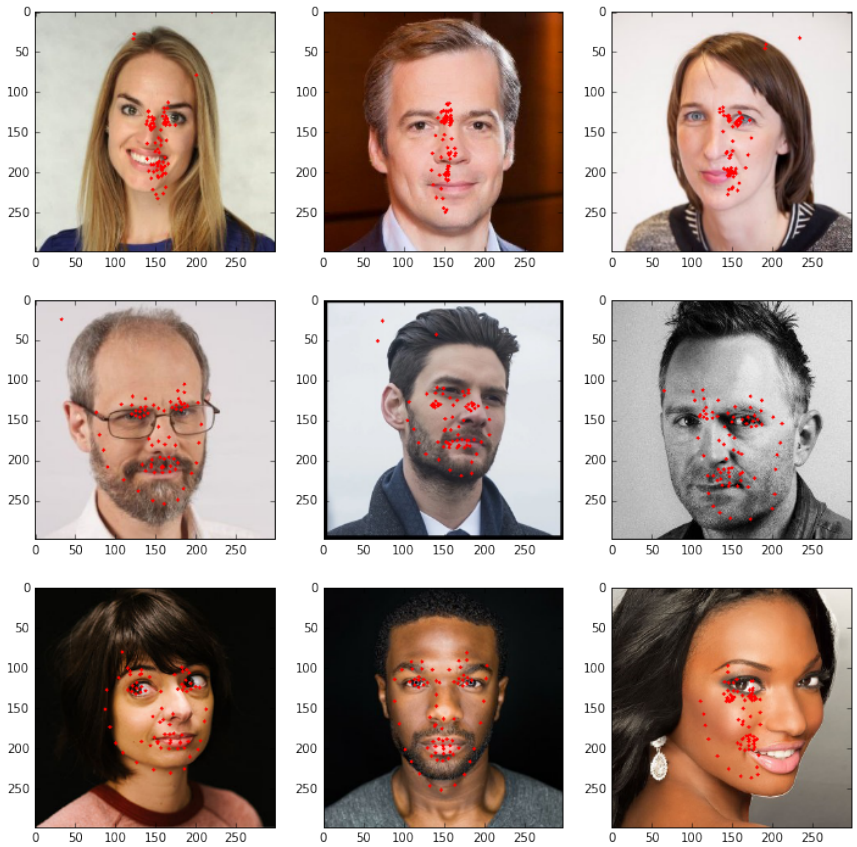
4. Nadam This optimizer as used for the benchmark model and showed good results (no tuning was made).

4 Results

4.1 Model evaluation and Validation

After many trials the model that did best was the model based on the ResNet50 retraining 2 blocks. this model not just had a good results as it takes a reasonable time to train. but the results steal not satisfactory as it does generalize enough on the test data it doesn't performs really well on data different from the on on the dataset.That is probably because all the images has the same background and this cam affect the outcome. At the end the model isn't a trustworthy model and some improvement has to be done.

for example when using random facial picture got from the Internet we get a result like this:



4.2 Justification

after all those tests and trials we can make a loss comparison for each model like:

model	training loss	validation loss
scratch	0.01547	0.02588
inception bottleneck	0.03599	0.03567
inception 2 blocks	0.0001108	0.003932
inception 4 blocks	0.0002229	0.003802
resnet50 0 blocks	0.01	0.013
resnet50 2 blocks	0.00005033	0.003779

By doing this comparison we see that the ResNet50 model has the most promising results. Probably is possible to get even better results by training more layers or even making some adjustments. Another step is collect more

datasets to combine with the MUCT dataset to improve the models generalization. But as this kind of training demands a lot of resources and keep the servers running started to get out of the budget the tests and development where halted at this point

5 Conclusion

This project throws in the spotlight a recurrent problem in the world of machine learning *the quality of the data* this is the most important detail and the work done here shows how it impacts. As pointed before we have a model that is capable to return a fairly accurate landmarks on pictures that it haven't seen but if the picture has the same *context* (i mean from the same dataset , the same background...) once it deviate from this the results are poor. and that is a problem that can be fixed by using some other datasets. as we can see for example in the project of Adrian Bulat on recognizing 3D facial landmarks [4] it uses more than 5 different datasets to reach an acceptable result.

5.1 Visualization

Some thing that was explored in this project and that did bring good improvements to the result is the use of concatenation, most of the examples of CNN you see today is straight forward (input->convolutional/pooling layers->fully connected->output) but when looking at big models like Inception or ResNet50 , those models have some trick under the sleeve. Like ResNet50 add the output from a previous convolutional block to the actual output, or how Inception run different convolutional stacks in a parallel to reach different features.

So with this in mind I did a use of such techniques like explained in the section 3.2.1 *CNN from scratch* to avoid big a convolutional layer stack and passing long time searching for a specific kernel size, I followed a different approach that improved the models performance by using two identical convolutional stack but with different kernel sizes (6 and 3) it was possible to get different features, and concatenating the output of each of those stacks before the fully connected layers.

This choice was made to be able to get the features with small and larger details.

Following this idea the same approach was taken in the 1 Bottleneck section where instead of using just the simple output from Inception a some manipulation was made to the data like using the *sine* and the *squared* of the

output and using one model that uses 3 sequences of fully connected layers to analyses those features separated before being concatenated.

The use of this kind of techniques was applied during all the process of this project and showed really good improvements and extreme important on the learning process as it was really useful to avoid the over-fitting of the model.

But anyway it was used with caution as it can relatively increase the training time, and the improvements aren't always paying off.

5.2 Reflection

Different that one outsider would think the challenge of this project is not to develop a machine learning but to understand the data you are working with, preparing the data to maximize the result. Mastering the data being used is more important them knowing the Machine learning / Deep learning structure you are going to use at the end this part is a lot of try and error. The second challenge is process power power convolutional networks really bring the home PCs to its limits and oblige us to use cloud solutions.

5.3 Improvement

This project shows that the approach used is correct and that with more resources is possible to delivery a reliable face alignment. The next step on such a project would be by starting by using some other datasets in conjunction to the MUCT so we can get a better generalization as all the images on this set has the same background it can bias the result.

And also by working on the hyper-parameters for the ResNet50 network and improving the results that would a start for getting a reliable face alignment

6 Hardware

For information propose all the training was executed on a virtual server hosted on google cloud The specs are the following:

8 CPUs with 60GB of memory and 1 NVIDIA Tesla P100 for the price of $1490.51 per month at a rate of 2.042 per hour

References

[1] Paul Ekman and Wallace V Friesen. Facial action coding system. 1977.

[2] Gareth J Edwards, Timothy F Cootes, and Christopher J Taylor. Face recognition using active appearance models. In *European conference on computer vision*, pages 581–595. Springer, 1998.

[3] Iain Matthews and Simon Baker. Active appearance models revisited. *International journal of computer vision*, 60(2):135–164, 2004.

[4] Adrian Bulat and Georgios Tzimiropoulos. How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230,000 3d facial landmarks). In *International Conference on Computer Vision*, 2017.

[5] S. Milborrow, J. Morkel, and F. Nicolls. The MUCT Landmarked Face Database. *Pattern Recognition Association of South Africa*, 2010. `http://www.milbo.org/muct`.

[6] Stephen Milborrow. Active shape models with stasm. *Stasm Version*, 3, 2009.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[8] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.