

# Machine Learning Engineer Nanodegree

---

## Capstone Project

---

Luiz Costa

January 29, 2017

## I. Definition

---

### Project Overview

Face Recognition is the task of identify or verify a person in a digital image or video stream through his face.

It's an important task for many areas, since social networks, where it can be used to label users in photos, until security, where it can be part of a program that reads a set of photos and try to identify a criminal.

The process can be divided in two phases: Detection and Recognition. For face detection most commonly used algorithm was proposed by Viola & Jones [1]. For face recognition, common algorithms include principal component analysis and eigenfaces.

With the explosion of cell phones with camera, thousands of photos are taken every day. Understand the objects and people in these photos, can play an important role in computer vision applications. In a social media application, a user, when upload a photo to his profile, an algorithm can automatically detect, recognize a person and asking for a user to confirm the label. With a lot of photos tagged, a search engine can take advantage this to find photos where specific persons appear.

My personal motivation for this project, is create a model to recognize famous faces in images. It can easily be extended to a video stream, where it can be used to label actors in films or tv series automatically. One example of this type of application is the ability of tag actors in video of Pornhub, a famous website for adults.[2].

### Problem Statement

Who is in the photo? This question summarizes the problem I'm proposing to be solved. Identify a person in image, can help to solve commons problems that we have today. For example, in a airport, is possible to use face recognition to identify criminals wanted by justice. In a search engine, is possible to find by images of specific person.

The task of face recognition can be very challenger. The person who appears in the image may be in different poses, with different light conditions, with eyeglasses. Because this, extract the important features of a face can be very difficult.

For this project, I'm proposing to build a model using a Convolutional Neural Network (CNN) for the task of face recognition. A model that uses a CNN, allow us to extract a wide range of unique features from an image and we can use this features to compare a face with others.

To solve this problem, my proposal is to construct a project that receives as input an image, add labels with the name of the person that appears in the image.

For this project I'm planning to use Deep Learning with a Convolutional Neural Network to recognize people in a image. A CNN model may be more effective to extract important features in a given image. The basic idea is:

- Receive an image as input
- Detect faces contained in the image
- Crop this faces and converts to the gray scales
- Use the cropped image in the gray scale in the CNN model
- Get the predicted label from the model and draw a box around the identified person

## Metrics

To compare the models, I'm using accuracy. I think it's a simple and straightforward metric, basically it's a fraction of how many labels the model has correctly identified. During the training phase, I will use Categorical Cross-Entropy [3] as the Loss Function. A loss function is important during training because it makes possible to understand whether or not the model is improving during the process. The important point here is to try to minimize the loss.

$$accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

## II. Analysis

---

### Data Exploration

The dataset used for this work was the PubFig83[4]. This dataset contains 11,071 face images of 83 public figures. The PubFig83[4] is a version of the dataset called PubFig[5], that contains

58,797 face images of 200 public figures. The decision of use a subset, was made with the goal to optimize the training time of the algorithm.

This dataset has some important characteristics. These characteristics make it a bit different of other face datasets, commonly used for face recognition benchmark.

- The images were taken in large variation in pose, light, expression, camera, etc
- the dataset classes are not balanced. The amount varies from 80 to 200 per class
- Duplicate images were removed

All the images in this dataset were resized to 100x100 pixels and are JPG images. In this dataset examples, are showed face images for Ashton Kutcher::



Note the variations in these images. There are a lot of noise and the face image is not completely isolated. It is a very challenger dataset, because all the noise can influence in the recognition task.

Here another set of image examples, this time, images of Barack Obama:



This dataset was divided as follows:

80% of data for training set  
10% of data for validation set  
10% of data for test set

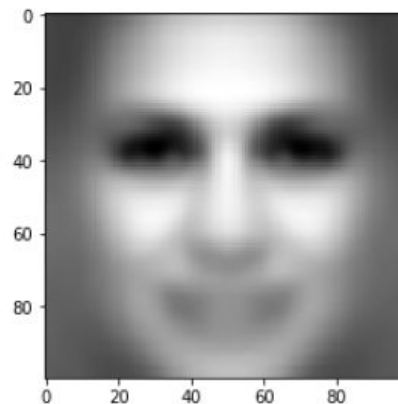
## Exploratory Visualization

As mentioned before, one of the characteristics of PubFig83 dataset, is the images are collected in large variation in pose, light, camera, etc. It can make very hard to recognize faces, because the faces can not be isolated.

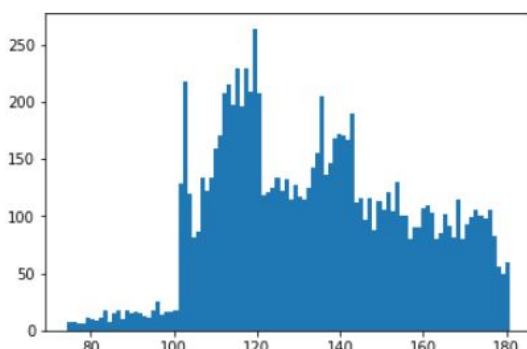
During the exploratory phase of this work, this problem was treated in different ways. I Started creating a baseline and benchmark model to compare with an approach using deep learning. One of the first things done, was to follow the recommendation of a paper[6] called Face Recognition using Eigenfaces. This paper describes an approach to use Eigenfaces[7] as a way to reduce dimensionality of face images.

We can think about the idea of Eigenfaces by fact that not all parts of a face are equally important or useful for face recognition[8].

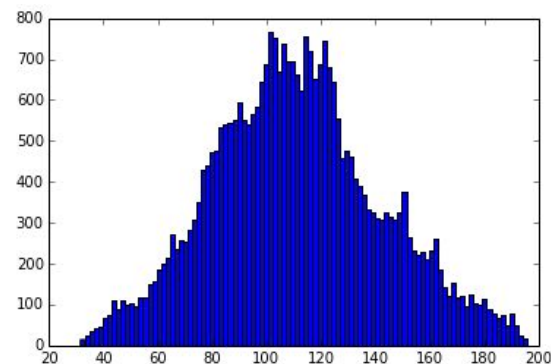
We starting centralizing the faces. To do this, we need to extract the mean face of the dataset.



The image above shows the mean face. If we take a look in the histogram, we can verify that the distribution of the mean is not a perfect normal curve. It occurs because the characteristic of the dataset, it does not have all faces isolated. It can be compared with an isolated face histogram.



*not isolated faces histogram*

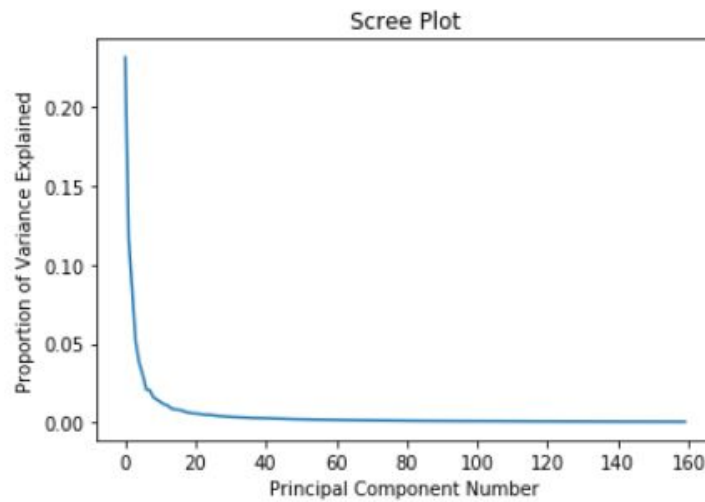


*isolated faces histogram*

After extract the mean, we normalize the dataset, it means, extract the mean and centralize the faces. The image below shows an example of faces after the normalization.



With a normalized dataset, we can start extracting the Eigenfaces. To do this, we used the PCA algorithm[9]. After run PCA using 160 components, we checked how variability are explained in by the components.



In this case, about the 20 first components account for most of the total variability in the data. The remaining components account for a very small proportion of variability.

We can finally check for the Eigenfaces. Here we are showing the first 9 faces.



The image above shows an example of the Eigenfaces extracted after the PCA. Note that the important parts of any faces was identified by the technique. It is possible to see eyes, nose and the mouth.

This data was used to train the classifier to the recognition task in a baseline model.




## Algorithms and Techniques

For solve the face recognition problem in this work, I used Deep Learning with a Convolutional Neural Network [10] (CNN) to recognize a person in a image.




### Load data and dataset organization

The idea was to load the data into arrays and feed the CNN models. To load the data, the dataset need be organized as follows:

- The training, test and validation data must be separated into folders with the names: Train, Test and Valid.
- Inside of each folder, all images of a class, must be in the same folder. A class in this case is the name of a person.

 test	83 items	Folder
 train	83 items	Folder
 valid	83 items	Folder

Example of the train, test, valid folders

 Adam Sandler	86 items	Folder
 Alec Baldwin	82 items	Folder
 Angelina Jolie	171 items	Folder

Example of the train folder

All the classes names was extracted from this folder structure.

I tested 3 possible architecture for my CNN model. VGG16[11], ResNet50[12] and a custom model.

## VGG16

The origin of VGG architecture was a paper published in 2014 [11]. This network is characterized by its simplicity, using only 3x3 convolutional layers stacked on top of each other, in increasing depth. Reducing the volume size is handled by max pooling. Two fully-connected layers, each with 4096 nodes are then followed by a softmax classifier.

I used Transfer Learning[13]. The idea was to starting with ImageNet weights and create a custom layers below that.

## RestNet50

ResNet is a short name for Residual Network. As the name indicates, the new terminology that this network introduces is residual learning.

What is Residual Learning?

In general, in a deep convolutional neural network, several layers are stacked and are trained to the task at hand. The network learns several low/mid/high level features at the end of its layers. In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of feature learned from input of that layer.[14] For this model, I used Transfer Learning too, starting with ImageNet weights.

## Custom CNN model

One of the experiments I did was to start from scratch. Creating my own CNN architecture.

My idea was to test a set of configurations and try to keep things simpler the other architectures and take advantage this on training time.

The custom model is a combination of a series of Convolutional Layers and Max Pooling layers. it increases the size of the filter as the layers get deeper.

## Parameters tuning

For all the models I used the Categorical Cross Entropy as loss function. As optimizers I used SGD,RMSPROP, ADADELTA.

The process to choose the optimizers was a lot of trial and error and checked the performance of the model. For SGD the best results I achieved was using learning rate about 0.005.

## **Benchmark**

I decide to compare my Deep Learning approach with a very know approach to recognize faces: the OpenCV Eigenfaces Recognizer[15]. In addition, I included, just for a baseline comparison, a simpler model using PCA and SVM approach to face recognition. I'm calling baseline model the PCA + SVM approach. The benchmark model is the OpenCV Eigenfaces Recognizer approach.

### **Problems with Eigenfaces Recognizer**

During the exploratory analysis for the benchmark model, I noted some problems with Eigenfaces Recognizer:

- 1) The training takes a lot of time
- 2) With this dataset, I got a very low accuracy

I tried reduce the training time and improve the accuracy changing the number of components parameter of the algorithm, but it does not work.

### **Using other Recognizer model**

Because the problems related above, I decide to try the other two options of OpenCV: FisherRecognizer and LBPHRecognizer [16].

After the tests, the best accuracy was achieved with LBPHRecognizer. Because this, different from the proposal, I will use the LBPHRecognizer as Benchmark model for this work.

For both baseline model and benchmark model, I used 11071 training, 1388 test images and I used accuracy as metric to compare the results.

### **Baseline Model**

For the baseline model, I used a data normalization process explained before. In summary, the data was load from dataset and:

- Extracted the face of each image using OpenCV Haarcascades Classifier[17]
- Converted all images to vector
- Calculated the mean of the image vector
- Centralized the images



After the data normalization process, the PCA was executed and extracted the Eigenfaces of this dataset. With the Eigenfaces, I plugged this in the SVM algorithm.

For this baseline model, I checked the the confusion matrix of the results. Here is a part of the matrix:

	precision	recall	f1-score	support
in/Adam Sandler	0.70	0.64	0.67	11
in/Alec Baldwin	0.89	0.80	0.84	10
in/Angelina Jolie	0.20	0.43	0.27	21
in/Anna Kournikova	0.41	0.41	0.41	17
in/Ashton Kutcher	0.42	0.50	0.45	10
in/Avril Lavigne	0.36	0.53	0.43	30
in/Barack Obama	0.51	0.70	0.59	27
in/Ben Affleck	0.56	0.42	0.48	12
in/Beyonce Knowles	0.36	0.38	0.37	13
in/Brad Pitt	0.30	0.43	0.35	30
in/Cameron Diaz	0.48	0.60	0.54	25
in/Cate Blanchett	0.30	0.38	0.33	16
in/Charlize Theron	0.35	0.35	0.35	20
in/Christina Ricci	0.33	0.43	0.38	14
in/Claudia Schiffer	0.67	0.33	0.44	12
in/Clive Owen	0.58	0.54	0.56	13
in/Colin Farrell	0.75	0.40	0.52	15
in/Colin Powell	0.75	0.55	0.63	11
in/Cristiano Ronaldo	0.86	0.71	0.77	17

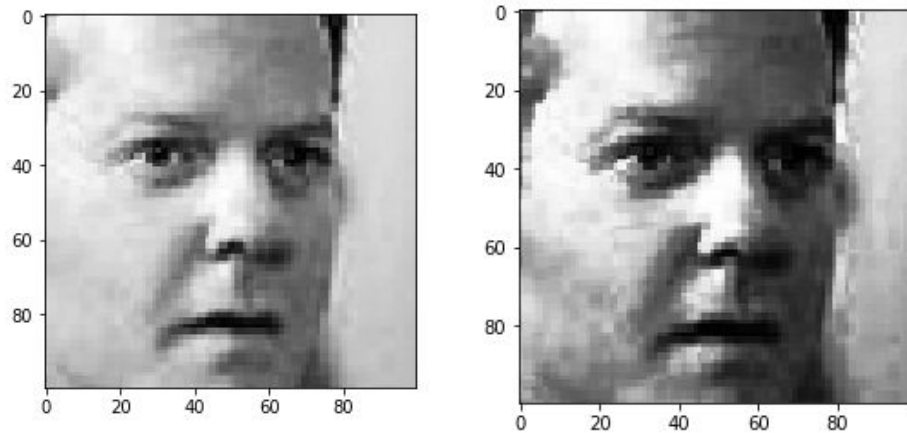
The final accuracy stayed around **50%**.

## Benchmark Model

For this model I starting loading the images from disk and start the training process using OpenCV LBPH Recognizer. To load the images from disk I used the OpenCV and applied a small set of normalization:

- Load the image from the disk with OpenCV
- Convert the image to gray scale
- Execute the Histogram Equalization
- Extract Face

An important part of this normalization process is the use of Histogram Equalization [18]. This technique adjusts the contrast of an image, it allows to enhance the image and get more detailed information.



The right image had a histogram equalization. In it is possible to see areas with more contrast.

The maximum accuracy achieved with this model was around **44%**.

The idea is to create a deep learning model to be compared with this model and the reference accuracy used is 44%.

### III. Methodology

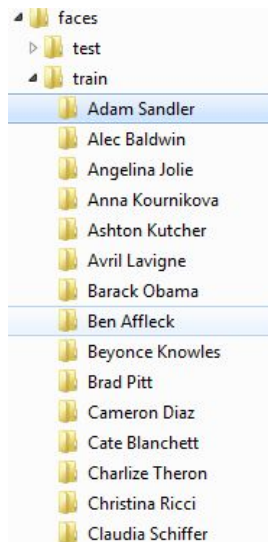
---

In this section I will describe the methodology used to create a deep learning model. Basically I tested 3 types of CNN architectures and choose the architecture with better accuracy.

#### Data Preprocessing

For the deep learning model, I used the same dataset organization for baseline and benchmark model. The dataset is organized with 2 levels:

*[Train, Test and Validation] / [person-name]*



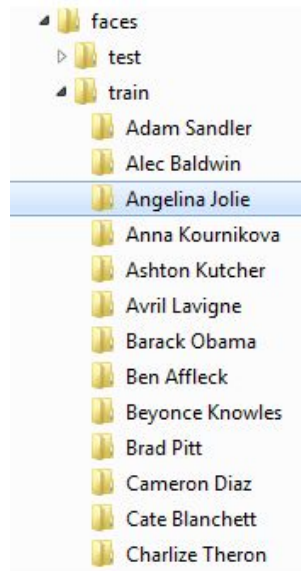
The figure shows the train folder with person names separated by folders

For each person folder, there are a set of images related with the person.

The data processing begins reading and extracting all image paths for the train, test and validation folders. It means loads an array with all the image paths for each person. For example, for the Adam Sandler, the preprocessing extract something like:

```
/faces/train/Adam Sandler/1.jpg  
/faces/train/Adam Sandler/11.jpg  
/faces/train/Adam Sandler/14.jpg
```

In the same process, besides load the images path, it creates a categorical representation for the training set. This categorical representation, is a list of arrays with values 0 and 1. The size of these arrays is 83, representing each person in the dataset. Each array has the value 1 in the position that represents the person and 0 for all other positions. Example:



In the image above, Angelina Jolie is highlighted. In this case, the position in the array that represents Angelina Jolie is 2. It because python array is zero based, then 0 is Adam Sandler, 1 is Alec Baldwin and 2 Angelina Jolie. It means that all images of Angelina will be represented by the same way in the categorical array:

```
../faces/train\Angelina Jolie\289.jpg  
[ 0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
```

Note that the array have only one value 1, all other values are zero and the position of value 1 in the array is 2.

The second part of the data processing is to load the images from disk and prepare the images to the feed the deep learning models. For each image path, this process can be summarized by:

- Load the image from disk with 100x100px size.
- Transform the image in a array
- Extract the face of the image. in train dataset, has just one face per image
- Expand the dimensions of the image array

One important thing is that I does not discard images that does not have face extracted. For this case, I add the original image in the array. In Tensorflow, these multidimensional image arrays are called tensors.

At the end of this preprocessing stage, the images are loaded into arrays and the categorical data is loaded in another array. With the training data loaded, is possible to start training the models.

## Implementation

In this section I will describe the approach to select the best model for the project. I divided the project in two parts: exploratory analysis / model selection and the build of a web project. In the first part I tested, did a parameter tuning, performed the training and selected the best model for the project. In the second part, I built a web application to demonstrate the model in “real world conditions”.

### Exploratory data analysis

During the exploratory data analysis, I tested 3 architectures of CNN: VGG16 and RestNet50 using Transfer Learning, and a custom model from scratch.

The hypothesis to test VGG16 and RestNet50 using transfer learning from ImageNet weights, was based on a idea that the patterns learned from ImageNet dataset are basic enough to be availed in the task of face recognition. It did not work as I thought.

For VGG16 I got a accuracy around 57%. It was better than the benchmark model, but it is not good enough to be used in production project.

For the ResNet50, an important step before the training was to resize the images to 224x224px. It because keras implementation uses this size to feed the model.

It can be complicated, if your machine does not have enough memory, because is necessary store in the memory all original data, convert to 224x224px and extract faces.

With this model, the achieved accuracy was around 42%.

I think the problem with the transfer learning approach is the dataset used for this project. It's very different from the imagenet and the weights does not generalizes well.

All these steps can be visualized with the jupyter notebook in the exploratory-analysis folder.

The custom model outperform these other two architectures. In this section I will describe how the custom model works.

### Model Selected: Custom

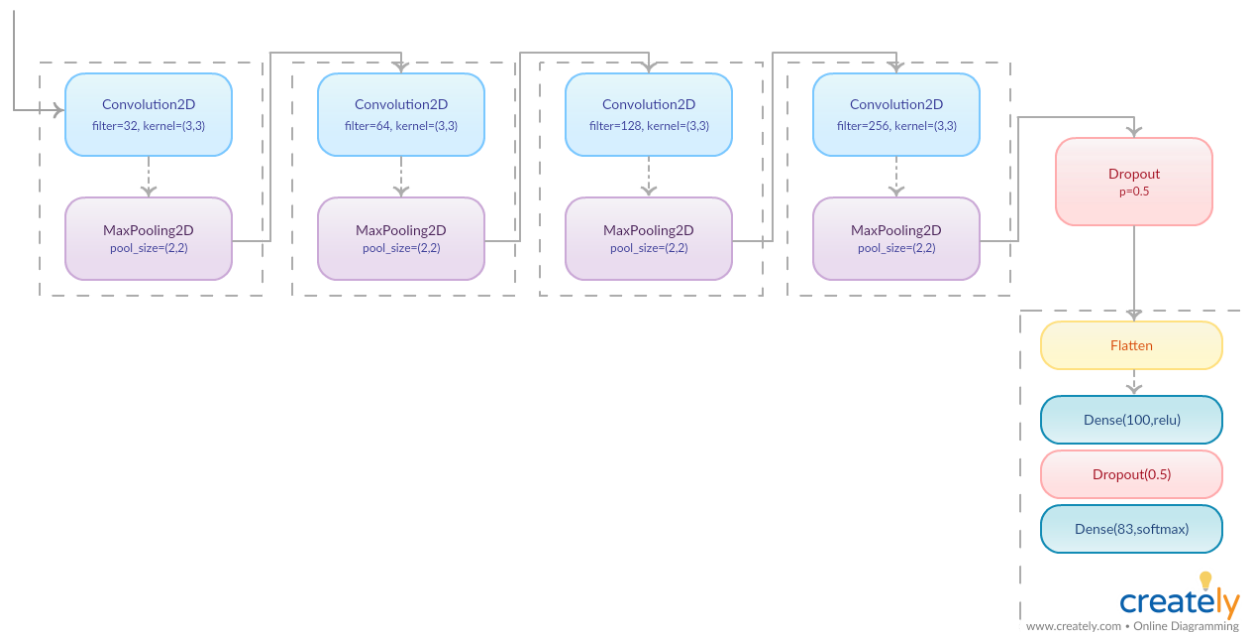
The creation of the customized model, had as a general guide, the premise of having a simpler architecture than the other two models. It can be explained by the type of task that must be performed by this model: Recognize Faces.

Faces generally has a set of standard characteristics and it does not have complicated pattern, because this, it does not need a very complicated transformations to learning this patterns.

The proposed model is a combination of 4 blocks of 3x3 kernel Convolutional Layers stacked on a max pooling layer to reduce the volume size, and being activated by a ReLu activation function. At each block the Convolutional Layer increases the number of filters, starting with 32 and end up with 256.

To avoid overfitting, I use a Dropout layer with  $p=0.5$  to dropping 50% of the connections.

Finally, a Fully Connected Layer with a Dropout and Dense layers. The image below shows the network architecture.



In the figure is possible to see the Convolutional layers with filter size increasing

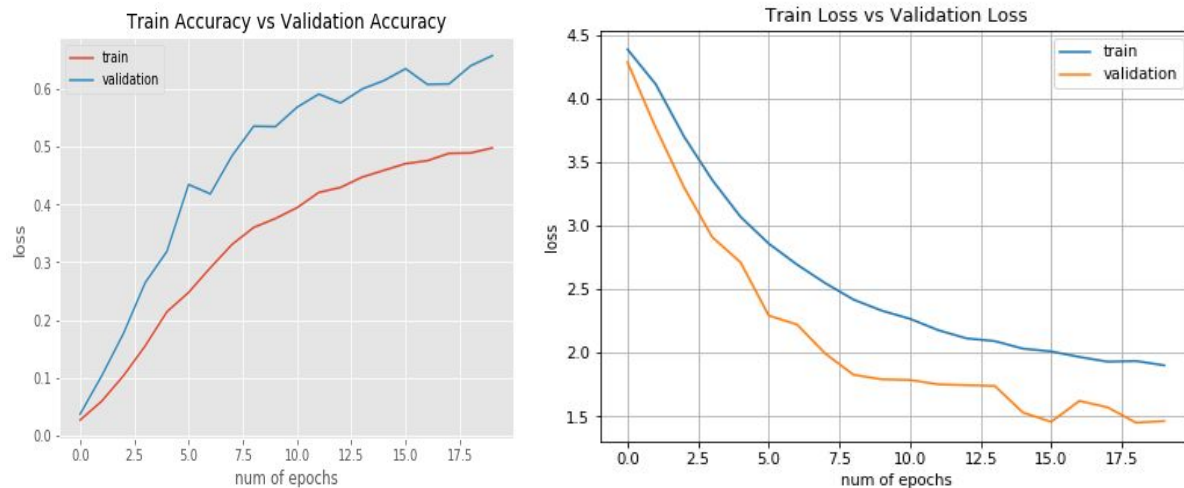
Initially, to train this model, I used color images with size 100x100px. The decision to use color images, was based in the fact, that using color, this model can learning more combinations of patterns, achieving better accuracy. It was a different decision used in baseline and benchmark model.

The metric used to evaluate this model was the accuracy.

I started training this model with 20 epochs using the Categorical Cross Entropy loss function and RMSprop optimizer.

I chose the Categorical Cross Entropy as loss function, was because my problem is a classification with more than 2 categories. For the other parameters, I used an experimental approach, looking at the performance of the model during training and check the final accuracy.

In the first attempt, I noted that with 20 epochs my model started underfitting. The figure below shows the model performance visualization for this configuration.

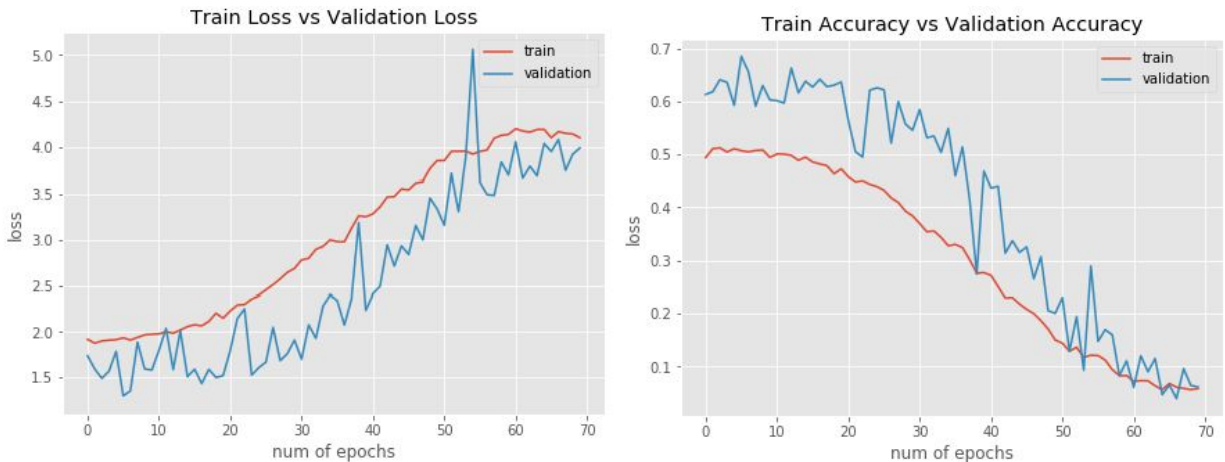


The first image shows the accuracy of model in train and validation. The other figure shows the behavior of the train loss and validation loss.

This model achieved 63% accuracy, but looking at the images above,, the model does not generalize very well. The train loss decrease more than the validation loss. It indicates an Underfitting.

## Refinement

In the refinement process ,again, I used a very experimental approach. First I just add more epochs. In the initial model, I used 20 epochs. The images above shows, the model was undefitting, but I decide to give a try and add more epochs. Setting to 70 epochs, the things got worse.



The images shows a very bumpy validation line. Both loss and accuracy was bumpy

With this model, the accuracy decrease to 0.06%. Very bad.

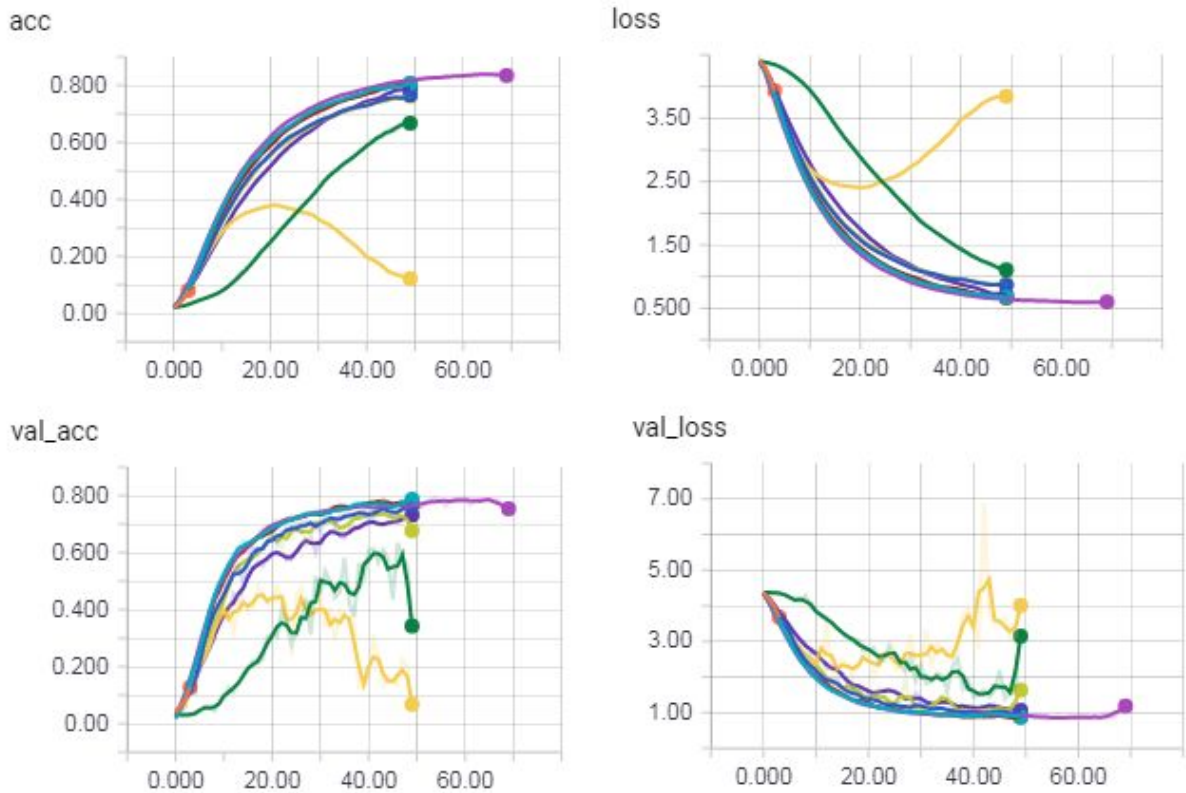
After this attempt, I started experimenting tuning parameters. The first adjust was try to use gray scale images. The Idea behind this was that, with gray scale images, a lot of noise can be reduced. To do this, I load the data in gray scale and set the epochs to 70 again and checked the results.



It seems, the result was not much different from the previous model. I kept trying to understand what I was doing wrong. This model achieved around 12% accuracy, better than previous. Because the result was better, I decided to keep gray scales images.

Another important parameter that I adjusted was the optimizer. I tested SGD with a small learning rate values and RMSprop. The figure below summarizes the evolution of these models in the tensorboard.





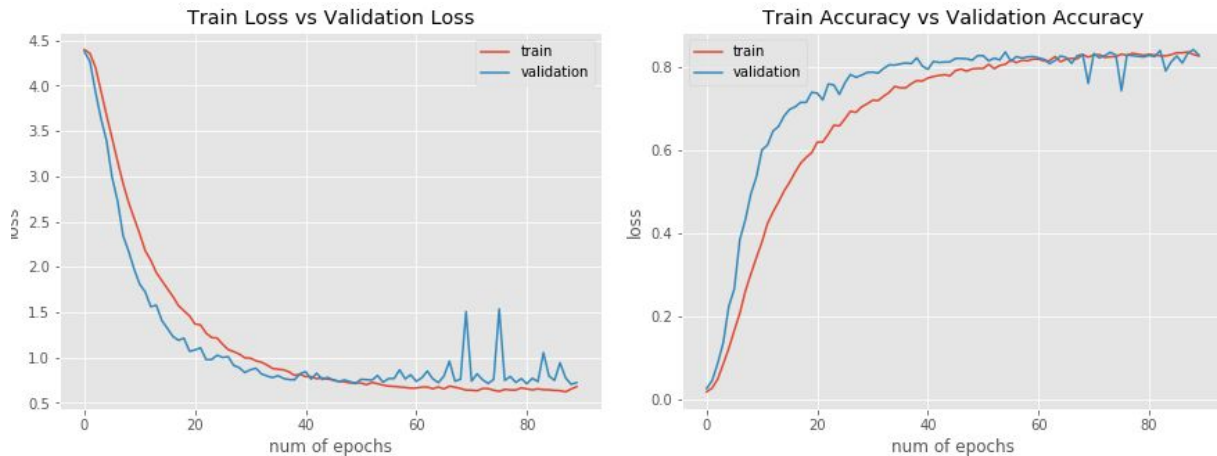
The figure shows a set of attempts to achieve the best model

In the image, one of the models achieved around 80% of accuracy. This model used a combination of tuning optimizer and number of epochs.

After a lot of attempts, the final model used:

- Gray scale images with isolated faces
- Uses **adadelat** as optimizer
- Set the number of epochs to 90

With this model, I achieved 83% of accuracy and the model visualization was better than the before models.



Best model result visualization

The image shows the behavior of the validation loss and accuracy very closed to the test.

## IV. Results

### Model Evaluation and Validation

To validate and evaluate the model, I created web project that can be used to see the results. It can be accessed by the url: <https://who-is-in-the-photo.herokuapp.com/> .

The final model described above, has a good performance with a different types of images. Images with isolated faces, works better, but it works with more than 1 face for image too.

Here a set of examples of faces recognized by the model.

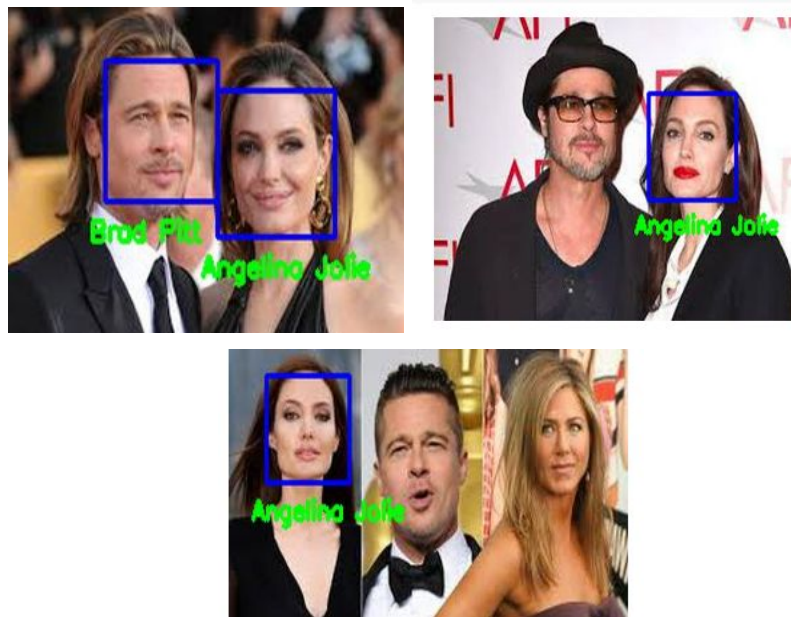


In this cases, all faces are isolated, and, the system had a very good performance. All 3 faces was recognized. This images was obtained by the internet, using google images.

Another important test is verify the accuracy of the model with different face conditions. This set of images does not have a clean face.



These two images Will Smith appears in different ways. Both the model identify correctly. For multi face detections, the performance can be improved. Take a look at these examples:



In the first image, the faces are correctly identified. Besides Brad Pitt hair is different from training images, the model recognizes him correctly. The second and third images, the model only recognized Angelina Jolie. It can happen because the face detection component does not work or because the model needs more examples of Jennifer Aniston and Brad Pitt.

## Justification

The deep learning model did a good job compared with the benchmark and baseline model. It generalizes better and besides the dataset nature, it works very well. To summarize the results we can see:

	<b>Baseline model (PCA+SVM)</b>	<b>Benchmark model (OpenCV+LBPHRecognitionizer)</b>	<b>Deep Learning Model (Custom Model)</b>
<b>Accuracy</b>	50%	44%	83%

Compared with Baseline and Benchmark model, the deep learning has the better performance.

## V. Conclusion

---

### Reflection

It was a very challenging project. First understand how the Face Recognition works. I had never implemented a face detection and recognition project before. After, I tried to understand how I can do face recognition using just traditional machine learning techniques.

During the exploratory data analysis, I had the opportunity to apply in practice several of the knowledge acquired in the classes. The most difficult part for me, was to create a CNN model from scratch. It takes a lot of time experimenting, tuning parameters to achieve a good performance. Another important part of this process was to see in practice the challenges of training a deep learning model. I had to learn how to set up a GPU in my personal computer to train the models. It was very important.

During the project phase, I had a chance to get out from jupyter notebook and write a production like code in python and deploy the application. It was very important for me, because I see in practice how to use the trained model in the production. During the classes, we don't have contact with production like projects, and it was very challenge to create one.

The good performance of the final model was a surprise for me. Initially I thought that this model would not be better than the current approaches like OpenCV or SVM. But the model proved itself better.

### Improvement

The final model has a good performance, but it needs some improvements to work better. The first point to improve would be the process of detecting faces with OpenCV. It does not working correctly when there are multiples faces and when the image conditions are no good. One of the improvement can be make is to use Dlib [19] to get landmarks points from the face.

Another improvement is to try another CNN architecture described in the paper Deep Face Recognition [20]. The architecture proposed, was prepared to handle set of variations in the images.

I think, there are some improvements to do in the web project to. It can show the last images uploaded to the server in the home. The error messages can be improved and the images can be uploaded to an external storage like Amazon S3.

## References

- [1] Paul Viola, Michael J Jones: International Journal of Computer Vision 57, pp. 137-154, Netherlands, 2004.
- [2] The Verge article about Pornhub, <https://www.theverge.com/2017/10/11/16459646/pornhub-machine-learning-ai-video-tagging>
- [3] Keras documentation about Categorical Cross Entropy, [https://keras.io/losses/#categorical\\_crossentropy](https://keras.io/losses/#categorical_crossentropy)
- [4] PubFig83 dataset page, <http://vision.seas.harvard.edu/pubfig83/>
- [5] PubFig dataset page, <http://www.cs.columbia.edu/CAVE/databases/pubfig/download/>
- [6] M. Turk and A. Pentland, "Eigenfaces for Recognition", Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71-86, 1991  
<https://www.cs.ucsb.edu/~mturk/Papers/mturk-CVPR91.pdf>
- [7] Wikipedia page for Eigenface, <https://en.wikipedia.org/wiki/Eigenface>
- [8] Citation from an post of <https://www.superdatascience.com/opencv-face-recognition/>
- [9] Wikipedia page for PCA, [https://en.wikipedia.org/wiki/Principal\\_component\\_analysis](https://en.wikipedia.org/wiki/Principal_component_analysis)
- [10] Wikipedia page for CNN, [https://en.wikipedia.org/wiki/Convolutional\\_neural\\_network](https://en.wikipedia.org/wiki/Convolutional_neural_network)
- [11] Karen Simonyan, Andrew Zisserman, Very Deep Convolutional Network for Large-Scale Image Recognition, <https://arxiv.org/abs/1409.1556>
- [12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, <https://arxiv.org/abs/1512.03385>
- [13] Transfer Learning, <https://machinelearningmastery.com/transfer-learning-for-deep-learning/>
- [14] Quora answer about how the ResNet works, <https://www.quora.com/What-is-the-deep-neural-network-known-as-%E2%80%9CResNet-50%E2%80%9D>
- [15] OpenCV Eigenfaces, [https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec\\_api.html#createeeigenfacerecognizer](https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_api.html#createeeigenfacerecognizer)
- [16] superdatascience.com - Face Recognition using OpenCV and Python: A beginner's guide, <https://www.superdatascience.com/opencv-face-recognition/>

- [17] OpenCV documentation - Face Recognition using Haar Cascades,  
[https://docs.opencv.org/3.3.0/d7/d8b/tutorial\\_py\\_face\\_detection.html](https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html)
- [18] Histogram Normalization,  
[https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram\\_equalization/histogram\\_equalization.html](https://docs.opencv.org/2.4/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html)
- [19] Dlib, <https://pypi.python.org/pypi/dlib>
- [20] Omkar Parkini, Andrea Vedaldi, Andrew Zisserman, Deep Face Recognition,  
<http://www.robots.ox.ac.uk/~vgg/publications/2015/Parkhi15/parkhi15.pdf>