Facial Keypoints Detection

Detect the location of keypoints on face image

Mattina Capparella, 1746513  
Federico Fontana, 1744946

Biometric Systems course project,  
teached by Prof. Maria De Marsico  
AY 2020/21

[Introduction 4](#_Toc63354951)

[External resources 5](#_Toc63354952)

[Datasets 5](#_Toc63354953)

[KAGGLE DATASET 5](#_Toc63354954)

[Tools, libraries and external models used 5](#_Toc63354955)

[Google Colab 5](#_Toc63354956)

[Pytorch and Torchvision 6](#_Toc63354957)

[Modus operandi 7](#_Toc63354958)

[Model selection 7](#_Toc63354959)

[Preprocessing 8](#_Toc63354960)

[Loss - Simple Split Trick Algorithm description 8](#_Toc63354961)

[2.15 8](#_Toc63354962)

[Our models 8](#_Toc63354963)

[Classifier only (*frm\_plain*) 9](#_Toc63354964)

[9](#_Toc63354965)

[Classifier with added noise (*frm\_n*) 10](#_Toc63354966)

[Classifier with added noise and denoiser (*frm\_n\_dn*) 11](#_Toc63354967)

[11](#_Toc63354968)

[Classifier with super resolution (*frm\_sr*) 12](#_Toc63354969)

[Classifier with added noise and super resolution (*frm\_n\_sr*) 13](#_Toc63354970)

[Classifier with denoiser and super resolution (*frm\_n\_dn\_SR*) 14](#_Toc63354971)

[14](#_Toc63354972)

[14](#_Toc63354973)

[14](#_Toc63354974)

[Classifier with super resolution and denoiser (*frm\_n\_SR\_dn*) 15](#_Toc63354975)

[Performance evaluation 16](#_Toc63354976)

[Open set identification 16](#_Toc63354977)

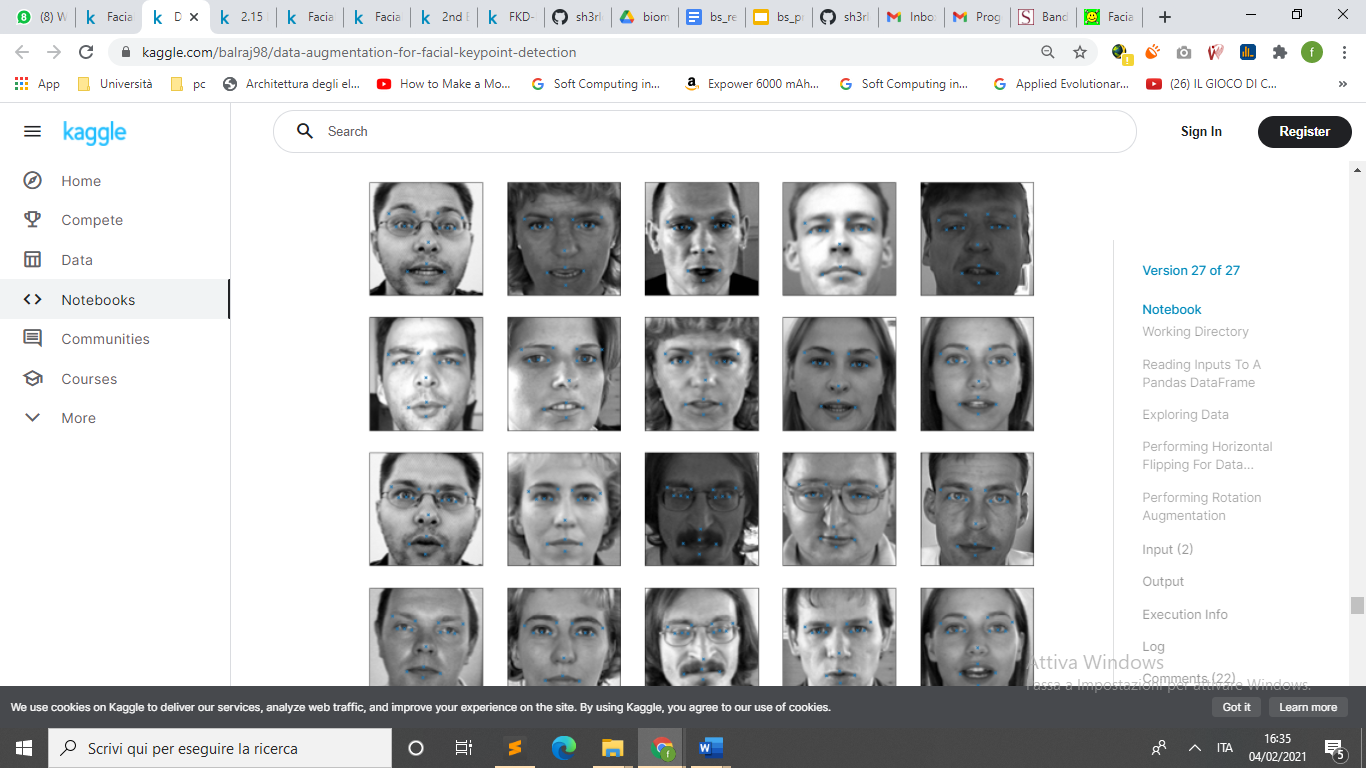
[Closed set identification 16](#_Toc63354978)

[Conclusions 18](#_Toc63354979)

[References 19](#_Toc63354980)

[Resources 19](#_Toc63354981)

# Introduction



Facial Key Points (FKPs) Detection

is an important and challenging problem in the fields of computer vision and

machine learning. It involves predicting the co-ordinates of the

FKPs, e.g. nose tip, center of eyes, etc, for a given face.

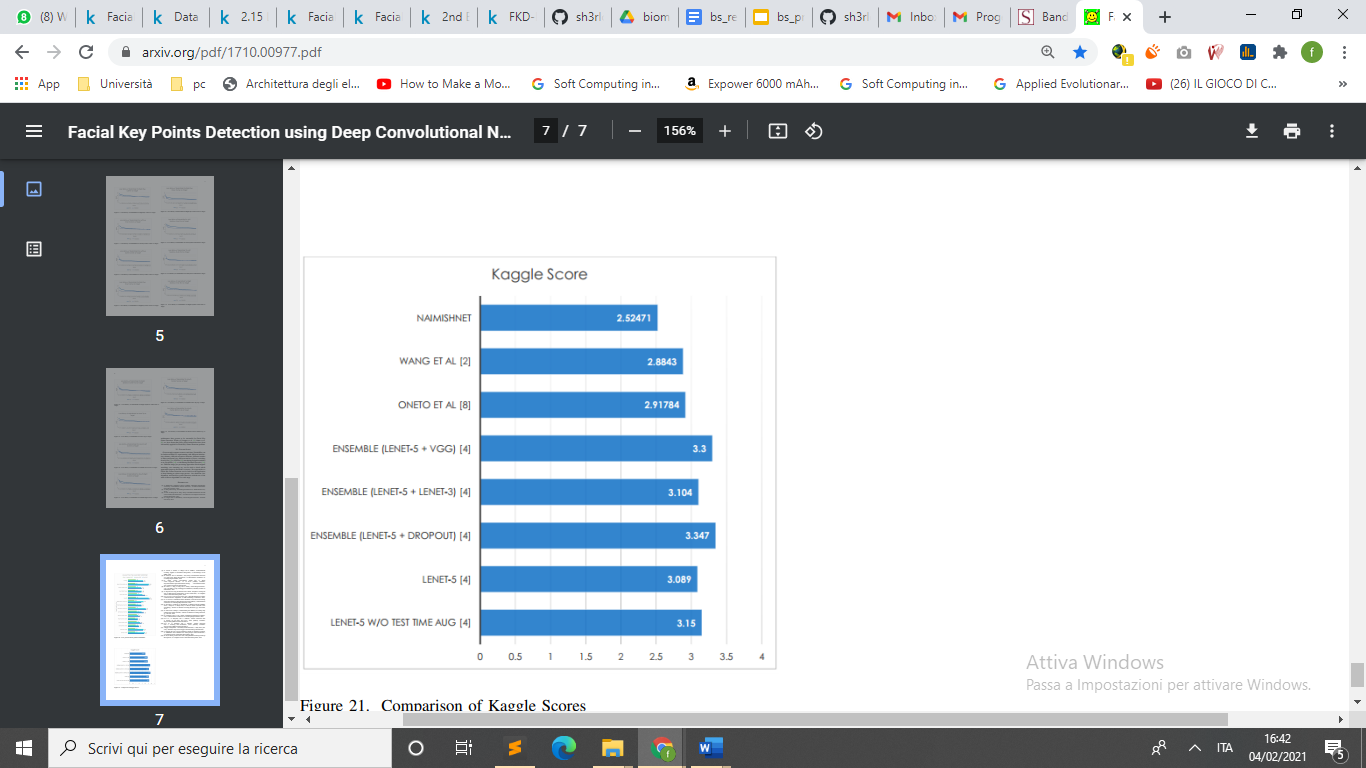
This project is about experimenting different models in FKPs Detection that is a critical element in face recognition.

However, there is difficulty to catch keypoints on the face due to complex influences from original images, and there is no guidance to suitable algorithms.

The **problem is** to predict the (x, y) realvalued co-ordinates in the space of image pixels of the FKPs for a given face image. It finds its application in tracking faces in images and videos, analysis of facial expressions, detection of dysmorphic facial signs for medical diagnosis, **face recognition**, etc.

Facial features **vary greatly from one individual to another**, and even for a single individual there is a large amount of variation due to pose, size, position, etc. The problem becomes even more challenging when the face images are taken under

different illumination conditions, viewing angles, etc.

With this report we want demonstrate that a good preprocessing can increase performance without modifying the model and without slowing down the algorithm too much.

# External resources

## Datasets

### KAGGLE DATASET

The dataset id taken from [Kaggle competition](https://www.kaggle.com/c/facial-keypoints-detection/data) :

Each predicted keypoint is specified by an (x,y) real-valued pair in the space of pixel indices. There are 15 keypoints, which represent the following elements of the face:

left\_eye\_center, right\_eye\_center, left\_eye\_inner\_corner, left\_eye\_outer\_corner, right\_eye\_inner\_corner, right\_eye\_outer\_corner, left\_eyebrow\_inner\_end, left\_eyebrow\_outer\_end, right\_eyebrow\_inner\_end, right\_eyebrow\_outer\_end, nose\_tip, mouth\_left\_corner, mouth\_right\_corner, mouth\_center\_top\_lip, mouth\_center\_bottom\_lip

Left and right here refers to the point of view of the subject.

In some examples, some of the target keypoint positions are misssing (encoded as missing entries in the csv, i.e., with nothing between two commas).

The input image is given in the last field of the data files, and consists of a list of pixels (ordered by row), as integers in (0,255). The images are 96x96 pixels.

Files:

**training.csv**: list of training 7049 images. Each row contains the (x,y) coordinates for 15 keypoints, and image data as row-ordered list of pixels.

**test.csv**: list of 1783 test images. Each row contains ImageId and image data as row-ordered list of pixels

**submissionFileFormat.csv:** list of 27124 keypoints to predict. Each row contains a RowId, ImageId, FeatureName, Location. FeatureName are "left\_eye\_center\_x," "right\_eyebrow\_outer\_end\_y," etc. Location is what you need to predict.

## Tools, libraries and external models used

### Google Colab

For the training and test phase we have used [Google Colab](https://colab.research.google.com/) because it offers an efficient platform that greatly reduces the time required for these steps since it gives high-capacity Nvidia GPUs usage for free.

We’ve made two notebooks: a [training one](https://colab.research.google.com/drive/1V4wo2cJpc9ANQ50VHoUyiMVtLwc_3uV8?usp=sharing) and a [demo one](https://colab.research.google.com/drive/1lPTYewjOPhMs33Tsmx1foCnmXPgT3uF3?usp=sharing) used to test each of the models explained further.

### Pytorch and Torchvision

For the development of the neural network’s part of the application we have used the [Pytorch framework](https://pytorch.org/) since greatly simplifies nearly all the parts of the processes by enabling computation on GPUs thus speeding algebraic operations up by many magnitudes.

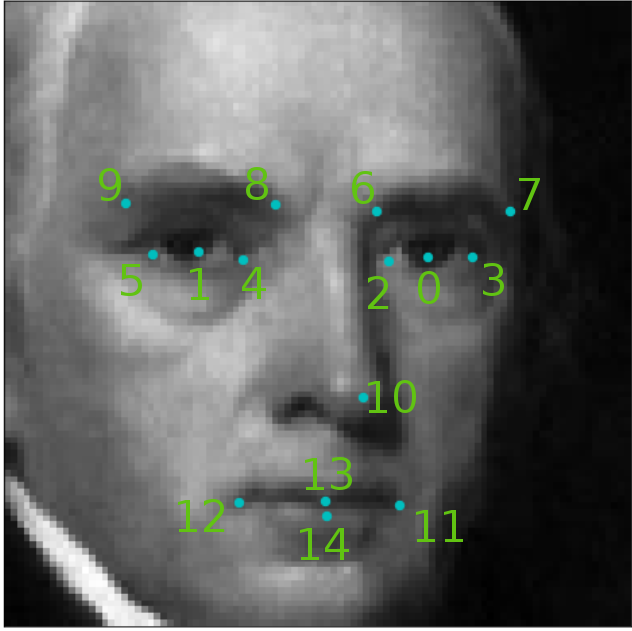
Pytorch comes with the [Torchvision library](https://pytorch.org/docs/stable/torchvision/index.html), a collection that contains many functions for images’ manipulation, used extensively in our preprocessing phases.

# Modus operandi

## Model selection

To demonstrate our idea of goodness of preprocessing we searched a model that perform well, but where no preprocessing was made.

Anyway, often preprocessing is not made because its time execution, meanwhile in FKPs detection is fundamental the speed of execution. If we need to detect the points in a video and in real time we need and extremely fast algorithm.

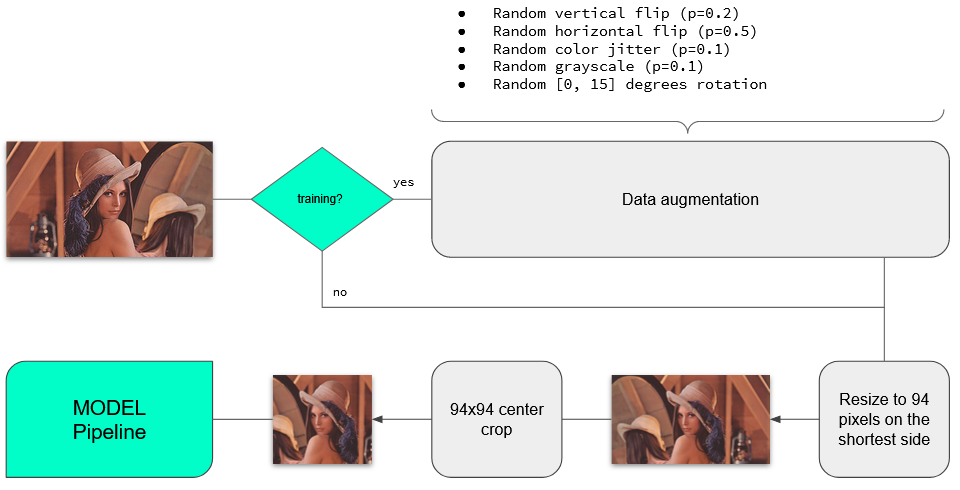


## 

## Preprocessing

We are concentrating our focus on preprocessing in particular 2 kind of o preprocessing:

* Real-time preprocessing: the speed of this kind of preprocessing is fundamental, we need to keep this process light and fast.
* Training preprocessing: we can use augmentation to increases dataset size and introducing natural image distortion to make the model generalize better.



(imamgine da cambiare)

## Loss - Simple Split Trick Algorithm description

Questa parte la faccio al volo perché non sono sicuro sia questo l’algo.

We choose [Simple Split Trick](https://www.kaggle.com/phylake1337/2-15-loss-simple-split-trick) because of the performance and the absence of augmentation and real-time preprocessing.

This solution is a **convolutional neural network** (CNN)

Using LeakyReLU, BatchNormalization, MaxPool2D

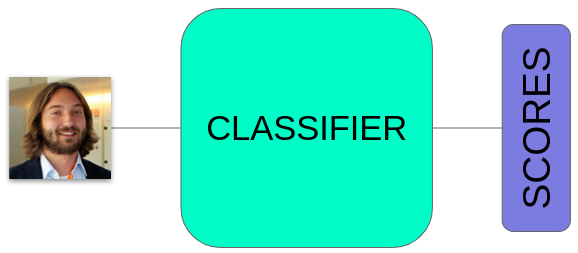
And in the final fully connected layer Dropout(.1),

**DOPO è ANCORA TUTTO VECCHIO**

# 2.15

# Our models

## Classifier only (*frm\_plain*)



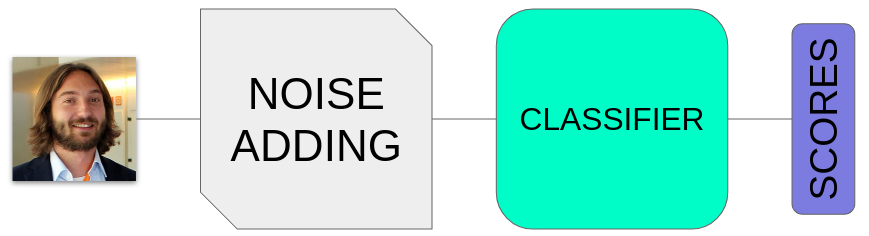
This model implements a simple **classifier** for the LFW dataset.

This classifier is a fine-tuned version of ResNet18 [6], basically the same architecture exposed in the paper but with the last linear layer replaced by another one that fits our classes’ number.

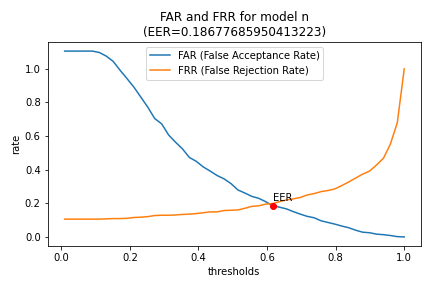
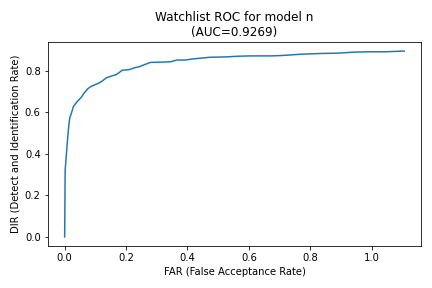
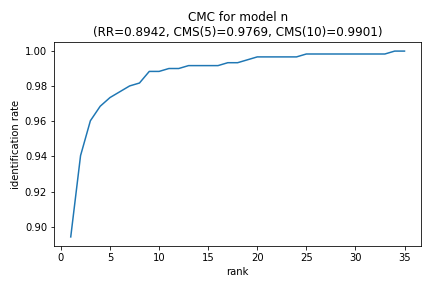
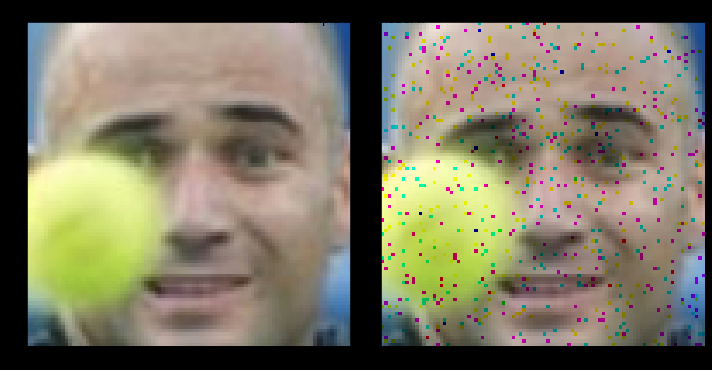
The output is then fed to a softmax function that basically returns a probability vector such that .

### 

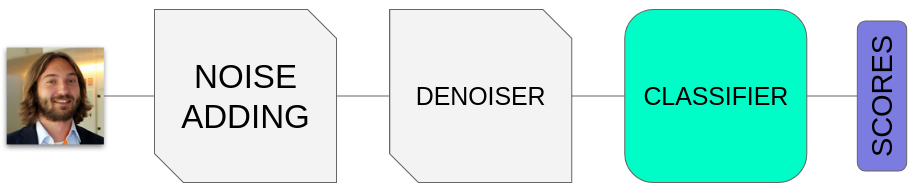
## Classifier with added noise (*frm\_n*)



This model adds **random noise** to the image and passes it to the classifier as before.

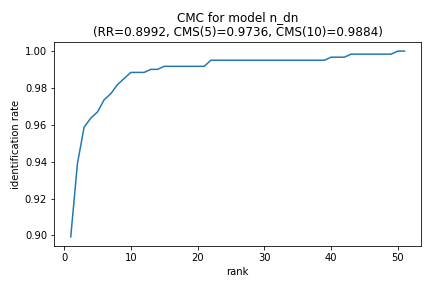
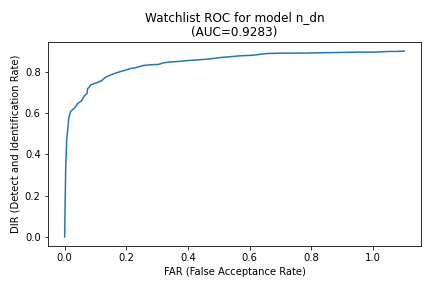
Each pixel of the image is randomly colored with a certain probability independent from the other pixels of the image, set as in our experiments. 

## Classifier with added noise and denoiser (*frm\_n\_dn*)



This model, similar to the previous adds, random noise to the image and passes it to the classifier as before.

Differently, in this case, we included a **denoiser** module based on [5].

In this situation we try to use only this module for removing the effect of the noise without the super resolution.

## 

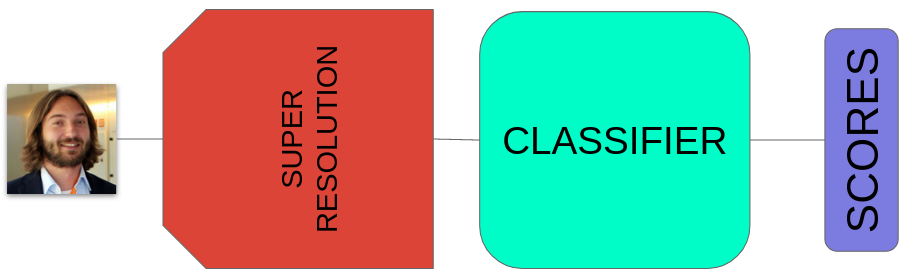
## 

## 

## 

## 

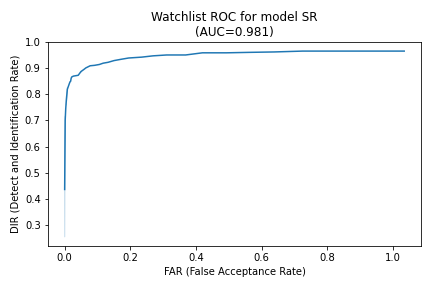
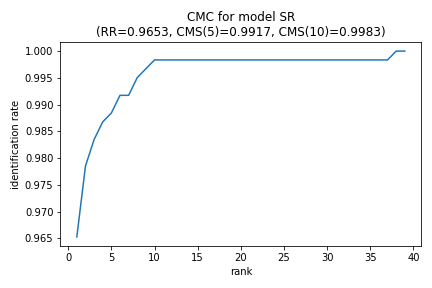
## Classifier with super resolution (*frm\_sr*)

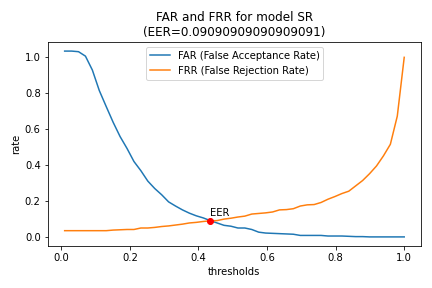


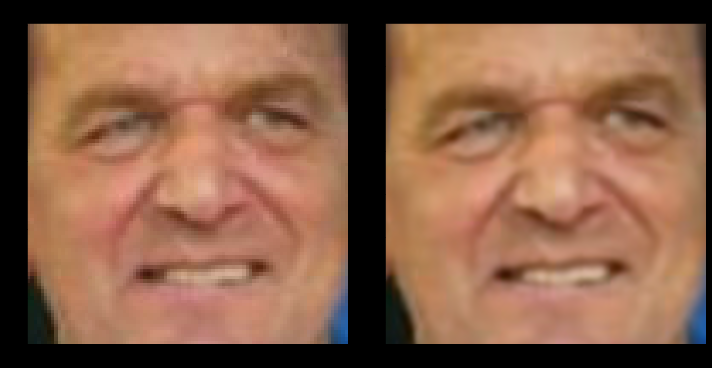
This model is a variant of the first.

In this case, before the classification each image in the dataset is upscaled by a **super resolution** module [4], and in this case we have verified that the results are better mainly because of the more features.

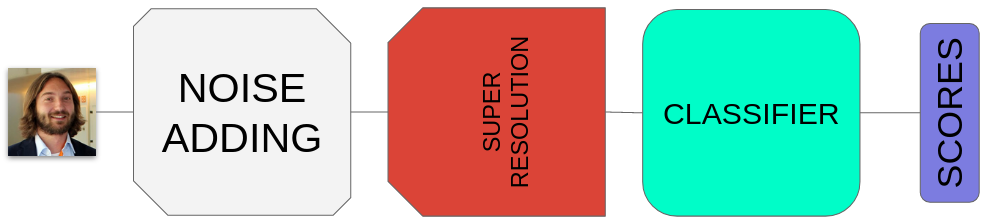
As it’s trained on classifier’s loss, this model is mainly for computers and not for the human eye as it “prefers” to add features to help the classification rather than beautiful-looking ones.



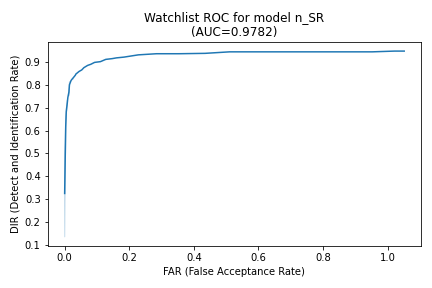
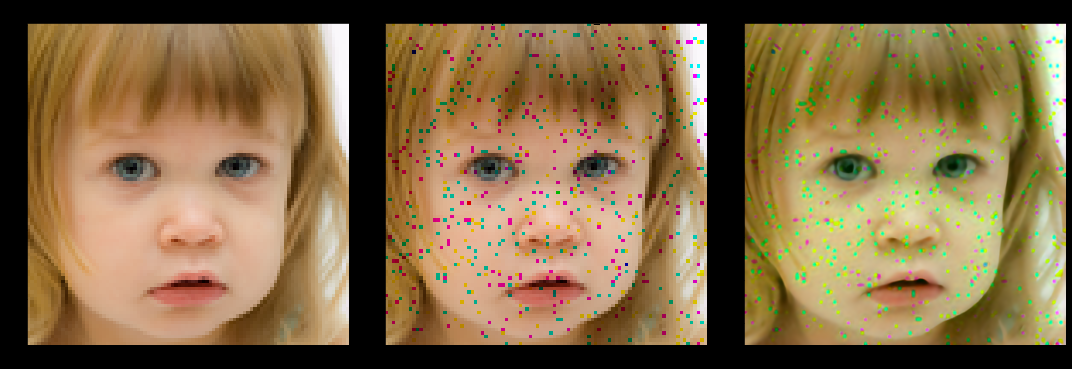
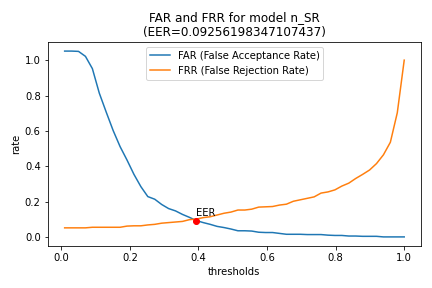
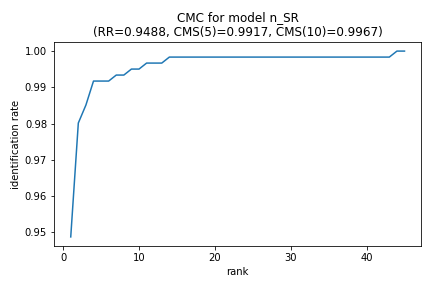




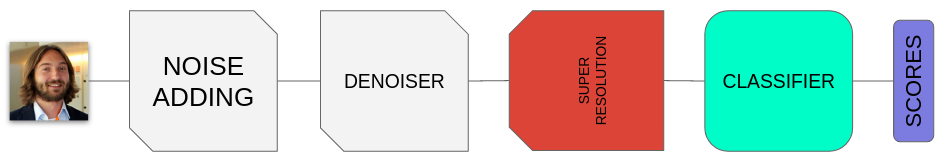
## Classifier with added noise and super resolution (*frm\_n\_sr*)



This model is a variant of the previous, that we have included to verify the performances of the super resolution in presence of noise.



## Classifier with denoiser and super resolution (*frm\_n\_dn\_SR*)



In this model we have included a denoiser module to mitigate the effect of the noise in the previous case.

## 

## 

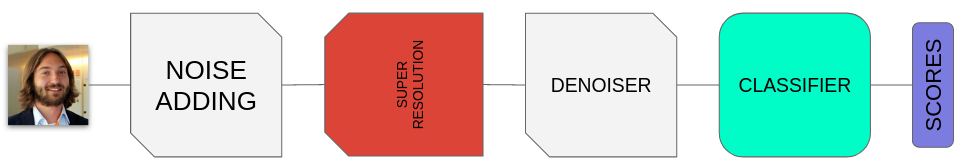
## 

## 

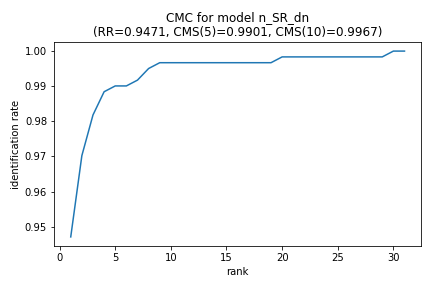
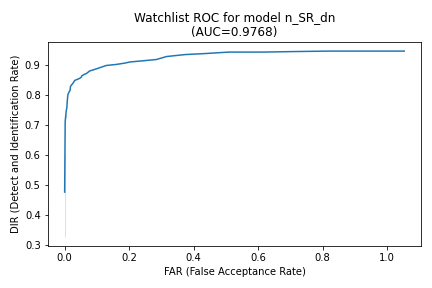
## 

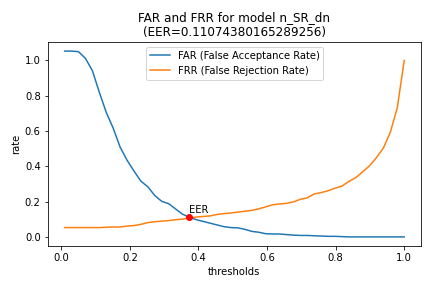
## 

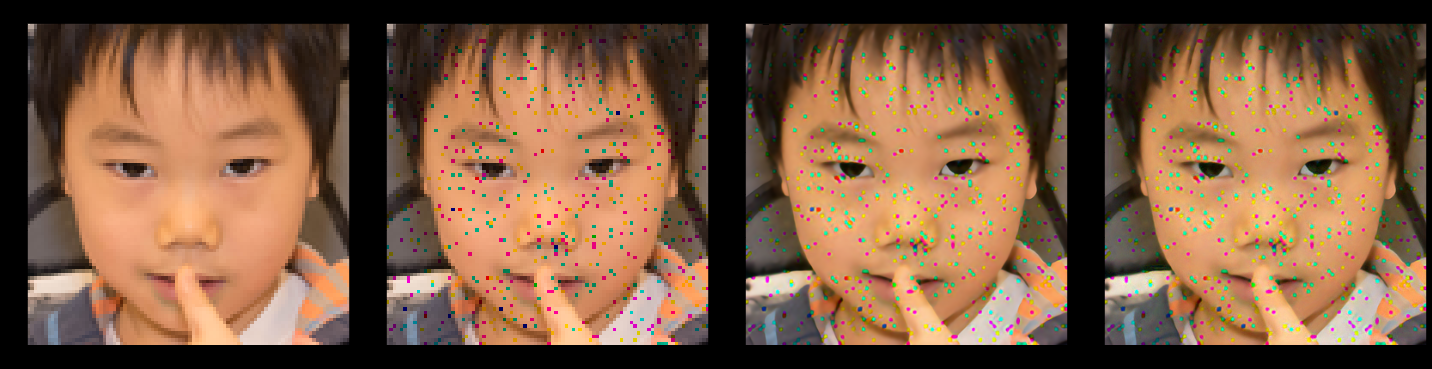
## Classifier with super resolution and denoiser (*frm\_n\_SR\_dn*)



In this model the denoiser module has been placed after the Super Resolution module.

This variant respect to the previous model did not change the performance significantly.





# Performance evaluation

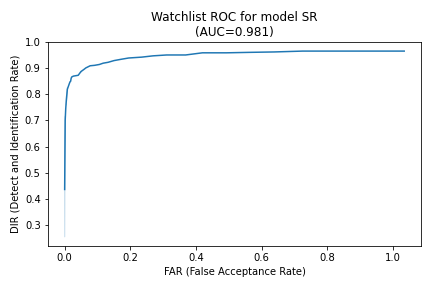
In this section we’ll discuss the performance of the models above.

We can already notice that the models including the super resolution module performs better than others, in numbers.

## Open set identification

We can look at the **AUC** to make a brief confront in the open set identification scenario:

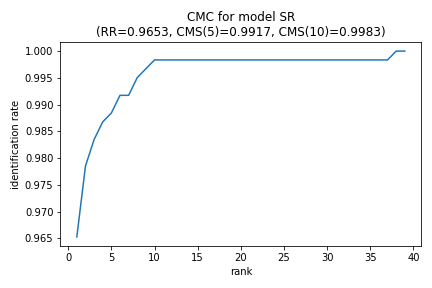
Thanks to this parameter we can see the goodness of our **ROC curve** in numbers.

This curve gives us an idea of all Detect and Identification Rates and the relative False Alarm Rates of the system. 

This is useful for the application in real contests because we can decide the watchlist threshold based on this plot; in fact each point of this curve represents a possible threshold and immediately we can see the DIR and the FAR associated with this value.

## Closed set identification

We can use the **Recognition Rate** (CMS(1)) for make a brief confront:

This parameter tells us the capacity of the system to recognize the classes (subjects) at the first position (rank): as we can see, the model with super resolution already at first position recognizes the 96% of subjects instead of the 92% of the plain model. 

We can see the performances at various ranks in the **CMC** plots.

## 

## Conclusions

Given these results, we can conclude that:

* **super resolution** module **can improve the performance** of face recognition systems and, for example, it can be used in some situations when we are in some circumstances of a non correct acquisition of subjects to make up for this lack;
* **noise seems not to worsen performances that much** in the open set identification task (), while in the closed set identification task this gap is more evident (), although they are different metrics;
* **denoising seems not to produce the desired effects**, with just a shy increase of in both AUC and RR in models without super resolution and nearly equal values in models with;
* **denoising after or before super resolution doesn't matter so much** in our experiments, since performances are nearly equal in both cases.

Below we can see a table that resume the results for all the models, with better scores in bold for each column:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **model** | **RR** | **CMS(5)** | **CMS(10)** | **AUC** | **ERR** |
| *plain* | 0.9289 | 0.9851 | 0.9917 | 0.9325 | 0.1669 |
| *n* | 0.8876 | 0.9752 | 0.9884 | 0.9294 | 0.1867 |
| *n\_dn* | 0.8942 | 0.9785 | 0.9901 | 0.9382 | 0.1801 |
| *SR* | **0.9653** | **0.9917** | **0.9983** | **0.981** | **0.090** |
| *n\_SR* | 0.9488 | **0.9917** | 0.995 | 0.9758 | 0.1041 |
| *n\_dn\_SR* | 0.9455 | **0.9917** | 0.995 | 0.9787 | 0.1041 |
| *n\_SR\_dn* | 0.9471 | 0.9901 | 0.9967 | 0.9768 | 0.1107 |

Overall, the model that performed best is , the one with just super resolution and classifier, confirming our conjecture.

# References

1. Huang et al., 2007,   
   Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments,  
   <http://vis-www.cs.umass.edu/lfw/>
2. Karras et al., 2018,   
   A Style-Based Generator Architecture for Generative Adversarial Networks,  
   <https://arxiv.org/pdf/1812.04948.pdf>
3. Singh et al., 2015,   
   Super Resolving Noisy Images,  
   <https://ieeexplore.ieee.org/document/6909760>
4. Wang et al., 2018,   
   ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks  
   <https://arxiv.org/pdf/1809.00219.pdf>
5. Ulyanov et al., 2020,   
   Deep Image Prior,  
   <https://arxiv.org/pdf/1711.10925v4.pdf>
6. He et al., 2015,   
   Deep Residual Learning for Image Recognition,  
   <https://arxiv.org/pdf/1512.03385.pdf>

# Resources

* Our Google Colab’s notebook used to train the models <https://colab.research.google.com/drive/1V4wo2cJpc9ANQ50VHoUyiMVtLwc_3uV8?usp=sharing>
* Our Google Colab’s notebook for demonstration  
  <https://colab.research.google.com/drive/1lPTYewjOPhMs33Tsmx1foCnmXPgT3uF3?usp=sharing>
* Our GitHub repo, home of all the code used  
  <https://github.com/rom42pla/bs_project>