Facial Keypoints Detection

Detect the location of keypoints on face image

Mattia Capparella, 1746513  
Federico Fontana, 1744946

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

[Introduction 3](#_Toc63518590)

[External resources 4](#_Toc63518591)

[Datasets 4](#_Toc63518592)

[KAGGLE DATASET 4](#_Toc63518593)

[Tools, libraries and external models used 4](#_Toc63518594)

[Google Colab 4](#_Toc63518595)

[Pytorch and Torchvision 5](#_Toc63518596)

[Modus operandi 6](#_Toc63518597)

[Model selection 6](#_Toc63518598)

[Preprocessing 7](#_Toc63518599)

[Luis’s FKPs Detection Algorithm description 7](#_Toc63518600)

[2.15 7](#_Toc63518601)

[Our methods 7](#_Toc63518602)

[Model baseline (plain) 7](#_Toc63518603)

[Model with Augmentation (*augmentation\_model*) 9](#_Toc63518604)

[Model with real-time filter (*real\_model*) 10](#_Toc63518605)

[Model with augmentation and real-time filter (*real\_aug\_model*) 11](#_Toc63518606)

[Performance evaluation 12](#_Toc63518607)

[(loss function image 1) (loss function image 3) (loss function image 3) 12](#_Toc63518608)

[(MSE image 1) (MSE image 2) (MSE image 3) 12](#_Toc63518609)

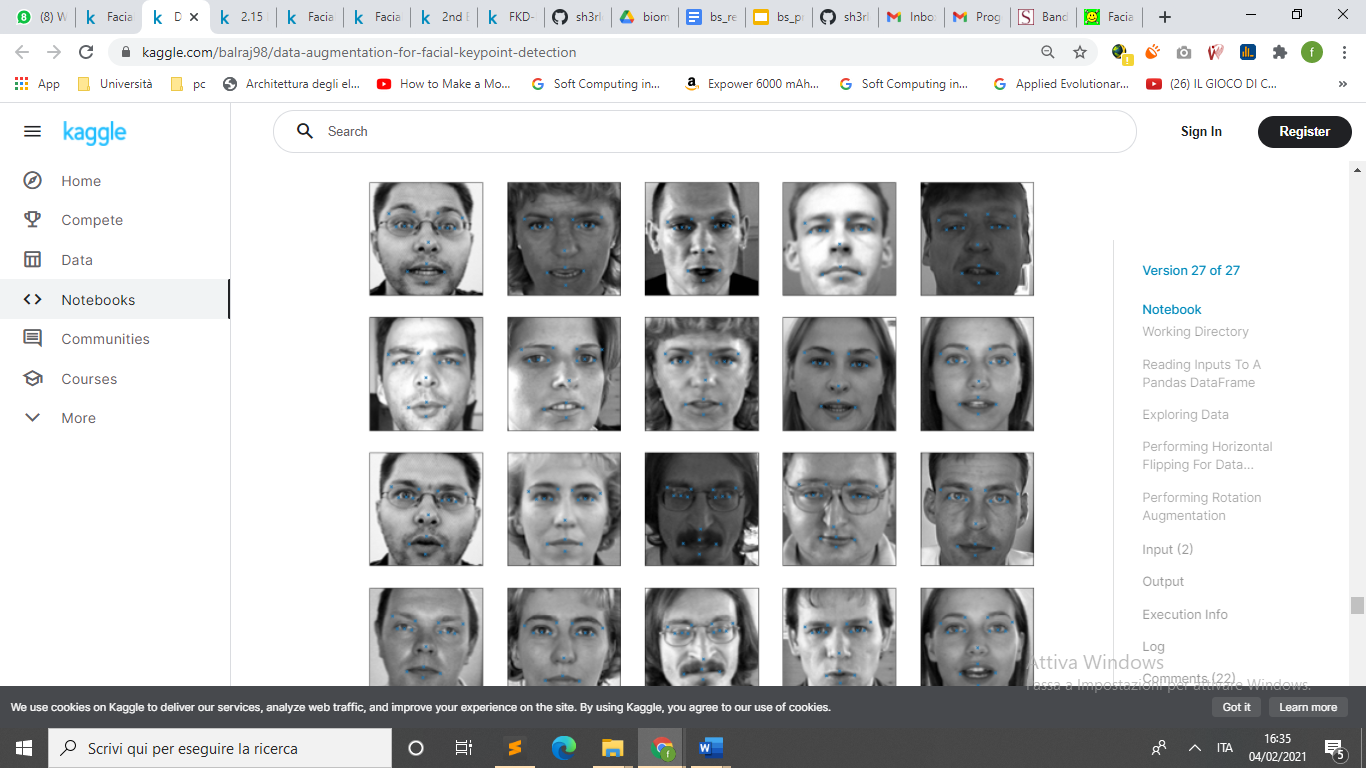
[(time of prediction image 1) (time of prediction image 2) (time of prediction image 3) 12](#_Toc63518610)

[Conclusions 13](#_Toc63518611)

[References 14](#_Toc63518612)

[Resources 14](#_Toc63518613)

# 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.

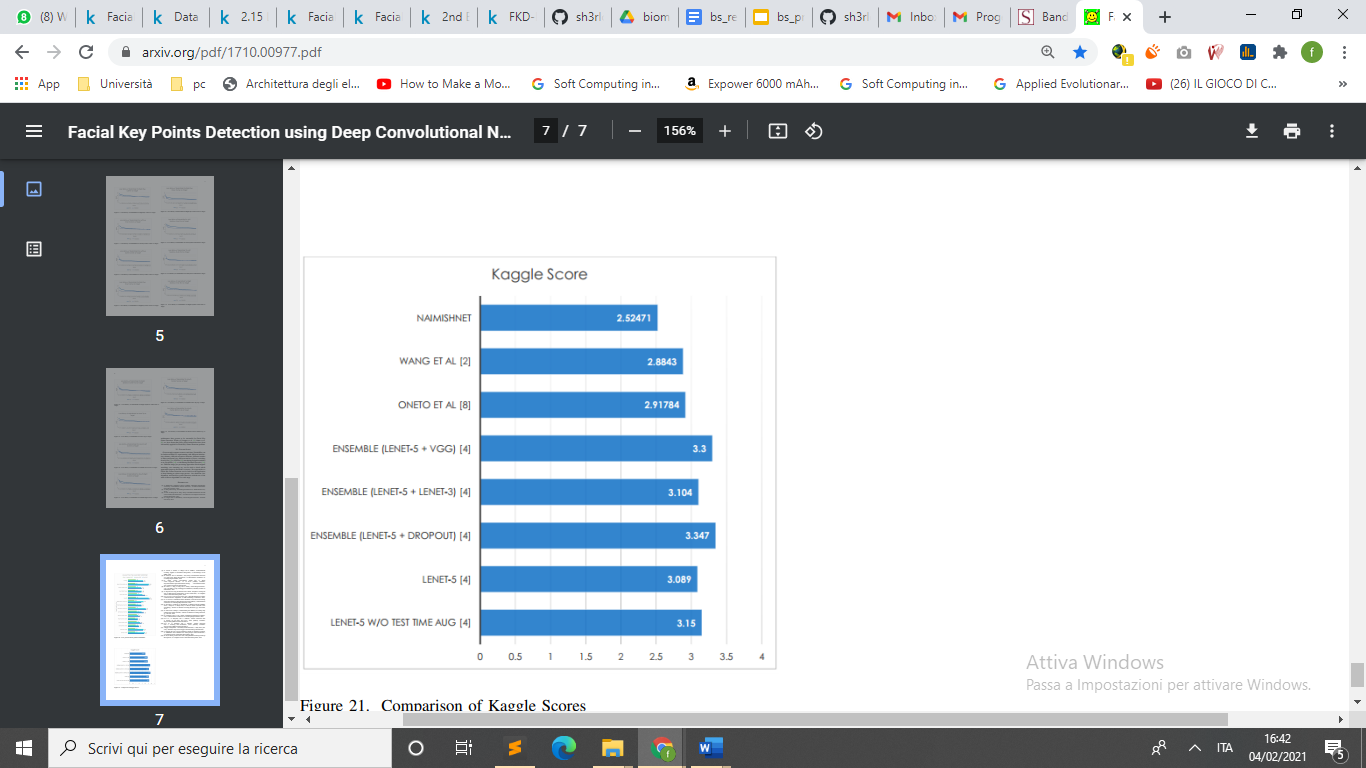
This project is about experimenting different models in FKPs Detection that is a critical element in face recognition.

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.

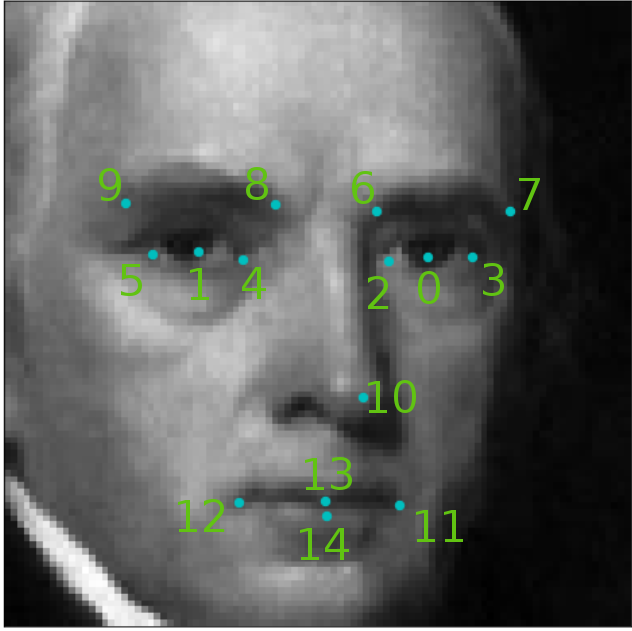
Pytorch comes with the [Torchvision library](https://pytorch.org/docs/stable/torchvision/index.html), a collection that contains many functions for images’ manipulation, used extensively in our preprocessing phases.

# 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:

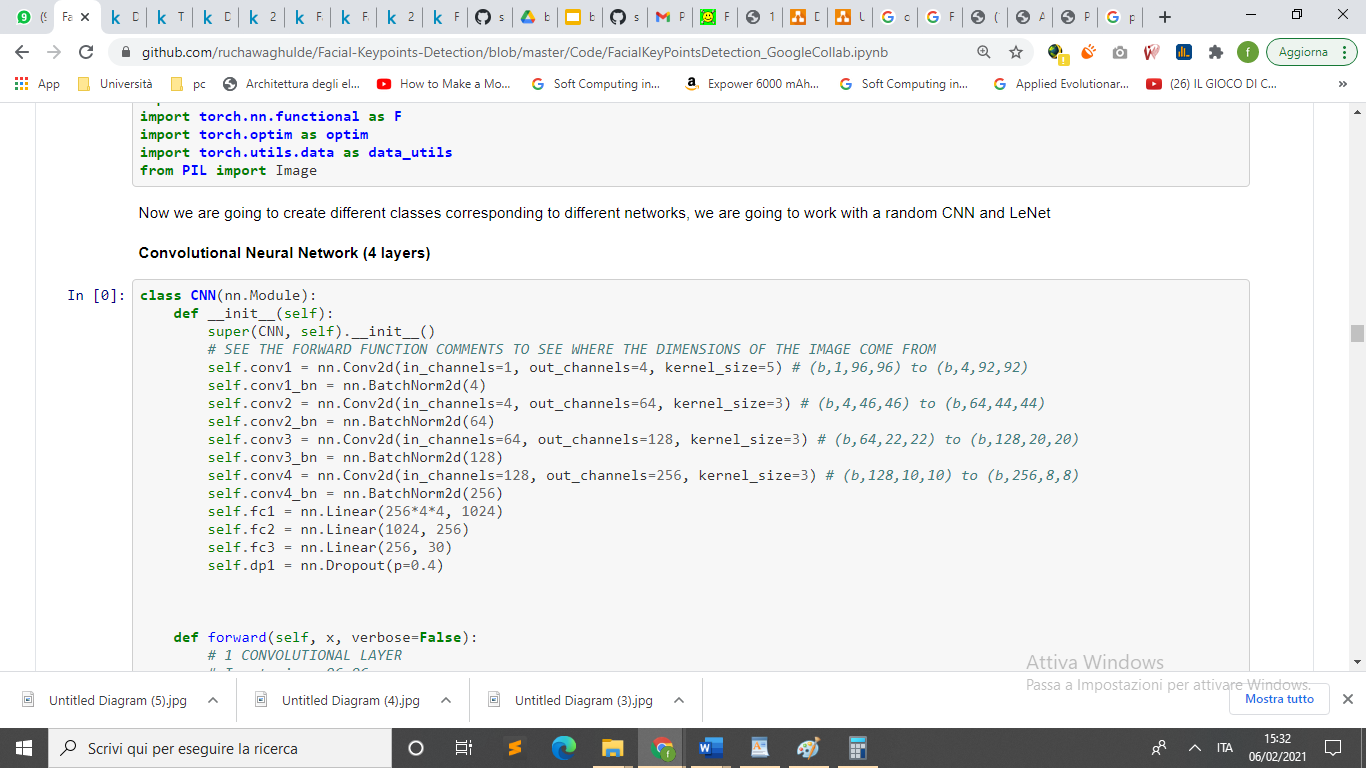
* Real-time preprocessing: the speed of this kind of preprocessing is fundamental, we need to keep this process light and fast.
* Training preprocessing: we can use augmentation to increases dataset size and introducing natural image distortion to make the model generalize better.

## Luis’s FKPs Detection Algorithm description

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

We choose siddh30’s algorithm because of the performance and the absence of augmentation and real-time preprocessing.

his solution is a **convolutional neural network** (CNN):



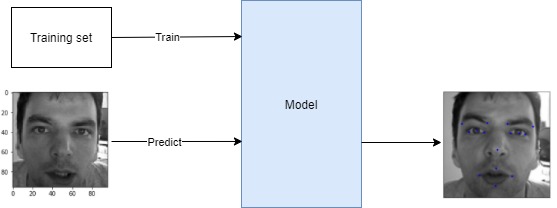
Convolutional Neural Network with 4 convolutional layer and 3 fully connected layer.

Each Convolutional layer performs ReLu activation and MaxPooling on its input to reduce the input size.

# 2.15

# Our methods

## Model baseline (plain)



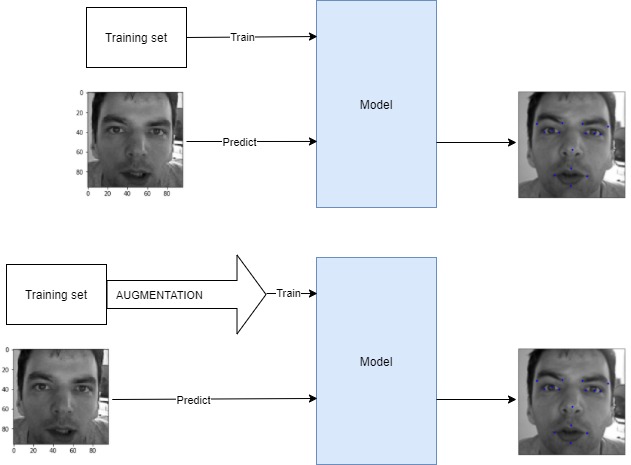
This model is the base model without preprocessing.

We are using it as baseline to make a comparison with our preprocessing.

(Image of loss on epoch) (image of MSE)

### 

## Model with Augmentation (*augmentation\_model*)



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.

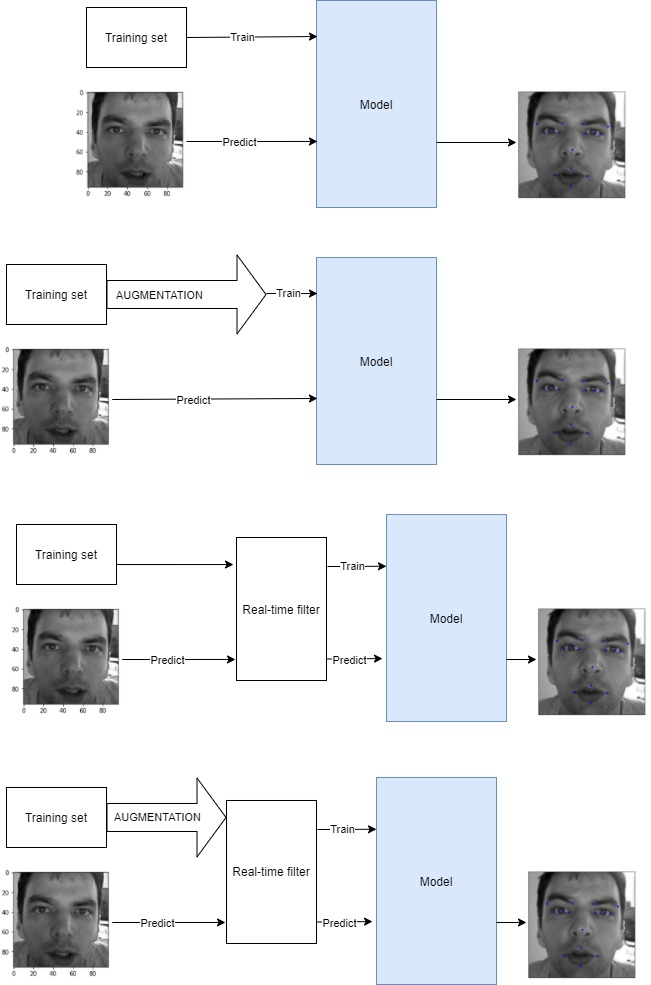
We are adding other modification the image as:

* Rotation
* Change in brightness
* Zoom in
* Random erasing zones

(Image of example of augmentation)

(Image of loss on epoch) (image of MSE)

## Model with real-time filter (*real\_model*)



This model does not use augmentation, but we will apply a filter just before the prediction (and training).

The execution time is a critical problem: we want use this on real-time ploblem.

The time of applying must be less than:

1 second / 30 frames per second = 30ms

Considering that the model takes 15 ms to predict one image we have:

30ms – 15ms = 15 ms

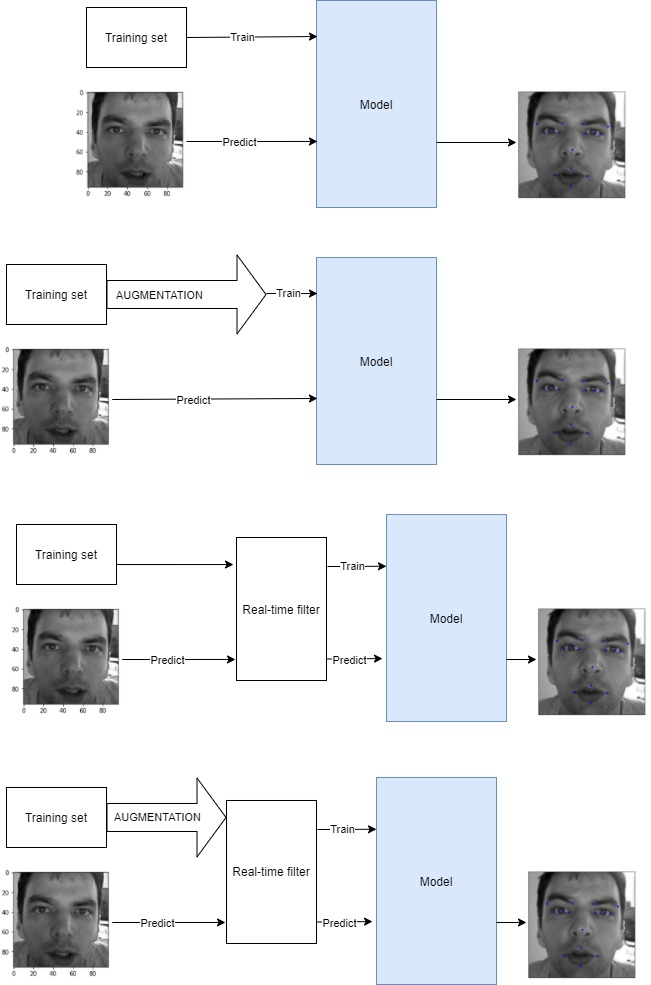
The filter must take less than 15ms

(evaluation ot time execution of real-time filter)

(example of real time filter)

(Image of loss on epoch) (image of MSE)

## Model with augmentation and real-time filter (*real\_aug\_model*)



This model uses real-time filter and augmentation too.

The idea is to bring together the advantages of augmentation and real-time filter

(Image of loss on epoch) (image of MSE)

# Performance evaluation

In this section we’ll discuss the performance of the model with different preprocessing above.

## (loss function image 1) (loss function image 3) (loss function image 3)

## (MSE image 1) (MSE image 2) (MSE image 3)

## (time of prediction image 1) (time of prediction image 2) (time of prediction image 3)

## Conclusions

Given these results, we can conclude that:

* **It is possible applying some real-time filter that increase the performance.**
* **The augmentation gets better generalization of the model.**

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

|  |  |  |
| --- | --- | --- |
| **model** | **MSE** | **RMSE** |
| *baseline\_model* |  |  |
| *Augmentation\_model* |  |  |
| *Real\_model* |  |  |
| *Real\_aug\_model* |  |  |

# References

1. Connor Shorten et al., 2019,   
   A survey on Image Data Augmentation for Deep Learning,  
   <https://www.researchgate.net/publication/334279066_A_survey_on_Image_Data_Augmentation_for_Deep_Learning>
2. Naimish et al., 2017,   
   Facial Key Points Detection using Deep Convolutional Neural Network - NaimishNet,  
   <https://arxiv.org/pdf/1710.00977.pdf>
3. Rucha Waghulde, Luis Oliveros Colón and Siddharth Mandgi, 2019,

Facial-Keypoints-Detection,  
<https://github.com/ruchawaghulde/Facial-Keypoints-Detection>

# Resources

* Our Google Colab’s notebook used to train the models
* Our Google Colab’s notebook for demonstration
* Our GitHub repo, home of all the code used