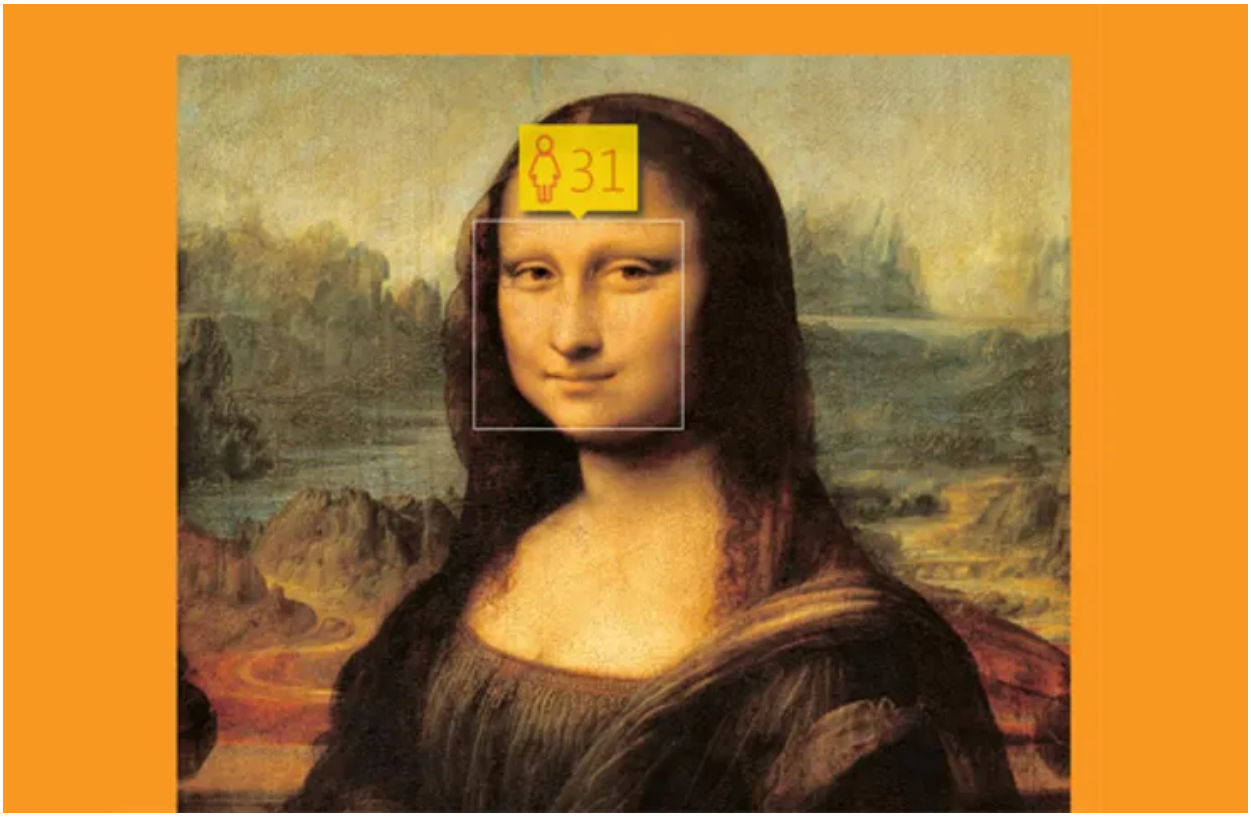
CAP 6655 – Project 2

Age detection model

Matteo Bassani



Introduction

Age detection is the process of automatically discerning the age of a person solely from a photo of their face. Typically, age detection is implemented as a two-stage process:

1. Detect faces in the input image/video stream.
2. Extract the face Region of Interest (ROI), and apply the age detector algorithm to predict the age of the person.

Basically, age estimation from the face is still a challenging problem, and guessing an exact age from a single image is very difficult due to factors like makeup, lighting, obstructions, and facial expressions.

Description

In this project, my goal is to build an age classification algorithm using images of people's faces (taken from public datasets found online). In particular, the algorithm should surround each face with a bounding box and label it with the age (or a range of ages) predicted. Also, it should work both for statical images and videos.

Faces are detected using the *face detection model of the DNN module of OpenCV*. In the first part of the project, we demonstrated that it works better than Haar cascade and is significantly less resource consuming than MTCNN.

Once obtained the bounding box coordinates of the faces, they are extracted ignoring the rest of the image/frame. Doing so allows the age detector to focus solely on the person's face and not any other irrelevant "noise" in the image. The face ROI is then passed through the model, yielding the actual age prediction.

There are several age detector algorithms, but the most efficient ones are deep learning-based age detectors. However, if we treat age detection as a regression problem, it is significantly harder for a model to accurately predict a single value representing that person's image. But if we treat it as a classification problem, defining buckets/age brackets for the model, our age predictor model becomes easier to train, often yielding substantially higher accuracy than regression-based prediction alone.

In particular in this project, we're going to compare the performance of a pre-trained model to the one of a model build and trained from scratch.

Models

- **Pretrained model**

Initially we will use a pre-trained Caffe model for age detection. If the performance is acceptable, we will test this model trying to analyze its pros and cons. Otherwise, we will try to train a new model from scratch, hoping to get better result than the pretrained one.

The pretrained model is pretty easy to use and it needs only two files:

- age_net.caffemodel: it is the pre-trained model weights for age detection. It can be downloaded from here.
- deploy_age.prototxt: it is the model architecture for the age detection model (a plain text file with a JSON-like structure containing all the neural network layer's definitions). It can be downloaded from here.

The model was implemented and trained by Levi and Hassner in their 2015 publication, Age and Gender Classification Using Convolutional Neural Networks. In the paper, the authors propose a simplistic AlexNet-like architecture that learns a total of eight age brackets: 0-2 / 4-6 / 8-12 / 15-20 / 25-32 / 38-43 / 48-53 / 60-100.

These age brackets are noncontiguous — this done on purpose, as the Adience dataset, used to train the model, defines the age ranges as such.

The network contains three convolutional layers, each followed by a rectified linear operation and pooling layer. The first two layers also follow normalization using local response normalization. The first Convolutional Layer contains 96 filters of 7×7 pixels, the second Convolutional Layer contains 256 filters of 5×5 pixels, the third and final Convolutional Layer contains 384 filters of 3 × 3 pixels. Finally, two fully-connected layers are added, each containing 512 neurons.

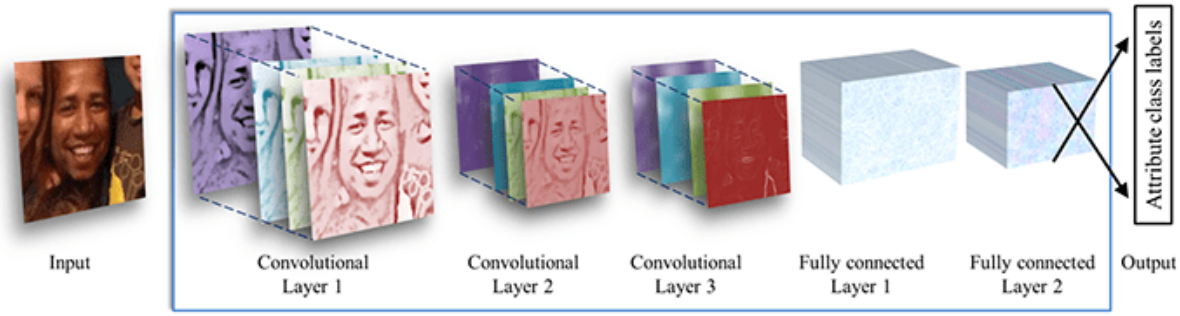
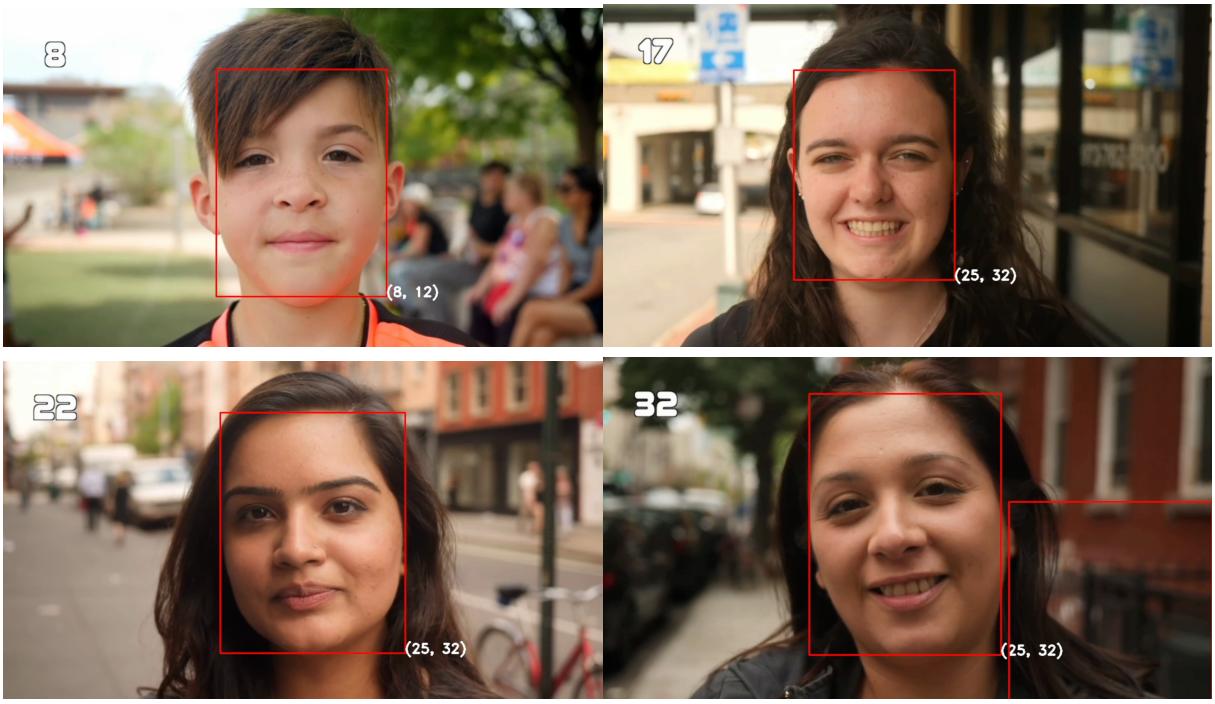


Figure 1: Architecture of this simple model.

To test this model we will adopt a cut version of this YouTube public video: https://www.youtube.com/watch?v=86qBAryeFco (*0 - 100 years in the USA*). All credits go to *ImagineVideoclips* who realized a video with the faces of 100 different person, aging from 1 to 100. We will use just some faces sampled from this video.

Figure 2: Here are some faces taken from the video, with their real age in the left high corner.

Here are some results of the pretrained model:



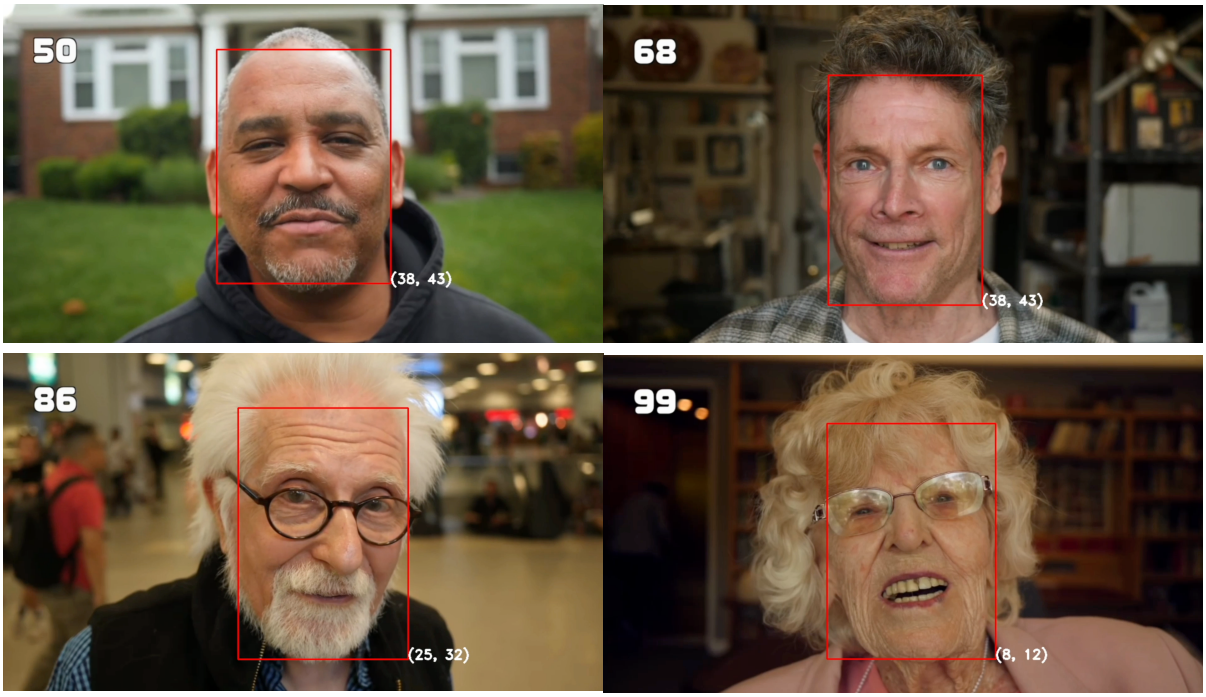


Figure 3: Some results of the pretrained model.

As we can see, the performance of the pretrained model is acceptable when dealing with young people. However, growing with age, the model tends to misclassify more and more often, reaching some outliers like the 99 years old woman who is classified as a 8-12 child.

Since this performance weren't enough good, we tried to build and train our own model.

- **New model**

1. *Dataset*:

This model is trained on a combination of *facial-age* dataset and the *UTKFace* dataset. Both datasets provide images that have already been cleaned and labelled properly.

facial-age dataset contains 9,778 RGB images of faces in PNG format of size 200x200 pixels each. This is the distribution of the dataset:

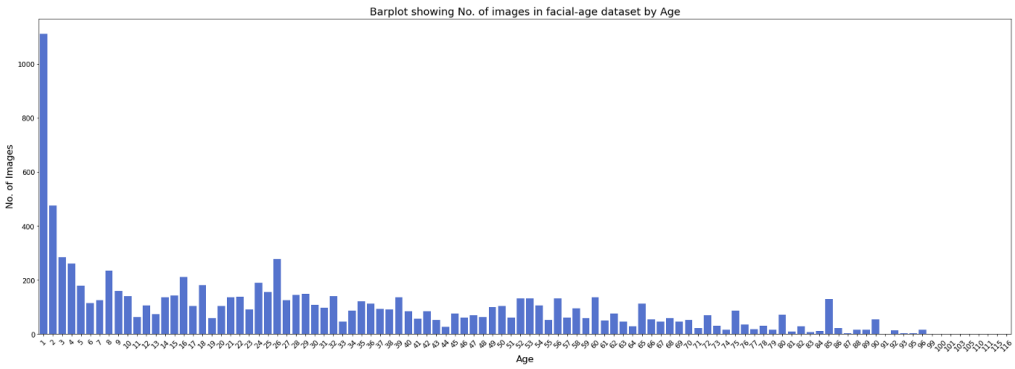


Figure 4: Distribution of facial-age dataset.

UTKFace dataset contains 23,708 RGB images of faces in JPG format of size 200x200 pixels each. This is the distribution of the dataset:

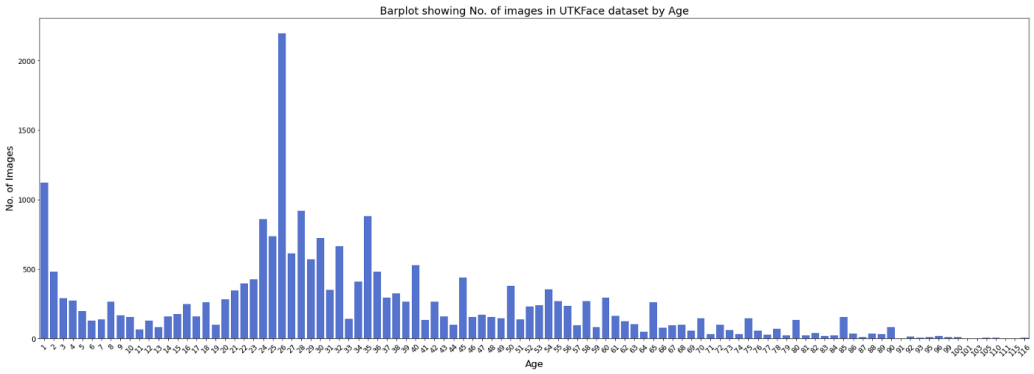


Figure 5: Distribution of UTKFace dataset

Then the two datasets are merged creating a bigger 33,486 images dataset of faces labeled with their age. The distribution of images after combining the two datasets is shown below.

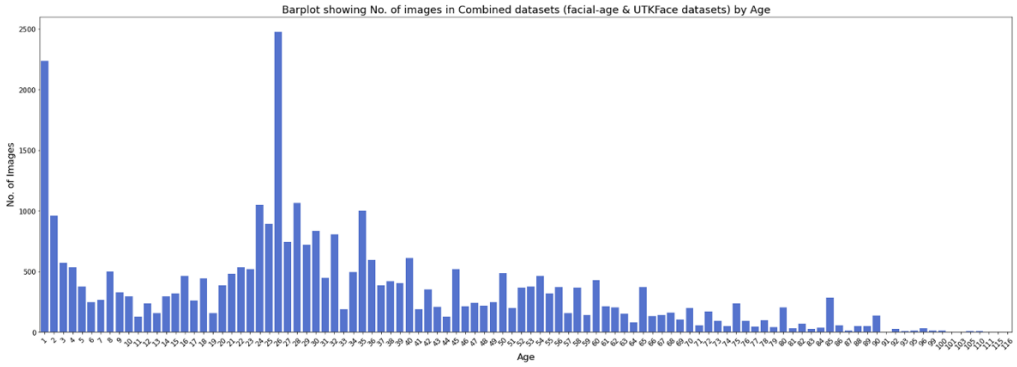


Figure 6: Distribution of the combined dataset.

Credits for this merging process goes to the author of this article which created a good dataset to train an age detection model: https://towardsdatascience.com/age-detection-using-facial-images-traditional-machine-learning-vs-deep-learning-2437b2feeab2. Also, age ranges are decided by the author in a way to get a good class balancing:

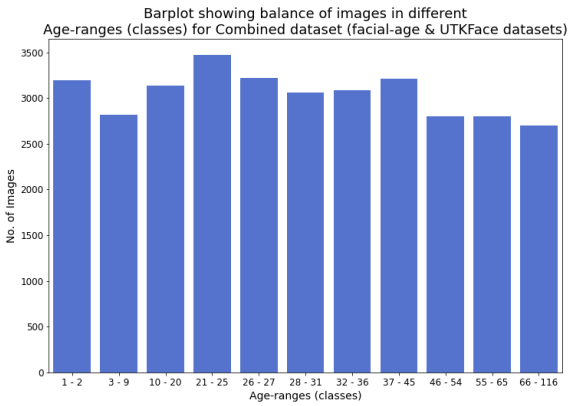


Figure 7: Age ranges of the combined dataset.

The dataset is split in the following way:

- 70% training set
- 30% test/validation set

We'll use the test set to validate the performance of the method after each epoch and then save the weights if the model improved its performance.

2. *Loss function and Metrics*:

Since we're dealing with a multi-class classification problem, we're using *categorical cross-entropy* as loss function. The dataset is quite balanced, so *accuracy* itself is enough as a metric.

3. Architecture:

The network contains four convolutional layers, each followed by a rectified linear operation and an average pooling layer. The first Convolutional Layer contains 16 filters of 3×3 pixels, the second Convolutional Layer contains 32 filters of 3×3 pixels, the third and the fourth layers both contains 64 filters of 3×3 pixels. Global average pooling is then performed. Finally, two fully connected layers are added, the first one containing 66 neurons and the last (output layer) only 7 (the total number of classes).

The number of epochs is set to 30, and the batch size to 64 (this will help the algorithm converging faster). The optimizer by default is SGD.

4. Results:

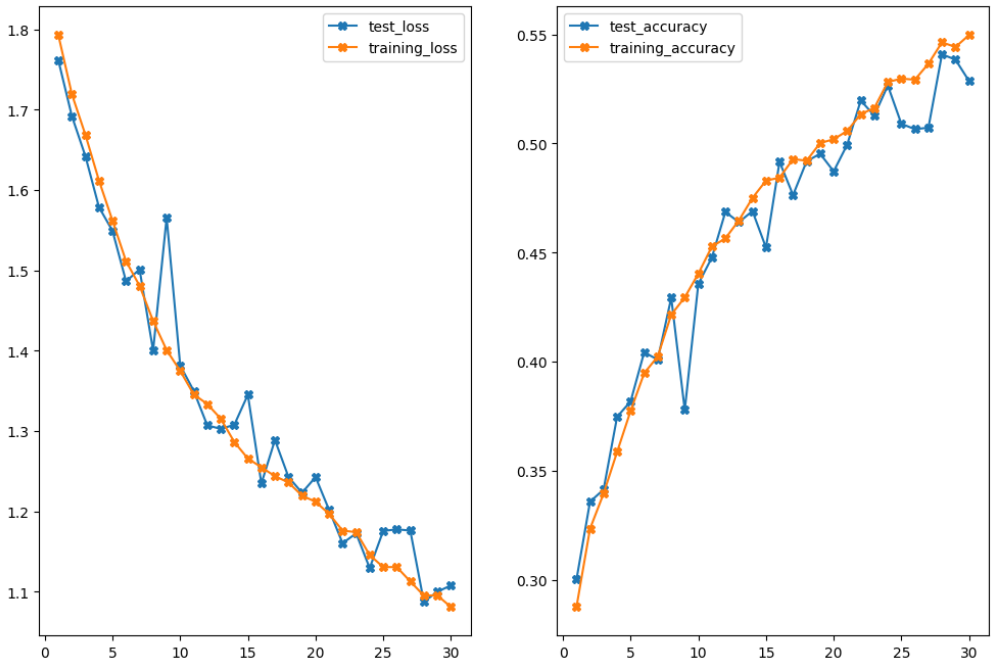


Figure 8: Loss and accuracy for training and test.

As we can see, the network obtained 0.54091 as validation accuracy after 30 epochs. The score is not bad, but we can try improving this result by training the network for more epochs. In fact the validation accuracy was still raising. We can see the results in our video used previously for the pretrained model:

Figure 9: Some results of the trained model.

These results are better than the pretrained model in some photos, but worse in others. Also there are some terrible errors in the prediction.

We can try with a new bigger model since this last model only has 64983 trainable weights. The architecture and the training are modified like this:

5. *Improvements*:

The new network contains five convolutional layers, each followed by a rectified linear operation and an average pooling layer. The first Convolutional Layer contains 32 filters of 3×3 pixels, the second Convolutional Layer contains 64 filters of 3×3 pixels, the third contains 128 filters of 3×3 pixels and the fourth layer contains 256 filters of 3×3 pixels. Global average pooling is then

performed. Finally, two fully connected layers are added, the first one containing 132 (double than before) neurons and the last (output layer) only 7.

The number of epochs is set to 60, and the batch size to 256 (our network is now bigger, and epochs are doubled, so we can adopt a bigger batch size). The optimizer by default is SGD.

Real-time data augmentation is also performed. This allow the model to be trained with a bigger dataset and getting better performance. New examples are generated from the existing ones with some manipulations such as horizontal flip, zoom, horizontal and vertical shift and small rotations.

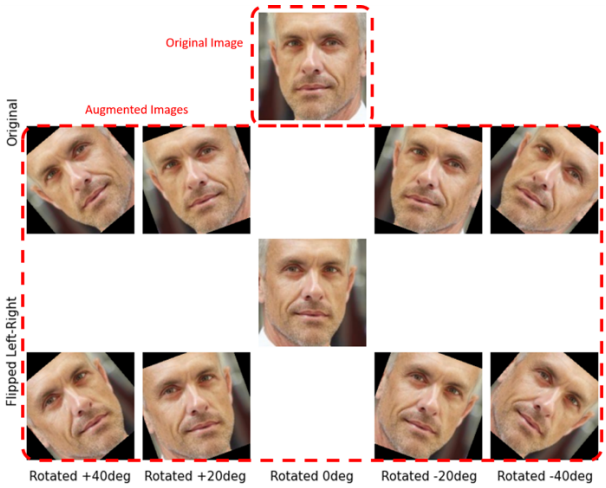


Figure 10: Image data augmentation: an example.

This network contains 422,695 trainable weights. But let's see the results:

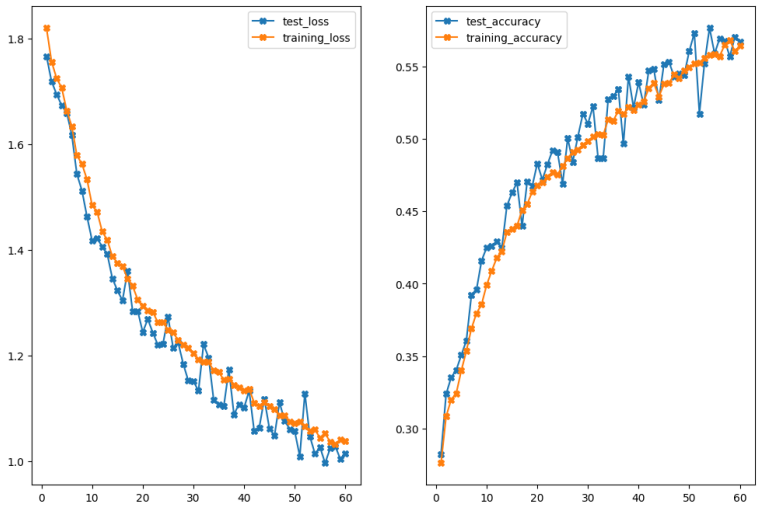


Figure 11: Figure 8: Loss and accuracy for training and test for the bigger model.

Now the curves (especially validation ones) are starting to converge. Also, this represents the limit for my hardware (MacBook Pro M1 Pro) but maybe with a desktop and a nice GPU many improvements are possible. Out best test accuracy now is 0.57655 which is not a bad result.

We can see this improvement also testing the model on our video:

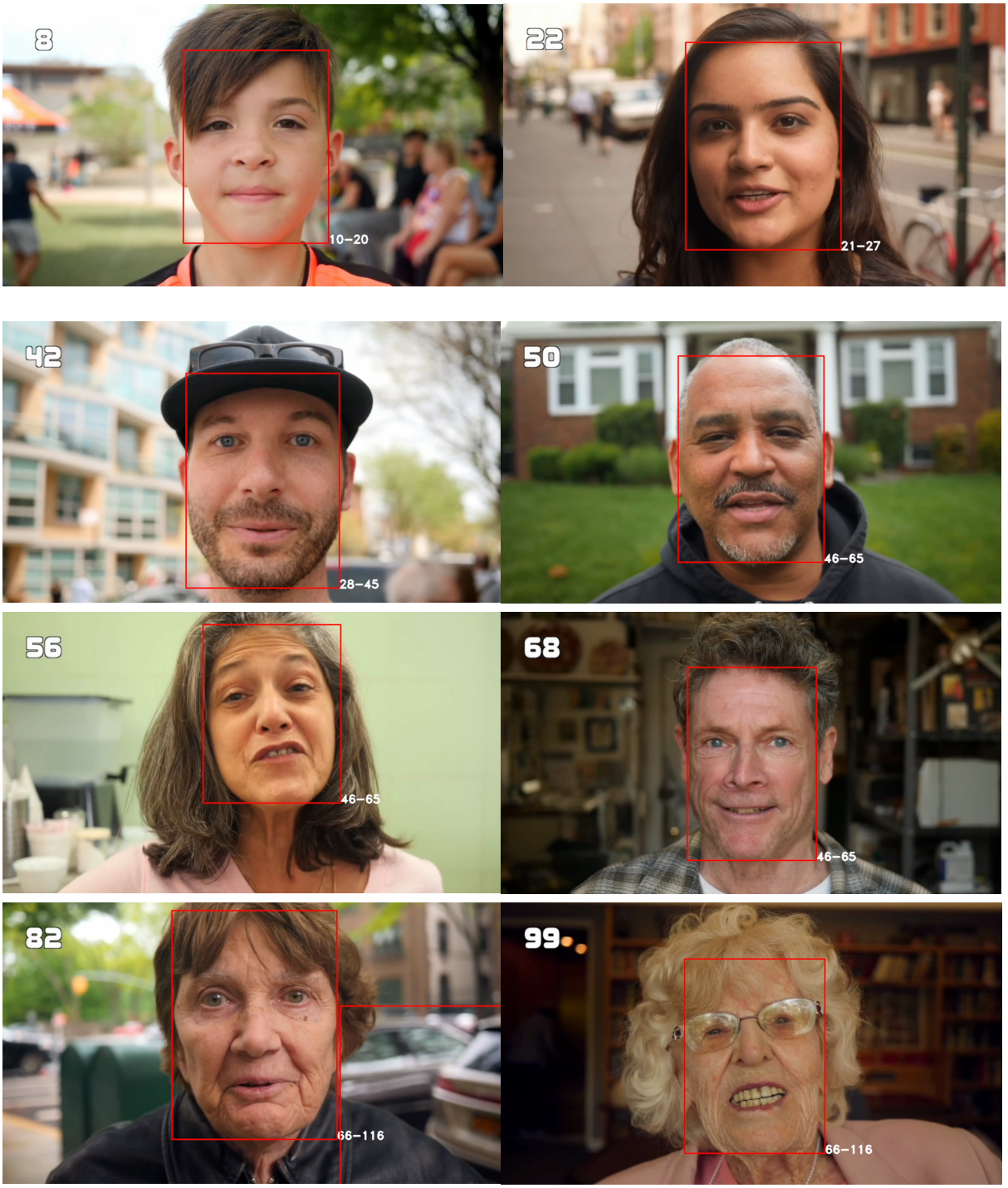


Figure 12: Some results of the bigger model.

Now the performance of the model seems improved a lot, there are still some minor errors, but we can be satisfied by this detection.

Analysis

Age prediction is a very hard task. The problem is that it is inherently subjective and based solely on appearance. In fact, appearance could be influenced by many factors: genetics, make-up, diet etc. It is a hard task also for humans, so teaching a machine (which is better in simple, detailed tasks) to estimate the age of a person is very challenging. The performance is still far by being perfect, but we can be satisfied by our empirical tests.

Before coming with the idea of a deep neural network, we performed some test with traditional machine learning algorithms (Random forest, SVM) using Canny edge detector as feature extractors. This algorithm consists in:

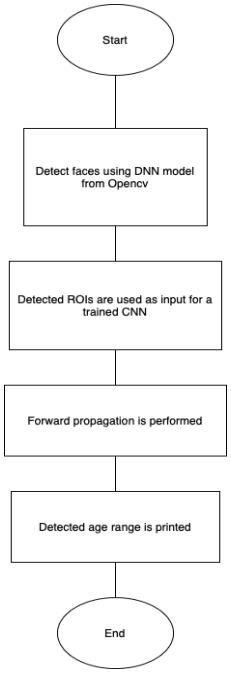
- Applying Gaussian filter to smooth the image to remove the noise
- Finding the intensity gradients of the image
- Applying gradient magnitude thresholding or lower bound cut-off suppression to get rid of spurious response to edge detection
- Applying double threshold to determine potential edges
- Tracking edge by hysteresis and suppressing all the weak edges.

A Random Forest model was trained on the features extracted using Canny on the facial-age dataset. However, the testing performance were very bad:

```
Training accuracy = 0.881 Testing accuracy = 0.36.
```

For this reason, our focus is then shifted from traditional machine learning to deep learning, which demonstrates to achieve better results in difficult task, such as age detection.

Flow Chart and code



To access the code of this project:

https://github.com/Matteo299/Age-Detection-model.git

- age_det_cnn_small.ipynb: first CNN trained
- age_det_cnn.ipynb: final CNN model
- video_age_detector.py: CNN model to detect ages in video
- video_age_detector_pretrained.py: pretrained model to detect ages in video
- webcam_age_detector.ipynb: CNN model to detect ages using webcam
- webcam_age_detector_pretrained.ipynb: pretrained model to detect ages using webcam