FINAL CHALLENGE: FACE RECOGNITION SYSTEM

Núria Codina Serra (U186655) & Maria Guasch Torres (U188325)

Universitat Pompeu Fabra - March 2024

1. INTRODUCTION

The aim of the *Face and Gesture Analysis Final Challenge* is to develop an automated face recognition system capable of identifying specified users within a given image dataset. This project entails the creation of a robust algorithm that can accurately determine the presence of pre-defined individuals from a test set comprising 1200 facial images.

At its core, the automatic system is tasked with two primary objectives: first, to detect and localize faces within an input image, and second, to match these faces against a training database containing identities of interest. Upon successful recognition, the system should return the corresponding identity of the detected individual. However, if no matches are found within the training database or if no faces are detected in the image, the system should return a designated indicator, such as "-1."

2. TRAINING AND TEST DATASETS

To meet the project's requirements, we were initially provided with a training dataset containing 1200 images, each accompanied by bounding box information and identity labels indicating either user identities (numbered 1 to 80) or impostors (labeled as -1). Additionally, a separate Test Dataset comprising 1200 images, sourced similarly but distinct from the training data, was prepared for evaluation purposes. While the training dataset was provided to us, it's worth mentioning that the test dataset was not directly provided; instead, it will be utilized by teachers for evaluation purposes.

Recognizing the challenge posed by the limited training data for training a deep learning model effectively, we took a strategy to expand the dataset's size and diversity. Initially, we manually obtained the names of 80 users by doing Google reverse image search. Subsequently, we developed a Python script named *'load_more_images.py'* aimed to scrape image thumbnails from three search engines—Google, Bing, and Yahoo—based on predefined search terms. This script iterates through the list of celebrity names, fetching image thumbnails from each search engine and organizing them into respective folders. The script leverages the requests library for handling HTTP requests, BeautifulSoup for parsing HTML content, and os for managing file operations. Furthermore, it utilizes a predefined list of celebrity names stored in the 'celebrities' module to facilitate targeted searches and image retrieval. Then, a manual revision of the obtained images was performed, deleting images which did not correspond to the identity.

This augmentation process significantly enriched the training dataset, enhancing its diversity and scalability for training the deep learning model effectively. So this new dataset was used to train and fine-tune our model's performance. And the initially provided training dataset, was used as our testing for the F1-score.

3. OUR DETECTION & RECOGNITION SYSTEM

In this section, we will explain the code and models used in our recognition system. To enhance clarity, we have divided the explanation into four blocks.

3.1. Detection Block

The first task was to create a detection block in order to detect the faces from the input model images. This was done in order to assign “-1” directly to those images not containing any face, and send those who contain faces to the next block, the face recognition model.

For this task, the detection function developed for lab1 was used. *'my_face_detection'* function is designed to identify faces within grayscale images using pre-trained Haar cascade classifiers for facial features such as eyes, nose, and mouth. Initially, the function employs the face cascade classifier to detect potential face regions within the image. Subsequently, it examines each detected face region and utilizes additional cascade classifiers for eyes, nose, and mouth to further validate the presence of facial features. A scoring mechanism is employed to assess the quality of detected faces based on the number of identified features, with faces containing at least three features considered valid. The function then prioritizes faces with the highest number of detected features, selecting the top two candidates. Finally, it returns the coordinates of bounding boxes encompassing these selected valid faces.

3.2. Face Recognition Model

After detecting faces and cropping the images by the delimited face area returned by the detection function, the `*my_face_recognition_function*` is called for each cropped face image. Within this function, face recognition is performed using the provided model (`my_FRmodel`), producing predicted identities and confidence scores for each face. These results are stored in `*our_ids*` and `*confidence*` lists, respectively, for further analysis. Upon processing all faces, the maximum confidence among them is determined and compared to a predefined threshold. If this maximum confidence falls below the threshold, indicating low confidence in recognizing any of the faces, the detected face is labeled as an impostor (autom_id = -1). However, if the maximum confidence surpasses the threshold, suggesting sufficient confidence in recognizing at least one face, the face with the highest confidence is identified based on its predicted class index. This identity (`*autom_id*`) signifies the recognized individual.

my_FRmodel

In the process of establishing our model for training and testing, we used the set of scrapped images. Subsequently, we created a dictionary associating each celebrity's name with the unique identifier. Following this, we partitioned our augmented dataset into distinct training and testing subsets utilizing pre-existing Python functionalities for such purposes.

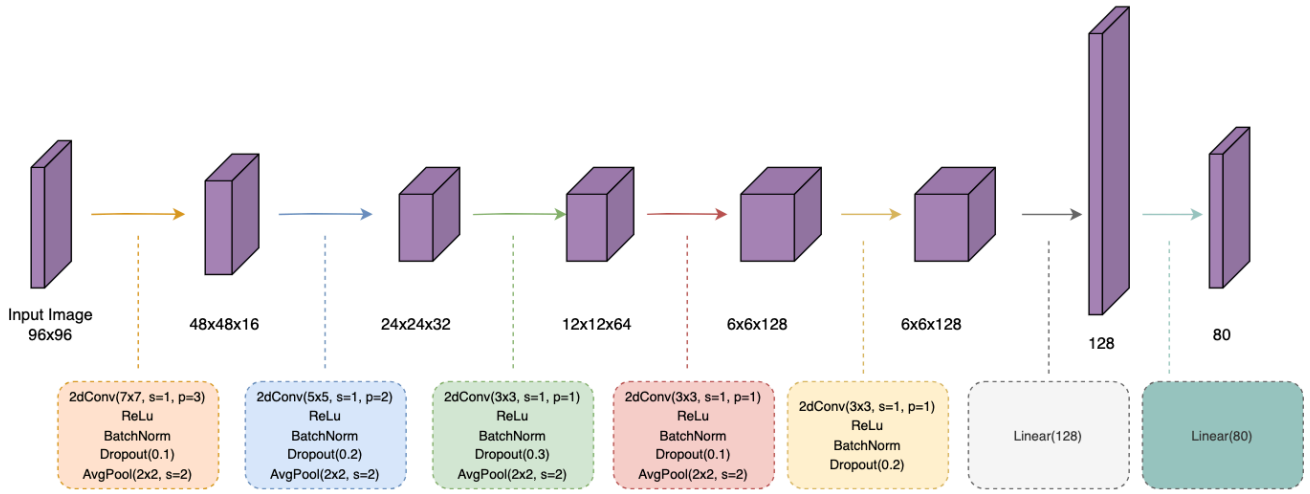
Subsequently, we performed preprocessing to the images to align them with the requirements of our model. This involved resizing them to a standardized resolution of 96x96 pixels, converting them to grayscale to simplify computational processing, and transforming them into tensors for compatibility with our neural network architecture. Additionally, a batch size of 250 was selected to balance computational efficiency with model optimization.

Finishing the preprocessing phase, we defined the functions of training and testing for our model. Finally, we developed our model: the *Convolutional Neural Network (CNN),* referred to as `*my_FRmodel*`.

The CNN architecture is structured with different layers, each serving specific purposes. Starting with *convolutional layers*, they aim to extract complex patterns from input images, using various settings to detect detailed spatial features in facial pictures (for instance, different kernel sizes, to capture both high level and lower level features). *Batch normalization layers* help stabilize the learning process, while *ReLU activations* add flexibility to the model. *Dropout layers* are strategically placed to prevent the model from becoming too specialized and improve its ability to recognize new faces. We have also added *average pooling layers*, to downsample the images and add robustness to variation.

After the convolutional layers, the extracted features are flattened and passed through *fully connected layers* responsible for identifying facial characteristics. This is where the heart of facial recognition lies, as these densely connected layers analyze the features to identify specific facial identities. Together, the

combination of convolutional and fully connected layers creates a robust system for facial recognition, allowing it to accurately distinguish between subtle facial features.



[1]

3.3. Gridsearch for optimal threshold

As the model does not account for the label -1 (representing impostors), a mechanism was developed to determine a threshold. This threshold serves to designate faces where the confidence (probability) for classification falls below a certain value, assigning the label -1 accordingly. To discover the optimal threshold for optimal performance, a *grid search* function was developed. This function systematically evaluates the recognition system across a range of thresholds, from 0 to 0.8 in increments of 0.025. Subsequently, it computes the detection and recognition outcomes, along with the corresponding F1-score for each threshold. The threshold producing the highest score is then selected as the optimal threshold. [2]

3.4. Plots and results

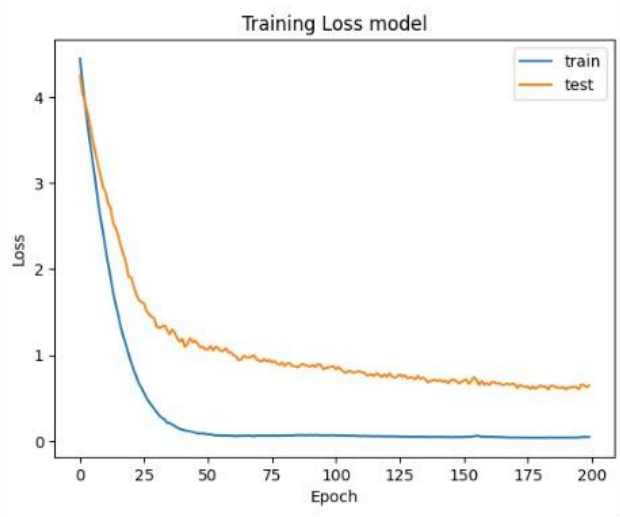
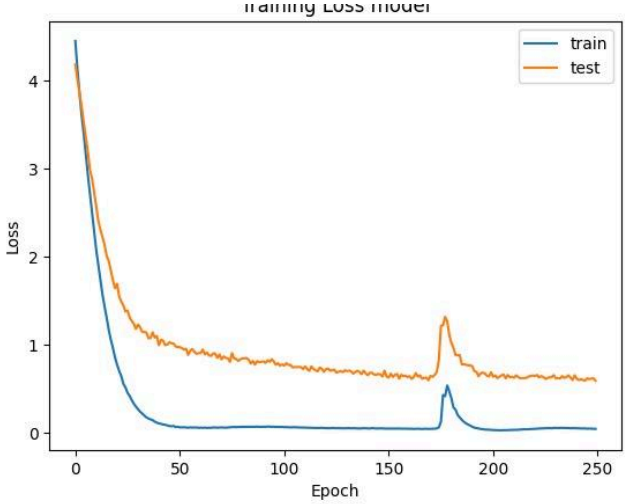
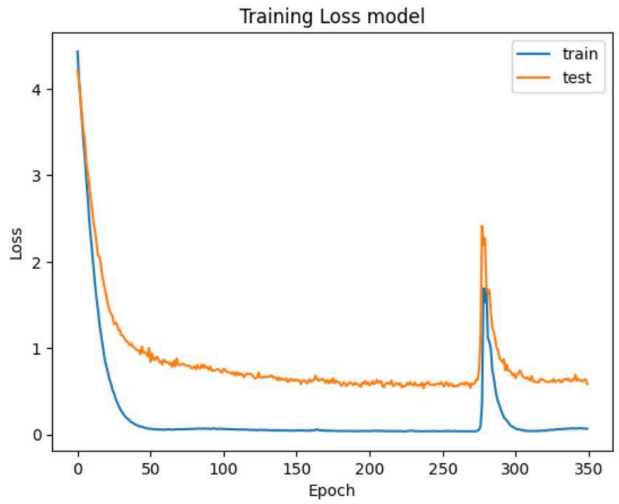
After training the code with different hyperparameters (number of epochs, batch sizes, etc.) and after finding the best threshold, the final model for our recognition system was the CNN explained above, trained with 350 epochs and 250 batch size, along with a 0.325 confidence threshold.

The requirements of the project were to obtain a F1-score above 0.2, with an average speed of less than 3 seconds per image, maximum size of 80MB, maximum number of 10 parameterizable layers and less than 1 million parameters.

For this final model, the number of parameters is 854576, the number of trainable layers is 7, processing time and maximum size are achieved, and the F1-score for the parameters defined above is 70.02.

We can also observe the losses of the model for different numbers of epochs in the table below:

[1] Complete structure and definition of CNN in annex 8.2.
[2] Results for the gridsearch function in annex 8.1

200 epochs	250 epochs
	
350 epochs	These three plots display the train and validation losses for our model. Even though there is a peak in the losses both for the 350 and 250 epoch models, we decided to keep the 350 one. This sudden increase in the loss is most likely due to some specific flawed image in our training data, but the tendency of the graph indicates that there is no overfitting.
	

4. OTHER APPROACHES

Prior to defining the final model, various alternative approaches were explored, producing inferior results. This section provides an overview of the preceding methodologies leading up to the selection of the ultimate model.

Lab4 Delivered Model[3]

The model we designed for the previous delivery of this subject was a very simple and reduced CNN model. It used 3 convolutional layers (of kernel size 3) and 2 fully connected linear layers. Of course, it also included ReLu as the activation function and average pooling layers.

This model allowed us to reach a F1 score of around 0.24. However, in the final evaluation with the professor's test set, it achieved only a 0.17 F1 score, which does not fulfill the requirements of the final project.

[3] Architecture and loss plot in annex 9.3

It is also important to note, that with this model, we did not set a confidence threshold, because the probabilities that the model gave as output were too similar between impostors and subjects with an id to define a cutoff value. We will dive further into this later in this report.

Pointwise Convolutions[4]

Another approach was to develop a CNN using pointwise convolutions in each block. One of the main benefits of using this type of convolutions is that they help maintain a controlled number of parameters. That is, they reduce the number of channels in the feature maps, serving as a form of dimensionality reduction, while still retaining important features.

Unfortunately, when we checked the loss plots for this model, we discovered that it was not correctly designed, as the training loss was very inestable, and the model did not seem to learn as the epochs went by.

No threshold

As we mentioned above, for the face recognition system delivered in lab 4, we did not define a threshold. This meant that all images that were passed through the CNN (images that contain a face according to our face detector) were assigned an id.

The main reason was that with our initial model, the confidence with which the CNN assigned a label to a face was generally low (possibly due to underfitting and also a model with low performance). We inspected by hand the confidence for each image, and there was not a clear probability cutoff value that divided true and false positives. Therefore, we decided to use no threshold.

However, once we improved our CNN, this problem vanished, as the model was assigning labels to faces included in the training data with very high confidence (on average), and with low confidence to unseen faces.

To pick the threshold which gave us the highest F1 score, we performed grid search, as explained above.

Augmented illumination dataset[5]

We also tried to improve the accuracy of the model by doing data augmentation. In particular, we added 3 new images for each original one but with different illumination. This way, the model could be trained with different instances of the same face.

However, once we tested different models trained with this dataset, we obtained worse results, so we ended up discarding this approach.

3 channel dataset

Another approach was to create a model that took coloured images (3 channels) instead of black and white. For the previous delivery, we opted for a model that took as input 1 channel images (black and white), to make things as simple as possible. In order to improve the accuracy, one of the options was to upgrade the network so that it could take 3 channel images as input.

However, after testing different architectures we were not able to achieve good results without trespassing the limit of 1M model parameters. So this approach was rejected as well.

[4] Model architecture and loss plot in annex 8.4
[5] Examples and implementation in annex 8.5

5. RESULTS AND INTERPRETATION

Following our commitment to the final model, we would like to provide additional insights into the outcomes. Beyond the F1 score, which stands at approximately 0.7, we have delved into more granular metrics such as the confusion matrix, detailing parameters like true positives and false negatives.

Applying a threshold of 0.325, the corresponding values are outlined in the table provided in annex 8.6. Notably, a substantial portion of the dataset is accurately classified as either an impostor or a non-face image (indicated by the label "-1"). This success can be attributed to two key factors: firstly, our face detection system adeptly identifies faces in images, and secondly, our carefully chosen threshold effectively designates the impostor label to images with low confidence.

It is imperative to acknowledge that our face detector, while proficient, is not flawless (with an accuracy of approximately 80%), contributing to some false negatives.

6. CONCLUSIONS

In conclusion, the Face and Gesture Analysis Final Challenge has been a comprehensive exploration of face recognition systems, incorporating various elements of deep learning, concretely convolutional neural networks. The project began with the reuse of a face detection system developed in earlier stages, followed by significant improvements through the incorporation of batch normalization layers and adjustments to convolutional layer sizes.

The process of model development involved the exploration of different architectures, hyperparameters, and augmentation techniques. Our final model, a Convolutional Neural Network (CNN), demonstrated robust face recognition capabilities, achieving an F1-score of around 0.7 while meeting the project's speed, size, and parameter constraints.

Training a deep learning model is a dynamic process with numerous hyperparameters to fine-tune and various architectural choices to consider. The journey presented in this report underscores the importance of experimentation, learning from failures, and the continual pursuit of optimization. As technology advances, future work in this domain will likely involve the integration of cutting-edge techniques and strategies to further enhance the accuracy and efficiency of face recognition systems.

7. FUTURE WORK

Fine tuning of hyper parameters

Our final model was tested with different combinations of hyper parameters such as the batch size. However, further experimentation with learning rates and even different optimizers could be done to try to improve the performance. We could perform grid search with them to find the best value.

Architecture modifications

Our current network is based on a very simple model. More experimentation could be done taking inspiration from other pre-existing models in the field of face recognition. We also have the limitations of parameters and number of layers, so we might not be able to implement as many layers as we would like.

Handling imbalanced data

As we have mentioned above, we created our own training dataset by scrapping online images of the subjects in the provided dataset. However, we had to manually remove images due to bad quality. This resulted in a somewhat imbalanced dataset, where some ids had less images than others. With more time, we could construct a better training dataset.

8. ANNEX

8.1. Gridsearch output

FOR 350 EPOCHS

```
***** COMPUTING SCORE FOR THRESHOLD: 0.0 *****
****** Threshold 0.0 calculated*******
F1-score: 67.44, Total time:  6 m 5.27 s
***** COMPUTING SCORE FOR THRESHOLD: 0.025 *****
****** Threshold 0.025 calculated*******
F1-score: 67.44, Total time:  5 m 36.91 s
***** COMPUTING SCORE FOR THRESHOLD: 0.05 *****
****** Threshold 0.05 calculated*******
F1-score: 67.44, Total time:  5 m 27.77 s
***** COMPUTING SCORE FOR THRESHOLD: 0.075 *****
****** Threshold 0.075 calculated*******
F1-score: 67.44, Total time:  5 m 29.84 s
***** COMPUTING SCORE FOR THRESHOLD: 0.1 *****
****** Threshold 0.1 calculated*******
F1-score: 67.44, Total time:  5 m 30.42 s
***** COMPUTING SCORE FOR THRESHOLD: 0.125 *****
****** Threshold 0.125 calculated*******
F1-score: 67.45, Total time:  5 m 27.99 s
***** COMPUTING SCORE FOR THRESHOLD: 0.15 *****
****** Threshold 0.15 calculated*******
F1-score: 67.59, Total time:  5 m 26.31 s
***** COMPUTING SCORE FOR THRESHOLD: 0.175 *****
****** Threshold 0.175 calculated*******
F1-score: 67.88, Total time:  5 m 32.02 s
***** COMPUTING SCORE FOR THRESHOLD: 0.2 *****
****** Threshold 0.2 calculated*******
F1-score: 68.48, Total time:  5 m 33.57 s
***** COMPUTING SCORE FOR THRESHOLD: 0.225 *****
****** Threshold 0.225 calculated*******
F1-score: 68.50, Total time:  5 m 27.47 s
***** COMPUTING SCORE FOR THRESHOLD: 0.25 *****
****** Threshold 0.25 calculated*******
F1-score: 68.74, Total time:  5 m 25.97 s
***** COMPUTING SCORE FOR THRESHOLD: 0.275 *****
****** Threshold 0.275 calculated*******
F1-score: 69.67, Total time:  5 m 23.07 s
***** COMPUTING SCORE FOR THRESHOLD: 0.3 *****
****** Threshold 0.3 calculated*******
F1-score: 69.94, Total time:  5 m 28.03 s
***** COMPUTING SCORE FOR THRESHOLD: 0.325 *****
****** Threshold 0.325 calculated*******
F1-score: 69.99, Total time:  5 m 28.91 s
***** COMPUTING SCORE FOR THRESHOLD: 0.35 *****
****** Threshold 0.35 calculated*******
F1-score: 69.40, Total time:  5 m 26.70 s
***** COMPUTING SCORE FOR THRESHOLD: 0.375 *****
****** Threshold 0.375 calculated*******
F1-score: 68.97, Total time:  5 m 24.01 s
***** COMPUTING SCORE FOR THRESHOLD: 0.4 *****
****** Threshold 0.4 calculated*******
F1-score: 69.26, Total time:  5 m 29.91 s
***** COMPUTING SCORE FOR THRESHOLD: 0.425 *****
****** Threshold 0.425 calculated*******
F1-score: 68.31, Total time:  5 m 29.30 s
***** COMPUTING SCORE FOR THRESHOLD: 0.45 *****
****** Threshold 0.45 calculated*******
F1-score: 67.25, Total time:  5 m 26.32 s
```

```
***** COMPUTING SCORE FOR THRESHOLD: 0.475 *****
****** Threshold 0.475 calculated*******
F1-score: 66.92, Total time:  5 m 32.18 s
***** COMPUTING SCORE FOR THRESHOLD: 0.5 *****
****** Threshold 0.5 calculated*******
F1-score: 66.92, Total time:  5 m 32.47 s
***** COMPUTING SCORE FOR THRESHOLD: 0.525 *****
****** Threshold 0.525 calculated*******
F1-score: 67.36, Total time:  5 m 25.42 s
***** COMPUTING SCORE FOR THRESHOLD: 0.55 *****
****** Threshold 0.55 calculated*******
F1-score: 66.23, Total time:  5 m 25.72 s
***** COMPUTING SCORE FOR THRESHOLD: 0.575 *****
****** Threshold 0.575 calculated*******
F1-score: 65.51, Total time:  5 m 24.81 s
***** COMPUTING SCORE FOR THRESHOLD: 0.6 *****
****** Threshold 0.6 calculated*******
F1-score: 64.31, Total time:  5 m 28.29 s
***** COMPUTING SCORE FOR THRESHOLD: 0.625 *****
****** Threshold 0.625 calculated*******
F1-score: 63.47, Total time:  5 m 25.58 s
***** COMPUTING SCORE FOR THRESHOLD: 0.65 *****
****** Threshold 0.65 calculated*******
F1-score: 62.31, Total time:  5 m 29.14 s
OPTIMAL THRESHOLD IS: 0.325
```

8.2. CNN Definition

Python

```python
class batchnorm_ReducedIdEstimationModel(nn.Module):
  def __init__(self, num_classes):
    super(batchnorm_ReducedIdEstimationModel, self).__init__()


    self.features = nn.Sequential(
      nn.Conv2d(1, 16, kernel_size=7, padding=3),
      nn.ReLU(inplace=True),
      nn.BatchNorm2d(16),
      nn.Dropout2d(0.1),
      nn.AvgPool2d(kernel_size=2, stride=2),


      nn.Conv2d(16, 32, kernel_size=5, padding=2),
      nn.ReLU(inplace=True),
      nn.BatchNorm2d(32),
      nn.Dropout2d(0.2),
      nn.AvgPool2d(kernel_size=2, stride=2),


      nn.Conv2d(32, 64, kernel_size=3, padding=1),
      nn.ReLU(inplace=True),
      nn.BatchNorm2d(64),
      nn.Dropout2d(0.3),
      nn.AvgPool2d(kernel_size=2, stride=2),
```

```
      nn.Conv2d(64, 128, kernel_size=3, padding=1),
      nn.ReLU(inplace=True),
      nn.BatchNorm2d(128),
      nn.Dropout2d(0.1),
      nn.AvgPool2d(kernel_size=2, stride=2),


      nn.Conv2d(128, 128, kernel_size=3, padding=1),
      nn.ReLU(inplace=True),
      nn.BatchNorm2d(128),
      nn.Dropout2d(0.2),
    )


    self.classifier = nn.Sequential(
      nn.Linear(128 * 6 * 6, 128),
      nn.Linear(128, num_classes)
    )


  def forward(self, x):
    x = self.features(x)
    x = x.view(x.size(0), -1)
    x = self.classifier(x)
    return x
```

8.3. Reduced CNN for Lab 4

Python

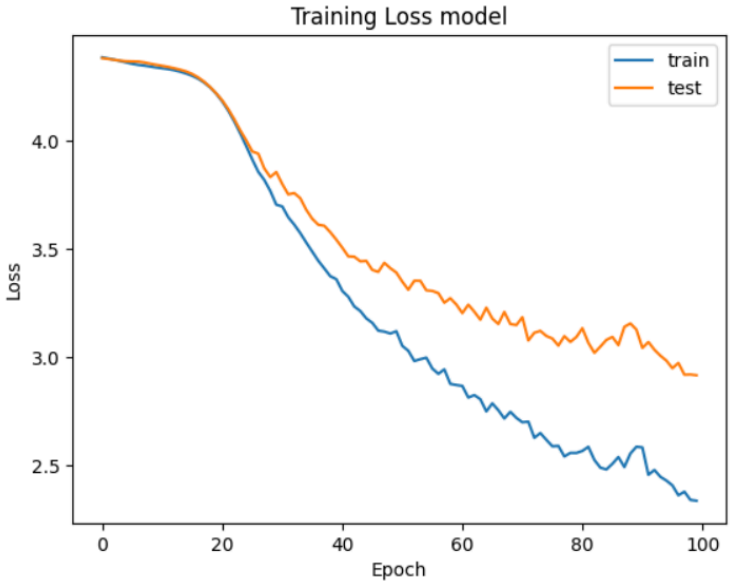
```
class ReducedIdEstimationModel(nn.Module):
  def __init__(self, num_classes):
    super(ReducedIdEstimationModel, self).__init__()


    self.features = nn.Sequential(
      nn.Conv2d(1, 16, kernel_size=3, padding=1),
      nn.ReLU(inplace=True),
      nn.AvgPool2d(kernel_size=2, stride=2),
      nn.Conv2d(16, 32, kernel_size=3, padding=1),
      nn.ReLU(inplace=True),
      nn.AvgPool2d(kernel_size=2, stride=2),
      nn.Conv2d(32, 64, kernel_size=3, padding=1),
      nn.ReLU(inplace=True),
      nn.AvgPool2d(kernel_size=2, stride=2),
      nn.AvgPool2d(kernel_size=2, stride=2)
    )


    self.classifier = nn.Sequential(
      nn.Linear(64 * 9 * 9, 128),
      nn.ReLU(inplace=True),
```

```
      nn.Dropout(p=0.3),
      nn.Linear(128, num_classes)
    )


  def forward(self, x):
    x = self.features(x)
    x = x.view(x.size(0), -1)
    x = self.classifier(x)
    return x
```



8.4. Point-wise convolution model

Python

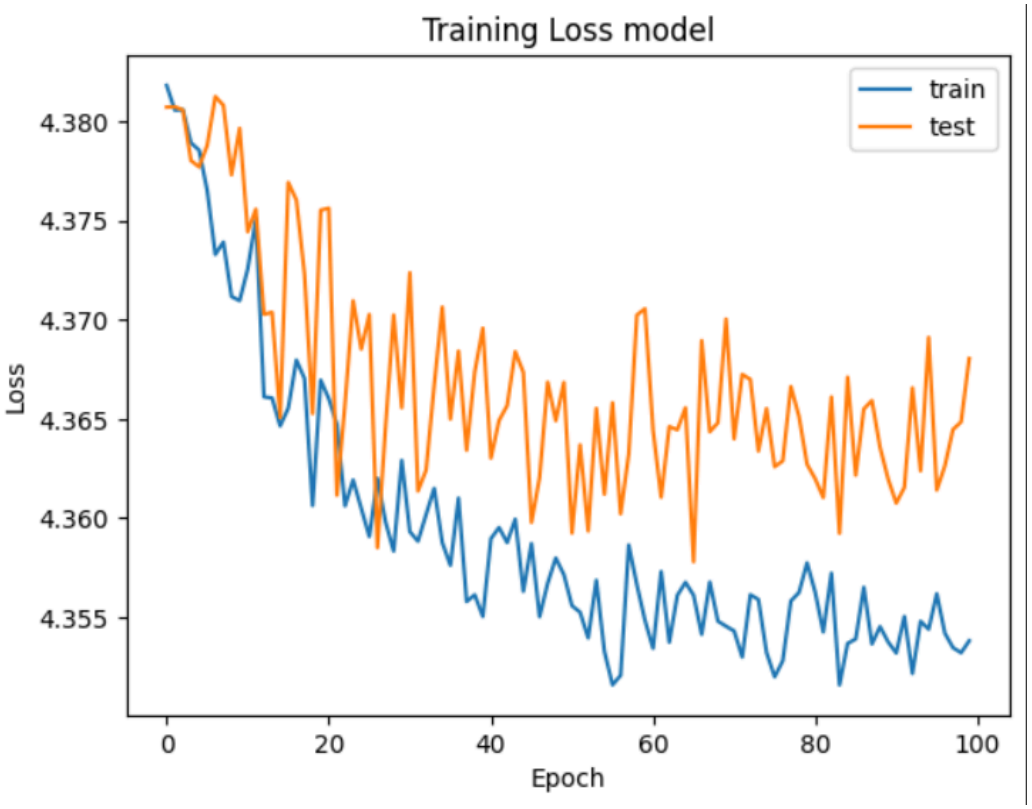
```
class PontWiseIdEstimationModel(nn.Module):
  def __init__(self, num_classes):
    super(PontWiseIdEstimationModel, self).__init__()

    self.features = nn.Sequential(
      nn.Conv2d(3, 16, kernel_size=1), # Point-wise convolution
      nn.ReLU(inplace=True),
      nn.Conv2d(16, 16, kernel_size=3, padding=1), # Normal convolution
      nn.ReLU(inplace=True),
      nn.AvgPool2d(kernel_size=2, stride=2),
      nn.Conv2d(16, 32, kernel_size=1), # Point-wise convolution
      nn.ReLU(inplace=True),
      nn.Conv2d(32, 32, kernel_size=3, padding=1), # Normal convolution
      nn.ReLU(inplace=True),
      nn.AvgPool2d(kernel_size=2, stride=2),
      nn.Conv2d(32, 64, kernel_size=1), # Point-wise convolution
      nn.ReLU(inplace=True),
      nn.Conv2d(64, 64, kernel_size=3, padding=1), # Normal convolution
```

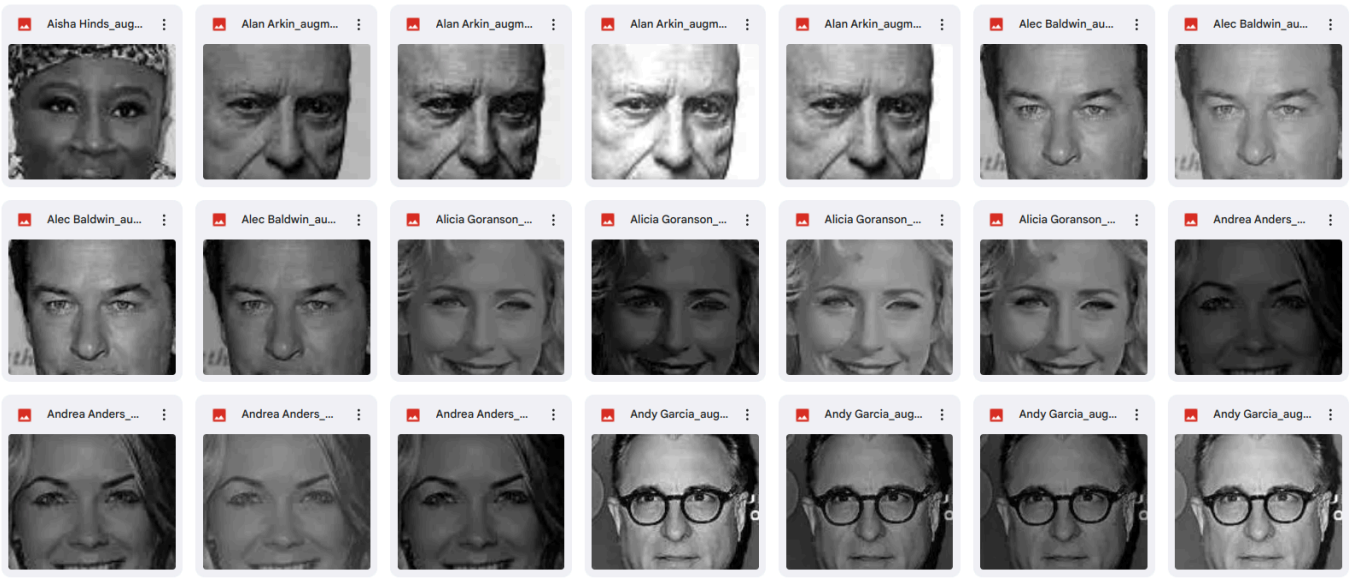
```
      nn.ReLU(inplace=True),
      nn.AvgPool2d(kernel_size=2, stride=2),
      nn.Conv2d(64, 128, kernel_size=1), # Point-wise convolution
      nn.ReLU(inplace=True),
      nn.Conv2d(128, 128, kernel_size=3, padding=1), # Normal convolution
      nn.ReLU(inplace=True),
      nn.AvgPool2d(kernel_size=2, stride=2)
    )

    self.classifier = nn.Sequential(
      nn.Linear(128 * 6 * 6, 128),
      nn.ReLU(inplace=True),
      nn.Dropout(p=0.5),
      nn.Linear(128, num_classes)
    )

  def forward(self, x):
    x = self.features(x)
    x = x.view(x.size(0), -1)
    x = self.classifier(x)
    return x
```



8.5. Data augmentation: illumination. Examples and implementation.



Python

```python
# Function to apply random illumination changes to an image
def random_illumination(image):
  # Randomly adjust brightness and contrast
  alpha = np.random.uniform(0.7, 1.3) # Random contrast factor
  beta = np.random.randint(-50, 50)  # Random brightness adjustment

  # Apply brightness and contrast adjustment
  augmented_image = cv2.convertScaleAbs(image, alpha=alpha, beta=beta)

  return augmented_image

input_path = '/content/drive/MyDrive/Analisi Gestos i Cares/model/Data/cropped_images'
files = os.listdir(input_path)

image_paths = []
labels = []

for file in files:
  if file.endswith('.jpg') or file.endswith('.jpeg') or file.endswith('.png'):
    image_path = os.path.join(input_path, file)
    image_paths.append(image_path)

    parts = file.split("_")
    name_index = None

    for i, part in enumerate(parts):
      if part.startswith("cropped"):
        continue
      else:
        name_index = i
        break

    name = parts[name_index]
```

```
    labels.append(celebrities_dict[name])
    original_image = cv2.imread(image_path)
    for i in range(3): # Augment with 3 additional images
      augmented_image = random_illumination(original_image)
      augmented_image_path = os.path.join(input_path, f"{name}_augmented_{i}.jpg")
      cv2.imwrite(augmented_image_path, augmented_image)
      image_paths.append(augmented_image_path)
      labels.append(celebrities_dict[name])

# Create a DataFrame with the image paths and labels
df = pd.DataFrame({'image_path': image_paths, 'id': labels})
df = df.astype({'id': int})

df.to_pickle('/content/drive/MyDrive/Analisi Gestos i Cares/model/Data/images_df.pkl')
print('Dataframe correctly created.')
```

8.6. Confusion Matrix

True Positives	False Positives
299	140
False Negatives	True Negatives
116	645