Project

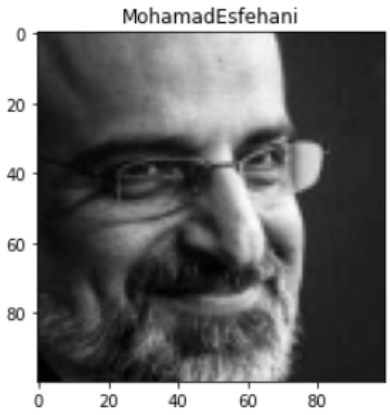
In this project, our primary goal is to classify images of famous individuals based on their facial features. To focus on the essential information for our classification task, we started by extracting and cropping faces from the images. Faces carry distinctive characteristics that are crucial for identifying and differentiating individuals.

Following the extraction process, we organized our dataset by reading images from each celebrity folder. To facilitate the learning process, we labeled each image with the corresponding celebrity folder name, establishing a clear association between the images and their respective identities.

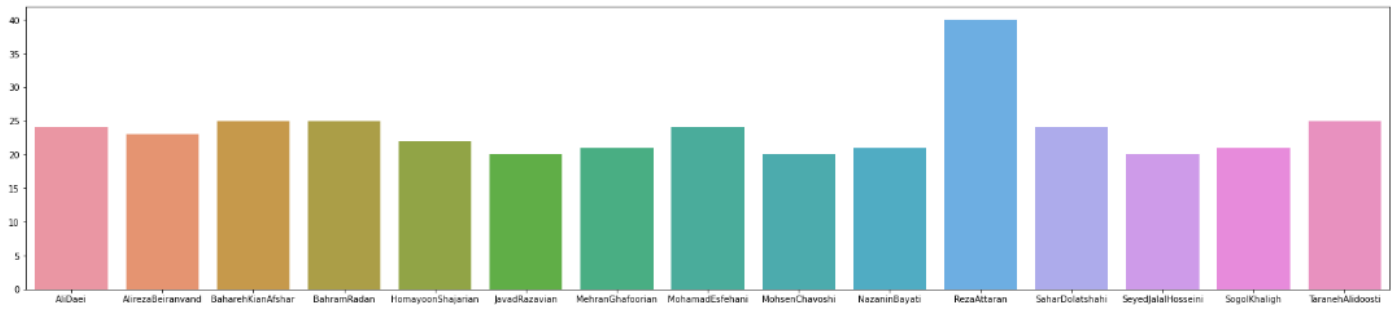
As part of our preprocessing steps, we decided to convert each image to grayscale. This decision was motivated by the understanding that, in our specific task, the color information might not be as informative as the facial features themselves. By working with grayscale images, we reduce the computational complexity associated with handling three RGB channels, while still capturing the essential facial characteristics needed for effective classification. This streamlined approach not only optimizes computational resources but also allows our model to focus on the distinctive patterns present in facial structures for accurate celebrity classification. We also capture the width and height of each image. These dimensions are appended to separate lists.

	height	width
count	355.000000	355.000000
mean	210.822535	210.715493
std	134.246967	134.145741
min	36.000000	36.000000
25%	107.000000	108.000000
50%	186.000000	186.000000
75%	268.000000	268.000000
max	958.000000	958.000000

With this information in hand, we move on to a statistical analysis of the collected widths and heights. This analysis allows us to identify key metrics such as the minimum, maximum, and mean values. Then we make a strategic decision to resize every image in our dataset to a uniform size of 100x100 pixels. This resizing operation ensures consistency in the input dimensions for further processing steps.



To identify any potential biases, we construct a bar plot that illustrates the count of images for each celebrity.we strategically partition our dataset into three distinct sets: a training set comprising 75% of the data, a test set with 15%, and a validation set with 10%. However, a crucial consideration comes into play due to the limited number of images available for each celebrity class. To decrease the risk of imbalances between classes within each set, we do a stratified split.

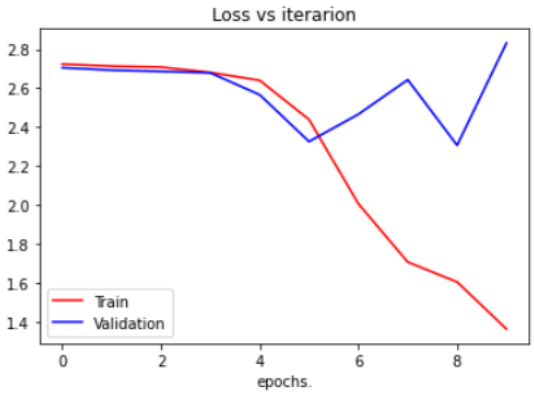


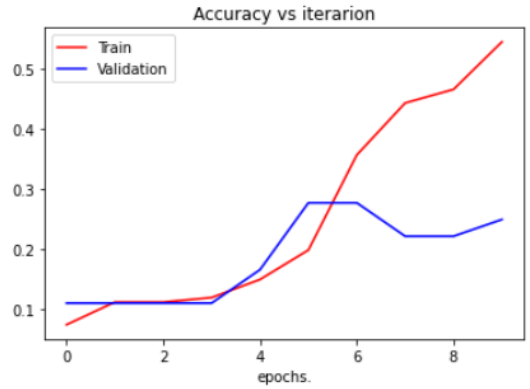
Afterward, we proceeded to build our initial version of a Convolutional Neural Network (CNN) to conduct an exploratory test. The design of our CNN is deliberately kept simple, following an architecture outlined as follows:

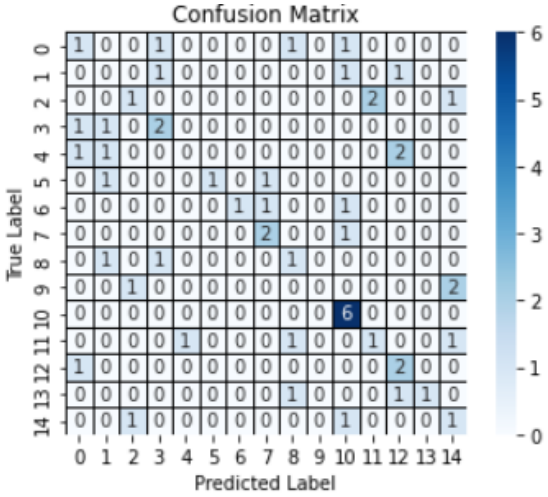
Layer (type)	Output Shape	Param #
conv2d_114 (Conv2D)	(None, 98, 98, 128)	1280
max_pooling2d_82 (MaxPooling2D)	(None, 49, 49, 128)	0
conv2d_115 (Conv2D)	(None, 47, 47, 64)	73792
max_pooling2d_83 (MaxPooling2D)	(None, 23, 23, 64)	0
conv2d_116 (Conv2D)	(None, 21, 21, 32)	18464
max_pooling2d_84 (MaxPooling2D)	(None, 10, 10, 32)	0
flatten_25 (Flatten)	(None, 3200)	0
dense_73 (Dense)	(None, 256)	819456
dense_74 (Dense)	(None, 15)	3855

This simplified CNN architecture serves as an introductory exploration into the capabilities of our model. By implementing this initial version, we aim to gain insights into the network's performance and its ability to extract meaningful features from the dataset.

	precision	recall	f1-score	support
0	0.00	0.00	0.00	4
1	0.00	0.00	0.00	3
2	0.00	0.00	0.00	4
3	0.67	0.50	0.57	4
4	0.33	0.75	0.46	4
5	0.40	0.67	0.50	3
6	0.25	0.33	0.29	3
7	0.25	1.00	0.40	3
8	0.33	0.33	0.33	3
9	0.00	0.00	0.00	3
10	1.00	0.17	0.29	6
11	0.00	0.00	0.00	4
12	0.40	0.67	0.50	3
13	0.00	0.00	0.00	3
14	0.00	0.00	0.00	3
accuracy			0.28	53
macro avg	0.24	0.29	0.22	53
weighted avg	0.28	0.28	0.22	53





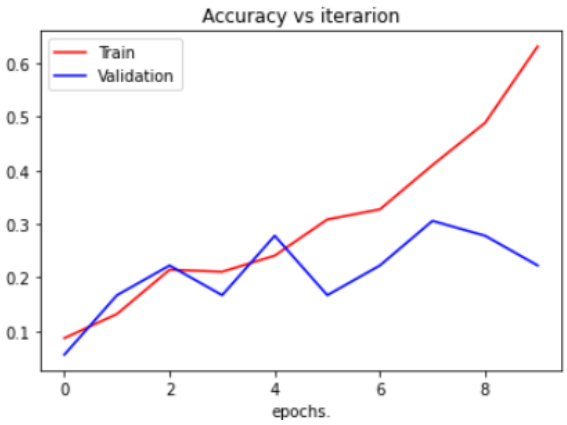


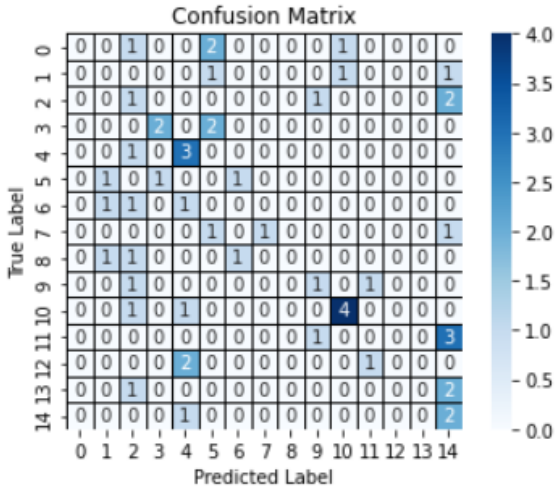
As evident from the results, the performance of the first version falls short of our expectations, exhibiting signs of potential overfitting. This suspicion is understood from the observed increase in validation loss. Additionally, the limited accuracy achieved on the training set further indicates that the model is not effectively capturing the underlying patterns within the data.

I subsequently simplified my CNN architecture, as shown below. Notably, I adjusted the kernel sizes in each convolutional layer to optimize feature extraction. In particular, the earlier layers now utilize larger kernel sizes, facilitating the capture of more generalized aspects such as edges, while the subsequent layers employ smaller kernel sizes to focus on extracting finer details.

Layer (type)	Output Shape	Param #
conv2d_136 (Conv2D)	(None, 98, 98, 32)	320
max_pooling2d_104 (MaxPooling2D)	(None, 49, 49, 32)	0
conv2d_137 (Conv2D)	(None, 47, 47, 32)	9248
max_pooling2d_105 (MaxPooling2D)	(None, 23, 23, 32)	0
flatten_35 (Flatten)	(None, 16928)	0
dense_93 (Dense)	(None, 32)	541728
dense_94 (Dense)	(None, 15)	495

	precision	recall	f1-score	support
0	0.00	0.00	0.00	4
1	0.00	0.00	0.00	3
2	0.12	0.25	0.17	4
3	0.67	0.50	0.57	4
4	0.38	0.75	0.50	4
5	0.00	0.00	0.00	3
6	0.00	0.00	0.00	3
7	1.00	0.33	0.50	3
8	0.00	0.00	0.00	3
9	0.33	0.33	0.33	3
10	0.67	0.67	0.67	6
11	0.00	0.00	0.00	4
12	0.00	0.00	0.00	3
13	0.00	0.00	0.00	3
14	0.18	0.67	0.29	3
accuracy			0.26	53
macro avg	0.22	0.23	0.20	53
weighted avg	0.25	0.26	0.23	53





Despite these modifications, the model's performance has not yet met the desired standards. Nevertheless, there is a positive trend emerging — a gradual decrease in both training and validation loss. This decreasing trend signifies that the model is beginning to adapt and learn more effectively from the provided data.

At this point, I faced a big problem in developing the model. Adding more training time or making the model more complicated didn't help. Trying those changes actually made things worse, like what happened in the first version with the increasing validation loss.p

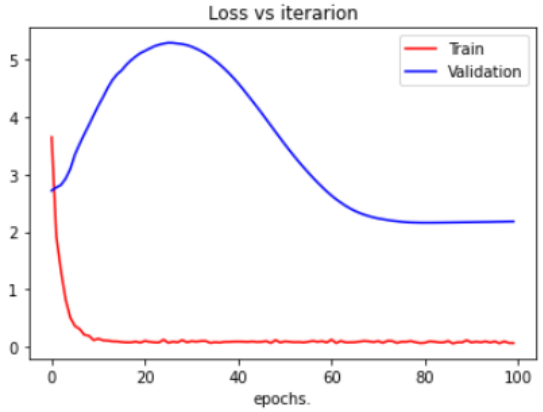
I proceeded to experiment with novel approaches, incorporating L1 normalization for convolutional layers and introducing batch normalization into the model architecture

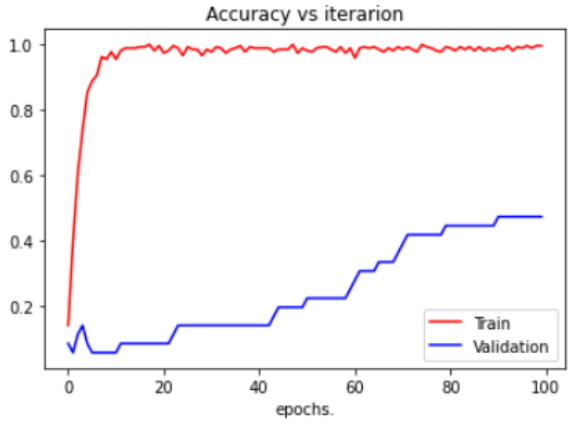
Both L2 regularization and Batch Normalization are techniques used in neural network models to improve training stability, generalization, and convergence.
L2 regularization encourages the model to use all features and avoids creating overly complex decision boundaries.
The regularization term discourages extreme weights, leading to a more generalized model that performs better on unseen data.
Batch Normalization (BatchNorm) is a technique to normalize the inputs of a layer by adjusting and scaling the activations.
Dropout is a regularization technique commonly used in neural networks to prevent overfitting. It involves randomly dropping out (setting to zero) a fraction of the input units (or neurons) at each update during training. This helps prevent complex co-adaptations on training data and forces the network to learn more robust features. I adopted a dynamic approach to learning rate management by setting the initial learning rate to 0.001 and implementing adjustments based on validation loss. This adaptive strategy entails reducing the learning rate if the validation loss shows an increasing trend, allowing the algorithm to take smaller steps for more precise convergence.

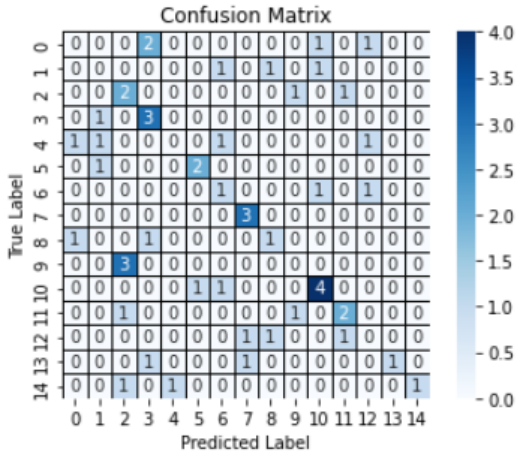
The final architecture, designed to harness the benefits of these learning rate adjustments and other enhancements, is outlined below:

Layer (type)	Output Shape	Param #
conv2d_138 (Conv2D)	(None, 96, 96, 128)	3328
max_pooling2d_106 (MaxPooling2D)	(None, 48, 48, 128)	0
batch_normalization_41 (BatchNormalization)	(None, 48, 48, 128)	512
conv2d_139 (Conv2D)	(None, 46, 46, 64)	73792
max_pooling2d_107 (MaxPooling2D)	(None, 23, 23, 64)	0
batch_normalization_42 (BatchNormalization)	(None, 23, 23, 64)	256
conv2d_140 (Conv2D)	(None, 21, 21, 32)	18464
max_pooling2d_108 (MaxPooling2D)	(None, 10, 10, 32)	0
batch_normalization_43 (BatchNormalization)	(None, 10, 10, 32)	128
flatten_36 (Flatten)	(None, 3200)	0
dense_95 (Dense)	(None, 256)	819456
dropout_16 (Dropout)	(None, 256)	0
dense_96 (Dense)	(None, 15)	3855

	precision	recall	f1-score	support
0	0.00	0.00	0.00	4
1	0.00	0.00	0.00	3
2	0.40	0.50	0.44	4
3	1.00	0.75	0.86	4
4	1.00	0.25	0.40	4
5	1.00	0.67	0.80	3
6	0.20	0.33	0.25	3
7	0.67	0.67	0.67	3
8	0.67	0.67	0.67	3
9	0.00	0.00	0.00	3
10	0.55	1.00	0.71	6
11	0.50	0.75	0.60	4
12	0.00	0.00	0.00	3
13	0.33	0.33	0.33	3
14	0.50	0.33	0.40	3
accuracy			0.45	53
macro avg	0.45	0.42	0.41	53
weighted avg	0.47	0.45	0.43	53





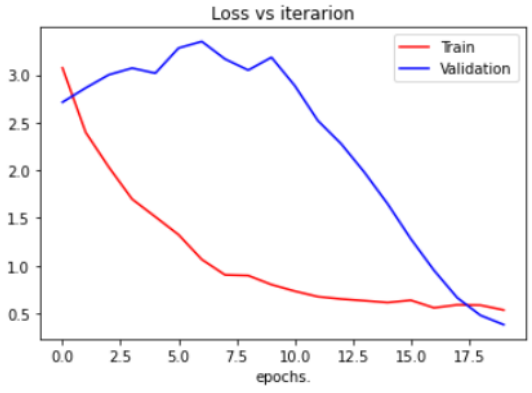


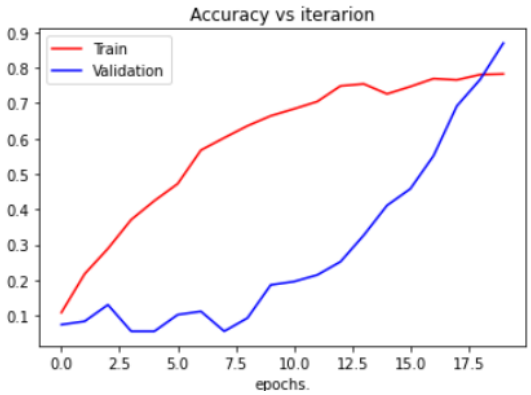
While recognizing that there is still room for improvement in the model's performance, I decided to explore the avenue of data augmentation. This involves expanding the dataset to three times its original size by applying the following configurations:

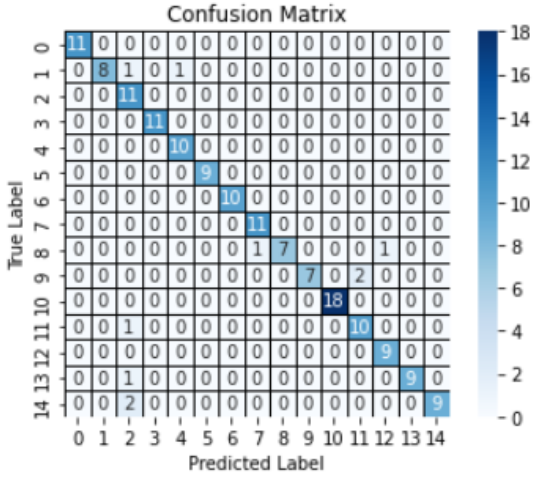
```
augmenter = ImageDataGenerator(
    rotation_range=30,
    horizontal_flip=True,
    vertical_flip=True,
    shear_range=0.2,
    zoom_range=0.2
   )
```

Now the mode is performing good as you can see bellow :

	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	1.00	0.80	0.89	10
2	0.69	1.00	0.81	11
3	1.00	1.00	1.00	11
4	0.91	1.00	0.95	10
5	1.00	1.00	1.00	9
6	1.00	1.00	1.00	10
7	0.92	1.00	0.96	11
8	1.00	0.78	0.88	9
9	1.00	0.78	0.88	9
10	1.00	1.00	1.00	18
11	0.83	0.91	0.87	11
12	0.90	1.00	0.95	9
13	1.00	0.90	0.95	10
14	1.00	0.82	0.90	11
accuracy			0.94	160
macro avg	0.95	0.93	0.94	160
weighted avg	0.95	0.94	0.94	160






Confusion Matrix
True Label
Predicted Label
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14
18 16 14 12 10 8 6 4 2 0