PAPARAZZI

Ashraful Firoz 2047398
Gaia Castelpietra 2053991

La Sapienza University of Rome, Department of Informatics
Applied Computer Science and Artificial Intelligence
https://github.com/AshrafAkon/paparazzi

ABSTRACT

Paparazzi is an actor's face recognition engine. The program is designed to analyze photos of actors, taken in real time from a mobile device. It aims to identify the actor and provide his full name along with the actor's ten most famous films.

TABLE OF CONTENTS

1. INTRODUCTION *2*
1.1 Objective struggle 2
1.2 New objective and aims 2
1.3 Project structure 2
2. DATA COLLECTION *3*
2.1 Dataset issues 3
2.2 Dataset collection 3
2.3 Dataset size 3
3. PREPROCESSING *3*
3.1 Face cropping 3
3.2 Face normalization 3
4. FACE RECOGNITION MODEL *4*
4.1 Key concepts 4
4.2 Architecture of ResNet50 4
4.3 Training 5
5. GUI *5*
5.1 Film API 5
5.2 Implementation 5
5.3 GUI design 5
5.4 GUI software 5
6. RESULTS AND CONCLUSIONS *6*
6.1 Training batch 6
6.2 Training epoch 6
6.3 Validation 7
6.4 Conclusions 7

7. FUTURE DEVELOPMENTS 7
8. REFERENCES 7

1. INTRODUCTION

Our software's goal is to answer the one question we all ask ourselves at least once while watching a movie or TV series: "Who's that actor?". It might have happened before that while watching a movie, you recognize a familiar face but can't quite remember where you saw them before. This face recognition engine aims to identify the actors and provide their most famous movies so that you can finally remember where you saw the actor before and not have to wonder for the next 20 minutes and get distracted from the current movie you are watching. It also serves the very intuitive function of discovering new actors that you might have never seen before and suggesting a couple of their best movies.

1.1 Objective struggle

At the beginning of our project, we thought of creating a totally different model. We wanted to build a music recommendation engine but soon gave up on the idea as the models required (mainly NLP models) were already fully developed and extremely efficient, and they were too big for us to rebuild or refine. Additionally, our first idea was heavily dependent on user input, thus leading to a lot of bias, and we figured out that it would impact negatively on the output, making it impossible to have reliable and precise outcomes.

1.2 New objective and aims

For the reasons described above, we decided to build a project based on face recognition. This technique correlates a face to a precise outcome, making it impossible for subjective aspects to come into play. No matter what picture the user provides, the result can either be right or wrong, giving us more tools to analyze the outcome and define a clear data graph.

1.3 Project structure

The project is divided into four parts:

- Scraping
- Face detection
- Face recognition
- Graphical User Interface (GUI)

The resources used for this project were Yunet for face detection, FacemarkLBF for landmark detection (i.e. detecting the eyes), Resnet50 for face recognition and Tkinter for the GUI.

2. DATA COLLECTION

2.1 Dataset issues

Most datasets were either dated, so to exclude the most recent actors, or included all celebrities in general, making it hard to work on our objective. Additionally, many datasets were built using Google scraping resulting on photos being completely unrelated or unusable, making it complicated to even reach 50% accuracy. The datasets would contain a lot of misleading photos for the model such as pictures of the wrong actor, pictures without face, morphed pictures, pictures with multiple faces and much more. To top this, the datasets were so big that our computing power was nowhere near being sufficient to analyse all the data.

2.2 Dataset collection

We were looking for a dataset that had more than 300 images for each class, didn't contain any duplicates, had photos from different angles (sideview, front view, top view and bottom view) and only contained images from the same person. Since the quality of the datasets scraped from Google was very low, we created our own dataset by scrapping the images from DuckDuckGo. [1]

2.3 Dataset size

We had to find the perfect ratio between database size and computing power. We needed around 200 photos for training and 30 for validation per class. A lower number of photos would have resulted in the validation accuracy to be lower than 50%. We found out that using 200 photos for training was a good compromise between good results and medium computing power.

3. PREPROCESSING

3.1 Face cropping

We were only interested in the facial features and ResNet50 expects input images to be 224x224 in size. We used OpenCV with pretrained YuNet model to find the coordinates of the face and crop it to the required size.

3.2 Face normalization

As the faces were not perfectly aligned horizontally, we needed to normalize all the faces by warping the image using both eyes as a horizontal landmark. We used pretrained FacemarkLBF [2] model to get the eye coordinates. Then use those coordinates in OpenCV to warp the image.

4. FACE RECOGNITION MODEL

ResNet50, short for Residual Network with 50 layers, is a deep convolutional neural network architecture that addresses the problem of training very deep networks. It was introduced by Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun in their 2015 paper "Deep Residual Learning for Image Recognition."

4.1 Key concepts

1. **Residual Learning**: The core idea behind ResNet is the use of residual learning. Instead of learning a direct mapping from input to output, the network learns the difference between the input and the output, making it easier to optimize very deep networks.
2. **Identity Shortcut Connections**: ResNet introduces shortcut connections that skip one or more layers. These shortcuts perform identity mapping, and their outputs are added to the outputs of the stacked layers. This helps in mitigating the vanishing gradient problem and enables training of very deep networks.

4.2 Architecture of ResNet50

ResNet50 consists of 50 layers, which can be broken down into several stages:

1. **Input Layer**: The ResNet50 architecture takes an input image, typically represented as a grid of pixels with three color channels (red, green, blue).
2. **Convolutional Layer**: The convolutional layers in ResNet50 are designed to extract and refine features from an input image through a series of convolution operations. First, the image passes through a 7x7 convolutional layer with 64 filters, detecting basic features like edges and textures, while reducing the image size. Then, a 3x3 max-pooling layer further downsamples the image, focusing on important features. The network includes several stages of residual blocks, each with three layers of convolutions (1x1, 3x3, 1x1 filters). These blocks capture features at different levels and use shortcut connections that add the input of each block to its output, helping to train deep networks effectively. This combination of layers helps the network learn detailed features, from simple shapes to complex objects, for accurate classification.
3. **Activation Function**: After each convolution operation, an activation function (ReLU) is applied, introducing nonlinearity to help the network learn complex feature relationships.
4. **Global Average Pooling**: A global average pooling layer reduces the spatial dimensions of the feature maps to 1x1, maintaining essential information while simplifying the data.
5. **Fully Connected Layer**: The fully connected (FC) layer in the ResNet50 model is replaced by a custom layer for generalizing our dataset. It consists of a sequence of linear and non-linear transformations designed to facilitate classification. Initially, a linear layer transforms the output from the previous layer (typically the global average pooling layer) into a 1024-dimensional vector. This is followed by a Rectified Linear Unit (ReLU) activation function, introducing non-linearity to help the network learn complex patterns. A dropout layer is then

applied, which randomly sets 50% of the elements to zero during training to prevent overfitting. This sequence of linear transformation, ReLU activation, and dropout is repeated once more, ensuring robustness and additional non-linearity. Finally, the transformed features are passed through another linear layer, which maps the 1024-dimensional vector to the number of classes. A softmax activation function is applied to this output to convert it into class probabilities, facilitating classification. This structured approach ensures that the model can effectively learn and generalize from the training data.

4.3 Training

For training, we wrote our own ImageDataset class based on PyTorch's Dataset class. We load all the images in memory at the beginning of training to reduce delays related to loading images from the disk. This reduced the training time around 10%. Each epoch we slightly modify the image so that the model can learn the complex features of a person's face rather than the composition of the image. We do this augmentation by slightly modifying the brightness, hue and sharpness of the image.

5. GUI

5.1 Film API

Our top choice was IMDB's API. The website is always up to date and its reputation guarantees us the best result when sorting the films and TV series for popularity. [3]

5.2 Implementation

We search for the actor's name, the output of the model, and we get its ID. With the ID, we can retrieve from "Movie Credits" all the movies and TV series the actor participated in. Successively, we sort the movies and TV series for popularity and select the first ten.

5.3 GUI design

The GUI is extremely simple and intuitive. We used Tkinter as it was the easiest and fastest way to achieve our design. A button makes you select the photo and, after some processing, the name and films appear under the uploaded photo.

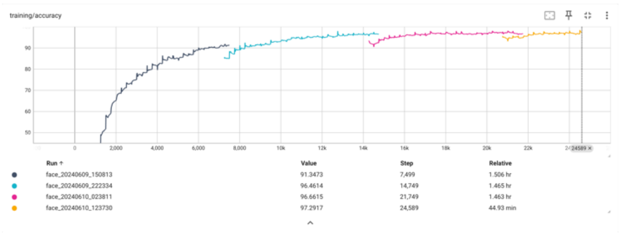
5.4 GUI software

To integrate the software with the GUI, we import the needed functions into the GUI class. Once we run the application, the selected photo will go through the same preprocessing as the training data, where the eyes get identified and the face is morphed and cropped to offer the best input for the model. The model then analyses the photo and outputs the predicted actor. From here, the code finishes the work by looking for the movies and TV series from the TMDB API, bringing it back to the GUI.

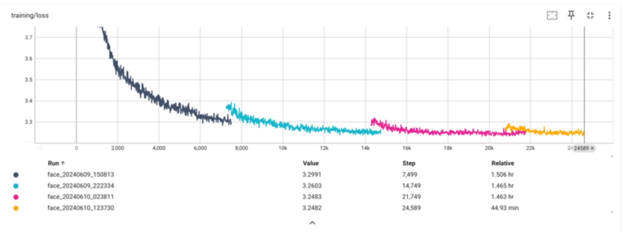
6. RESULTS AND CONCLUSIONS

6.1 Training batch

We used a batch size of 64 with a learning rate of 0.0001. With each batch, there are small drops due to new data being input into the training model, but overall the accuracy keeps increasing. In the first 2000 batches, there is a very steep increase, after that the accuracy increases steadily until reaching 97.29%. For the loss, we see a similar behaviour: a steep drop in the first 2000 batches and a steady decrease until reaching 3.25%.



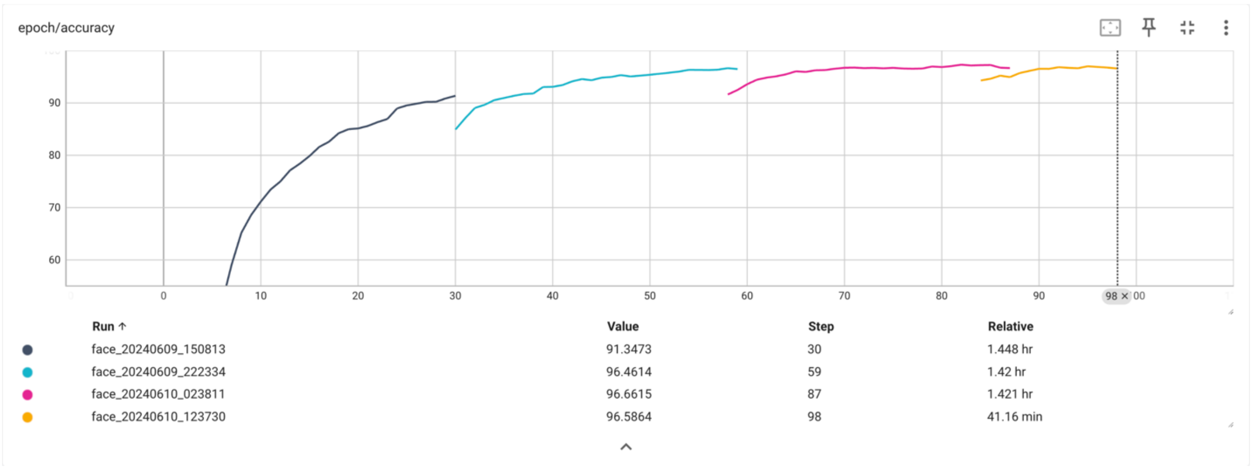
	Run ↑	Value	Step	Relative
●	face_20240609_150813	91.3473	7,499	1.506 hr
●	face_20240609_222334	96.4614	14,749	1.465 hr
●	face_20240610_023811	96.6615	21,749	1.463 hr
●	face_20240610_123730	97.2917	24,589	44.93 min



	Run ↑	Value	Step	Relative
●	face_20240609_150813	3.2991	7,499	1.506 hr
●	face_20240609_222334	3.2603	14,749	1.465 hr
●	face_20240610_023811	3.2483	21,749	1.463 hr
●	face_20240610_123730	3.2482	24,589	44.93 min

6.2 Training epoch

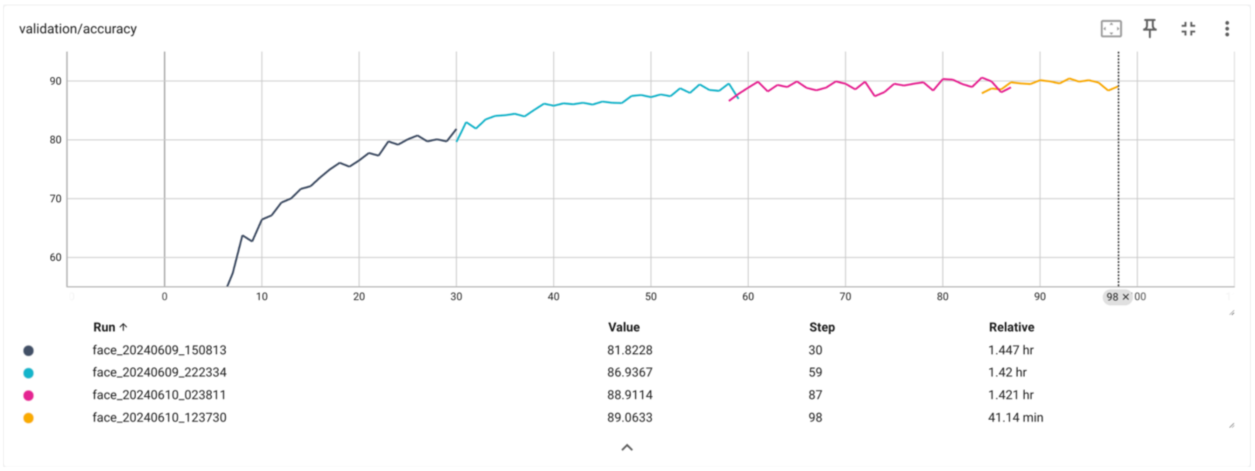
We went through 100 epochs. For every iteration of the data, the accuracy of the model increased, reaching a good 96.59%. This suggests that the model was effectively learning the features of the data over multiple passes. Unfortunately, for the loss, there was an error in the data collection and the data is inconclusive.



	Run ↑	Value	Step	Relative
●	face_20240609_150813	91.3473	30	1.448 hr
●	face_20240609_222334	96.4614	59	1.42 hr
●	face_20240610_023811	96.6615	87	1.421 hr
●	face_20240610_123730	96.5864	98	41.16 min

6.3 Validation

For the validation accuracy, we see a less steep increase than in the training graphs, reaching only 89.06%. This discrepancy could indicate potential overfitting to the training data, where the model performs better on seen data than on unseen data.



	Run ↑	Value	Step	Relative
●	face_20240609_150813	81.8228	30	1.447 hr
●	face_20240609_222334	86.9367	59	1.42 hr
●	face_20240610_023811	88.9114	87	1.421 hr
●	face_20240610_123730	89.0633	98	41.14 min

6.4 Conclusions

The model's high training accuracy suggests that it has learned well from the provided data, however we need to put more attention to the gap between training and validation accuracies. Strategies such as regularization techniques or adjusting the model architecture can mitigate overfitting and improve the generalization of unseen data.

Additionally, the errors in loss data collection highlight the importance of robust monitoring throughout the training process. Clear and accurate metrics are essential to understand the model performance and make eventual fixes.

7. FUTURE DEVELOPMENTS

For the future, we aim to integrate the software with a mobile app, both for Apple and Android, to reduce the time spent between the photo being taken and it being analyzed, prolonged by having to send the photo from the mobile device to the computer.

Additionally, we want to expand our model dataset and computational power, in order to achieve more accurate results and offer a bigger range of actors that can be identified.

8. REFERENCES

[1] DuckDuckGo scrapper: src/murec/scrapper/scrap.py

[2] FacemarkLBF:

https://github.com/php-opencv/php-opencv-examples/blob/master/models/opencv-facemark-lbf/lbfmodel.yaml

[3] IMDB website: https://www.imdb.com/?ref_=nv_home