Age Recognition Project Documentation

Konrad Adamiec, Szymon Tuz, Arkadiusz Ułanowski, Bartosz Wzorek

March 7, 2024

Contents

1 Methodology

1.1 Data Collection

After reviewing publicly available datasets accessible without payment, we opted for UTKFace due to its inclusion of diverse faces and a substantial quantity of them, which allowed us to modify it according to our needs (based e. g. on quality). Many other datasets predominantly featured faces of celebrities, which we aimed to avoid, as celebrities often do not accurately reflect their age. UTKFace, consisting of faces of random individuals, seemed promising. However, we have later discovered that the dataset had some flaws, as discussed in subsequent sections.

1.2 Face Recognition

1.2.1 Algorithm

Initially, we tried to use dlib face recognition, which was used in gathering UTKFace dataset. However, our tests revealed significant performance issues, particularly in tasks involving age recognition from video and live feeds. Consequently, we transitioned to using Haar Cascade from OpenCV for more efficient face detection.

To ensure alignment and consistency with our training set, we employed a method based on the average left eye center position, right eye center position and mouth center position from a random selection of 10 000 samples from the cropped UTKFace dataset (again using Haar Cascade). To get better results we divided each face into top part (upper half, for the eyes) and bottom part (lower third, for the mouth). Our results were:

Listing 1: Results for average positions in UTKFace

```
Average left eye coordinates: (57.310087173100875,
    57.82689912826899)
Average right eye coordinates: (135.36114570361147,
    57.82689912826899)
Average mouth coordinates: (97.13935681470137, 155.28483920367535)
```

Our code for the algorithm below can be found in repository.
Algorithm for Face Alignment:

1. **Detect Faces:**
   - Use OpenCV Haar Cascade algorithm to identify faces within the image.
2. **For Each Detected Face:**
   - **Divide into Upper and Bottom Parts:**
     - Split the detected face into two sections: upper (upper half of picture) and bottom (the lowest third) parts.

- **Scan Upper Part for Eyes:**
  - Analyze the upper part of the face to detect eyes using an eye detection mechanism (Haar Cascade).
- **If Two Eyes are Found:**
  - Calculate the center of the left and right eyes.
  - Transform the original image so that the left and right eyes of the face are aligned horizontally.
  - Scale the image to make the distance between the eyes equal to the average distance in the reference dataset (UTKFace).
  - Adjust the image so the left eye position matches the average left eye position in the reference dataset.
  - Crop the aligned face to a size of 200 x 200 pixels.
- **Scan Bottom Part for Mouth:**
  - Examine the bottom part of the face to detect the mouth using a mouth detection process (Haar Cascade).
- **If One Mouth is Detected (and not Two Eyes):**
  - Align the image so that the detected mouth center matches the average center from UTK.
- **If neither One Mouth or Two Eyes were detected:**
  - Return original face resized to 200 x 200.

Figure 1: Image after rotation

Figure 2: Scaled image

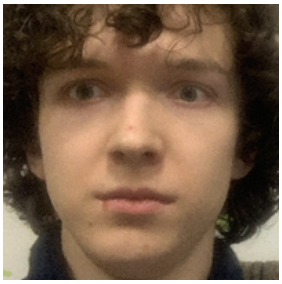

Figure 3: Result of the final transformation

1.2.2 Quality evaluation

To assess the effectiveness of our face recognition algorithm, we compiled a dataset consisting of 50 photos featuring random individuals, including both famous personalities and ordinary people. We tried to make the dataset diverse for the testing.

In addition, we incorporated another 50 photos featuring random animals to test the algorithm's ability to discriminate between human faces and other objects.

To evaluate the performance of the OpenCV Haar Cascade, we conducted experiments with different `minNeighbors` settings, specifically exploring values of 15, 20, and 25. Below are the results:

minNeighbors	**False Positives (Animals)**	**True Negatives (Animals)**
15	1	49
20	1	49
25	0	50

minNeighbors	**True Positives (Humans)**	**False Positives (Humans)**	**False Negatives (Humans)**
15	40	4	10
20	37	0	13
25	36	0	14

The occurrence of false negatives in face recognition results can be attributed to:

1. **Low-Quality Photos:** Faces in images with poor quality, including low resolution and lighting, contribute to false negatives.

2. **Obstructed Faces:** Partially covered facial features, by hands, sunglasses and some other, hinder accurate detection and recognition.

3. **Unusual Facial Expressions:** Faces displaying unconventional expressions pose a challenge for recognition, especially in cases where the algorithm is not trained for extreme variations.

Figure 4: Unusual Facial Expressions example.

Figure 5: Covered Face Example.

minNeighbors	Accuracy	Precision	Recall	F1 Score
15	0.8	0.91	0.8	0.8511
20	0.74	1	0.74	0.8506
25	0.72	1	0.72	0.8372

After consideration of the performance metrics, we have opted to set `minNeighbors` to 15 for our face recognition algorithm. This decision was based on our preference to prioritize higher face detection rates, even at the expense of potential false positives. It aligns with our goal of maximizing face detection.

1.3 Data Preprocessing

Our initial dataset came to us with disadvantages like corrupted data, duplicate images, incorrectly labeled data and such. We've performed some cleaning up to reduce the cardinality of bad data examples.

Figure 6: Examples of corrupted images removed from dataset manually.

The result dataset had a heavily non-uniform distribution amongst different age groups with extreme spikes at around 26 yrs old and almost no examples of people above 80 yrs old. To help the problem we've decided to perform the following data augmentation methods on the training set: example removal, horizontal flip, blur, brightening, darkening, mixup.

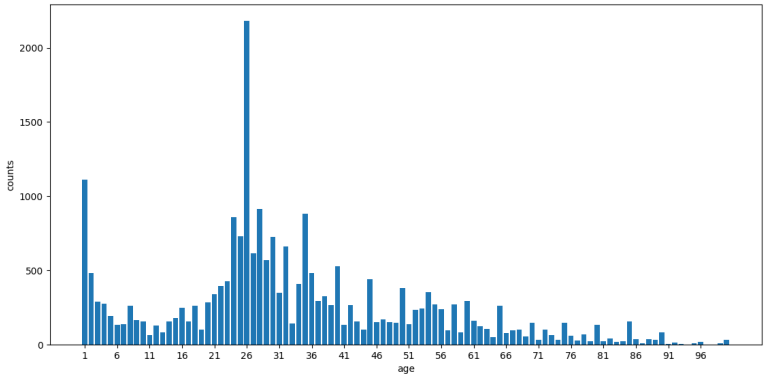

Figure 7: Initial distribution of the dataset.

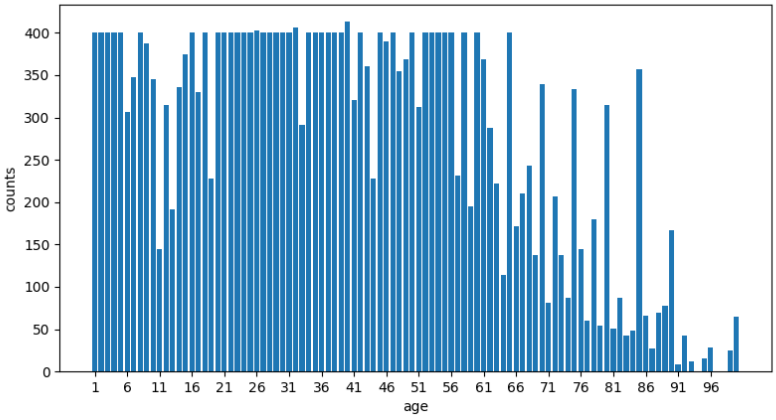

Figure 8: Distribution of the training set after clean-ups and augmentation.

This results in a dataset of a nice distribution, of 200x200 colored images of face examples. Our initial research has shown that the color can be skipped, as it presents no valuable information about the age of a person. However, there were differences in contrast, saturation and contour amongst various age groups. We've decided to extend our images to size 206x206 and in the freshly added pixels we've encoded means and standard deviations of original images' grayscale, eight power of the contour and saturation channel pixel intensities, so we could help the model do the job. The eight power was chosen specifically

based on our empirical observations as histograms of pixel intensities seemed to be the most separated in that power.

To sum things up, our data after processing is a set of 206x206 grayscale images with some statistical measures encoded in them – images nicely distributed among the training set.

Figure 9: An example of preprocessed image with encoded information on new pixels.

1.4 Model Training

We formulated the problem as an image classification task with 90 discrete classes, where the first class corresponds for people age 1 or lower and the ninetieth class corresponds for people age 90 or above. As an architecture we have chosen ResNet-18, defining the FC as: firstly, linear layer consisting of 512 neurons that are then ReLUd, dropped out with the rate of p=0.5, and connected to a linear layer consisting of 90 output neurons. In those we expect raw logits that are passed after to a cross entropy loss which we've been trying to minimize.

Then, softmaxed logits from the output layer form a probability distribution, i. e. we know how sure the model is that a face belongs in each of the classes. As an age prediction we take expected value of the mentioned distribution, rounded to the nearest integer.

A final touch to the model was to take mentioned prediction as an 'early prediction' that is to be corrected. We have calculated biases for each of the early predictions (for example, an average age of a person from the test set for whom age 60 was predicted, was actually 52) and corrected the output by subtracting those from respective groups.

2 Case Study

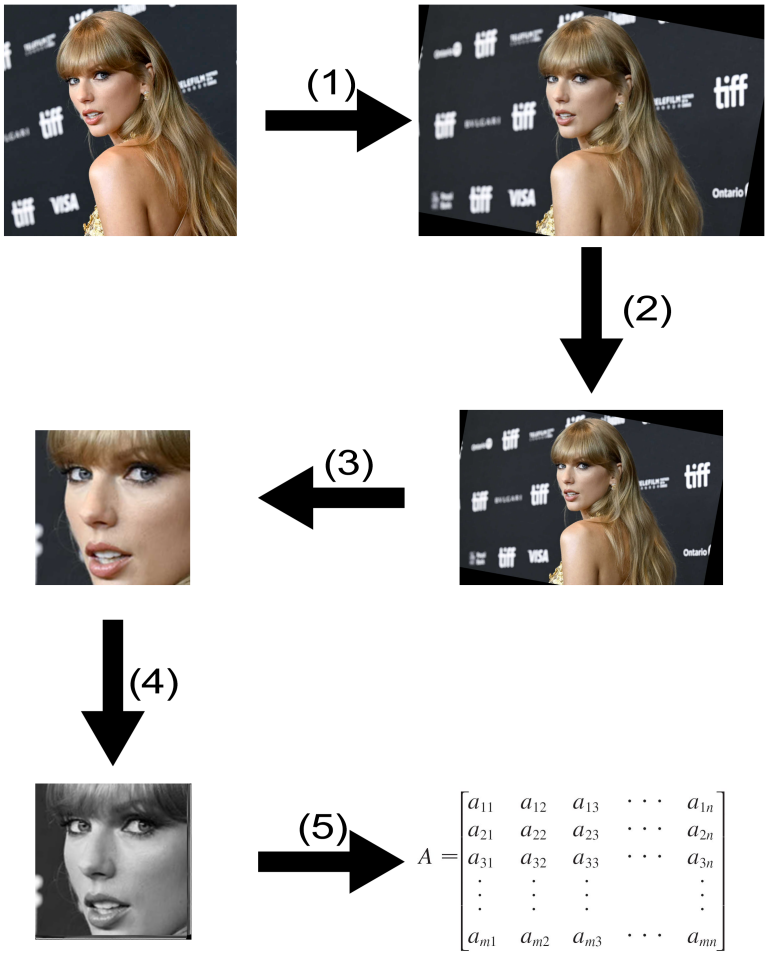

Figure 10: Application pipeline

1. After a face is detected, the app is trying to correct face's rotation
2. The image is scaled so it fits to the dataset's interpupillary distance
3. The image is cut so it fits to the dataset's average eyes positions
4. The image is preprocessed
5. The image is made into 206x206 matrix with values normalized to the (0, 1) interval

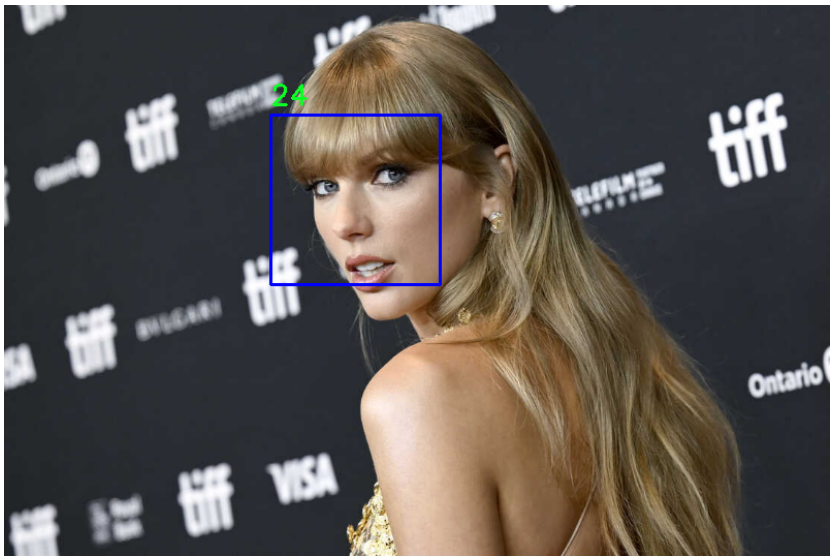

Figure 11: Final result seen by the user. Bounding box is inexact and do not represent well what was passed to the model.

3 Results

Actions performed	MAE achieved on the test set
Training on raw data	5.8436
Data preprocessing	5.6603
Data augmentation, then preprocessing	5.5420
Augmentation, preprocessing, bias substraction	5.3397

References

[1] Castrillón Santana, M., Déniz Suárez, O., Hernández Tejera, M., Guerra Artal, C., *ENCARA2: Real-time Detection of Multiple Faces at Different Resolutions in Video Streams*, *Journal of Visual Communication and Image Representation*, 2007, **18**(2), April, 130-140.

[2] `https://www.youtube.com/playlist?list=PLkt2uSq6rBVctENoVBg1TpCC7OQi31AlC`

[3] `https://twitter.com/karpathy/status/1013244313327681536`

[4] `http://karpathy.github.io/2019/04/25/recipe/`

[5] `https://pytorch.org/docs`

[6] https://medium.com/nerd-for-tech/optimizers-in-machine-learning-f1a9c549f8b4

[7] https://www.geeksforgeeks.org/face-alignment-with-opencv-and-python/