# Guess The Age Contest: Group 8

Faiella Ciro, Giannino Pio Roberto, Scovotto Luigi and Tortora Francesco

{c.faiella8, p.giannino, l.scovotto1, f.tortora21}@studenti.unisa.it

Jan, 2023

Department of Computer Engineering, Electrical Engineering and Applied Mathematics (DIEM), University of Salerno, Fisciano, Italy

## 1 Introduction

The Age Estimation problem turns out to be a complex task even for humans, in certain cases. With computer vision, we aim to simplify the difficulties even for samples that are hard to discriminate. The most common answers to this problem expect the usage of *DeepLearning Algorithms* to find the correct solutions. Our endeavour was to implement a network that tried to carry out age estimation, trying to reduce error to a minimum. Using the CNNs (Convolutional Neural Networks), the network automatically learn the effective image representations.
The solution proposed by us resolves the age estimation task as a regression problem, working with deep neural network. Our choice is based on the minimum distance that could occur between the predict value and the true one.
The network was trained onto the MIVIA Age Dataset [1]. The base model is the SeNet50 [2], pre-trained onto VGG-Face Dataset [3], fine-tuned adding three fully-connected layers at the end. We started from the above pre-trained network, provided by keras-vggface[4], because using network trained for the face recognition problem could be a good starting point for achieving our aim [5]. The loss function used is a custom one, built to maximize the $AAR$ (Age Accuracy and Regularity), because the goal of the contest is to maximize this value, which depends on $mMAE$ and $\sigma$.

## 2 Description of the method

### 2.1 General architecture

**Single experts** As said in the brief introduction, we start from the SeNet50 architecture[2], provided by Keras VGG-Face[4]. Starting from this network, the last layer was removed (the output layer) and the *Average Pooling* layer has been taken as output layer. This layer is linked with a *Flatten* layer, used to

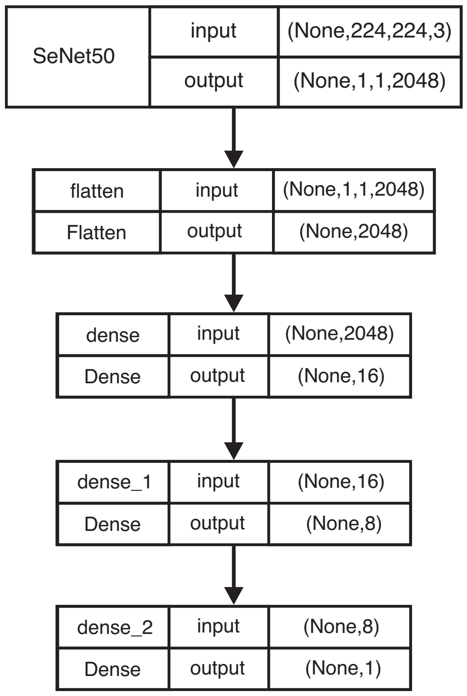


Figure 1: Model architecture. *The SeNet50 architecture is whole compressed in the SeNet50 block.*

convert a multi-dimensional tensor into a one-dimensional tensor and it allows the model to learn from the data more effectively by providing it in a more digestible form. The *Flatten* layer is linked with three fully-connected layer (*Dense*), where the last one has only one neuron, with linear activation function, used for the regression problem solution. The whole architecture is shown in fig.1. The number of neurons chosen in each *Dense* layer was respectively: 16, 8 and 1 (only one neuron for the regression). We use three *Dense* layers instead of only one, because by adding more neurons to the model, we are giving it more capacity to learn from the data. This can be especially useful when the data is very complex or has a lot of features, as in our case. Moreover, in our experiments (as can be seen in the section 4) we noticed an improvement increasing the numbers of fully-connected layers.
Our solution is based on solving a regression problem, instead of a classification one. The root of the choice is considering the gap error: a classifier could predict a class wrongly and the distance between the correct class and the predicted one could be huge. Instead, in a regression problem since the solutions space is a continuous function, the distance, between the predict age and the true one, could be less.

## 2.2 Training procedure

**Dataset** We use the VGG-Face2 MIVIA Age Dataset [1]. It consists of 575 073 images of more than 9 000 identities, collected at different ages. The images are extracted from VGGFace2[3] and annotated with the person's age by means of a knowledge distillation technique[1]. The VGG-Face2 MIVIA Age Dataset is the most accurate facial age dataset currently available at this scale in terms of sample size and heterogeneity.
Since we have a very large dataset, we chose to split training and validation so as to give 90% of the data to the training set, in order to have a large amount of data for training. We might think that the remaining 10% for the validation set is a minimal amount, but the size of our dataset allowed us to make this choice, since that the 10% of it is about 57k images. The sets were split with a constant seed, value 8, so that in all the network tests we performed the split was the same and the performance comparison was unbiased. All labels, containing the age of each face, were normalized to obtain values between 0.01 and 0.81.

**Face pre-processing** We choose the pre-processing function provided by Keras VGG-Face [4]. It makes a copy of the input data, x, and performs mean subtraction on the red, green, and blue channels (assuming the data is in the RGB color space). The resulting preprocessed data is then returned. The *data format* argument is used to specify whether the channels are the first or last dimension in the data. If *data format* is not provided, the image data format in Keras is used instead.

In addition, a *brightness*, that could vary by ±30%, was added during pre-processing, and we specified a scaling factor to be applied to each image. Specifically, the rescale parameter is set to $\frac{1}{255}$, which means that each pixel in the image will be divided by 255, so defined in range $[0, 1]$, after the image is read in.

**Data augmentation** We consider that applying any type of augmentation, such as either horizontal flip or vertical flip, may not be very useful, because the faces are specular and the upside down faces are not common too. Furthermore, the rotation could not be useful because in the dataset there are already images not only in a clear frontal view, but also in a 'wild' mode. The dataset is left unbalanced because the options analyzed were: over-sampling, under-sampling or using some technology able to generate synthetic faces. These operations were discarded due to some problems: in order to use the first one, we had too few elements of age at the extremes of the range to bridge the gap with the middle ages; the second one would have reduced the dataset drastically worsening the middle ages, which are generally the ages most analyzed in real-world environments; using some advanced technology, as the GAN (Generative Adversarial Network), designed to generate faces of a specific age, can be a nice way to balance the set [6], but it have would required a lot of work to reach an high accuracy level such to use it.

**Fine tuning** Since the aim of the contest was the age estimation, we considered the option to use an already pre-trained network for the face recognition aim. Indeed, we consider that start from a network formerly trained to recognize faces[5], contrariwise the normal-used weight for object detection like *ImageNet* released by Keras, the network would be more confident to extract features inherent with our scope. In order to do this, we relied on the Keras VGG-Face package[4].

The keras-vggface is a collection of models that have been implemented with Keras, pre-trained on VGGFace. It provides pre-trained models, such as VGG16, Resnet50 and Senet50[2], used in our experiment. We used these models by loading the pre-trained weights, using them as the starting point for our own model. In particular, we analyzed two models (Resnet50 and Senet50) both pre-trained with VGG-Face dataset.

**Training of the single experts** For the training of our network, we developed a custom loss function (in detail at section 3) to make sure that we maximized the $AAR$, the goal of the contest; certainly using an existing loss function, such as either $MAE$ or $MSE$, would have allowed us to have acceptable results, but considering that the $AAR$ depends not only on $MAE$, but also on $\sigma$, it would have been more intricate to achieve the same results.

Regarding the optimizer, the choice falls on SGD. We found out that SGD with learning rate 0.0001 momentum 0.92 was optimal for finding high-quality local minimum. This thanks to the momentum that prevents the network from converging to sharp local minimum. We also tried ADAM with the same learning rate, but it got stuck in a local minimum greater then the one find out with SGD. The learning rate was set at $10^{-4}$, because using a bigger one for the step size of each iteration, while moving toward a minimum of a loss function, might be too large and bring the function to a larger local minimum, or rather, far from an optimal local minimum. There is also experimentally results to support this choice, proven to be the best one.

The data were normalized passing from the values' range $[1, 81]$ at the $[0.01, 0.81]$ one. This kind of normalization is useful when the loss is calculated, because if we maintain the original values' range, the loss increases drastically and, consequently, reducing it becomes very difficult. Instead, with the new range the loss value is quite manageable, and also improve the entire training process.

The choice of the batch size of 32 is due to experimental tests. In particular, comparing it with a batch size of 64, we saw that the network works better with 32.

The number of epochs totally done were 37, but thanks to early-stopping checkpoint, the network stopped at the 33 epochs in which the best local minimum was reached.

To avoid over-fitting, a forward check has been added, i.e. early stopping. It saves the best model and, from that, keep training looking for better results; if the latter are not found in 5 epochs the training stops maintaining the last good result.

## 3 Loss function design

**Our choices** We designed our loss function taking inspiration from the Age Accuracy and Regularity ($AAR$) metric, defined within the constest as:

$$AAR = \max(0; 5 - mMAE) + \max(0; 5 - \sigma) \tag{1}$$

In this way, by evaluating the values of $mMAE$ and $\sigma$, we maximize the value of $AAR$. Considered that the $mMAE$ is calculate with the follow method:

$$mMAE = \frac{\sum_{j=1}^{8} MAE^j}{8} \tag{2}$$

where $MAE^j$ is the $MAE$ calculated between the 8 age ranges distributed as follows:

1. $MAE^1 \in [1, 10]$;
2. $MAE^2 \in [11, 20]$;
3. $MAE^3 \in [21, 30]$;
4. $MAE^4 \in [31, 40]$;
5. $MAE^5 \in [41, 50]$;
6. $MAE^6 \in [51, 60]$;
7. $MAE^7 \in [61, 70]$;
8. $MAE^8 \in [71, 81]$.

Instead, the $\sigma$ is:

$$\sigma = \sqrt{\frac{\sum_{j=1}^{8}(MAE^j - MAE)^2}{8}} \tag{3}$$

In order to evaluate the loss and back-propagate the gradients, we use the following equation:

$$AAR_{loss} = 0.5 * mMAE + 0.5 * \sigma \tag{4}$$

We used as loss function the eq.4, instead of the eq.1, because considering the maximum between the two values and their sum, if the $mMAE$ and $\sigma$ are greater then 5, the $AAR_{loss}$ would be always 0 and the gradients will not update until the two values descend to 0. Indeed, we consider that the loss will update its values always, even if the $mMAE$ and $\sigma$ are both greater then 5, as in the eq.4. Among the choices made, we decided not to limit the maximum value of $mMAE$ and $\sigma$ to 5 so as to always have a loss other than 0. In addition, the loss function, as performance improves, should decrease, while the $AAR$ (eq.1) increases.

**Description of the algorithm** The operations performed by the custom loss function are as following:

1. The function takes as input two arguments: $y_{true}$, a tensor of true values, and $y_{pred}$, a tensor of predicted values. Both $y_{true}$ and $y_{pred}$ are cast to *dtype float32* because the loss function must have the two tensors of this type.

2. The $MAE$ loss function is imported from the *tf.keras.losses* module and assigned to the variable $mae_{fun}$.

3. The $MAE$ between $y_{true}$ and $y_{pred}$ is calculated using $mae_{fun}$.

4. Based on the value of $y_{true}$, we calculate the $MAE$ relative to the age range normalized (more details here) (e.g. for $y_{true}$ between 0.01 and 0.1, we calculate $MAE^1$)

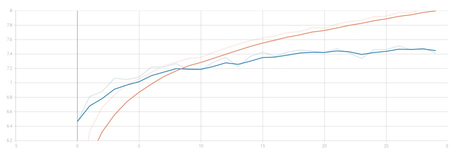
   - $MAE^1$: values of $y_{true}$ $]-\infty, 0.1]$;
   - $MAE^2$: values of $y_{true}$ $]0.1, 0.2]$;
   - $MAE^3$: values of $y_{true}$ $]0.2, 0.3]$;
   - $MAE^4$: values of $y_{true}$ $]0.3, 0.4]$;
   - $MAE^5$: values of $y_{true}$ $]0.4, 0.5]$;
   - $MAE^6$: values of $y_{true}$ $]0.5, 0.6]$;
   - $MAE^7$: values of $y_{true}$ $]0.6, 0.7]$;
   - $MAE^8$: values of $y_{true}$ $]0.7, +\infty[$.

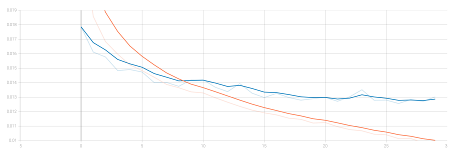
   Again based on the range of $y_{true}$, we calculate $MAE^j_{diff}$ which is the quadratic difference between $MAE$ and $MAE^j$.

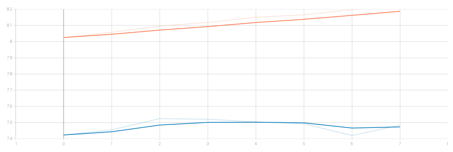
5. All the values of $MAE^j$ are saved within a stack; then all indices of $MAE^j$ whose value is not *None* are identified and the mean is performed, thus going on to calculate the value of $mMAE$ (eq.2).

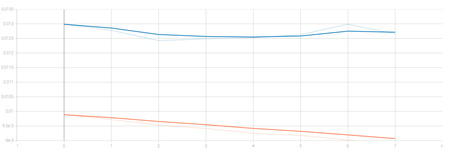
6. All the values of $MAE^j_{diff}$ are saved within a stack; then all indices of $MAE^j_{diff}$ whose value is not *None* are identified and the quadratic mean is performed, thus going on to calculate the value of $\sigma$ (eq.3).

7. Finally, the function returns mean between $mMAE$ and $\sigma$.


(a) $AAR$ first 30 epochs


(b) LOSS first 30 epochs


(c) $AAR$ additional 8 epochs


(d) LOSS additional 8 epochs

● Train
● Validation

Figure 2: SeNet50 tensorboard graphs

## 4 Experimental results

### 4.1 Experimental framework

The entire dataset made available was considered; it has been divided in training set and validation set respectively: 90% and 10%. The random seed used to split the dataset in train and validation were blocked at 8, so in this way, trying different networks, with several hyper-parameters were analyzed on the same validation set and trained on the same set too.
During the train, we developed a custom metric capable of evaluating the $AAR$ as in the eq.1, so we were able to understand if the network was learning properly. This metric calculates the $mMAE$ and the $\sigma$ as in eq.2 and eq.3. Based on the evaluations of this metric we also chose the best model.

**Choice of base model** Starting from the base models of SeNet50 e ResNet50 we made some evaluations. After all, we start from the ResNet50, adding a *Dense* layer at the end, and evaluating it on the validation after about 30 epochs, the $AAR$ value reached was 7.09. Then, we tried the SeNet50 with the same hyper-parameters and, after the same number of epochs, the $AAR$ value was about 7.11. Thus, as can be seen, the two networks get roughly the same performance.

**Ensemble with SeNet50 and ResNet50** Therefore, there was a try to implement a network with ensemble, in order to benefit from both of theirs advantages. This approach, however, turned out to be worse than the network we chose, because the $AAR$ achieved was 7.25. This new network we have defined, had a layer capable of averaging the results coming from two network (SeNet50 and ResNet50, with the above configurations), which having the same input, as the final layer; this last one brought us lower $AAR$ values than we expect from. Moreover, we noticed, evaluating this network on the little test set (fig.3), that this network returned always values in between 30/35 for all the samples, even if the real age was 25 or 57. So, this solutions was rejected and we re-started from the best one single network: SeNet50.

**Dense layers** We applied *Dense* layers to our network instead of only one because due to experiments we notified the performance variations among those two solutions. In fact, in the one with only one *Dense* the $AAR$ was of 7.11 evaluate on the validation set, otherwise, with the three *Dense* layers the $AAR$, after the same number of epochs, reached 7.25. Starting from this point, latter network was trained again until it reached an $AAR$ of 7.52 at the 33 epoch, always evaluated on the validation set.

Figure 3: MIVIA test images

## 4.2 Results

Overall, the results presented show an $AAR$ of 7.5245, with $mMAE$ of 1.1256 and $\sigma$ of 1.3499. It was deemed acceptable by us, although it has limitations with regard to the ages at the extremes of the range proposed in the dataset. This is due to several factors, the most important of which is certainly the unbalance of the dataset used.
We came up with the following final results:

| $MAE$ | $mMAE$ | $\sigma$ | $AAR$ |
|---|---|---|---|
| 1.7258 | 1.1256 | 1.3499 | 7.5245 |

## 4.3 Repeatability and stability of the results

As already mentioned, the system is not particularly stable at the extremes of the age range; in addition, we can assume that the system will not work perfectly with samples where faces are not cropped or samples presenting only a portion of the face. We have not assumed the use of samples with multiple faces in them. What for us is a big point in favour of our solution, is the fact that, applicatively, the ages where our network performs better are precisely the most common ages that could interface with an age estimation system.

## 4.4 Prediction of the results on the test

Our predictions are based on both the results obtained and the known problems in our network. Having encountered problems at the extremes of the age range, we expect to have reduced performance precisely at those points. For ages in the middle part of the age range, we expect reasonable performance, in agreement with the $AAR$ results obtained. We expect minimal variations in $MAE$ and $\sigma$, thus implicitly also in $AAR$, as the network is certainly not perfect and given the presence of the sample problems at the extremes of the age range.

We can draw the following conclusions: if the test set will be unbalanced in the same way as the dataset used, then we expect performance similar to the results we obtained; if, on the other hand, the test set that will be used is a perfectly balanced dataset, we expect considerable deterioration. To show our confidence in our network, we can make an approximate prediction: we think that the test set will be balanced in the same way of the dataset proposed in the contest, so we estimate an $AAR \cong 6.5$.

## References

[1] Greco, A., Saggese, A., Vento, M., Vigilante, V. *Effective training of convolutional neural networks for age estimation based on knowledge distillation. Neural Comput.* 2021.

[2] Hu, J., Shen, L., Albanie, S., Sun, G., Wu, E. *Squeeze-and-Excitation Networks.* 2019.

[3] Cao, Q., Shen, L., Xie, W., Parkhi, O.M., Zisserman, A. *VGGFace2: a dataset for recognising faces across pose and age.* 2018.

[4] Refik Can Malli. *VGGFace implementation with Keras Framework.* 2017. URL: https://github.com/rcmalli/keras-vggface.

[5] Carletti, V., Greco, A., Percannella, G., Vento, M. *Age from faces in the deep learning revolution.* 2020.

[6] Lin, Y., Tang, C., Chen Z., Hsu, J., Shopon, M., Gavrilova, M. *Age-Style and Alignment Augmentation for Facial Age Estimation.* 2021.