Guess The Age

Age Estimation from Facial Images with DCNN

Group 9

Avella Antonello	a.avella19@studenti.unisa.it	**0622701703**
Carpentieri Eugenio	e.carpentieri6@studenti.unisa.it	**0622701804**
Costantino Valerio	v.costantino2@studenti.unisa.it	**0622701675**
De Pisapia Claudio	c.depisapia1@studenti.unisa.it	**0622701712**

Course
Artificial Vision

Lecturers
Percannella Gennaro
Vento Mario
Greco Antonio

Summary

1 Introduction

In this report, we present the work carried out in the age estimation contest for the Artificial Vision course.

Age Estimation is the task of estimating the age of a person from an image or some other kind of data and is nowadays a relevant problem in several real applications. In the era of deep learning, many DCNNs for age estimation have been proposed, so effective to achieve performance comparable to those of humans.

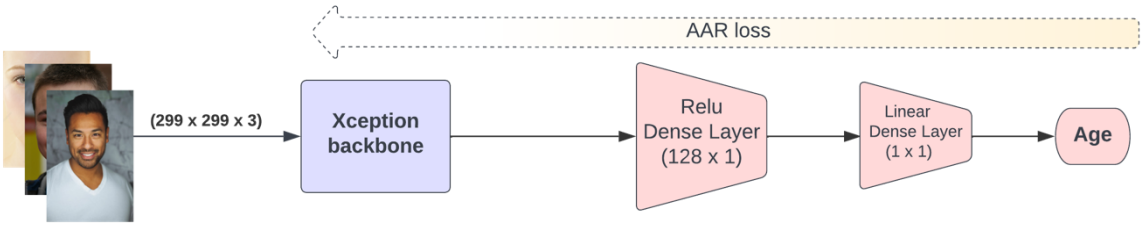
Therefore, we first analyzed the state of the art regarding this problem, with particular attention paid to the methods proposed in the GTA2021 contest [1].

Next, we reviewed several possible architectures, both single-expert and multi-expert, and designed a loss function that could take into account the contest requirements. At this point, the implementation and testing phase of the proposed ideas began.

We decided to start by testing several backbones from among those pre-tested for both face recognition and image classification tasks, choosing the one that looked the most promising. After that, we compared the performance of the backbone with the performance of the latter with different fully connected layers, to see if there was any improvement.

2 Description of the method

2.1 General Architecture



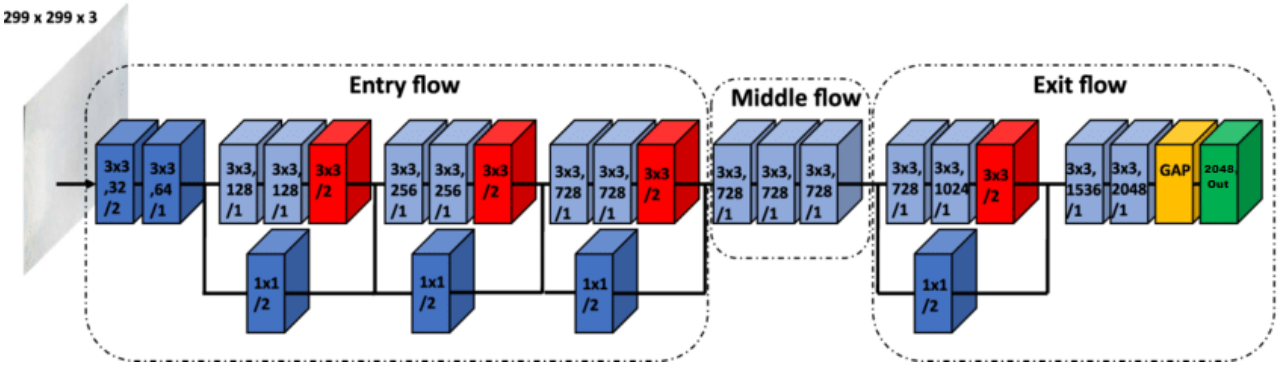
Our architecture is depicted in the image above: the samples of the dataset provided for the contest are fed into a Xception backbone based on ImageNet dataset weights without top layers so that it can be customized for the concerned task; after that, the feature vector resulting from the backbone is passed as input to a regressor (composed by fully connected layers) that is responsible for providing the final output, i.e. the age associated with the current input.

Let's look in detail our architecture.

2.1.1 Backbone

The first part of the architecture was structured using the Xception network as a backbone to extract feature vectors followed by a global average pooling and two dense layers to specialize the network to the facial age estimation task.

Xception is a deep convolutional neural network architecture developed by Google researchers, inspired by Inception. Its architecture is shown below:



As it is known, Xception is able to perform very well on image classification tasks, but it is still a good choice for our regression task for several reasons. First of all, because we are working with image data and our regression task is closely related to an image classification one, but also because it is a very deep neural network, with a large number of layers, that has already been trained on a large dataset and has learned to extract useful features from images for seizing complex relationships in the data that can be useful for a wide range of image-based tasks.

Additionally, it makes use of depthwise separable convolutions (reversed from the original, since it involves a pointwise convolution followed by a depthwise convolution), which can help reduce the number of parameters in the model and make it more computationally efficient.

Based on some research carried out on this architecture, we noticed that Xception is not necessarily the best choice for all regression tasks, so we analyzed a variety of different models and architectures to find the one that works best for our specific case, but this will be further discussed later in this report.

2.1.2 Regressor

For the second part of our architecture, we employed a regressor that takes as input the feature vector returned by Xception and provides the prediction on the age of the current input.

In particular, this is composed by:

- ReLu Dense Layer with 128 neurons
- Linear Dense Layer with 1 neuron

One advantage of using a fully connected part with a regressor for age estimation is that it can handle a wide range of ages and can predict ages with a high degree of precision. This is because the output of a regressor is a continuous numerical value, rather than a discrete category, so it can make predictions for any age within a certain range. In contrast, using a classifier for age estimation would require discretizing the age range into a fixed set of categories, which may not be as accurate or flexible.

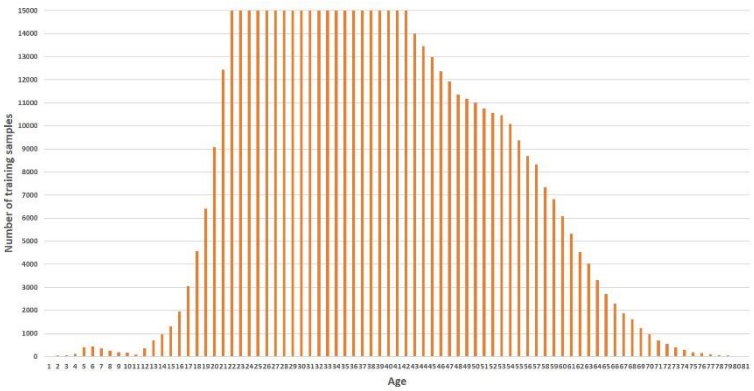
In addition, by using a classifier, we do not have the possibility of differentiating a 100-year error from a 2-year error, which we can do by using an optimized regressor with a MAE-like loss, as in our case.

2.2 Training Procedure

2.2.1 Dataset

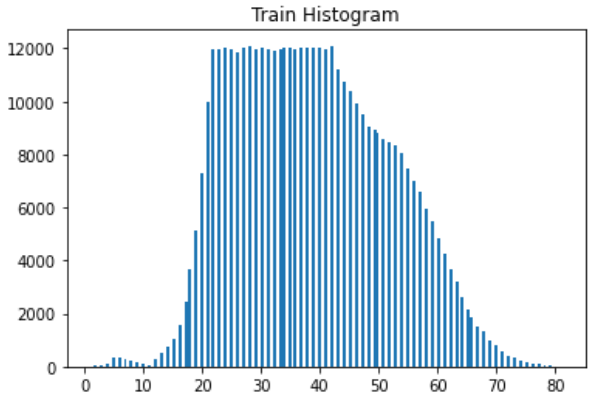
The MIVIA Age Dataset is composed of 575.073 images of more than 9.000 identities, got at different ages. They have been extracted from the VGGFace2 dataset and annotated with age by means of a knowledge distillation technique, making the dataset very heterogeneous in terms of face size, illumination conditions, facial pose, gender and ethnicity. Each image of the dataset contains a single face, already cropped.

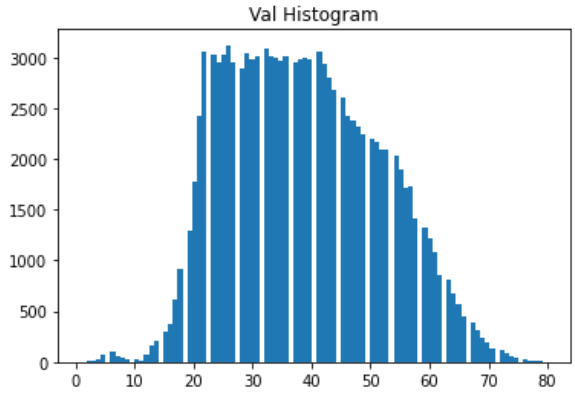
For each age group, we took 80% of the samples for the training set and 20% of the images for the validation set (respectively for a total of 460058 and 115015 samples), taking into account the original distribution of the samples over the age, depicted on the right.



To evaluate performance on the validation set, we used the loss described in the next section.

The distributions of the samples in training and validation sets after splitting are shown below:





2.2.2 Face pre-processing

We performed an online pre-processing for each sample in the batches. In particular, each sample has been resized with a bilinear interpolation to obtain an image with the dimension required for the network input and then each pixel has been normalized between 0 and 1.

2.2.3 Data Augmentation

To apply data augmentation to the samples in the dataset, we used Keras ImageDataGenerator. Specifically, we randomly apply a change in image brightness between 0.5 and 1 and a horizontal flip. These transformations are applied with a probability of 50%.

2.2.4 Data Generator

To avoid loading the entire dataset (and consequently saturating the GPU memory) a custom data generator has been created that performs batch creation. In addition, this custom data generator is

entrusted to execute data augmentation and all pre-processing steps when an image is loaded to be included in the batch.

Moreover, to deal with dataset unbalancing and so to obtain each batch balanced, i.e. with the same number of elements for each age group, we carried out an undersampling for the samples of the most represented classes and an oversampling for those of the least represented classes. In this way, the set of all batches in an epoch does not cover the entire dataset and so in an epoch the set of samples that the network sees no longer has a Gaussian distribution, but a Uniform one that has the same number of samples of the original training and validation set.

To ensure that the network receives as input approximately all the samples in the dataset, a randomization of the order of the samples in the training and validation set is performed at the end of each epoch.

To prevent the loss gradient from rising too much, we preferred to normalize the output expected from the network as well.

Normalizing labels can be useful in a neural network training procedure because it can help the model to learn more effectively. There are a few reasons for this:

- Numerical stability: Normalizing the labels can help to ensure that the model doesn't become numerically unstable, which can make it difficult for the model to learn.
- Faster convergence: Normalizing the labels can also help the model to converge faster, as the optimization algorithm will be able to make larger steps in the right direction.

2.2.5 Training

To train the previously described architecture, we opted for a sequential training of the two parts, as we needed the best feature vector obtainable from the backbone to train the regressor. Below we see in detail the training procedures adopted.

Training of the backbone

The training followed two steps. First, we trained only the final dense layers, leaving the pre-trained weights of the backbone blocked, with the following parameters:

- Optimizer: Adam
  - Learning rate: 0,001
  - Clipvalue: 1
- Loss: AAR_loss
- Metrics:
  - MAE
  - AAR_metric
- Callbacks:
  - ModelCheckpoint:
    - Monitor: val_loss
  - EarlyStopping:
    - Monitor: val_loss
    - Patience: 3
- Max Epoch: 100
- Batch size: 32

Then, we unfroze the backbone weights and apply a fine tuning with a smaller learning rate:

- Optimizer: Adam
  - Learning rate: 0,0001
  - Clipvalue: 1
- Loss: AAR_loss
- Metrics:
  - MAE
  - AAR_metric

- Callbacks:
  - ModelCheckpoint:
    - Monitor: val_loss
  - EarlyStopping:
    - Monitor: val_loss
    - Patience: 5
  - Reduce Learning Rate on Plateau:
    - monitor="val_loss"
    - factor=0.1
    - patience=2
    - mode="auto"
    - min_delta=0.0001
    - cooldown=0
    - min_lr=0
- Max Epoch: 100
- Batch size: 32

We decided to apply a fine-tuning policy because the task at hand was complex and therefore it was convenient to start from an optimum that is closer to the global optimum than an optimum that we could find from random weights.

Moreover, we decide to reduce the learning rate during training to improve the performance of our model to avoid the model's weights oscillation or divergence and local minima.

Furthermore, we used early stopping to avoid overfitting and so to improve the generalization performance of the model on unseen data.

Training of the regressor

Taking the feature vector obtained from Xception, we structured the following training for the regressor as follows:

- Optimizer: Adam
  - Learning rate: 0,00001
  - Clipvalue: 1
- Loss: AAR_loss
- Metrics:
  - MAE
  - AAR_metric
- Callbacks:
  - ModelCheckpoint:
    - Monitor: val_loss

    - EarlyStopping:
        - Monitor: val_loss
        - Patience: 3
- Max Epoch: 100
- Batch size: 32

For the training procedure of this regressor, it has been used the dataset with the same split used to train the backbone, with the difference that in this case there are neither balanced batches nor data augmentation, thus the data fed to these networks follow the original distribution (there is not undersampling or oversampling).

3 Loss function design

The loss function was designed referring to the metric AAR used in the contest:

$$\boldsymbol{L_{AAR}}(y,\hat{y}) = \boldsymbol{\gamma \cdot mMAE}(y,\hat{y}) + \boldsymbol{\lambda\sigma}(y,\hat{y})$$

Where:

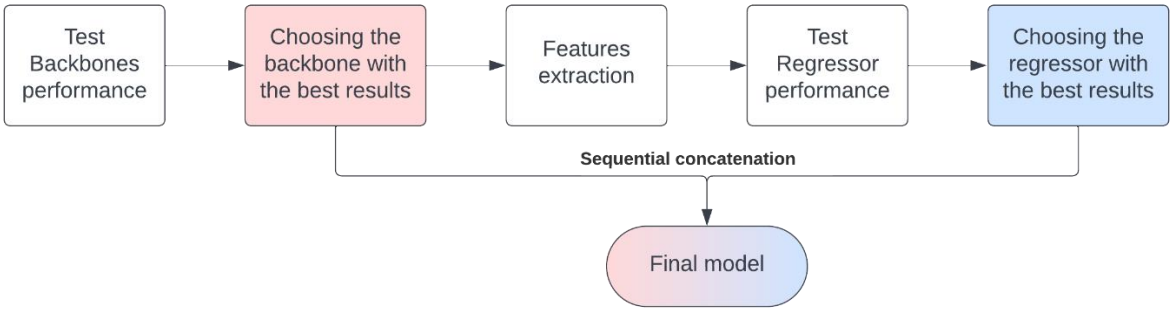
- $mMAE(y,\hat{y}) = \frac{1}{8}\sum_{j=1}^{8} MAE_j\ (y,\hat{y})$
    - $MAE_j(y,\hat{y}) = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \cdot I(y_i \epsilon G_j)$
    - I is the indicator function
    - G_j refers to age groups
- $\sigma(y,\hat{y}) = \sqrt{\frac{\sum_{j=1}^{8}(MAE_j(y,\hat{y}) - MAE(y,\hat{y}))^2}{8}}$
    - $MAE(y,\hat{y}) = \frac{\sum_{i=1}^{N}|y_i - \hat{y}_i|}{N}$

γ and α are regularization parameters and we set them to $\gamma = \frac{1}{2}$ and $\alpha = \frac{1}{2}$. y_i is the true age label of the i-th sample and its prediction is $\hat{y}_i$. We compute the MAE of each group by averaging the absolute errors of all the samples over each group: I is the indicator function which is 1 if the condition is true, and 0 otherwise. G_j represents the j-th age group, the one we are computing the MAE over. The average over the 8 groups, namely the $mMAE$, is part of the loss function together with the variance over the MAE of each group, σ. The variance is computed by averaging the square error between the j-th MAE and the total MAE.

We designed the loss function in this way because by definition a loss function must be minimized, we know that in order to obtain a high AAR value we have to minimize $mMAE$ and variance and so we thought of defining the loss function as a linear combination of precisely these two terms.

4 Experimental results

4.1 Experimental framework



First of all, we selected a set of pre-trained backbones to test in order to select the most promising one for our task.

The selected backbones are the following:

- VGGFace ResNet 50
- SeNet 50
- MobileNet
- Xception

For each of the backbones, several types of training have been performed with different hyper-parameters to have an idea of which network could be the best performing among the different ones.

To test the performance of the backbones, the following 2 Dense layers have been added:

- Relu Dense layer (128 x 1)
- Linear Dense layer (1 x 1)

Unfortunately, due to limited time and computational resources, it has not been possible to try all the settings for the hyper-parameters for each network since the training time is quite high given the large dataset.

In the next section, the different settings and their respective result will be shown.

Given the best-performing model, we removed the last two layers of it to extract the feature vectors for each sample of the dataset. These features have been used to train a set of different regressors that can be used in place of those last layers of the network to give the final age prediction. We analyzed the impact (or the non-impact) of different combinations of fully connected layers.

The selected fully connected regressors are the following:

1. Relu Dense layer (128 x 1) + Linear Dense layer (1 x 1)
2. Relu Dense layer (1024 x 1) + Linear Dense layer (1 x 1)
3. Relu Dense layer (1024 x 1) + Dropout(0.2) + Relu Dense layer (128 x 1) + Dropout(0.2) + Linear Dense layer (1 x 1)

The regressor in step 1 has the same structure as the one used for choosing the backbone. What we tried was to see how it initially behaved on the original Gaussian distribution knowing that it had been trained on a uniform distribution, and then performed further fine tuning on the original

Gaussian distribution with a very low learning rate, as we will show in the results. Instead, the other regressors were trained directly on the original Gaussian distribution.

Obviously, different fully connected use different sets of hyper-parameters which will be shown in the next section.

At the end, to assemble the final model we chose the best backbone and the best associated regressor, which was chosen using the highest AAR Metric on the validation set.

Although not allowed within the scope of the contest, other (computationally more complex) methods were also tried to corroborate the result. The one with the best result was the CatBoost regressor. The results obtained with these methods will also be shown in the results section.

4.2 Results

After training the different models with various backbones, we report the hyper-parameters of the distinct tests and their relative results:

Model name	Batch size	Optimizer(Learning rates)	Training backbone	Train AAR loss	Val AAR loss	Val AAR metric	Val MAE
VGGFace	64	Adam(0.001,0.0001,0.00001)	No/Yes/Yes	0.0052	0.02	6.19	0.025
Senet 50	32	Adam(0.001 with reduce LR)	Yes	0.0085	0.02	6.46	0.022
Mobilenet	128/64	Adam(0.001/0.0001 with reduce LR)	No/Yes	0.0071	0.02	6.3	0.027
Xception	32	Adam(0.001/0.0001 with reduce LR)	No/Yes	**0.0068**	**0.0072**	**8.56**	**0.012**

As can be seen in the results, the best-performing network, considering AAR Metric and AAR Loss on the validation set, is the one with the Xception backbone.

Next, we extracted the feature vectors for each sample of the dataset from the Xception backbone. These features were used to train the previously mentioned regressors. We then report the hyper-parameters of the different tests and their relative results:

Booster name	Batch Size	Parameters	Train AAR metric	Train MAE	Val AAR metric	Val MAE
Relu(128)+Linear Base	32	Adam(0.001/0.0001 with reduce LR)	7.99	0.017	7.99	0.017
Relu(128)+Linear Finetuned	32	Adam(0.00001)	**8.11**	**0.016**	**8.11**	**0.017**
Relu(1024)+Linear	32	Adam(0.001/0.0001 with reduce LR)	2.28	0.03	2.24	0.03
Relu+Relu+Linear	32	Adam(0.001/0.0001 with reduce LR)	2.94	0.03	2.90	0.03

In this case, to realize how much improvement the use of the regressors brought relative to the base model i.e., Xception with 2 dense layers, we computed the AAR Metric on the former using the validation set with the original distribution to have a fair comparison with the results of the regressors.

It is clear that ReLu(128)+Linear Finetuned gives us the highest improvement of the AAR Metric with respect to the base model, with an increase of 0.12.

It is important to emphasise that the results in the table were obtained, following the training phase, by administering the training set and the validation set to the network in the form of a single batch in an attempt to obtain results as faithful as possible to the results obtainable on the test set.

We can also note that the AAR on the training set and on the validation set is approximately the same, we can then deduce that the network is not overfitting.

To analyze with more precision the error of the age estimation, we show the MAE for each group age:

Group Age	Train MAE	Val MAE
1 - 10	0.46	0.7
11 - 20	1.4	1.4
21 - 30	1.5	1.6
31 - 40	1.8	1.8
41 - 50	1.8	1.8
51 - 60	1.5	1.5
61 - 70	1.1	1.2
70 +	0.6	0.63

Furthermore, we compute the mMAE and the variance from the data shown above to better understand the quotas that compose the AAR. The value of mMAE is 1.34, while the value of the variance is 0.54.

As mentioned earlier, to emphasise our previous results, various boosters were also tried:

Booster name	Train AAR metric	Val AAR metric
XGBoost	9.98	8.05
LightGBM	8.27	8.16
CatBoost	8.18	8.16

From these results, it can be seen that with our training procedure, there is an underlying bias that could not be bridged even with ensemble methods with more than 3 classifiers, although these are not permitted in this contest.

4.3 Repeatability and stability of the results

As described in the “Data generator” paragraph in Section 2.2, the training procedure of our first model heavily relies on randomization.

The operations of undersampling and oversampling let the data fed to the network follow an uniform distribution at the cost of selecting only a subset of samples in the dataset. The selection of this subset is partially mitigated by shuffling the samples at the end of each epoch in order to feed, on average, all the data to the network at the end of the training phase.

The split between training set and validation set also plays an important role in the stability of the results. Given the two levels of randomization, the best approach to obtain stable results is a repeated k-fold cross-validation which consists in multiple k-fold iterations and averaging the results.

The training of the final regressor, however, does not suffer of the same instability as the base model. Thus, a simple k-fold cross-validation is enough to mitigate the problem of holding out a validation set. These techniques have not been applied due to limited training time.

4.4 Prediction of the results on the test

Given the 8.11 AAR value obtained on the validation set with the original distribution, knowing the size of the test set which is approximately the same as our validation set, and being aware of the decrease in performance on new data, we estimate that our model obtains an AAR score of 6.

References

[1] A. Greco, «Guess the Age Contest,» in *19th International Conference, CAIP 2021*, 2021.