Introduction to Machine Learning

Machine Learning Project Final Report

Szymon Markiewicz
Justyna Pokora
Jacek Zalewski
Błażej Misiura

Faculty of Mathematics and and Information Science
Warsaw University of Technology
Poland
January 27, 2023

Contents

1 Introduction **2**

2 Dataset **2**
2.1 Description of dataset . . . 2
2.2 Cleaning data . . . 3
2.3 Augmentations . . . 3

3 Convolutional Neural Networks **3**
3.1 What are Convolutional Neural Networks? . . . 3
3.2 Separate models for women and men . . . 4
3.3 Dataset . . . 5
3.4 Model structure . . . 12
3.5 Final Model . . . 14

4 Transfer learning **15**
4.1 What is transfer learning? . . . 15
4.2 Models used in project . . . 16
4.3 ResNet . . . 16
4.3.1 ResNet - short description . . . 16
4.3.2 ResNet - models used in project . . . 16
4.4 MobileNet . . . 23
4.4.1 MobileNet - short description . . . 23
4.4.2 MobileNet - models used in project . . . 23
4.5 Comparisons . . . 29

5 CNN vs Transfer Learning **29**

6 Software description **29**
6.1 Introduction . . . 29
6.2 Application prerequisites . . . 29
6.3 Graphical user interface . . . 30
6.4 Algorithm . . . 30
6.4.1 Model choice . . . 30
6.4.2 Frame processing algorithm . . . 31
6.4.3 Resizing and cropping of image . . . 31

7 Use case scenarios **31**
7.1 Starting the application . . . 31
7.2 Real-time video analysis . . . 32
7.3 Processing a pre-supplied video . . . 33
7.4 Processing an image array . . . 34

8 References **36**

1 Introduction

The aim of this project was to gain experience in developing machine learning models and implementing them in algorithms that would serve specific use cases. The project required us to deliver two main products:

- an age detection model that is trained and tested by us, every step being documented and with the possibility to be recreated
- a software application which detects human faces and displays predictions, implementing some arbitrary face recognition model and the one mentioned above for providing information about the age

2 Dataset

2.1 Description of dataset

In our project, we decided to use the UTKFace dataset. It consists of over 20,000 photos tagged with age, gender and race. Images include large differences in pose, facial expression, lighting, occlusion, resolution, etc. It covers people from 1 year old to over 100 years old. In the case of both men and women, the largest number of people is aged 26. The age distribution is shown in the histogram below.

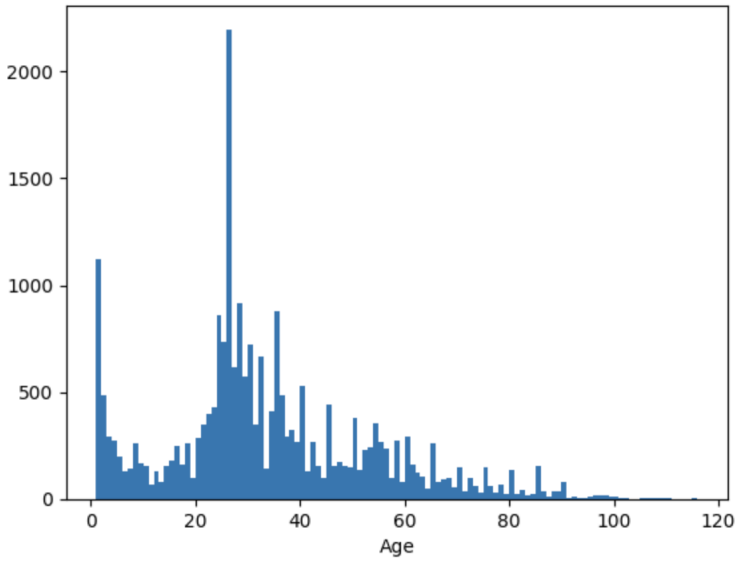


Figure 1: Age distribution in a whole sample

As it can be seen, the elderly (above about 90 years of age) appear only as single individuals in the dataset. The box plot shown below marks these observations as outliers. It graphically presents locality, spread and skewness groups of numerical data through their quartiles.

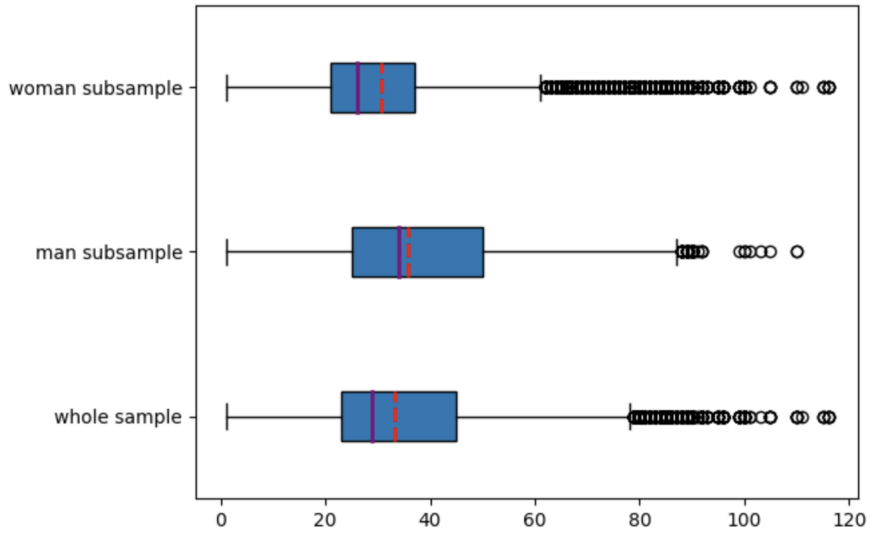


Figure 2: Boxplot of age variable

The mean is the red dashed line. The median is the purple line. The first quartile is the left edge of the blue rectangle. The third quartile is the right edge of the blue rectangle. The interquartile range is the length of the blue rectangle. The range contains everything from left to right. The outliers are the dots to the left and right.

Some modifications to the data were made during the modeling, but they are described appropriately.

2.2 Cleaning data

In order to remove erroneous observations, we decided to manually verify the dataset. During it, we removed 28 photos from the collection, due to i.a. incorrect age classification, no face in the photo, blurred photo. We also verified whether the images were labeled correctly and removed additional 3 which had missing information in their name.

2.3 Augmentations

Our first step to making the dataset more uniform was to limit it to people up to 75 years of age - whch resulted in removing additional 850 pictures. Initially, we wanted to even out the dataset (by extending it) by augmenting the data and saving it as a pickle so that it could be fed into the modeling process. Thus, the dataset had over 50000 photos and loading the entire dataset and using it for modeling caused RAM problems as it used all the 32GB we had at our disposal. This made the whole process very long, so we had to augment and align the data in a different way. We made the alignment by using appropriate weights. Data were augmented in the form of a horizontal random flip and rotation by a random amount in the range [-20% * 2pi, 20% * 2pi].

3 Convolutional Neural Networks

3.1 What are Convolutional Neural Networks?

A Convolutional Neural Network (CNN) is a type of Artificial Neural Network (ANN). CNNs are generally used to solve image-driven pattern recognition problems. (Keiron O'Shea, Ryan Nas, 2015)

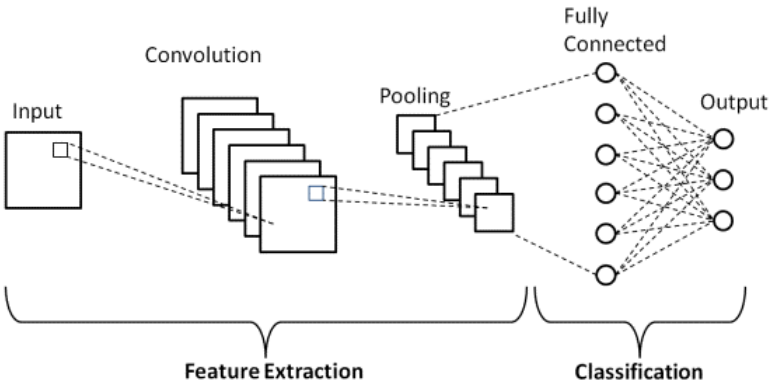


Figure 3: CNN architecture

In the CNN architecture, not counting the input and output, there are 3 main types of layers - convolution, pooling and fully connected layers. Each of the 3 layers can appear multiple times in the model structure.

- Convolution layer - used to extract numerous features from the input image. The output of this layer, called the feature map, is obtained by applying a set of filters to the input using a process called convolution, where each filter is detecting some sort of feature of the image. (Keiron O'Shea, Ryan Nas, 2015)
- Pooling layer - used to downsample the spatial dimensionality of the passed down input - the feature map, to reduce the time needed for computation. (Keiron O'Shea, Ryan Nas, 2015)
- Fully connected Layer - used to connect the neurons between two different layers, it is the part of the CNN where classifications based on the produced earlier feature maps takes place. (Keiron O'Shea, Ryan Nas, 2015)

Other important layer is the Dropout layer, which is used to prevent overfitting of the model during training by randomly choosing a set of neurons 'drop' each time it is called. (MK Gurucharan, 2022)

One additional key component of the CNN is the activation functions, which when applied to a layer of neurons decide what information is moved forward. For example the Rectified Linear Unit (ReLU) activation returns 0 for a negative input and otherwise it returns the input itself. (MK Gurucharan 2022)

3.2 Separate models for women and men

At the beginning we made 3 separate models - for detecting the gender of the person and for predicting the age for both women and men separately, as we though it might increase the overall accuracy. We tested this hypothesis on by training 3 models - detecting age, detecting age for women only and detecting age for men only. We used the same model structure for all of them.

We used the Sequential model from the tensorflow.keras.models library. The initial structure of the model is based on one of the Lectures from an ML course by Joaquin Vanschoren (https://ml-course.github.io/master/notebooks/09%20-%20Convolutional%20Neural%20Networks.html).

```
model = Sequential()
model.add(Resizing(100,100))
model.add(Rescaling(1./255))
model.add(RandomFlip("horizontal"))
model.add(RandomRotation(0.2))
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(1, activation='relu'))


model.compile(loss="mae",
              optimizer = "adam",
              metrics =["accuracy"])


history = model.fit(train_image,train_age, batch_size=32,
                    epochs = 100, validation_split = 0.1,
                    callbacks = [tensorboard_callback, early_stopping])
```

Figure 4: Model structure

For the age detection we used early stopping callback with the patience set to 10 epochs. The resulting model had 6.4054 Mean Absolute Percentage Error (MAE) after 57 epochs.

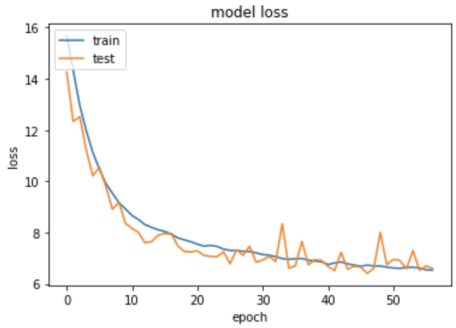


Figure 5: Model loss for age detection

As for the models detecting ages for both genders separately - we did not use the early stopping callback and set them up to go for 100 epochs. The lowest MAE for age detection for men was 6.4157 and for women 6.5062, which meant the model did not improve as we wanted it to, so we decided to move forward with only the age detection, not taking the gender into account.

3.3 Dataset

As described in the 2.3 section we wanted to even out our dataset. Our first approach oversampling and undersampling the dataset until the distribution for each class was more even. Some of the datasets created were

1. Dataset with limited number of augmentations per picture - to try to prevent overfitting. We created the train and test set by first removing random pictures from classes with over 1000 images, proportionally diving the images between the train and test sets for each age class (8/10 to train class, 2/10 to test class) and augmenting the images in the train class. The augmentation was done for random pictures

using ImageDataGenerator from keras.preprocessing.image library. Initial configuration used is shown in Figure 6.

```
datagen = ImageDataGenerator(width_shift_range=0.03,
                             height_shift_range=0.03,
                             horizontal_flip=True,
                             rotation_range=50,
                             brightness_range=[0.5,1.5],
                             zoom_range=[0.7, 1.1]
                             )
```

Figure 6: Augmetations applied to the images

This dataset is in greyscale with image size 100x100. All other datasets were based on this one.

2. Datasets with different sizes (50x50 and 224x224, both in greyscale) and in rgb (100x100)
3. Dataset with all classes in the train set evened out to 800 pictures
4. Dataset with all classes in the train set evened out to 1000 pictures, but still only keeping at most 800 of the original pictures - the rest were augmented, as to not have some classes with almost none augmented pictures compared to the others.
5. Dataset with different augmentations applied

In Figure 7 are the age distributions for the sets (1), (3), (4). The test dataset has the same distribution for all of the sets and is shown in Figure 8.

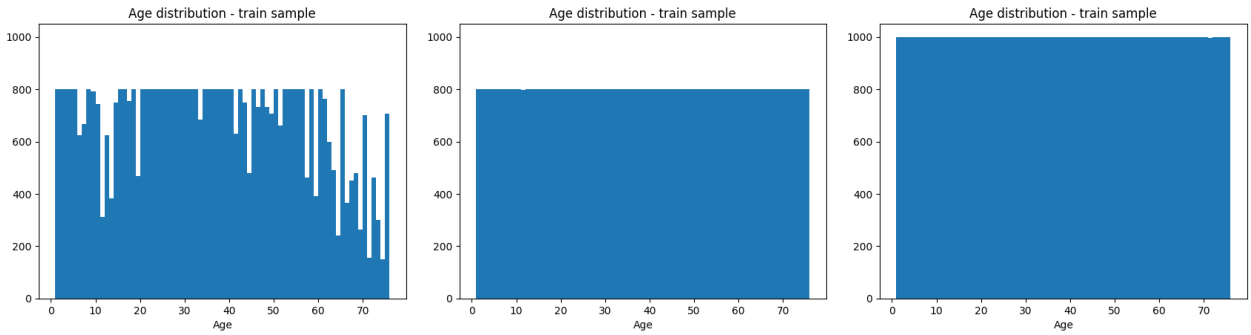


Figure 7: Age distributions for the train samples of datasets (1), (3) and (4)

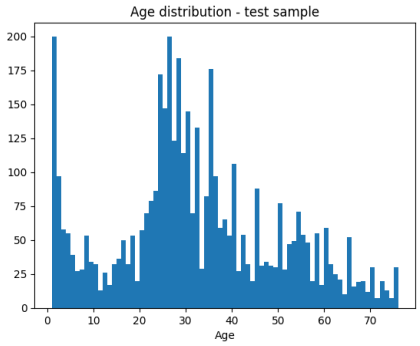


Figure 8: Age distribution for the test samples

For training the models we used the same model structure as in Figure 4, but without the RandomFlip and RandomRotation Layers and number of epochs set to 25.

First we compared the models trained on the base greyscale sets with different sizes - 50x50, 100x100 and 224x224. In Figure 9 are the plots of model loss (MAE) in all of these sets. Contrary to our expectations the dataset with the biggest image size - 224x224 did the worst. For the 50x50 and 100x100 sets the model loss got close to a 7, but for the 224x224 set it did not go below 8. Based on the error distribution and not wanting to lose too much information and because it did not take more time to train on the 100x100 pictures the 100x100 size for images was chosen.

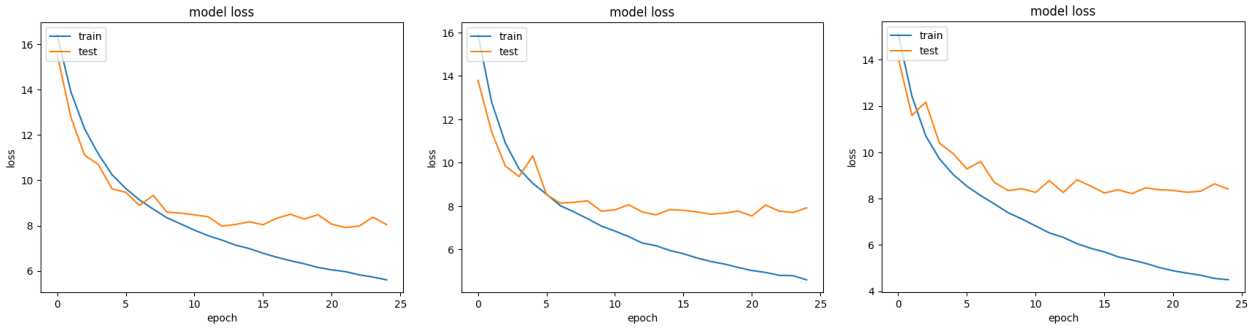


Figure 9: Model loss (MAE) for the greyscale datasets with image sizes 50x50, 100x100 and 224x224

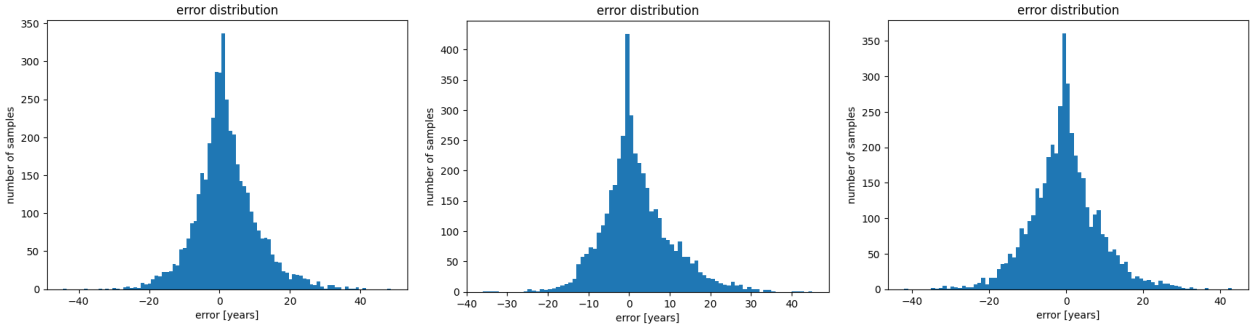


Figure 10: Error distribution for the greyscale datasets with image sizes 50x50, 100x100 and 224x224

We also decided between greyscale and rgb images. In Figure 11 and Figure 12 are the plots for model loss and error distribution. Because both the MAE and the error distribution were very similar the rgb images were used from now up until the final model. Computations for greyscale are faster, but useful information might be lost. (Joseph Nelson, 2020)

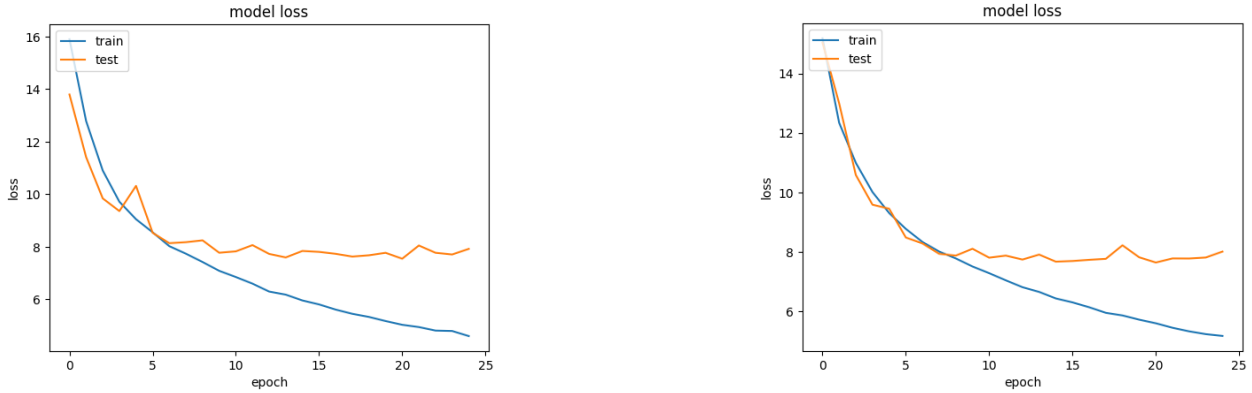


Figure 11: Model loss (MAE) for the greyscale and color datasets with image sizee 100x100

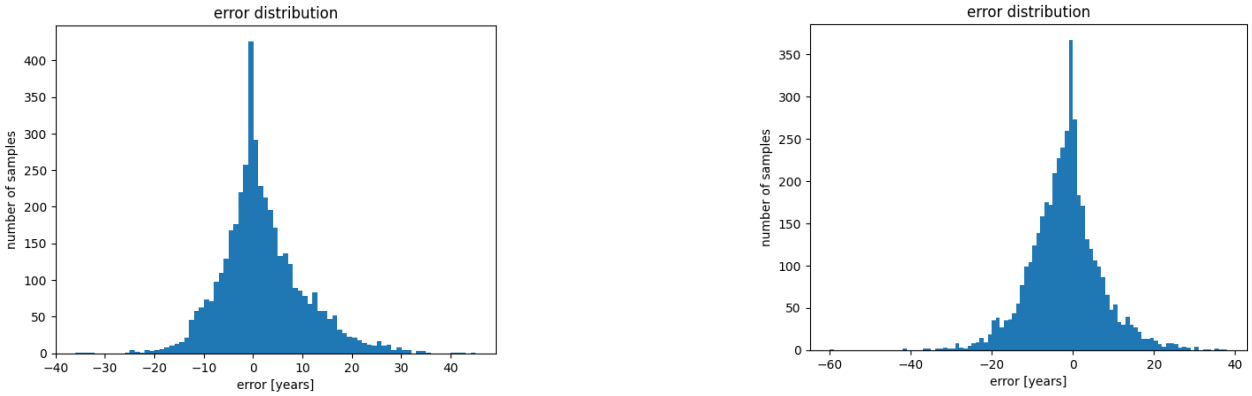


Figure 12: Error distribution for the greyscale and color datasets with image size 100x100

However we found that this approach of evening out the distribution of the dataset caused overfitting in the models, as shown in the MAE plots in Figure 9, 11. In Figure 13 there are MAE plots for 3 additional models trained on augmented datasets - (1) with train dataset evened out to 800, (2) with train dataset evened out to 1000 and (3) with different augmentations. On all of them we can see the same tendency of the model loss of the test set resembling a horizontal line, which means overfitting of the model. From now on we used only the original images with only underfitting the classes to a 1000 images per class (only 2 classes were underfitted) and dividing them proportionally 9/10 to train set and 1/10 to test set. The new distributions are shown in Figure 14.

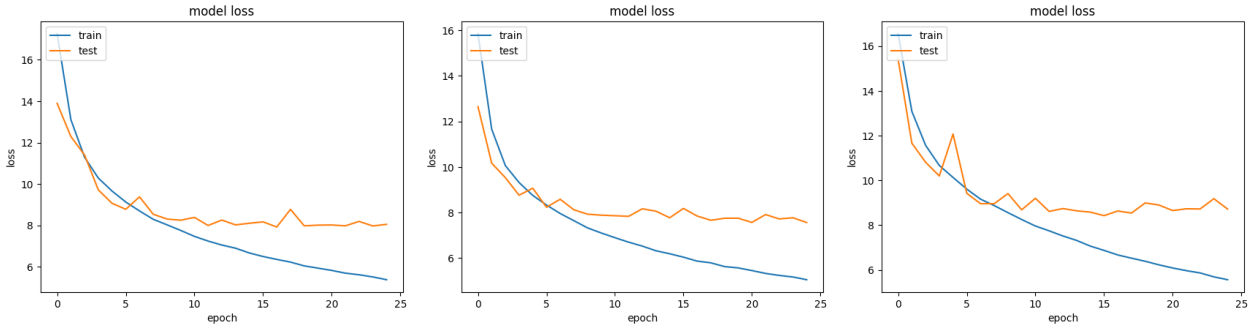


Figure 13: Model loss (MAE) for the greyscale datasets with train dataset - (1) evened out to 800, (2) evened out to 1000 and (3) with different augmentations

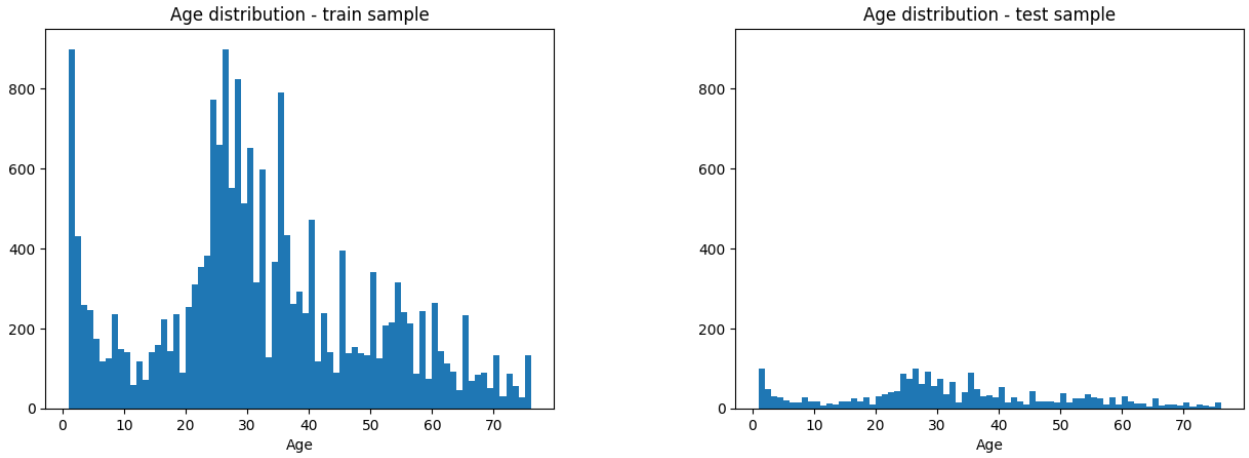


Figure 14: Dataset distribution

One other way to 'fix' the uneven dataset is to add weights to the classes. The weights for different ages were computed using the function compute_class_weight() from sklearn.utils.class_weight library with the class_weight parameter set to 'balanced'. To test this approach we compared two models trained on the same dataset mentioned above and the with the same model structure - shown in Figure 4. The augmentations were applied directly to the model and these were: random horizonatal flip and random rotation in the range [-20% * 2pi, 20% * 2pi]. The patience of the early stopping callback was set to 10 epochs. In Figure 15 we can see that for the no weighted model the error distribution is much higher around 0 than it is for the weighted model. In Figure 16 we see that the number of errors for the unrepresented classes is a little bit smaller for the weighted model, overall the unweighted model made less misclassifications with error over 10 years. The MAE and MAPE (Mean Absolute Percentage Error) for the not weighted model are equal to 6.235 and 0.306 respectively and for the weighted model 6.987 and 0.406. Thus we discard the weighted classes in the model and use the dataset as it is without evening out the distribution.

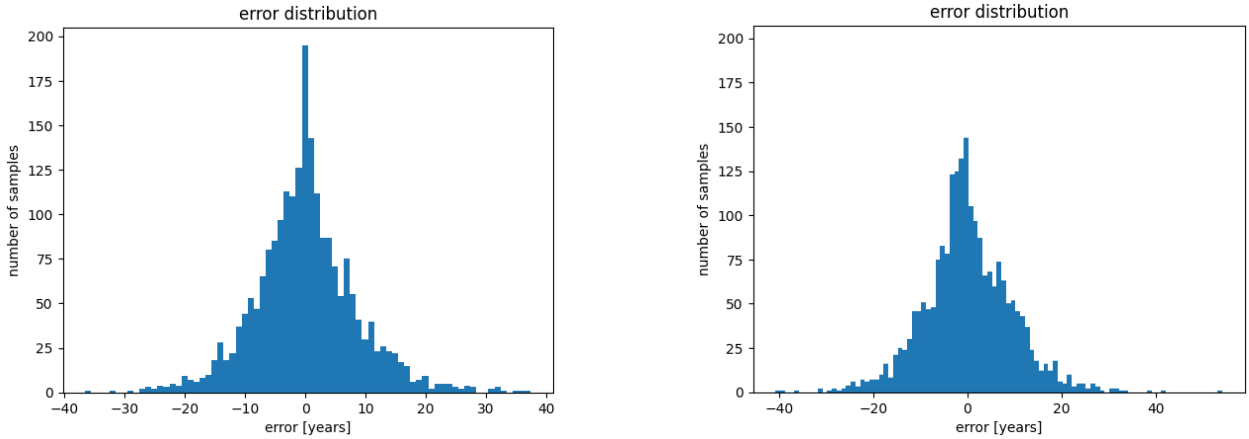


Figure 15: Error distribution for weighted and not weighted models

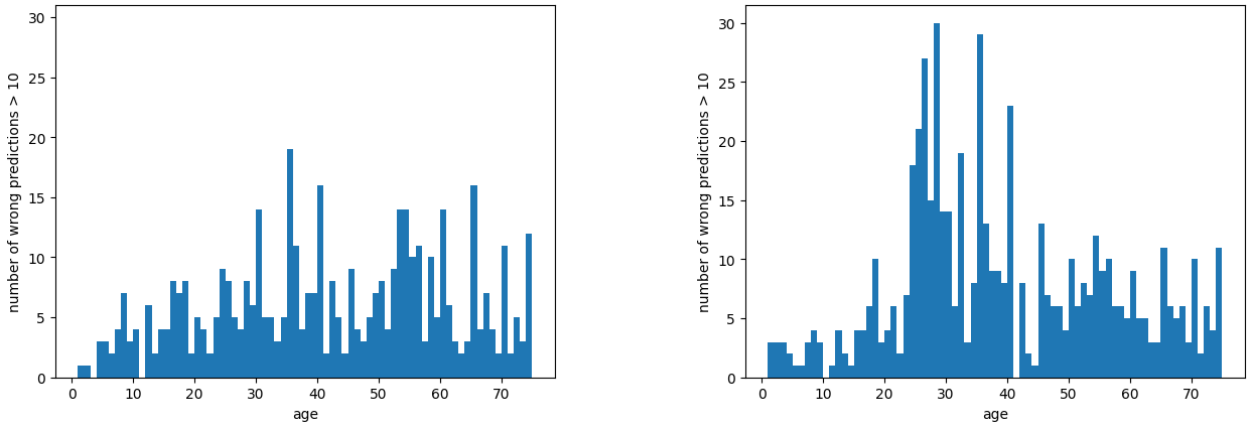


Figure 16: Number of wrong predictions with difference of minimum 10 for weighted and not weighted models

For additional augmentations we added some additional layers to the model like adding random brightness, translations and zoom - Figure 17. However this made the trained model have the MAE of 14 and always predict the age to be around 29 years old - we can see that in the graphs in Figure 18. The error distribution graph looks like our test sample bu shifted to the left by about 30 years.

```
model.add(RandomFlip("horizontal"))
model.add(RandomRotation(0.4))
model.add(RandomTranslation(height_factor=0.1, width_factor=0.1, fill_mode="nearest"))
model.add(RandomZoom(((-0.3, 0.1))))
model.add(RandomContrast(0.4))
model.add(RandomBrightness(0.3))
```

Figure 17: Augmentations layer added to the model

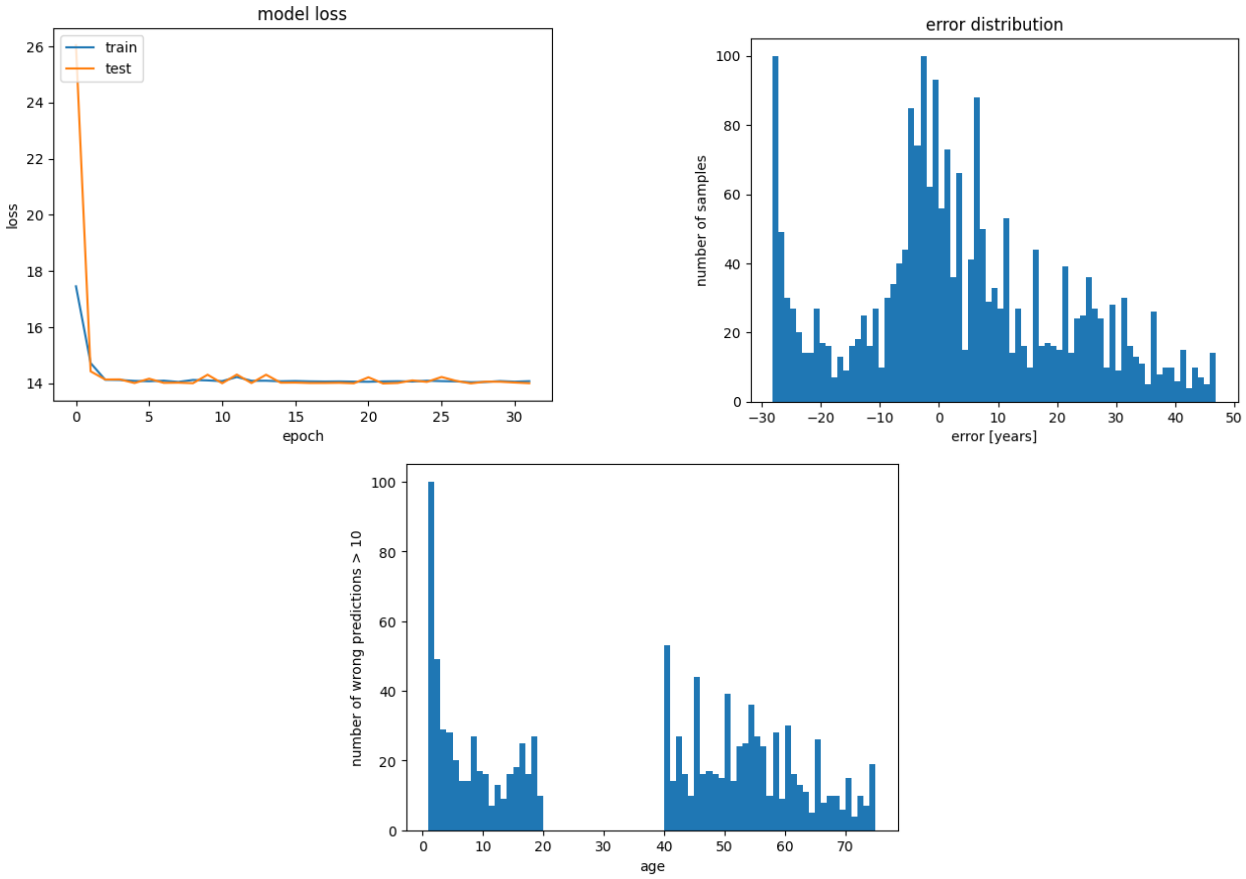


Figure 18: Statistics of the augmented model

The augmentations that seemed to work the best are shown in Figure 19. In Figure 20 we see the model loss and error distribution for this model. The final MAE for the model is 6.358 and MAPE 0.307, which is somewhat worse that the model with less augmentations, however we decided that it was important to leave them as the real world data will vary from what we have in the test sample.

```
model.add(RandomFlip("horizontal"))
model.add(RandomRotation(0.2))
model.add(RandomZoom(((-0.1, 00))))
model.add(RandomContrast(0.1))
```

Figure 19: Remaining augmentation layers on the model

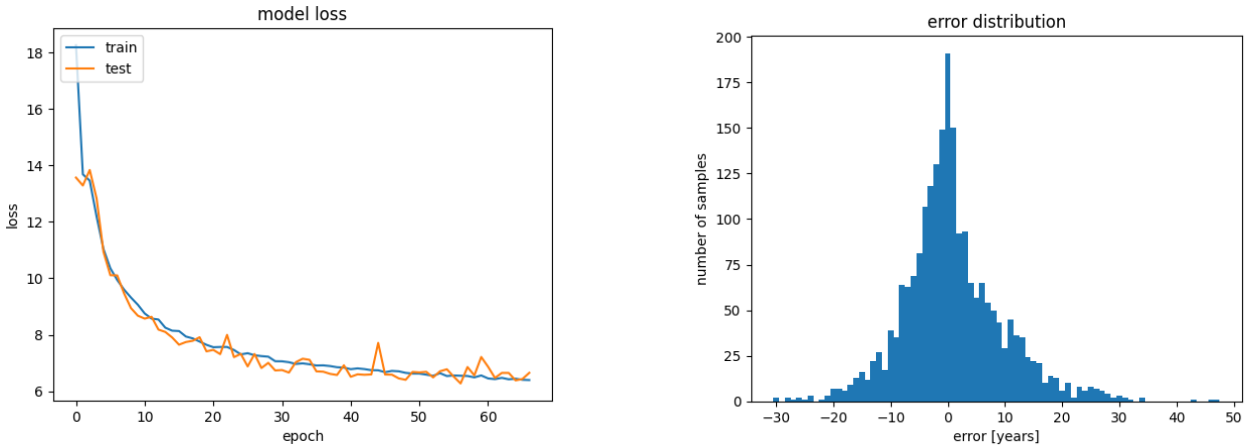


Figure 20: Model loss and error distribution for the model

Next we added an augmentation layer with Gaussian noise with standard deviation of 0.1, but that seemed to significantly increase the MAE to above 7 as shown in Figure 21. Therefore for now we left the model without additional noise added to the augmentations of the images.

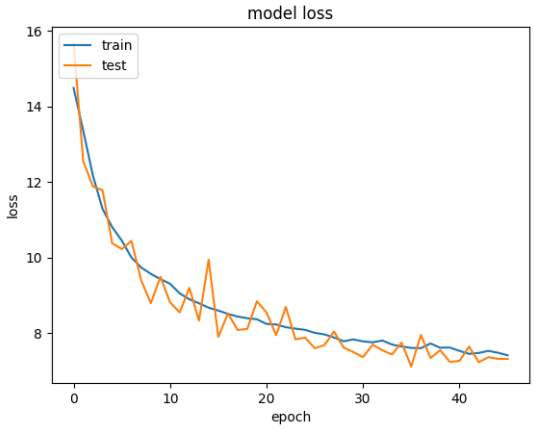


Figure 21: Model loss for the model with added Gaussian noise

3.4 Model structure

To improve the model from Figure 4 additional layers were added:
- Convolutional input layer, 256 feature maps with a size of 3×3, a rectifier activation function
- Max Pool layer
- Fully connected layer with 128 units and a rectifier activation function
- Fully connected layer with 256 units and a rectifier activation function

The Conv2D layers are added in order of increasing feature maps. (Adrian Rosebrock, 2018) The result was improved MAE to 6.081 and MAPE to 0.256. The model loss and error distribution are shown in Figure 21.

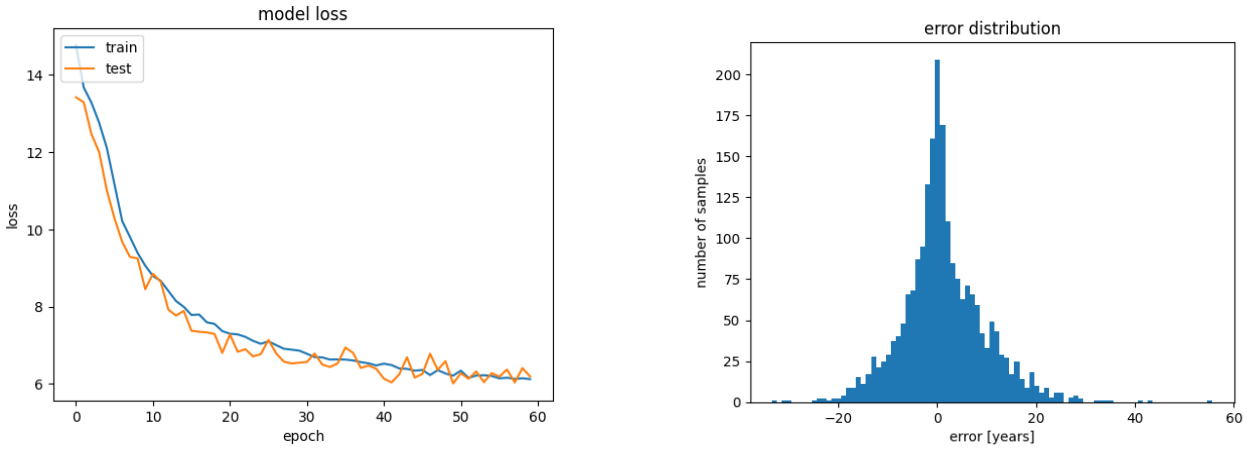


Figure 22: The model loss and error distribution for model with some additional layers

Following that we tried adding even more layers to the model, namely: - Convolutional input layer, 64 feature maps with a size of 3×3, a rectifier activation function
- Convolutional input layer, 256 feature maps with a size of 3×3, a rectifier activation function
- 2x Fully connected layer with 256 units and a rectifier activation function
- Fully connected layer with 128 units and a rectifier activation function
- 2x Fully connected layer with 64 units and a rectifier activation function
The new model is shown in Figure 23.

```
model = Sequential()
model.add(Resizing(100,100))
model.add(Rescaling(1./255))
model.add(RandomFlip("horizontal"))
model.add(RandomRotation(0.2))
model.add(RandomZoom(((-0.1, 00))))
model.add(RandomContrast(0.1))

model.add(Conv2D(32, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(128, (3,3), activation='relu'))
model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(256, (3,3), activation='relu'))
model.add(Conv2D(256, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.1))
model.add(Dense(1, activation='relu'))
```

Figure 23: Model structure

However this has caused the trained model to drop in performance coompared to the previous model. In Figure 24 we can see the model loss and error distribution. Both MAE (6.351) and MAPE (0.309) are close to the ones from the original model. We decided to use the previous model structure for the final model.

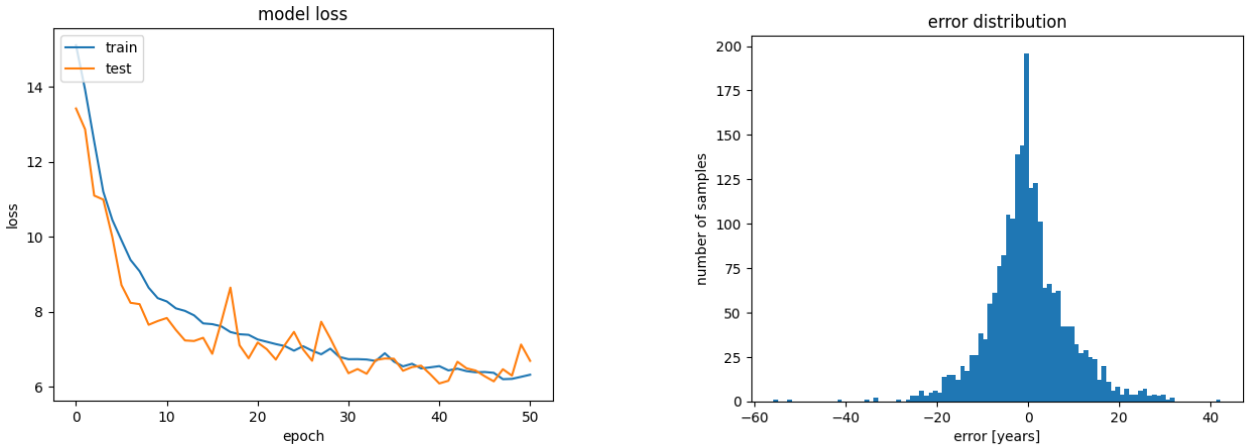


Figure 24: The model loss and error distribution for model with many additional layers

3.5 Final Model

In our final model we added back the augmentation layer with Gaussian noise - this time with the standard deviation equal to 0.02, as not to interfere with the image too much, but still introduce some random noise to the train sample. Introducing noise to the training sample should help it work on the noisy real world data. (Sovit Ranjan RathSovit Ranjan Rath, 2020) The model structure used is shown in Figure 25. The trained model had the MAE equal to 6.061 and MAPE equal 0.259. In Figure 27 we can see that the the average error for age prediction is less than 10 for most classes. In Figure 28 there are all of the misclassified images with error bigger than 30 years. Out of over 2000 test images there were only 10 with that big of an error.

```
model = Sequential()

model.add(Resizing(100,100))
model.add(Rescaling(1./255))

model.add(RandomFlip("horizontal"))
model.add(RandomRotation(0.2))
model.add(RandomZoom(((-0.1, 00))))
model.add(RandomContrast(0.1))
model.add(GaussianNoise(0.02))

model.add(Conv2D(32, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(64, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(128, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Conv2D(256, (3,3), activation='relu'))
model.add(MaxPooling2D())

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.1))

model.add(Dense(1, activation='relu'))
```

Figure 25: Structure of final model

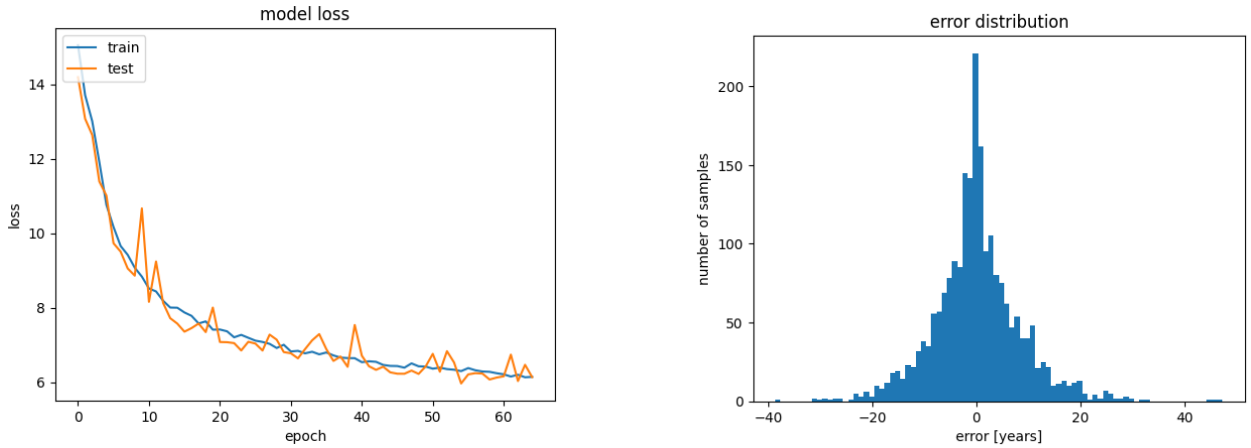


Figure 26: Model loss and error distribution for the final model

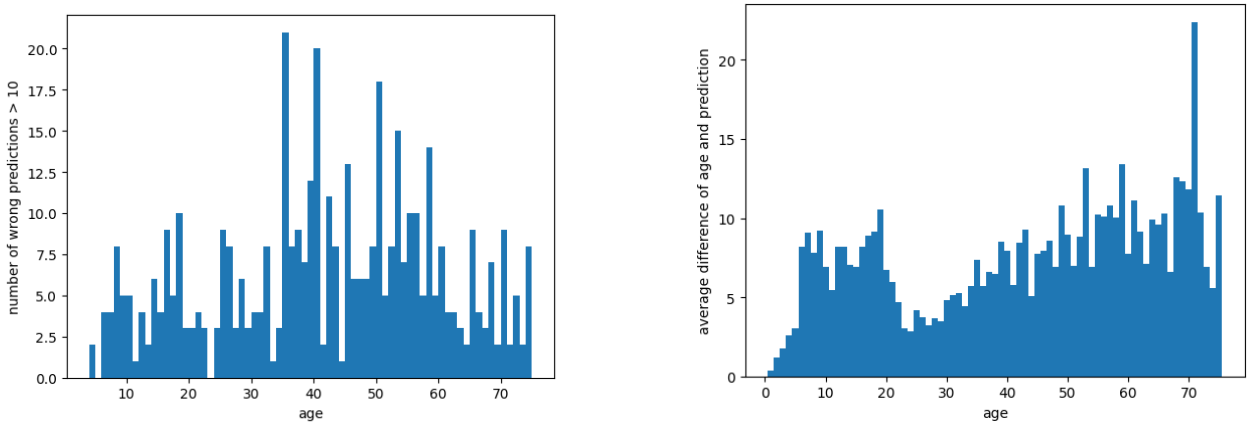


Figure 27: Number of images misclassified with the error ¿ 10 and the average eroor for each class

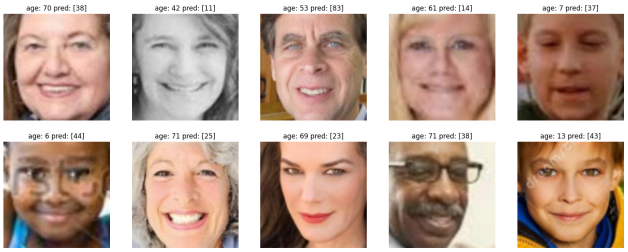


Figure 28: Misclassified photos with over 30 year difference

4 Transfer learning

4.1 What is transfer learning?

Transfer learning is an approach to machine learning that has its origins in 1976, when Stevo Bozinovski and Ante Fulgosi published a paper dealing with this topic for the first time. In their work (Zhuang F. et al., 2020), authors show that the main assumption of this technique is to use the results obtained in one study

and use these results to analyze another, similar problem. What is more, this approach makes it possible to obtain very satisfactory results, while reducing the necessary volume of data, compared to the situation when the researcher would like to create a model from scratch. This is especially valuable in many real-life situations, when collecting a large set of data is associated with high costs, or even impossible due to the specificity of the analyzed problem. Also this approach speeds up the entire modeling process. All this meant that transfer learning quickly gained great popularity and is constantly developing.

4.2 Models used in project

In our work, we decided to focus on implementing two types of neural networks - ResNet and MobileNetV3. We chose the residual neural network (ResNet) because it is the most cited neural network of the 21st century (Schmidhuber J., 2021). In the article "MobileNetV3 for Image Classification" (Qian S. et al., 2021), the authors present results showing that, considering the fact that MobileNetV3 networks are lightweight neural networks, they are characterized by high efficiency compared to large neural networks. Therefore, we decided to choose the second group of neural networks.

4.3 ResNet

4.3.1 ResNet - short description

ResNet is an artificial neural network (ANN) that is an open or frameless variant of HighwayNet. HighwayNet was the first very deep coupled neural network with hundreds of layers to work with. ResNet omits some layers and typical models are implemented with a two- or three-layer omission. There are two main reasons for adding skip connections: to avoid the problem of vanishing gradients, which leads to easier optimization of neural networks, where gating mechanisms facilitate the flow of information through multiple layers ("information highways") or mitigate the problem of degradation (accuracy saturation); where adding more layers to a sufficiently deep model leads to a larger learning error. What distinguishes the ResNet network is the ratio of the number of trained layers to performance, because as it turns out, it is possible to train even thousands of layers while maintaining satisfactory performance. Especially the area covered by our project, namely facial recognition, benefited from the breakthrough brought by the discussed network.

4.3.2 ResNet - models used in project

We decided to choose a network consisting of 50 layers (ResNet50). Discussing the network architecture, it consists of 48 convolutional layers, one MaxPool layer and one average pool layer. The network architecture accordingly consists of (ResNet-50: The Basics and a Quick Tutorial, https://datagen.tech/guides/computer-vision/resnet-50/):
- A 7×7 kernel convolution alongside 64 other kernels with a 2-sized stride.
- A max pooling layer with a 2-sized stride.
- 9 more layers—3×3.64 kernel convolution, another with 1×1.64 kernels, and a third with 1×1.256 kernels. These 3 layers are repeated 3 times.
- 12 more layers with 1×1,128 kernels, 3×3,128 kernels, and 1×1,512 kernels, iterated 4 times.
- 18 more layers with 1×1.256 cores, and 2 cores 3×3.256 and 1×1.1024, iterated 6 times.
- 9 more layers with 1×1.512 cores, 3×3.512 cores, and 1×1.2048 cores iterated 3 times.
- Average pooling, followed by a fully connected layer with 1000 nodes, using the softmax activation function (not counted as a layer).

According to the article "Detailed Guide to Understand and Implement ResNets" by Ankit Sachan the architecture can be divided into 4 parts. The image dimensions and channel width are accepted as the network's input. First thing which is performed is initial splices and max pools using 7×7 and 3×3 kernel sizes, respectively. Next, the first stage of the network begins. It consists of 3 blocks containing 3 layers each. In first two layers the convolution operation is performed using kernels of the size 64, whereas in the third layer using kernel of the size 128. The curved arrows refer to the identity connection. The dashed connected arrow means that the convolution operation in the residual block is performed in step two, hence the input size will be halved in height and width, but the width of the channel will be doubled. Going from one stage to

another means doubling the channel width and halving the input signal size.

Deeper networks like ResNet50, ResNet102 etc. are using the bottleneck design. 3 layers are stacked on top of each other for every residual function F. The three layers are 1×1, 3×3, 1×1 turns. The 1×1 weave layers are responsible for reducing and then restoring the dimensions. The 3×3 tier remains a bottleneck with smaller I/O dimensions.

At the end, the network has an average pooling layer followed by a fully connected layer of 1000 neurons (ImageNet class output).

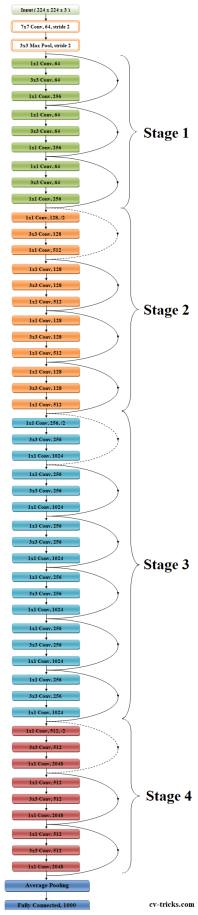


Figure 29: Architecture of ResNet50

In order to implement the model for our needs, we loaded the base model ResNet50. Then we set the trainable = False parameter to avoid a situation where in subsequent iterations we would lose the information contained in the base model. In addition, the include_top = False parameter, so that it is possible to add new layers to the top, which we will train using our dataset. Before starting modelling, it is necessary to process the data. This is done using the preprocess_input function, which will convert the input images from RGB to BGR, then will zero-center each color channel with respect to the ImageNet dataset, without scaling.

The data used for modeling came from the previously described UTKFace dataset. Erroneous and incorrectly classified photos were eliminated from it, such as: significantly incorrect age, no face in the photo. Data were augmented in the form of a horizontal random flip and rotation by a random amount in the range [-20% * 2pi, 20% * 2pi]. Other modifications to the dataset were also made in some iterations, but this will be described in the following steps. Validation split is set to 0.2.

An early modeling stop criterion has been added when a monitored metric (validation loss) has stopped improving, in our case decreasing, with patience set to 12.

The process of model improvement looked as follows:

1. First, we added a Dropout layer (with probability 0.5) before the classification layer, for regularization and a layer which is responsible for Global Average Pooling. We also add a Dense layer which specifies the dimensionality of the output space. This specification proved to be not very effective, with a validation loss of 13.

2. Changed the probability in a Dropout layer to 0.2. This allowed us to lower the validation loss to around 8.7.

3. In the next iteration of the model, it was decided to add more layers, Dense layers with different units value (256, 128 and 64), BatchNormalization layers, which are used to make training faster and more stable through normalization of the layers' inputs by re-centering and re -scaling, and also more Dropout layers with probability set to 0.1 to avoid overfitting. We used those layers in this and next models. The specification is presented below. It resulted in a validation loss of around 5.4.

```
base_model = ResNet50(
    weights='imagenet',  # Load weights pre-trained on ImageNet.
    input_shape=(224, 224, 3),
    include_top=False)
base_model.trainable = False

#preprocessing layers + augmentations
inputs = Input(shape=(224, 224, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)

x = base_model(x, training=False)

x = GlobalAveragePooling2D()(x)
x = BatchNormalization()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.1)(x)

x = BatchNormalization()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.1)(x)

x = BatchNormalization()(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.1)(x)
outputs = Dense(1)(x)

model = Model(inputs, outputs)
model.compile(optimizer='adam', loss='mae', metrics=['accuracy'])
```

Figure 30: Final structure of model

4. The next step was to modify the dataset. As described in the chapter on the analysis of the UTKFace set, there are significantly fewer people aged over 75, so it was decided to limit the set used for modeling to people from 1 to 75 years of age. This change increased the validation loss to around 5.6.

5. In this iteration, the dataset was modified so that classes greater than 500 were limited to 500 photos. The obtained validation loss was 6.8, so it did not bring any improvement, also in the next iteration, this modification was abandoned and the whole dataset was reconsidered.

6. It was decided to add equalizing weights for each class because the data set was characterized by the fact that the discrepancy in the number of classes was significant. To be more precise, there were significantly more people aged 20-40 and 1 year than, for example, at the age at the upper limit of the

analyzed set (75 years). The introduction of weights caused that the impact of uneven distribution was leveled. A validation loss of around 6 has been achieved.

As it can be seen, the value from the last iteration is not the lowest value, but this is most likely due to the fact that the validation dataset was better suited to the model from point 3 or 4 because these models learned on sets with a similar distribution as the validation set. In order to decide which of the models will be the most optimal, their effectiveness was analyzed - how often and to what extent they incorrectly predict age. The choice was made between the model from the fourth and sixth iterations.

For both models, an analysis was made for how many photos the prediction error is over 10 years. The results are shown in histograms and misclassified photos are also presented.

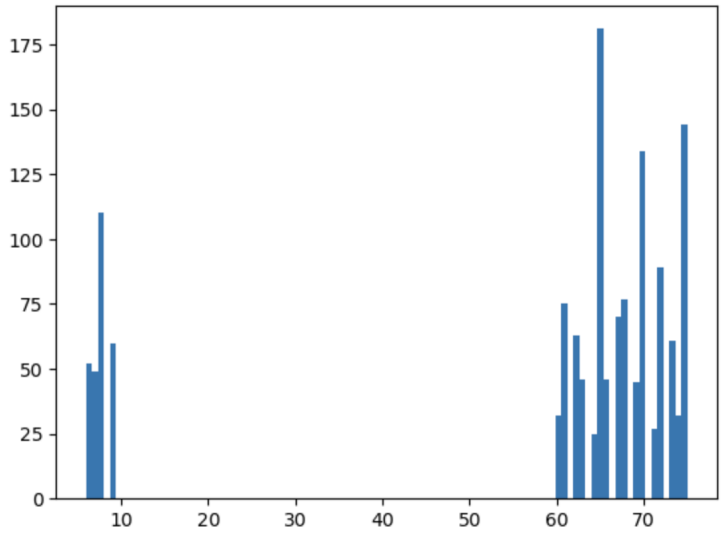


Figure 31: Prediction error over 10 years - model 4

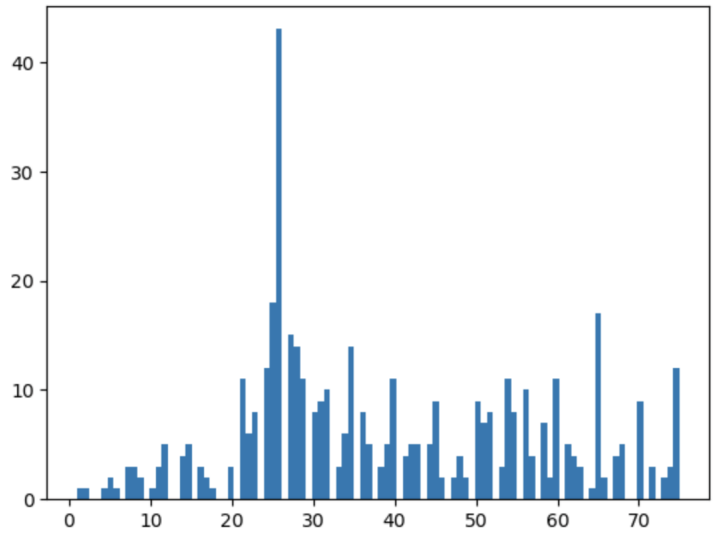


Figure 32: Prediction error over 10 years - model 6

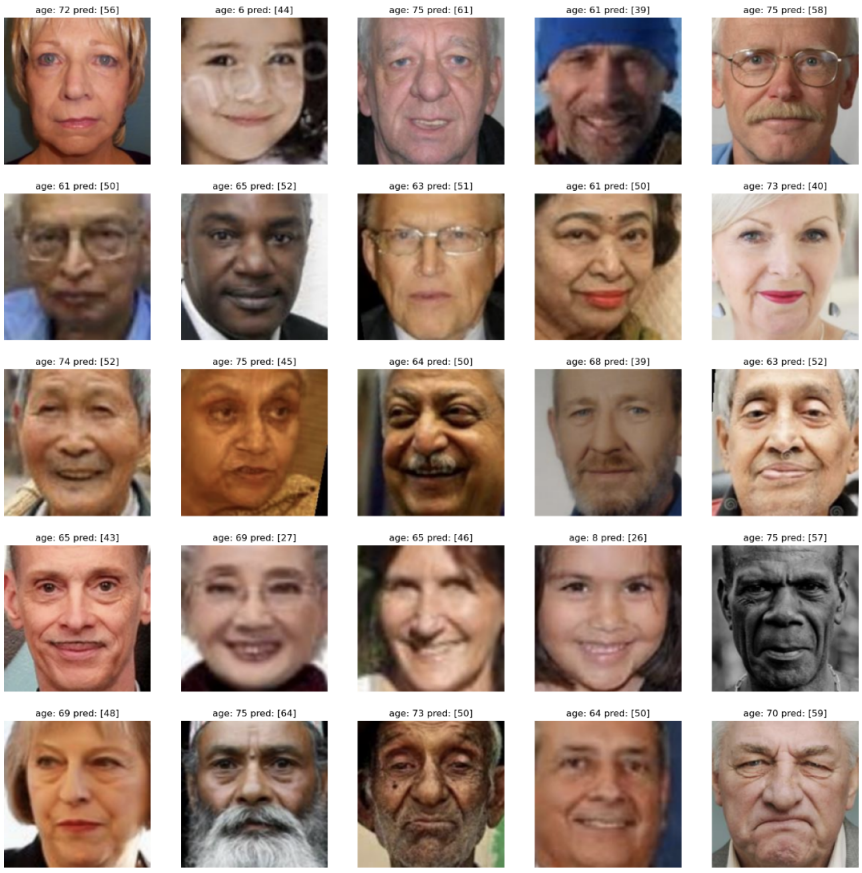


Figure 33: Misclassified photos - model 4

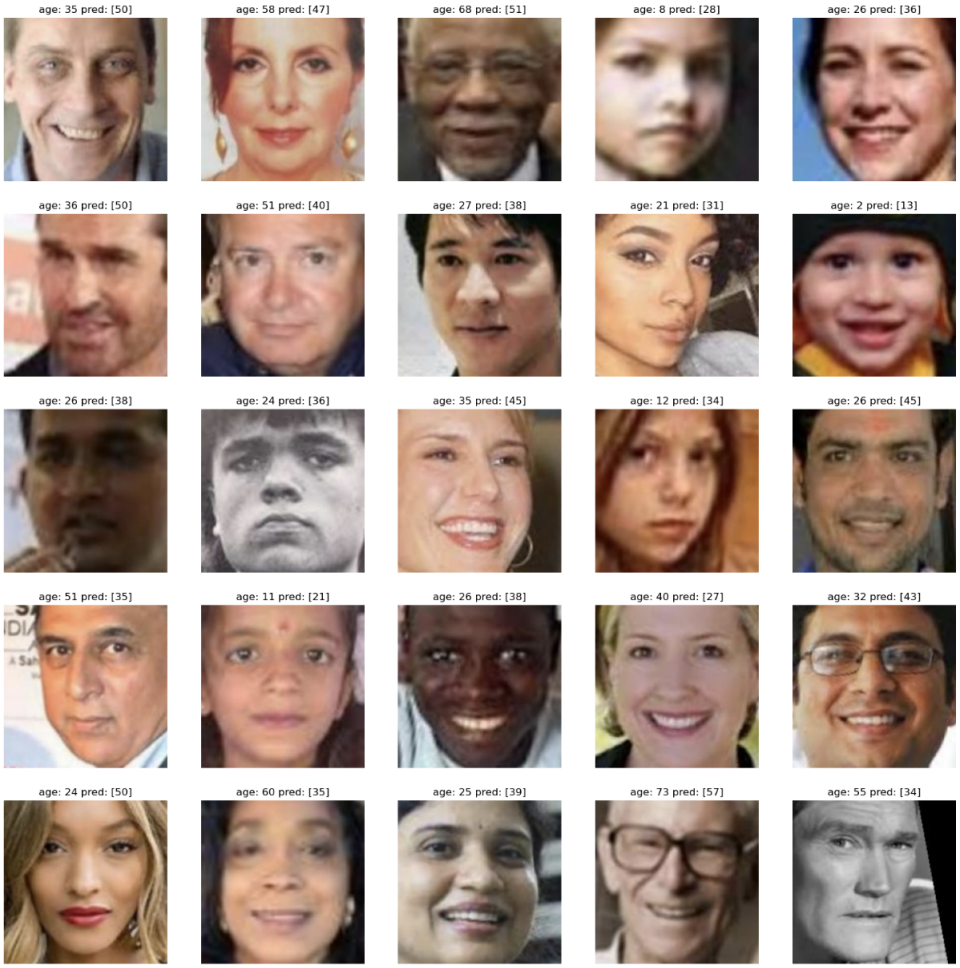


Figure 34: Misclassified photos - model 6

It can be seen that the histograms of the prediction error over 10 years differ significantly between the fourth model and the sixth model. In the case of the fourth model, errors are made more often for younger and older people, while for middle-aged people (which dominates the test set) they do not make false predictions for more than 10 years. This is because this model was trained on a predominately middle-aged set and was trained without using weights to even out this set. Therefore, the model performs better for middle-aged people, but has significant errors for younger and older people. In contrast, the sixth model, for each age group, is wrong for a certain percentage of observations, but in the case of young and old people, these errors are less frequent than in the fourth model. Based on these observations, it can be concluded that with a high degree of probability, the sixth model is a more universal model for classifying people in the full age range of 1-75 years.

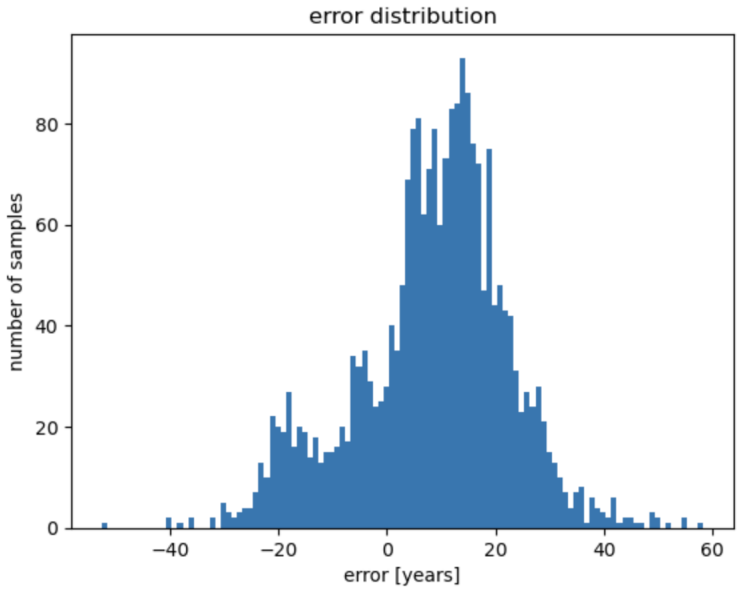


Figure 35: Error distribution in prediction - model 4

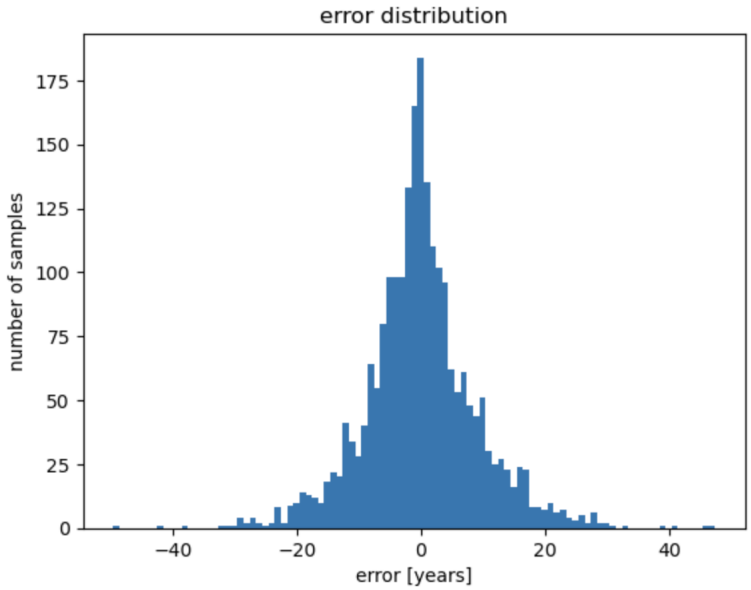


Figure 36: Error distribution in prediction - model 6

As can be seen, the age prediction error distribution using the model from point six is close to the normal distribution with an average of 0, while in the case of the model from point four, the average exceeds 10 years. This means that despite the higher validation loss by about 0.4, the sixth model is on average more effective in predicting the age of people, making a noticeably smaller error. For this reason and the reason mentioned in the paragraph before, it was finally decided to choose a model with the specification from point six.

4.4 MobileNet

4.4.1 MobileNet - short description

MobileNet is a group of models intended for mobile and embedded vision applications. They belong to the group of lightweight deep neural networks, which are based on streamlined architecture that uses depthwise separable convolutions (Howard A. G., et al., 2017). Thanks to the use of depthwise separable convolutions in these networks, it was possible to significantly reduce the number of parameters compared to the network with regular convolutions while maintaining the same depth. Depthwise separable convolutions consist of depthwise convolution and pointwise convolution as shown in the diagram below (Pujara A., 2020).

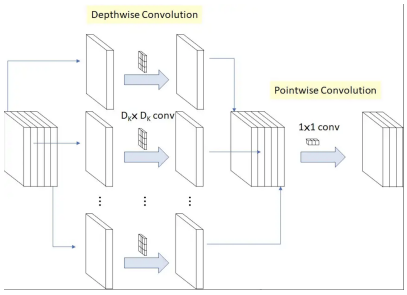


Figure 37: Architecture of MobileNet

Depthwise convolution is a type of convolution where for each input channel we apply a single convolution filter. After the convolution filter is applied to each input channel, convolved outputs are stacked together.

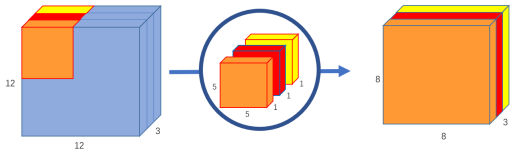


Figure 38: Structure of depthwise convolution

Pointwise convolution is a type of convolution that has a kernel size of 1x1. What it does is combine the features which were created in the first step by a depthwise convolution.

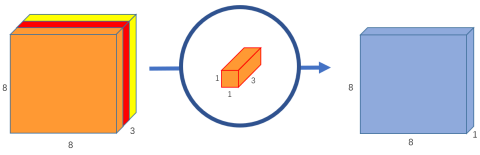


Figure 39: Structure of pointwise convolution

4.4.2 MobileNet - models used in project

MobileNetV3Large is the latest model designed to introduce a new generation of high-performance, high-accuracy neural network models. As research has shown, MobileNetV3 works more effectively than previous versions. The innovation that MobileNetV1 was installed was the use of separable coils in depth. The next version, MobileNetV2, introduced a linear bottleneck and an inverted residual structure. This was dictated by the construction of sandwich structures. This was done using the low rank of the problem. MobileNetV2 supervision of the MnasNet configuration, which introduced lightweight attention modules based on squeeze

and excitation in the bottleneck structure. The MobileNetV3 we implemented is a combination of all these layers, which results in the best models for all applications. (Howard A., 2019)

In order to implement the model for our needs, we loaded the base model MobileNetV3Large. Then we set the trainable = False parameter to avoid a situation where in subsequent iterations we would lose the information contained in the base model. In addition, the include_top = False parameter, so that it is possible to add new layers to the top, which we will train using our dataset. Before starting modelling, it is necessary to process the data. In MobileNetV3 it is done automatically. Pixels are scaled between -1 and 1.

The data used for modeling came from the previously described UTKFace dataset. Erroneous and incorrectly classified photos were eliminated from it, such as: significantly incorrect age, no face in the photo. Data were augmented in the form of a horizontal random flip and rotation by a random amount in the range [-20% * 2pi, 20% * 2pi]. Other modifications to the dataset were also made in some iterations, but this will be described in the following steps. Validation split is set to 0.2.

An early modeling stop criterion has been added when a monitored metric (validation loss) has stopped improving, in our case decreasing, with patience set to 12.

Moving on to the modeling process, after several iterations, a choice was made between two models. They differed in whether or not weights were used to balance the data set. Besides, the specification of both of them was the same and looked like this (the same as in the case of ResNet50, where the structure of the model was described in more detail).

```
base_model = MobileNetV3Large(
    weights='imagenet', # Load weights pre-trained on ImageNet.
    input_shape=(224, 224, 3),
    include_top=False)
base_model.trainable = False

#preprocessing layers + augmentations
inputs = Input(shape=(224, 224, 3))
x = data_augmentation(inputs)

x = base_model(x, training=False)
x = GlobalAveragePooling2D()(x)
x = BatchNormalization()(x)
x = Dense(256, activation='relu')(x)
x = Dropout(0.1)(x)
x = BatchNormalization()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.1)(x)
x = BatchNormalization()(x)
x = Dense(64, activation='relu')(x)
x = Dropout(0.1)(x)
outputs = Dense(1)(x)

model = Model(inputs, outputs)

model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
```

Figure 40: Final structure of model

In order to compare the models, a histogram was prepared showing how many photos the prediction error is over 10 years. Some misclassified photos are also presented. In addition, a chart with the age prediction error distribution is presented.

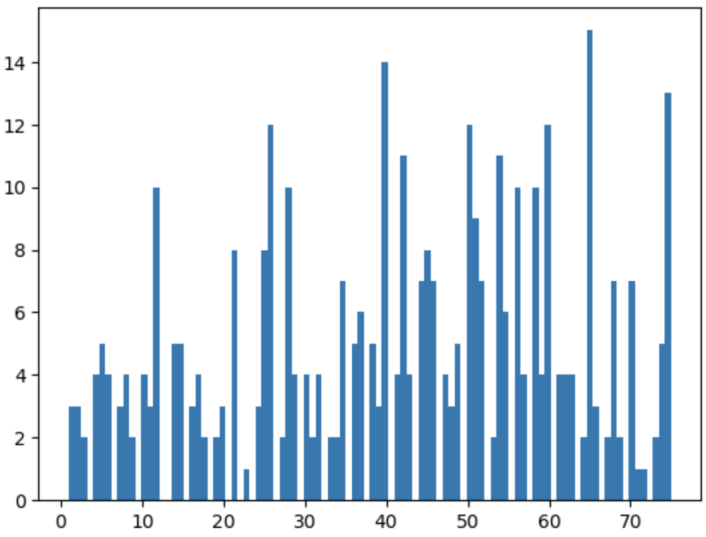


Figure 41: Prediction error over 10 years - model without weights

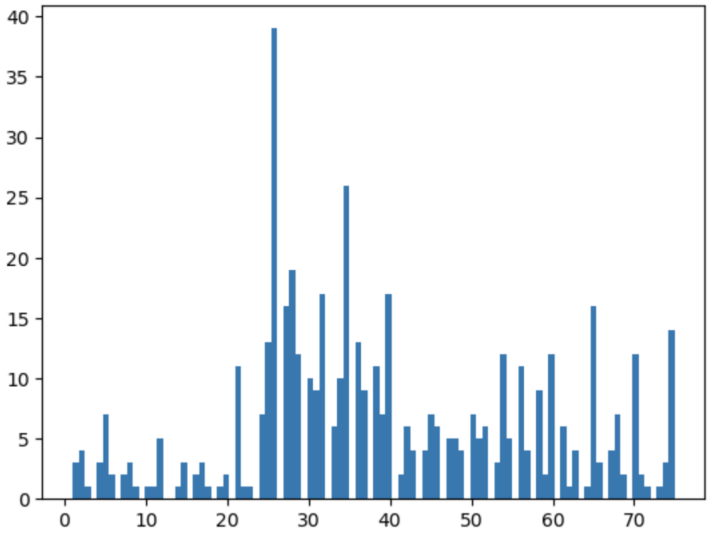


Figure 42: Prediction error over 10 years - model with weights

Based on the histogram showing for how many photos the prediction error is over 10 years, it can be seen that the model without weights is more evenly wrong when considering the entire age range, while the model with weights is wrong proportionally more often for age groups that are more in the test set.

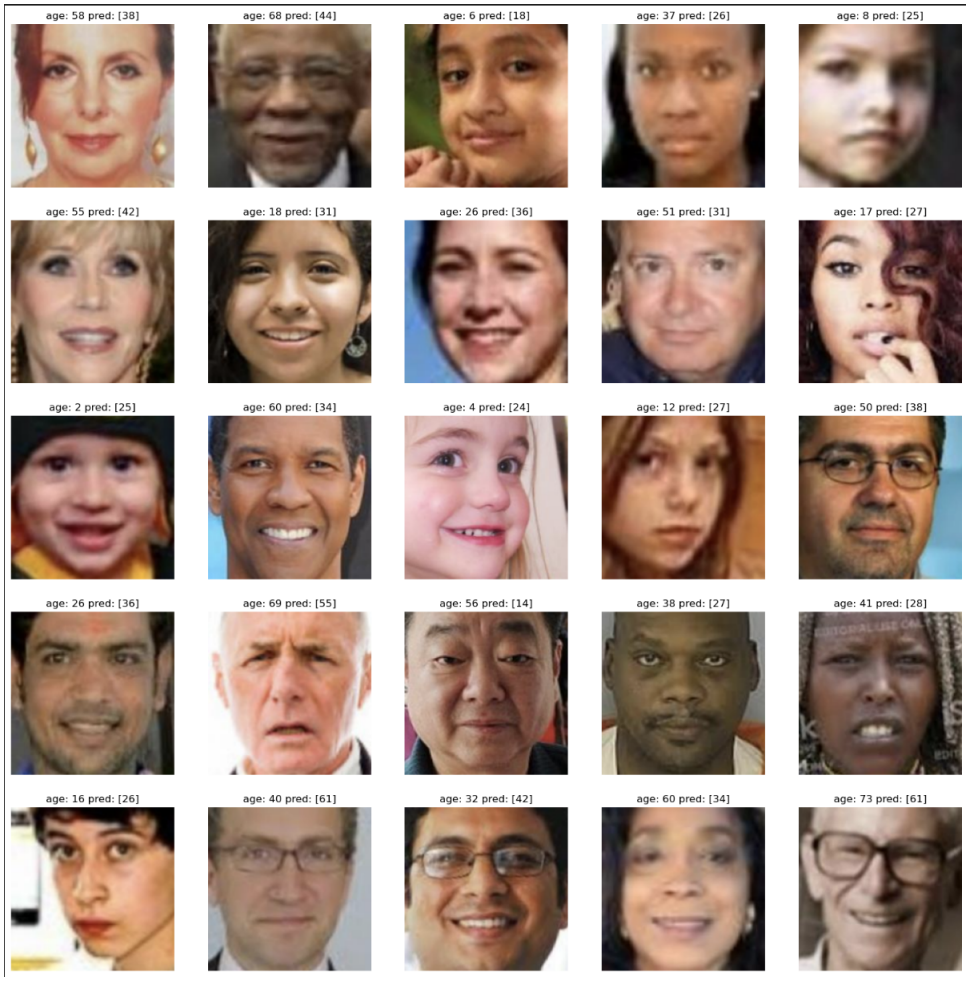


Figure 43: Misclassified photos - model without weights

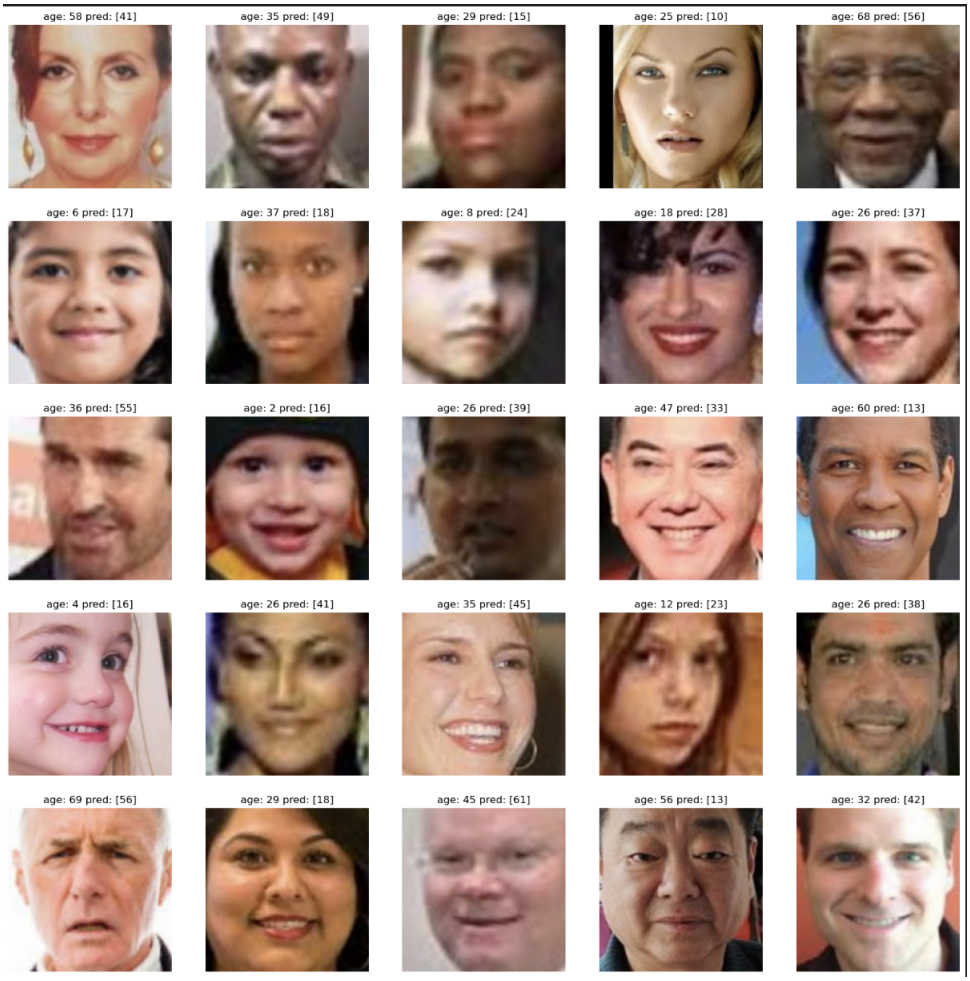


Figure 44: Misclassified photos - model with weights

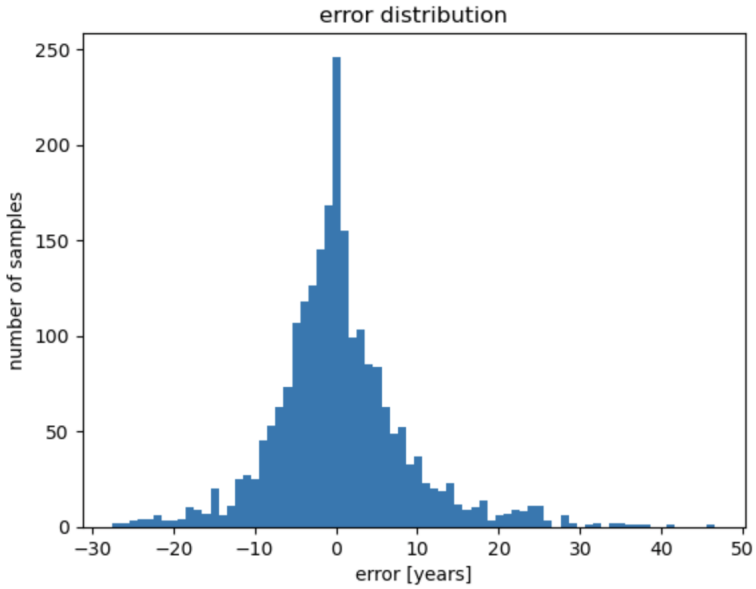


Figure 45: Error distribution in prediction - model without weights

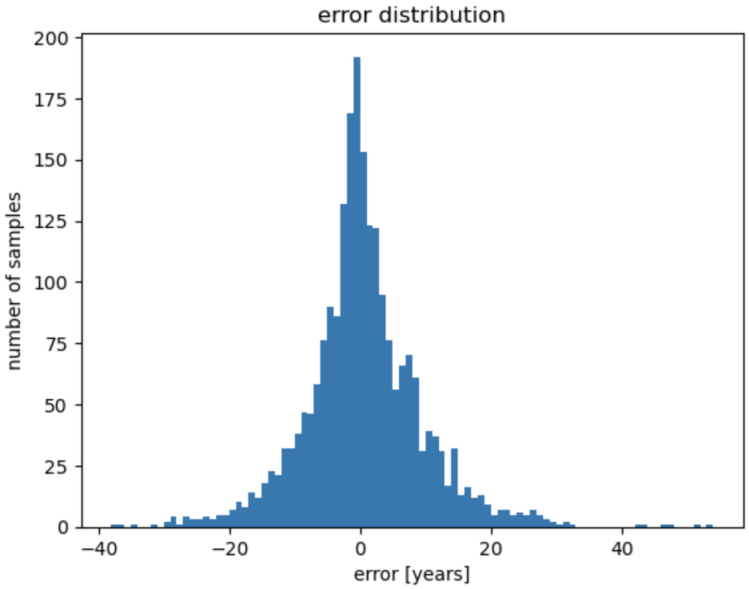


Figure 46: Error distribution in prediction - model with weights

For both models, the error distributions are close to each other, with the mean close to zero.

In order to choose one of the two models, we decided to use two error measures - MAE (Mean Absolute Error) and MAPE (Mean Absolute Percentage Error). MAE measures the mean error between the actual value and the model estimate, while MAPE measures the mean absolute percentage deviation. The MAE for the model without weights was 5.74, and for the model with weights it was 6.43. MAPE for the model without weights was 30.89%, and for the model with weights it was 36.63%. In the case of both measures, the lower the score, the better, so we decided to choose a model without weights.

4.5 Comparisons

The next step was to choose one model from the two - a model based on ResNet50 or a model based on MobileNetV3Large. The selection was made on the basis of MAE and MAPE measures. The MAE for ResNet50 is 6.04 and for MobileNetV3Large it is 5.74. MAPE for ResNet50 is 31.06% and for MobileNetV3Large it is 30.89%. Due to the fact that both measures are lower for the model based on MobileNetV3Large, we decided to choose this model as our final model based on Transfer Learning.

5 CNN vs Transfer Learning

The last step in choosing the final model for our project was to decide whether the CNN model or the model based on Transfer Learning would be better. Again, we will make our decision based on the MAE and MAPE values. The MAE for the CNN model is 6.06, and for the model based on Transfer Learning it is 5.74. MAPE for the CNN model is 25.94%, and for the model based on Transfer Learning it is 30.89%. The MAE value is lower for the model based on Transfer Learning, while MAPE is lower for the CNN model. We decided to choose a model with a lower MAE value. We did this because the MAPE measure has some problems with lower volume data. More precisely, the point is that, for example, if the actual age is 1 and the predicted age is 2, the error is 100%, while when the actual age is 50 and the predicted age is 51, the error is only 2%, even though in both cases missed by 1 year. Therefore, we decided that in our case it would be more optimal to choose a model with a lower MAE error, which measures the average absolute error between the real and predicted values. To sum up, we decided to choose a model based on Transfer Learning, and more specifically using the MobileNetV3Large base model.

6 Software description

6.1 Introduction

The project was implemented in Django framework, which is native to Python language. The main concept of Django is processing HTTP requests of a user. This framework is extensive, quite often used in large scale machine learning models which serve responses based on various algorithms. In our case, the application is an image-processing tool that, based on graphical input, detects human face and predicts both age and gender.

The application is of the form of a self-hosted web server. After launching the website it provides the user with all needed functionalities and a simple GUI providing navigation and utility buttons. The correct usage of this application will be described in the chapters that follow.

As a requirement for this project we were to produce software for three different use cases, namely:

- **Live stream from user's built in camera** - the application collects frames provided by the user's camera, processes them and outputs on the screen, close to real time.
- **Processing of a video selected by user** - the user supplies a single video to the application. The program analyzes each frame and outputs predictions, saving processed video a hard drive. The video can be played via GUI.
- **Processing of multiple images** - the user supplies multiple images contained in a directory. The application processed each image and outputs predictions, saving the images to the user's hard drive. The images can be accessed via GUI.

6.2 Application prerequisites

In order for the user to access the application's functionalities, there are a few external packages or libraries necessary to download beforehand, of course, including Python language interpreter and `django` web framework. The full list is presented below.

Prerequisites:

- Python programming language, version 3.9 or higher
- `django` framework, version 4.1
- `OpenCV` package
- `imutils` package
- `tensorflow` package
- `numpy` package

6.3 Graphical user interface

The application is divided into three main use flows corresponding to the scenarios introduced in Section 6.1. Apart from that, the user is greeted with a home screen that allows navigation to specific functionalities (Fig. 47)

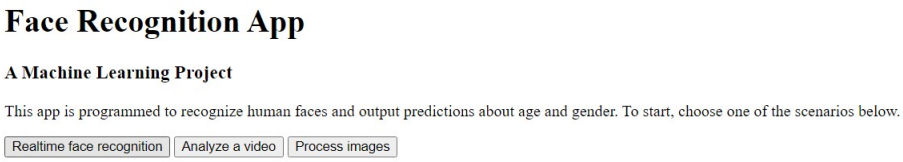


Figure 47: Home screen - user starting place in our application.

The website views are preceded with an informational header. Moreover, in each case there is a *"Go back"* button that allows the user to navigate back to the home screen (Fig. 48).

Figure 48: An example of a "Go back" button.

In general, all actions of the program are performed by activating HTML buttons. Those will be described in detail in future sections.

6.4 Algorithm

6.4.1 Model choice

Three models were used for the implementation of this project. The first one was a pre-trained, ready face recognition model in the Caffe deep learning framework based on ResNet-50 architecture. The predictions from this model allow for determining, with a certain confidence level, the boundary box in which a human face is contained.

The next models were used in order to predict the gender and age of a human, based on the frame extracted using the Caffe one.

The second model was the one that predicts the gender of a human. It is CNN type and we did not put much attention into it, since the results were not satisfactory. Nevertheless, we have decided not to remove it and treat it like an additional feature of the application.

The third, age model, was prepared and described in earlier chapters. In general, one configuration of models is used throughout the whole application; the user can not change them via the GUI.

6.4.2 Frame processing algorithm

The vast majority of code which is of importance is contained in `views.py` file located in *face_rec_app* folder. The function of interest is called `process_frame()` and is located at the very bottom of the file. It is used throughout all implementations in the project, that is for live stream, video and image processing. The rest of the files

It takes a frame (any) as an argument and processes it in the following way:

1. Compute the height and width of the frame.
2. Convert it to 300x300 blob, then pass it to the face recognition model.
3. For every detection, if the confidence is satisfactory (in our case we set it so that we must be 70% confident that the detection is valid), compute the coordinates of the rectangle box containing a human face.
4. Using the coordinates, we extract the part of the frame containing the rectangle with the face (at the same time normalising it so that it is close to a square; see next section) and resize it so that we obtain two images of size 224x224 (for age model) and 100x100 (for gender model).
5. Expand the dimensions of prepared images so that they can be passed to the models.
6. Pass the face images to the models - gender and age.
7. Using OpenCV library, display the bounding box and the predictions around the face contained in the frame.
8. Return processed frame.

After all processing is finished, the files are saved to a specified folder. The user may access them and display either on the website or in the local photo/video viewer. For images, it is the "images_out" folder and for the video there is the "video_out" folder. Both are located in the project directory "face_rec_project".

6.4.3 Resizing and cropping of image

Due to the boundary box often being in the shape of a rectangle, it was necessary to normalize it for the results to be accurate. While preparing the solution we noticed that the boundary box almost always contain a large part of human forehead and some part of the chin, making the box larger in the vertical direction. Therefore, we are normalizing the boundary box by:

1. Computing the difference between the height and the width of the face image, depending on what value is larger.
2. Applying 80% and 20% of that value to the height end coordinates (shrinking the height for when height is larger) and applying 50% of that value to the end coordinates of the width (shrinking the width for when the width is larger).

7 Use case scenarios

7.1 Starting the application

Once user satisfies the prerequisites of this application, the very first step would be to open up their console. Once in the terminal, one needs to navigate to the project directory and execute the following command:

```
python manage.py runserver
```

The server will then begin its start-up procedure. Once finished, the desired output should look similar to the one presented in Fig. 49. What is crucial, at the end there is an instruction regarding quitting the

server manually. The user can achieve this by entering "CTRL + C" combination in the terminal on which the server is run. It may be useful in case any unexpected behaviours occur and the server instance needs fresh start.

```
System check identified no issues (0 silenced).
January 27, 2023 - 05:20:06
Django version 4.1, using settings 'face_rec_project.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Figure 49: Output of the server starting procedure.

The project website is now up. Notably, in the prompt from Fig. 49 the user is given the server address. The server is hosted on local machine, hence the address *http://127.0.0.1:8000/*. It is left for the user to enter this address in the browser to access the application. Once they have done that, they are greeted with the home screen containing three navigation buttons.

Note that all sample images and videos in the following sections are either self-taken or downloaded, free stock content.

7.2 Real-time video analysis

The first functionality of the application can be acccessed by clicking "Realtime face recognition" button in the home screen. It redirects the user to the screen presented in Fig. 50.

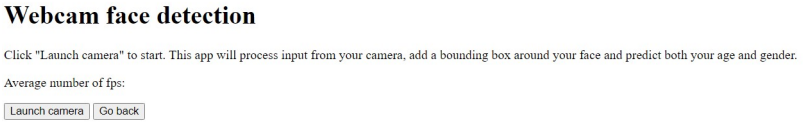


Figure 50: Initial screen for the webcam streaming functionality.

The user now has the possibility to to click the "Launch camera" button, which will activate one's camera sensor and began displaing the frames from it after a shor while. Please note that the program is designed for the camera to warm up, therefore it may take a couple of seconds to fully begin to work.

The correct output of this action would be the view presented in Fig. 51. The program will continuously display camera input while simultaneously displaying the boundary box with certainty rate and age/gender predictions, as described earlier.

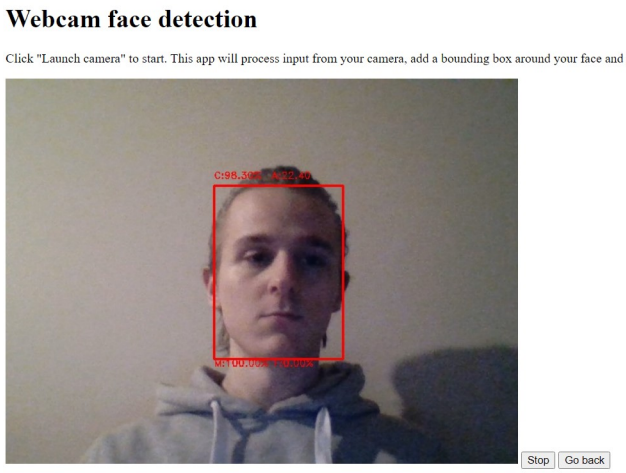


Figure 51: View of the webcam stream functionality while running.

If the user wishes to stop the stream, it is required to click the "Stop" button. on the right. After stopping, the average framerate is computed and displayed to the user (Fig. 52).

As a sidenote: one bug is known to occur when the user's face is too close to the camera sensor. In that case, sometimes a second boundary box is displayed that is offset in the east-south direction. Our guess is that this issue derives from some part of the rescaling process that is used for passing the frame to the Caffe model.

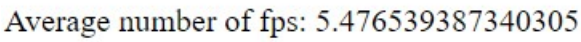


Figure 52: The number of fps achieved in realtime face recognition scenario.

7.3 Processing a pre-supplied video

The second case scenario is accessed by clicking the "Analyze a video" button. The user is greeted with a simple description and a view which enables to upload a file to the application. This view is programmed to obtain precisely one file from the user. As we can observe in Fig. 53 it is possible by clicking the area with the "Video" label. The usual, OS specific file viewer will pop up and the user will be able to choose the file freely. Please note that the text in this case is in polish; this is because the default HTML buttons are rendered in the browser according to local language.

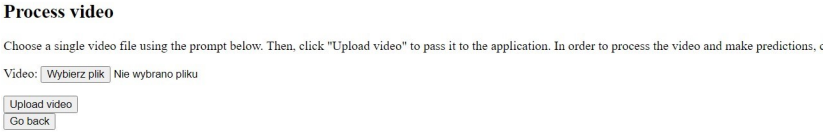


Figure 53: Initial screen for video processing scenario.

After choosing the video file, one must perform the next steps precisely in their order. First, the user must click the "Upload video" button below. The browser will begin to reload. Once finished the video file is uploaded to the app. After the reload two new buttons will appear - "Process video" and a prompt asking the user whether they would like to view the processed video in the browser. The latter will have no effect when clicked now; one must first click the "Process video" button in order for the application to prepare the predictions. Depending on the size of the video, it may take up to few minutes.

After the video processing is finished the page will reload. The "Show me!" button is now functional. The user may click it and the newly processed video will play underneath, as shown in Fig. 54. The user may at

any time return to main menu. What is crucial though, for the sake of consistency, the files are cleared on each new upload. Therefore, one will lose progress once they upload and process a new video.

Figure 54: Displaying the processed video.

7.4 Processing an image array

The last use case is very similar to the previous one, only this time the program processes a sequence of images supplied by the user.

The user is greeted with a near-identical screen with the possibility to upload several image files using the default OS file explorer (Fig. 55). What may be useful, the user may double-check whether he has chosen the correct images by looking on the prompt on the right hand side (containing the number of chosen files) and placing the cursor over the area with the images label - the hint with file names will then appear. The area of interest is shown in 56.

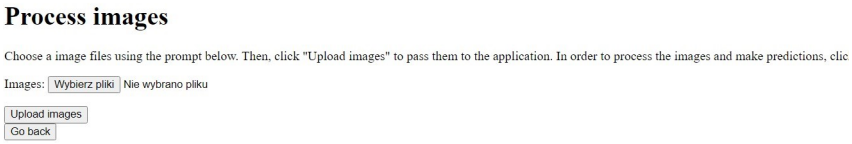


Figure 55: Initial screen for images processing scenario.

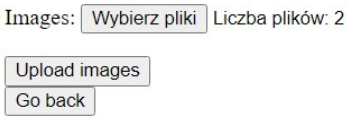


Figure 56: The area enabling user to choose and upload files.

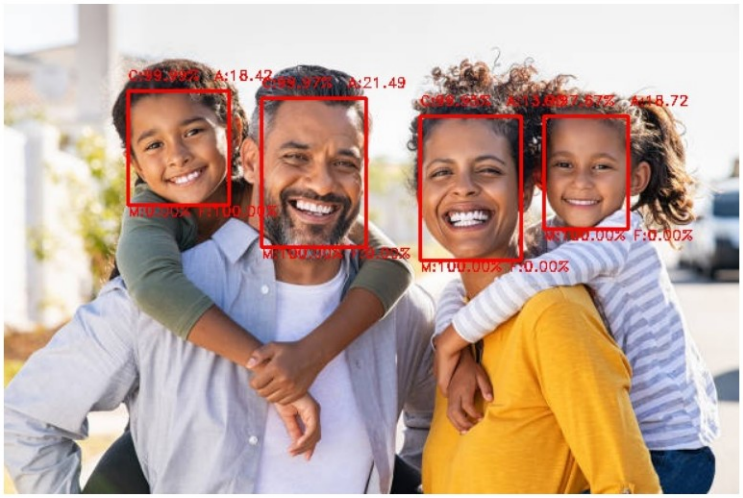
The next steps are all according to the previous case scenario. It is crucial that the user completes all of those in consecutive order to effectively use the application.

After completing the uploading and processing of the files, the prompt to show them will appear with the "Show me!" button (57). Once clicked, the processed images will be listed vertically with labels corresponding to the output folder and image file name.

Images: Wybierz pliki Nie wybrano pliku

Upload images
Process images Would you like to view the images? Show me!

/images_out/out_stock.jpg



/images_out/out_stock2.jpg

Figure 57: Displaying processed images.

8 References

Howard A. G., et al. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. 2017. [https://arxiv.org/pdf/1704.04861.pdf]

Howard A., et al. “Searching for MobileNetV3”. 2019. [https://arxiv.org/pdf/1905.02244.pdf]

Pujara A. “Image Classification with MobileNet”. 2020.
[https://medium.com/analytics-vidhya/image-classification-with-mobilenet-cc6fbb2cd470]

Sachin A. ”Detailed Guide to Understand and Implement ResNets”.

Schmidhuber J. “The most cited neural networks all build on work done in my labs”. 2021.

Qian S., Ning C., Hu Y. “MobileNetV3 for Image Classification”. 2021.

Zhuang F., Qi Z., Duan K., Xi D., Zhu Y., Zhu H., Xiong H., He Q. “A Comprehensive Survey on Transfer Learning”. 2020.

Keiron O’Shea, Ryan Nash “An Introduction to Convolutional Neural Networks”. 2015.
[https://arxiv.org/pdf/1511.08458.pdf]

MK Gurucharan, ”Basic CNN Architecture: Explaining 5 Layers of Convolutional Neural Network”. 2022.
[https://www.upgrad.com/blog/basic-cnn-architecture/]

Joaquin Vanschoren, ”Lecture 9: Convolutional Neural Networks”. 2021.
[https://ml-course.github.io/master/notebooks/09%20-%20Convolutional%20Neural%20Networks.html]

Joseph Nelson, ”When to Use Grayscale as a Preprocessing Step”. 2020.
[https://blog.roboflow.com/when-to-use-grayscale-as-a-preprocessing-step/]

Adrian Rosebrock, ”Keras Conv2D and Convolutional Layers”. 2018.
[https://pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/]

Sovit Ranjan RathSovit Ranjan Rath, ”Adding Noise to Image Data for Deep Learning Data Augmentation. 2020. [https://debuggercafe.com/adding-noise-to-image-data-for-deep-learning-data-augmentation/]

Django Web Framework. 2023.
[https://www.djangoproject.com/]