Demographical Characteristics' Detection (Gender, Age and Ethnicity) using Image Classification

Course: Machine Learning and Content Analytics FT 2021-2022

Professors: Haris Papageorgiou, Georgios Perakis

Students:

Mandyli Chrysoula – Charikleia (F2822106)

Papaloukas Ioannis (F2822110)

Papidaki Christina (F2822111)

Taklakoglou – Chidiroglou Argyrios (F2822114)

Table of Contents

Introduction 3

Our project 3

Our Vision/Goals 4

Methodology 5

Data Collection 6

Uncolored dataset 7

Dataset Overview 7

Data Processing/Annotation/Normalization 10

Algorithms, NLP architectures/systems 15

Multioutput model 16

Experiments – Setup, Configuration 16

Results & Quantitative Analysis (incl. visualizations) 22

Qualitative & Error Analysis 26

Single-output models 29

Gender Detection 29

Ethnicity Detection 40

Age Group Detection 51

Sum up of the best models' performance for Gender, Ethnicity and Age Group 62

Colored dataset 63

Dataset Overview 63

Data Processing/Annotation/Normalization 63

Algorithms, NLP architectures/systems 63

Experiments – Setup, Configuration 64

VGG16 Model 64

GoogleNet/Inception Model 67

ResNet50 Model 68

EfficientNetB0 Model 70

Model Evaluation and Prediction using unseen data 72

Discussion, Comments/Notes and Future Work 74

Members/Roles 74

Time Plan 75

Bibliography - References 76

Appendices 77

Introduction

The specific project uses neural network models and image classification, aiming to recognize and predict demographic characteristics of human depicted on images. More specifically, using face-data images, the features detected are “gender”, “age” and “ethnicity”. For that purpose, the image classification methods which were implemented are Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Two datasets have been used, the first containing uncolored and the second containing colored face-images, for which a multioutput and a range of single-output models have been developed. The pretrained models deployed, used the architectures of VGG16, GoogleNet/Inception Model, ResNet50 and EfficientNetB0. The models were compared based on their metric results, and those with the best performance were selected, consisting of the models deployed with ResNet50 architecture, using the colored dataset, for “gender” detection, and built CNN models detecting “age”, “ethnicity” and “gender”, using the uncolored dataset. New unseen test data was used to implement predictions on those best models and to calculate further metrics. The programming language used is Python and the external tools used are mainly keras and tensorflow.

Our project

Our project focuses on detecting the demographic characteristics of “gender”, “ethnicity” and “age”, from face-image data. Different models and architectures have been used, providing either simultaneously the above characteristics, or individually.

The specific project could be categorized under the umbrella of computer vision, which is a sector of Artificial Intelligence that uses Machine Learning and Deep Learning to allow computers to see, recognize and analyze things in photos and videos, in the same way that people do. More specifically, it was approached as an Image Classification problem, as its task was to categorize and assign labels to groups of pixels or vectors within an image, depending on specific rules.

As an overview, the analysis and modeling begun by using an **uncolored dataset** of face-image data, depicting men and women from all 5 continents’ ethnicities, and from a wide variety of ages (1-116 years old). A multioutput CNN model has been deployed, predicting “gender”, “age” and “ethnicity”. Following, nine single-output models were deployed, using CNN and RNN (GRU) architectures - three models for detecting each desired feature (“gender”, “age”, “ethnicity”), out of them the model with the best predictive ability (lowest validation loss) was selected as the best - resulting to one model for each feature.

As the “gender” was the feature for which the multioutput and single-output models had the best performance, in order to proceed more CNN architectures concerning gender detection and to use pretrained models, a new dataset consisting of **colored images** had been used and presented.

Our Vision/Goals

The main goal of our project, is to deploy models with high performance, classifying the depicted human demographic characteristics appropriately, in order to reveal his/her demographic profile.

Our vision is to strengthen the effort of providing a customized experience in ecommerce interfaces (dynamic landing pages) and social media ads, based on the demographic characteristics of the users. More specifically, the two main cases we target, are to facilitate the demographic characteristics' identification of:

A) A user who is in front of the screen while e-shopping. The detection of the user's "gender", "ethnicity" and "age" features, by using image classification, would provide to the ecommerce company a user demographical profile, even if he is not registered or signed by any account to the specific e-shop. Having this information, the company will be able to dynamically modify its landing page interface, customized ideally for the specific user's demographic profile (based on A/B testing that the company or its partners have already conducted). This will allow them to improve significant KPIS and metrics, such as achieving a higher conversion rate and gaining customer engagement.

   More specifically, our goal is to strengthen their effort by recognizing and identifying each user's demographics characteristics, just by their face-image, so that the company/customer will be able to efficiently apply its research results, categorize him to one of their customer segments, improve customer satisfaction and turn more easily potential customers into actual customers, by appropriate dynamic landing page transformation. So, even in the case that a different member of the family or a friend uses the user's account or device while e-shopping, the appropriate demographical characteristics would be identified, so that the ideal interface transformation will happen.

   Already, a range of companies are trying to engage a dynamical landing page based on users' segments, using characteristics gathered such as gender, transforming the interface. Some transformation may contain the interface's color or buttons' shape (i.e. if it has been observed that women or men tend to push more easily the "shop now" button when it is in a specific color or size, then the interface would dynamically transform. Additionally, based on cultural characteristics, in Asian countries red color evokes feelings of joy, while in Europeans, feelings of fear and anger. So, the "ethnicity" information could be useful for the interface's visualization and coloring, or even for an automated language changing), the size of letters or level of detail in descriptions (for example, based on the "age" feature, very young or old human may find it more difficult to read small letters or long descriptions).

   A barrier on this trial, would be that the user should have provided access to his/her device's camera, in order to enable capturing his face and analyze it real time. Additionally, the camera's quality and the light conditions, could be possible difficulties to the analysis as well.

B) A social media user who has uploaded public photos on his account. In the specific approach, the business question aiming to answer is "How users of desired demographic characteristics can be targeted through marketing actions, using photos posted on social media". The detection of the user's "gender", "ethnicity" and "age" features, from the

uploaded images, would assist in targeting customers or discovering trends, using demographics extracted from the users' uploaded photos. In that way, social media advertising companies will be able to target the desired user profiles with appropriate social media ads, as a demographical profile will be created to each user based on the features detected and the personas they have already developed.

Methodology

In more detail, the methodology followed could be divided in two main parts.

1. For the first part of analysis and modeling, the **uncolored dataset** of face-image data was used. During the data overview, preprocessing and exploratory data analysis, the data was normalized, reshaped, had one hot encoding transformation and spited to train, validation and test.

   The first model deployed was a **multioutput CNN model**, predicting "gender", "age" and "ethnicity" using Keras. For this model, the architecture implemented was composed of three major branches one for each feature of the dataset. For the building of the final model, which gives three different outputs, unique functions had been created. Three of them correspond to the building of the model for each one of the features "age", "ethnicity" and "gender". A fourth function combines these three models to create the final multioutput model. For the model's evaluation, validation loss, validation accuracy and mean absolute error metrics were used. Some predictions made by the model were displayed, as predicted vs real values written above and below the corresponding images.

   Concerning the **single-output models**, different models had been developed, each of those detecting just one of the desired features (gender, age, ethnicity). Nine of the univariate predictive models implemented have been kept and compared, and more specifically, two CNNs and one RNN (GRU) for each predicted feature. As it was observed that the multioutput model's performance in age detection was very poor without the use of age buckets, in single-output models' deployment the age groups were used. In the models that one hot encoded data was used during the training process, multiclass classification had been applied. In addition, in one of the models detecting ethnicity and gender, a tuner and a function were used in order to implement hyperparameter tunning. After the model's evaluation, for each feature, the model with the lowest validation loss was selected as the best for gender, age and ethnicity detection accordingly. For every model selected as the best, the corresponding predictions and analyzes were made.

2. The second part of the analysis uses a new dataset consisting of **colored face-images**. As the "gender" detection had the best performance in the uncolored dataset, we proceeded in more CNN architectures concerning "gender" detection using this new dataset. After a data pre-processing and data augmentation implementation, using this data, four pre-trained CNN models (detecting gender) were deployed and the architectures used were VGG16, GoogleNet/Inception Model, ResNet50 and EfficientNetB0. The selected models were downloaded, and modified slightly to suit the specific task, while the layers of the pre-trained models were frozen. The model with the best performance on the validation

data, was used to make predictions on the new unseen images. This model was chosen, via the comparison of the accuracy and the loss scores, of the four models.

The models' performance was evaluated using the new unseen test data for calculating the metrics: loss, accuracy, the classification report, and the confusion matrix.

Data Collection

The data finally used have been collected from two Kaggle datasets.

- The first dataset consists of 23705 pictures in .csv format. It includes the columns of age, ethnicity, gender, images, and pixels. The pictures are uncolored and present women and men from all 5 continents' ethnicities, and from a wide variety of ages (from babies to over 100 years old people).
- The second dataset consists of colored images of men and women. It is split into training and validation directories. Each directory includes two folders, that contain pictures of males and females separately. The number of pictures, in the training directory, is 23243 and 23766 for female and male accordingly. In the validation directory, there are 5841 images of females and 5808 of males included.

For both datasets, the data collection process consisted of downloading the csv files or the respective pictures, uploading them to google drive storage and, following, connecting the google drive file to the google collab account, in order to proceed with the data analysis and the project's implementation.

Uncolored dataset

Dataset Overview

Data characteristics/features

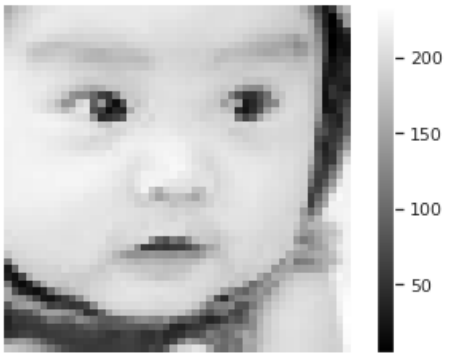
As it was mentioned before the uncolored dataset contains information about the age, the ethnicity and the gender of the person that is presented in each image. Additional features included in the dataset is the image name (which was not useful for the analysis) and the pixels (used as an independent variable).

	age	ethnicity	gender	img_name	pixels
0	1	2	0	20161219203650636.jpg.chip.jpg	129 128 128 126 127 130 133 135 139 142 145 14...
1	1	2	0	20161219222752047.jpg.chip.jpg	164 74 111 168 169 171 175 182 184 188 193 199...
2	1	2	0	20161219222832191.jpg.chip.jpg	67 70 71 70 69 67 70 79 90 103 116 132 145 155...
3	1	2	0	20161220144911423.jpg.chip.jpg	193 197 198 200 199 200 202 203 204 205 208 21...
4	1	2	0	20161220144914327.jpg.chip.jpg	202 205 209 210 209 209 210 211 212 214 218 21...

***Output 1**: Uncolored dataset characteristics/columns*

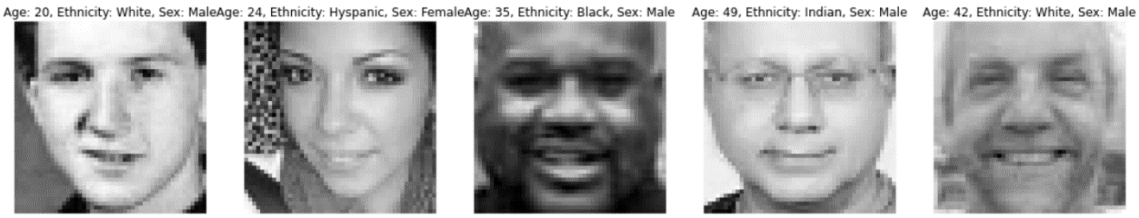
Images

Plotting some pictures while exploring the data, we can observe that the image color values range from 0 (black) to 255 (white), as expected.



***Picture 2**: Plotting the 10th picture from the uncolored dataset*

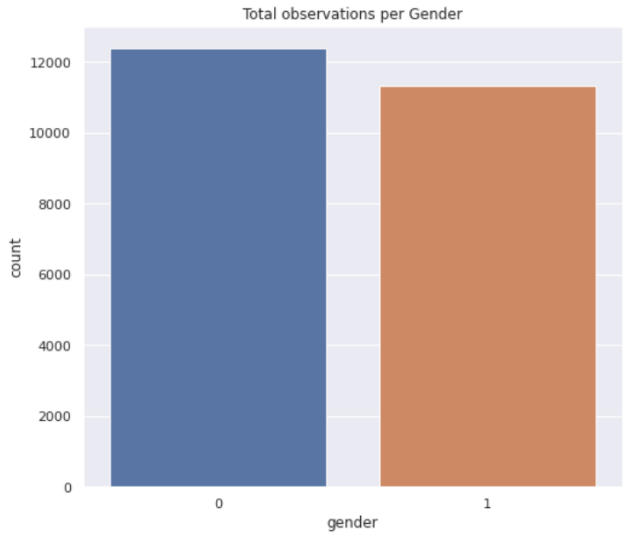
Plotting some random images with their labeled features, we can have an idea of the variety of ages, ethnicities and genders that are going to be analyzed.



***Picture 2**: Plotting random pictures with their labels from the uncolored dataset*

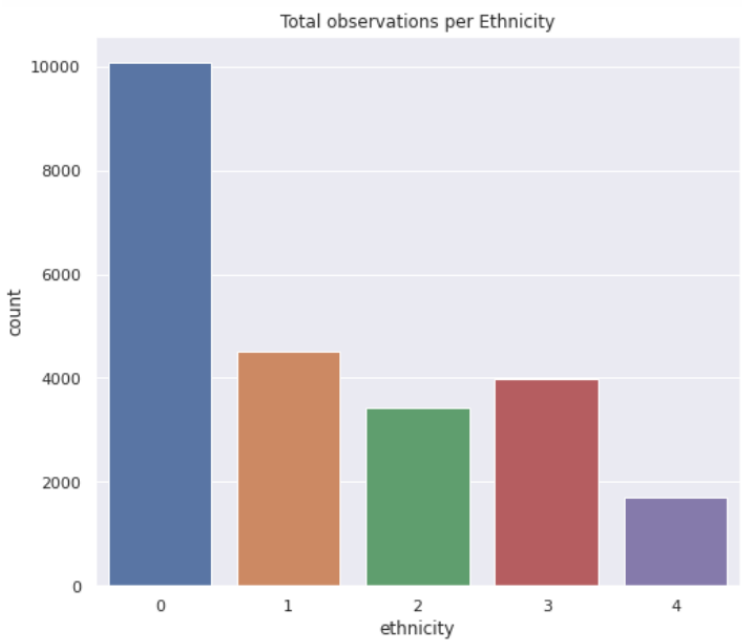
Dataset descriptive statistics

Regarding the gender, it is a binary variable that takes the values 0 and 1, where 0 represents the males and 1 the females. As it can be seen in the figure below, the gender variable is quite balanced, as there are 12391 males and 11314 females in the data.



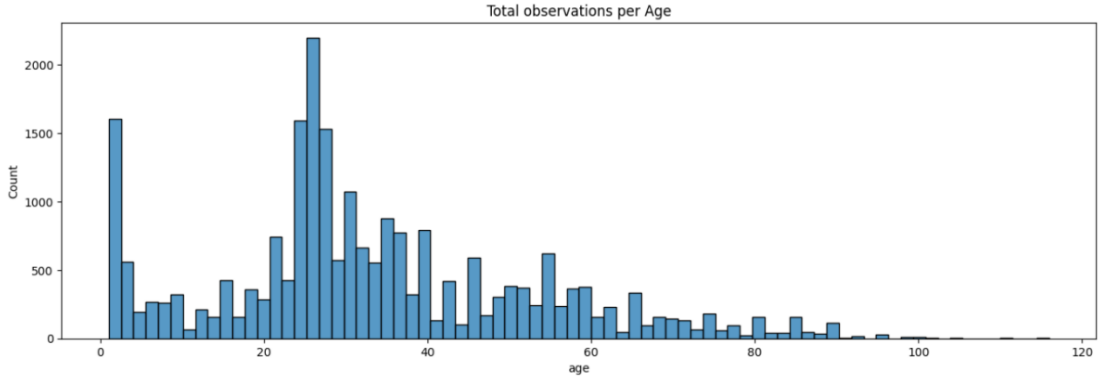
***Plot 3**: **Uncolored dataset** Total observations per Gender*

The ethnicity column takes 5 values, which are numbers from 0 to 4. Each number corresponds to an ethnicity. Specifically, the value 0 represents White people, the value 1 Black people, 2 is for the Asians, 3 for the Indians and 4 for the Hispanic ones. The total count of the observations that refer to each ethnicity are depicted in the **Plot 2** below. The category with the most observations is the first one that includes 10078 pictures of white people. The dataset contains 4526 images of black people, 3975 Indians, 3434 Asians and 1692 Hispanics.



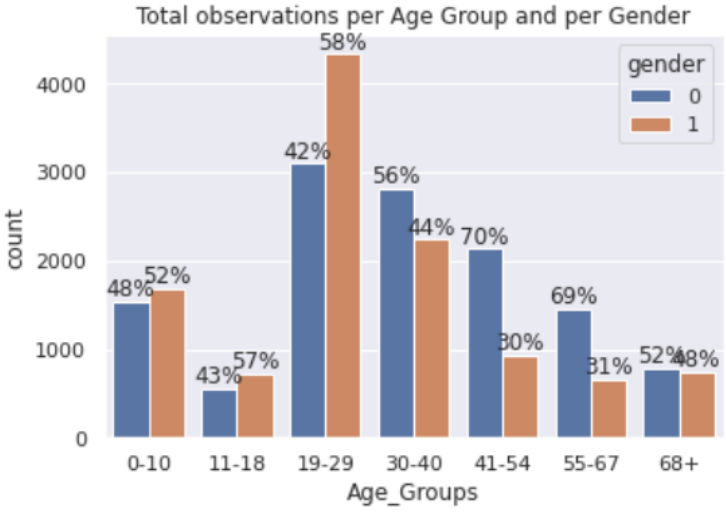
***Plot 2**: **Uncolored dataset** Total observations per Ethnicity*

Concerning the age attribute, it takes values from 0 to 116. Thus, there are images of people in a variety of ages, from babies to very old ones. In the plots below, it can be seen, that most of the observations are gathered in the age range 20-40. Also, there are a lot of 1 year old babies.



***Plot 3**: **Uncolored dataset** Total observations per Age*

Specifically for the age variable, some of the following models will be made to predict the exact age of each person, while some others are going to predict a range of values that might be a person's age. For that reason, the data are transformed, and some bins are created that categorize the values of the 'age' variable into groups. These groups were selected based on which ages the appearance of a person starts to change. The aforementioned age groups are 0-10, 11-18, 19-29, 30-40, 41-54, 55-67, 68+. In the following barplot, the total number of males and females for each age group is counted and the percentage that corresponds to each gender is depicted. The age group 19-29 is the one that gathers the most observations, while the 11-18 has the least.



***Plot 4**: **Uncolored dataset** Total observations per Age Group and per Gender*

Additional/complementary datasets that would probably strengthen the analysis, but are out of reach within the project cycle, would be datasets consisting of columns describing a wider range of demographic characteristics (such as language speaking, or country of origin) from municipalities, or even psychographic, concerning the people depicted on the images from social media databases.

Data Processing/Annotation/Normalization

Cleansing processes

As we can observe in **Output 2**, there are no missing values to any of the dataset columns.

```
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   age         23705 non-null  int64
 1   ethnicity   23705 non-null  int64
 2   gender      23705 non-null  int64
 3   img_name    23705 non-null  object
 4   pixels      23705 non-null  object
```

***Output 2**: Uncolored dataset -output of info command*

In addition, in order to find out if there are any outliers on the data, the value counts data revealed that gender and ethnicity distinct categories values were those expected and while checking for age values, 21 people had an age greater than 100 (**Output 3**) and the highest age found was 116, which is a not impossible value for a human's age.

	age	ethnicity	gender	img_name	pixels	Age_Groups
1290	101	2	1	20170105174739309.jpg.chip.jpg	222 222 223 222 218 195 130 81 222 241 233 236...	68+
1291	101	0	0	20170112213500903.jpg.chip.jpg	183 184 177 174 172 165 169 172 174 178 168 17...	68+
1292	103	2	0	20170112213001988.jpg.chip.jpg	195 248 124 87 86 33 57 67 53 56 53 54 43 44 4...	68+
1293	105	0	1	20170112213507183.jpg.chip.jpg	103 113 125 124 129 120 119 120 129 132 139 12...	68+
1294	105	0	1	20170112213021902.jpg.chip.jpg	251 250 251 251 250 244 226 210 192 166 162 15...	68+
1295	105	1	1	20170112213303693.jpg.chip.jpg	187 171 158 187 196 198 198 193 175 155 168 19...	68+
1296	105	0	0	20170112213001988.jpg.chip.jpg	161 115 157 145 165 203 154 137 150 153 160 12...	68+
1297	105	0	1	20170112213001988.jpg.chip.jpg	175 172 159 148 144 135 134 126 124 133 131 12...	68+
1363	110	1	1	20170110155201038.jpg.chip.jpg	38 38 39 38 39 92 12 11 43 52 47 36 19 23 20 2...	68+
1364	110	0	0	20170112213500903.jpg.chip.jpg	35 60 66 67 78 173 183 173 157 153 154 158 156...	68+
1365	110	2	0	20170112223734562.jpg.chip.jpg	163 177 169 142 141 147 141 135 156 139 137 14...	68+
1366	110	0	1	20170120134701015.jpg.chip.jpg	5 10 17 22 10 18 34 46 36 44 45 42 59 63 83 11...	68+
1367	110	3	1	20170110155139762.jpg.chip.jpg	1 0 3 28 94 125 145 159 78 153 98 213 219 212 ...	68+
1368	111	0	1	20170120134646399.jpg.chip.jpg	101 96 95 93 94 97 100 103 106 108 110 111 112...	68+
1369	115	1	1	20170112213257263.jpg.chip.jpg	209 209 205 193 191 145 62 51 43 53 71 60 68 5...	68+
1370	115	0	1	20170120134725991.jpg.chip.jpg	218 212 204 195 181 155 165 174 144 153 155 14...	68+
1371	115	0	1	20170120134725990.jpg.chip.jpg	170 153 145 141 133 140 141 146 144 150 171 17...	68+
1372	116	0	1	20170120134921760.jpg.chip.jpg	81 91 101 106 113 115 117 121 122 122 122 128 ...	68+
1373	116	2	1	20170112220255503.jpg.chip.jpg	207 200 197 181 186 174 184 185 162 164 164 16...	68+
1374	116	3	1	20170120134744096.jpg.chip.jpg	133 147 143 150 206 60 45 101 144 76 25 85 114...	68+
1375	116	0	1	20170112213001988.jpg.chip.jpg	105 86 95 135 159 150 147 147 153 117 148 160 ...	68+

***Output 3**: Uncolored dataset – Images depicting people of ages over 100*

Data pre-processing steps

Before starting the models' deployment, it was essential to process the data in order to be in the proper format. To begin with, to ensure that every row in the `pixels` column of the dataset has 2304 data points, the following command of **Script 1** was executed, with an empty list as an output. We inspect the pixels of each observation because the aim is to reshape the pixels into the format (48,48,1).

```
lista = []
for i in range(len(df)):
    if len(df['pixels'][i])!=2304:
        lista.append(i)

lista
```

***Script 1**: Checking if all images' pixels consist of 2304 values*

We also created dictionaries with the labels of the classes of each column, as displayed In **Output 4.** This process had to be done in order to have a better understanding of the confusion matrix in the models' predictions later on.

```
dataset_dict['gender_alias']
```

```
{'male': 0, 'female': 1}
```

```
dataset_dict['ethnicity_alias']
```

```
{'white': 0, 'black': 1, 'asian': 2, 'indian': 3, 'hispanic': 4}
```

***Output 4**: Class labels in dictionary format*

Afterward, it was necessary to convert each observation in the `pixels` column into a numpy array and change their data type to float.

```
df['pixels'] = df['pixels'].apply(lambda x:  np.array(x.split(), dtype="float32"))
```

***Script 2**: Converting 'pixels' column to numpy array*

Then, we rescaled the pixel values (between 0 and 255) to the [0, 1] interval, as neural networks prefer to deal with small input values, normalizing the data.

```
df['pixels'] = df['pixels'].apply(lambda x: x/255)
```

***Script 3**: Data normalization*

The one-dimensional numpy array of images had to be converted into a three-dimensional numpy array, because this is the appropriate format for the models' input. So, a numpy array was created containing the lists of the pixels and then those were reshaped in the desirable format (23705,48,48,1). To explain it more, the first number (23705) corresponds to the number of lists of pixels or equally to the number of observations in the `pixels` column. The second number (48) corresponds to the output width of each of the images. The third number (48) corresponds to the output height of each of the images. The last number (1) corresponds to the channel dimension. Thus, we end up having a four-dimensional tensor that will be used as input for the models, as presented in **Output 5.**

```
X = np.array(df['pixels'].tolist())

X = X.reshape(X.shape[0],48,48,1)

X.shape
```

```
(23705, 48, 48, 1)
```

***Output 5**: Shape after reshaping*

Another method that was tried to use, was data augmentation to have more observations and make the models more robust and avoid overfitting but finally, the method was not use because of the dataset's format.

Following, one hot encoding was performed twice, in two different ways. One time for the multioutput model (**Output 6** for gender) and one time for the single output models (**Script 5** for gender).

In addition, a stratified splitting was performed for each column of the dataset, splitting it into train, test, and validation (**Script 4 and 5** correspondingly). For the single output models, also weights were assigned to the classes of each column (**Script 7** for gender), and the ages were assigned to 7 bins as described earlier.

One hot encoding and stratified splitting for the multioutput model

```
age_label = np.array([a[0] for a in labels], dtype='uint8')
eth_label = np.array([e[1] for e in labels], dtype='uint8')
gen_label = np.array([g[2] for g in labels], dtype='uint8')
```

```
gen_label
```

```
array([0, 0, 0, ..., 1, 1, 1], dtype=uint8)
```

```
age_label_ohe = MinMaxScaler().fit_transform(age_label.reshape(-1, 1))
eth_label_ohe = LabelBinarizer().fit_transform(eth_label)
gen_label_ohe = LabelBinarizer().fit_transform(gen_label)
gen_label_ohe = to_categorical(gen_label_ohe)
```

```
gen_label_ohe
```

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [0., 1.],
       [0., 1.],
       [0., 1.]], dtype=float32)
```

***Output 6**: One hot encoding for multioutput model - Gender*

The procedure of **Output 6** is corresponding for age and ethnicity.

```
X_train_val, X_test, y_train_val_age, y_test_age, y_train_val_eth, y_test_eth, y_train_val_gen, y_test_gen, = train_test_split(X,
                                                                                                                               age_label_ohe,
                                                                                                                               eth_label_ohe,
                                                                                                                               gen_label_ohe,
                                                                                                                               test_size=0.2,
                                                                                                                               stratify=eth_label_ohe,
                                                                                                                               random_state=42)
```

```
X_train, X_val, y_train_age, y_val_age, y_train_eth, y_val_eth, y_train_gen, y_val_gen, = train_test_split(X_train_val,
                                                                                                           y_train_val_age,
                                                                                                           y_train_val_eth,
                                                                                                           y_train_val_gen,
                                                                                                           test_size=0.2,
                                                                                                           stratify=y_train_val_et
                                                                                                           random_state=42)
```

***Script 4**: Stratified splitting of data for the multioutput model*

One hot encoding and stratified splitting for the single output model:

The following steps refer to the “gender” column, but the procedure is corresponding for the “age_group” and “ethnicity” columns. The random state was set as 123 for all those splittings.

- Stratified splitting of data

```
yg= df['gender']
```

```
# Stratified splitting for gender
X_train_val_g, X_test_g, y_train_val_g, y_test_g = train_test_split(X,yg,
                                                    test_size=0.2,
                                                    random_state=123,
                                                    stratify=yg)
```

```
X_train_g, X_val_g, y_train_g, y_val_g = train_test_split(X_train_val_g,
                                            y_train_val_g,
                                            test_size=0.20,
                                            random_state=123,
                                            stratify=y_train_val_g)
```

Script 5: *Stratified splitting of data for the single-output model (Gender)*

- Using one hot ecnoder to turn the variable into categorical

```
y_enc_g = OneHotEncoder(sparse=False)
```

```
y_enc_g
```

```
OneHotEncoder(sparse=False)
```

1) At first we run fit_transform on the Training data

```
y_train_enc_g = y_enc_g.fit_transform(y_train_g.values.reshape(-1, 1))
y_train_enc_g
```

```
array([[1., 0.],
       [1., 0.],
       [1., 0.],
       ...,
       [1., 0.],
       [1., 0.],
       [0., 1.]])
```

2) Then we use the fitted One-hot-Encoder to transform the rest of the data

```
y_val_enc_g = y_enc_g.transform(y_val_g.values.reshape(-1, 1))
y_test_enc_g = y_enc_g.transform(y_test_g.values.reshape(-1, 1))
```

```
print(f'y_train_g shape: {y_train_enc_g.shape}')
print(f'y_val_g shape: {y_val_enc_g.shape}')
print(f'y_test_g shape: {y_test_enc_g.shape}')
```

```
y_train_g shape: (15171, 2)
y_val_g shape: (3793, 2)
y_test_g shape: (4741, 2)
```

Script 6*: One hot encoding for single-output model (Gender)*

- Assigning weights to classes

```
class_totals_g = y_train_enc_g.sum(axis=0)
class_weight_g = class_totals_g.max() / class_totals_g
```

Script 7*: Assigning weights to classes for single-output model (Gender)*

The functions numpy.save() and numpy.load() were also used, to save X_train and y_train in a format that is quite easy to store and retrieve.

For “Age” column:

```
np.save('X_train_a.npy', X_train_a)
np.save('y_train_enc_a.npy', y_train_enc_a)
```

```
X_train_a = np.load('X_train_a.npy')
y_train_enc_a = np.load('y_train_enc_a.npy')
```

For “Gender” column:

```
np.save('X_train_g.npy', X_train_g)
np.save('y_train_enc_g.npy', y_train_enc_g)
```

```
X_train_g = np.load('X_train_g.npy')
y_train_enc_g = np.load('y_train_enc_g.npy')
```

For “Ethnicity” column:

```
np.save('X_train_e.npy', X_train_e)
np.save('y_train_enc_e.npy', y_train_enc_e)
```

```
X_train_e = np.load('X_train_e.npy')
y_train_enc_e = np.load('y_train_enc_e.npy')
```

***Script 8**: Saving x_train and y_train for Age, Gender and Ethnicity*

So, for the single-output model, we end up having a training dataset, a validation dataset, and a test dataset for each feature. The test dataset will be used as an out-of-sample dataset in order to observe the performance of the models in unseen data later on.

- **For “Gender” column:** y_train_enc_g, y_val_enc_g, y_test_enc_g ~ X_train_g, X_val_g, X_test_g.
- **For “Ethnicity” column:** y_train_enc_e, y_val_enc_e, y_test_enc_e ~ X_train_e, X_val_e, X_test_e.
- **For “Age” column:** y_train_enc_a, y_val_enc_a, y_test_enc_a ~ X_train_a, X_val_a, X_test_a.

Algorithms, NLP architectures/systems

Computer vision - image classification

A) CNN

- Multioutput model (predicting 3 characteristics): ethnicity, gender, age (not buckets)
- Single-output model (predicting one characteristic)
  - Gender (with and without hyperparameters tunning - multiclass classification)
  - Age (buckets) using class weights and without - multiclass classification
  - Ethnicity (with and without hyperparameters tunning - multiclass classification)

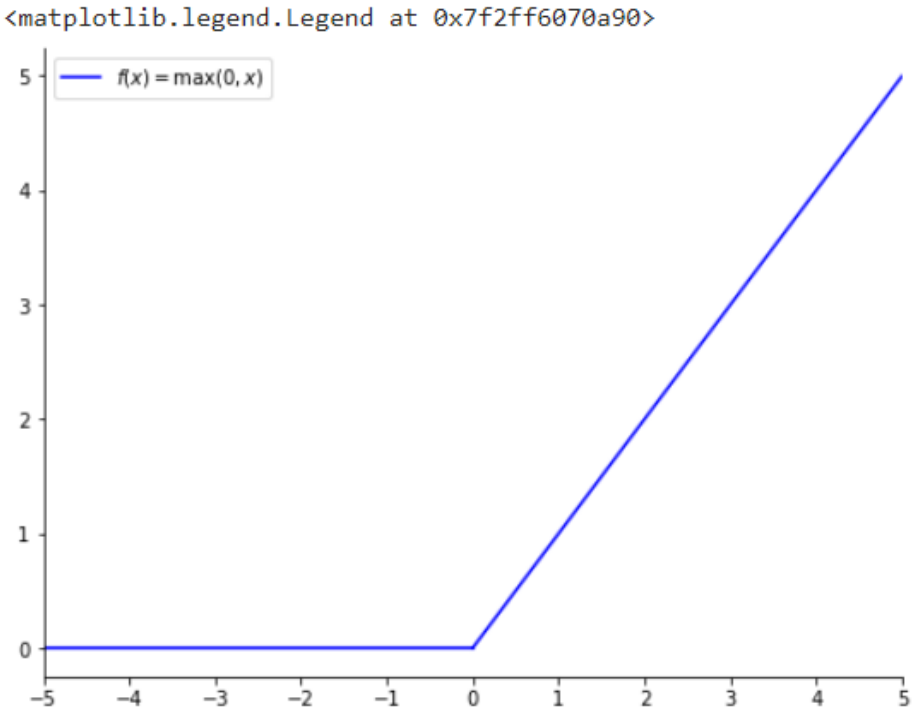
B) RNN

- GRU for age (buckets), gender, ethnicity

The above architectures and systems are going to be presented in more detail in each corresponding model.

Note

For the activation of our neurons we are going to use the rectifier function: $f(x)=x+=\max(0,x)$. This neuron, or unit, that employs the rectifier is also called a rectified linear unit (ReLU) and can be seen in **Plot 5**. As we can observe, all negative values and zero inputs are snapped to 0.0, whereas the positive outputs are returned as-is, resulting in a linearly increasing slope, given that we created a linearly increasing series of positive values.



***Plot 5**: Relu non-linear activation function*

Multioutput model

Experiments – Setup, Configuration

To perform a multioutput prediction with Keras, a special architecture has been implemented. In this architecture, the model is composed of three major branches one for each feature of the dataset. For the building of the final model which gives three different outputs, unique functions have been created. Three of them correspond to the building of the model for each one of the features “age”, “ethnicity” and “gender”. A fourth function combines these three models to create the final multi-output model.

The model’s final summary and architecture are presented in **Summary 1** and **Figure 1** respectively.

Model: "cnn"

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 48, 48, 1)]	0	[]
conv2d_3 (Conv2D)	(None, 48, 48, 16)	160	['input_1[0][0]']
activation_4 (Activation)	(None, 48, 48, 16)	0	['conv2d_3[0][0]']
batch_normalization_4 (BatchNo rmalization)	(None, 48, 48, 16)	64	['activation_4[0][0]']
max_pooling2d_3 (MaxPooling2D)	(None, 16, 16, 16)	0	['batch_normalization_4[0][0]']
dropout_4 (Dropout)	(None, 16, 16, 16)	0	['max_pooling2d_3[0][0]']
conv2d_4 (Conv2D)	(None, 16, 16, 32)	4640	['dropout_4[0][0]']
activation_5 (Activation)	(None, 16, 16, 32)	0	['conv2d_4[0][0]']
batch_normalization_5 (BatchNo rmalization)	(None, 16, 16, 32)	128	['activation_5[0][0]']
conv2d (Conv2D)	(None, 48, 48, 16)	160	['input_1[0][0]']
max_pooling2d_4 (MaxPooling2D)	(None, 8, 8, 32)	0	['batch_normalization_5[0][0]']
conv2d_8 (Conv2D)	(None, 48, 48, 16)	160	['input_1[0][0]']
activation (Activation)	(None, 48, 48, 16)	0	['conv2d[0][0]']
dropout_5 (Dropout)	(None, 8, 8, 32)	0	['max_pooling2d_4[0][0]']
activation_10 (Activation)	(None, 48, 48, 16)	0	['conv2d_8[0][0]']
conv2d_6 (Conv2D)	(None, 4, 4, 64)	18496	['dropout_6[0][0]']
batch_normalization_11 (BatchN ormalization)	(None, 16, 16, 32)	128	['activation_11[0][0]']
max_pooling2d_1 (MaxPooling2D)	(None, 8, 8, 32)	0	['batch_normalization_1[0][0]']
activation_7 (Activation)	(None, 4, 4, 64)	0	['conv2d_6[0][0]']
max_pooling2d_8 (MaxPooling2D)	(None, 8, 8, 32)	0	['batch_normalization_11[0][0]']
dropout_1 (Dropout)	(None, 8, 8, 32)	0	['max_pooling2d_1[0][0]']
batch_normalization_7 (BatchNo rmalization)	(None, 4, 4, 64)	256	['activation_7[0][0]']

dropout_10 (Dropout)	(None, 8, 8, 32)	0	['max_pooling2d_8[0][0]']
conv2d_2 (Conv2D)	(None, 8, 8, 32)	9248	['dropout_1[0][0]']
conv2d_7 (Conv2D)	(None, 4, 4, 64)	36928	['batch_normalization_7[0][0]']
conv2d_10 (Conv2D)	(None, 8, 8, 32)	9248	['dropout_10[0][0]']
activation_2 (Activation)	(None, 8, 8, 32)	0	['conv2d_2[0][0]']
activation_8 (Activation)	(None, 4, 4, 64)	0	['conv2d_7[0][0]']
activation_12 (Activation)	(None, 8, 8, 32)	0	['conv2d_10[0][0]']
batch_normalization_2 (BatchNo rmalization)	(None, 8, 8, 32)	128	['activation_2[0][0]']
batch_normalization_8 (BatchNo rmalization)	(None, 4, 4, 64)	256	['activation_8[0][0]']
batch_normalization (BatchNorm alization)	(None, 48, 48, 16)	64	['activation[0][0]']
conv2d_5 (Conv2D)	(None, 8, 8, 32)	9248	['dropout_5[0][0]']
batch_normalization_10 (BatchN ormalization)	(None, 48, 48, 16)	64	['activation_10[0][0]']
max_pooling2d (MaxPooling2D)	(None, 16, 16, 16)	0	['batch_normalization[0][0]']
activation_6 (Activation)	(None, 8, 8, 32)	0	['conv2d_5[0][0]']
max_pooling2d_7 (MaxPooling2D)	(None, 16, 16, 16)	0	['batch_normalization_10[0][0]']
dropout (Dropout)	(None, 16, 16, 16)	0	['max_pooling2d[0][0]']
batch_normalization_6 (BatchNo rmalization)	(None, 8, 8, 32)	128	['activation_6[0][0]']
dropout_9 (Dropout)	(None, 16, 16, 16)	0	['max_pooling2d_7[0][0]']
conv2d_1 (Conv2D)	(None, 16, 16, 32)	4640	['dropout[0][0]']
max_pooling2d_5 (MaxPooling2D)	(None, 4, 4, 32)	0	['batch_normalization_6[0][0]']
conv2d_9 (Conv2D)	(None, 16, 16, 32)	4640	['dropout_9[0][0]']
activation_1 (Activation)	(None, 16, 16, 32)	0	['conv2d_1[0][0]']
dropout_6 (Dropout)	(None, 4, 4, 32)	0	['max_pooling2d_5[0][0]']
activation_11 (Activation)	(None, 16, 16, 32)	0	['conv2d_9[0][0]']
batch_normalization_1 (BatchNo rmalization)	(None, 16, 16, 32)	128	['activation_1[0][0]']
batch_normalization_12 (BatchN ormalization)	(None, 8, 8, 32)	128	['activation_12[0][0]']
max_pooling2d_2 (MaxPooling2D)	(None, 4, 4, 32)	0	['batch_normalization_2[0][0]']
max_pooling2d_6 (MaxPooling2D)	(None, 2, 2, 64)	0	['batch_normalization_8[0][0]']
max_pooling2d_9 (MaxPooling2D)	(None, 4, 4, 32)	0	['batch_normalization_12[0][0]']
dropout_2 (Dropout)	(None, 4, 4, 32)	0	['max_pooling2d_2[0][0]']
dropout_7 (Dropout)	(None, 2, 2, 64)	0	['max_pooling2d_6[0][0]']
dropout_11 (Dropout)	(None, 4, 4, 32)	0	['max_pooling2d_9[0][0]']
flatten (Flatten)	(None, 512)	0	['dropout_2[0][0]']

```
flatten_1 (Flatten)            (None, 256)          0           ['dropout_7[0][0]']

flatten_2 (Flatten)            (None, 512)          0           ['dropout_11[0][0]']

dense (Dense)                  (None, 128)          65664       ['flatten[0][0]']

dense_2 (Dense)                (None, 128)          32896       ['flatten_1[0][0]']

dense_4 (Dense)                (None, 128)          65664       ['flatten_2[0][0]']

activation_3 (Activation)      (None, 128)          0           ['dense[0][0]']

activation_9 (Activation)      (None, 128)          0           ['dense_2[0][0]']

activation_13 (Activation)     (None, 128)          0           ['dense_4[0][0]']

batch_normalization_3 (BatchNo  (None, 128)         512         ['activation_3[0][0]']
rmalization)


batch_normalization_9 (BatchNo  (None, 128)         512         ['activation_9[0][0]']
rmalization)

batch_normalization_13 (BatchN  (None, 128)         512         ['activation_13[0][0]']
ormalization)

dropout_3 (Dropout)            (None, 128)          0           ['batch_normalization_3[0][0]']

dropout_8 (Dropout)            (None, 128)          0           ['batch_normalization_9[0][0]']

dropout_12 (Dropout)           (None, 128)          0           ['batch_normalization_13[0][0]']

dense_1 (Dense)                (None, 1)            129         ['dropout_3[0][0]']

dense_3 (Dense)                (None, 5)            645         ['dropout_8[0][0]']

dense_5 (Dense)                (None, 2)            258         ['dropout_12[0][0]']

age_out (Activation)           (None, 1)            0           ['dense_1[0][0]']

eth_out (Activation)           (None, 5)            0           ['dense_3[0][0]']

gen_out (Activation)           (None, 2)            0           ['dense_5[0][0]']

==================================================================================================
Total params: 265,832
Trainable params: 264,328
Non-trainable params: 1,504
```

***Summary 4**: **Multioutput** Model for Gender, Age and Ethnicity detection.*

The default structure for the convolutional layers is based on a Conv2D layer with a "Relu" activation, followed by a "BatchNormalization" layer, a "MaxPooling" and then finally a Dropout layer. This sequence is repeated three times. More specifically:

1. **Input:** Images of dimensions (48,48,1)
2. **1ST Block**
    a. **Convolution layer Conv2D:** 16 filters using 3x3 kernel
    b. **Activation**: The "Relu" function is being put on top, as activation function
    c. **BatchNormalization**
    d. **MaxPooling2D:** The layer cuts the size of the convolutional layers by producing a 16x16x16 output
    e. **Dropout:** The rate is set 0.25 to regularize the model
3. **2nd Block**
    a. **Convolution layer Conv2D:** We double the filters to 32 using 3x3 kernel
    b. **Activation**: The "Relu" function is being put on top, as activation function
    c. **BatchNormalization**
    d. **MaxPooling2D:** The layer cuts the size of the convolutional layers by producing a 8x8x32 output
    e. **Dropout:** The rate is set 0.25.
4. **3rd Block**
    a. **Convolution layer Conv2D:** We keep the filters at 32 using 3x3 kernel
    b. **Activation**: The "Relu" function is being put on top, as activation function
    c. **BatchNormalization**
    d. **MaxPooling2D:** The layer cuts the size of the convolutional layers by producing a 4x4x32 output
    e. **Dropout:** The rate is set 0.25.

The structure of the following layers differs between the branches.

For the **ethnicity** branch, the layers following the default structure are:

5. **4th Block**
    a. **Convolution layer Conv2D:** We double the filters to 64 using 3x3 kernel
    b. **Activation**: The "Relu" function is being put on top, as activation function
    c. **BatchNormalization**
6. **5th Block**
    a. **Convolution layer Conv2D:** We keep the filters at 64 using 3x3 kernel
    b. **Activation**: The "Relu" function is being put on top, as activation function
    c. **BatchNormalization**
    d. **MaxPooling2D:** The layer cuts the size of the convolutional layers by producing a 2x2x64 output
    e. **Dropout:** The rate is set 0.25
7. **6th Block**
    a. **Flatten**
    b. **Dense:** Adding a Dense Layer with 128 neurons
    c. **Activation**: The "Relu" function is being put on top, as activation function
    d. **BatchNormalization**
    e. **Dropout:** The rate is set 0.5
    f. **Dense:** This is the final layer with 5 neurons because of the 5 classes of ethnicity
    g. **Activation**: The "softmax" function is being put as activation function

For the **gender** branch, the layers following the default structure are:

5. **4th Block**
    a. **Flatten**
    b. **Dense:** Adding a Dense Layer with 128 neurons
    c. **Activation**: The “Relu” function is being put on top, as activation function
    d. **BatchNormalization**
    e. **Dropout:** The rate is set 0.5
    f. **Dense:** This is the final layer with 2 neurons because of the 2 classes of gender
    g. **Activation**: The “sigmoid” function is being put as activation function

For the **age** branch, the layers following the default structure are:

5. **4th Block**
    a. **Flatten**
    b. **Dense:** Adding a Dense Layer with 128 neurons
    c. **Activation**: The “Relu” function is being put on top, as activation function
    d. **BatchNormalization**
    e. **Dropout:** The rate is set 0.5
    f. **Dense:** This is the final layer with 1 neuron
    g. **Activation**: The “linear” function is being put as activation function

Figure 1 shows the architecture diagram of the final multi-output model as it is described previously. Out of the three branches, the left one is responsible for classifying the ethnicity category, the branch in the middle for the age category and the one on the right for the gender category.

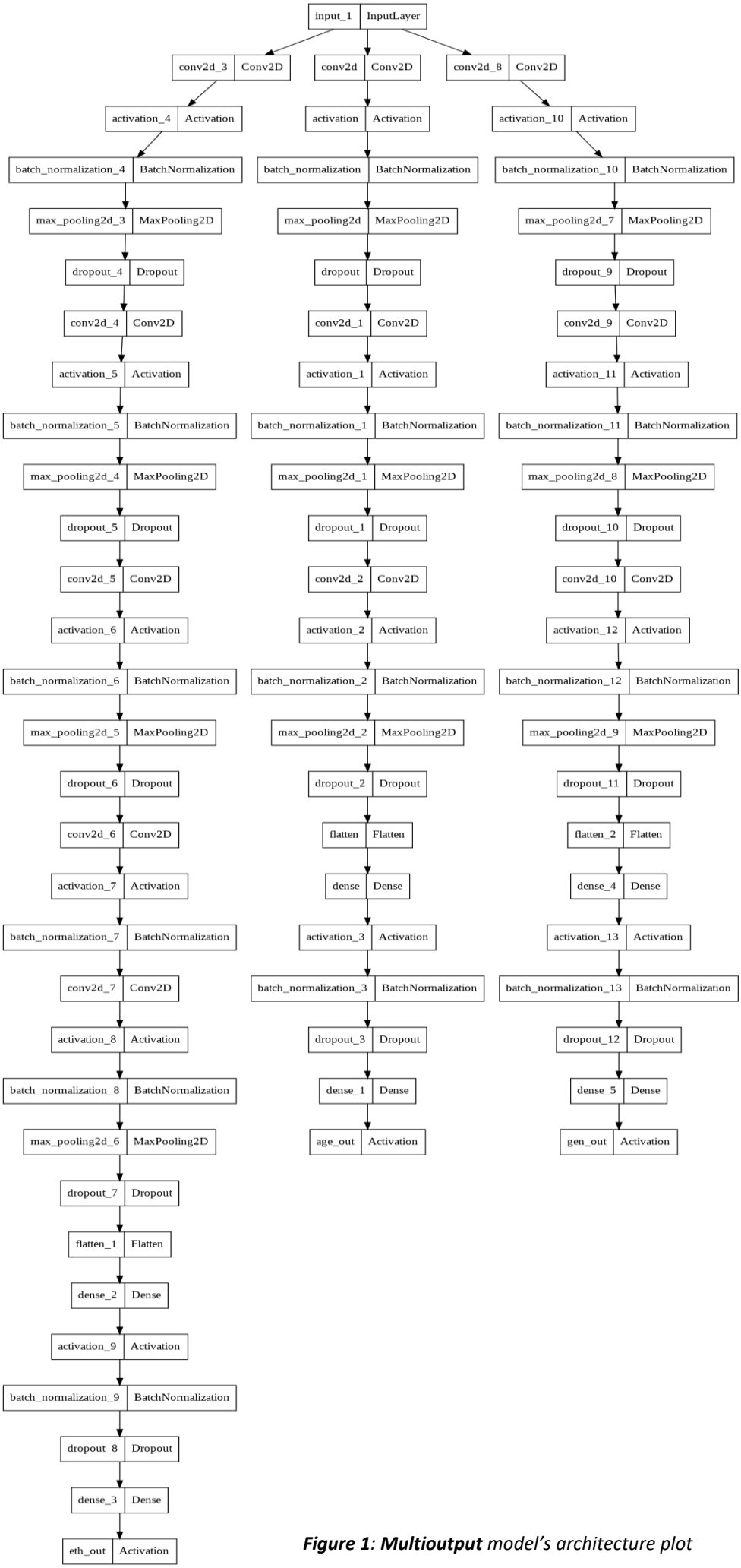


***Figure 1**: **Multioutput** model's architecture plot*

Model's Compilation

For the model's compilation, mean squared error has been used as loss function for the "age" prediction, and categorical crossentropy as loss function for the other two features because of the multi class classification problem. A quick reminder that feature "age" is not distributed in bins. The optimizer "Adam" is being used and the metrics are mean absolute error and accuracy. Also, some weights have been put according to difficulty of the predictions. Since the age is the most difficult to predict, its weight is 8.0. Ethnicity's weight is 2.0 and age's weight is 1.

Model's Training

The hyperparameters that were used for the training of the model are:

- Batch Size, set at 200
- Epochs, set at 100
- Early Stopping, using "val_loss" as a criterion to stop training and patience as 10 epochs before stopping

Results & Quantitative Analysis (incl. visualizations)

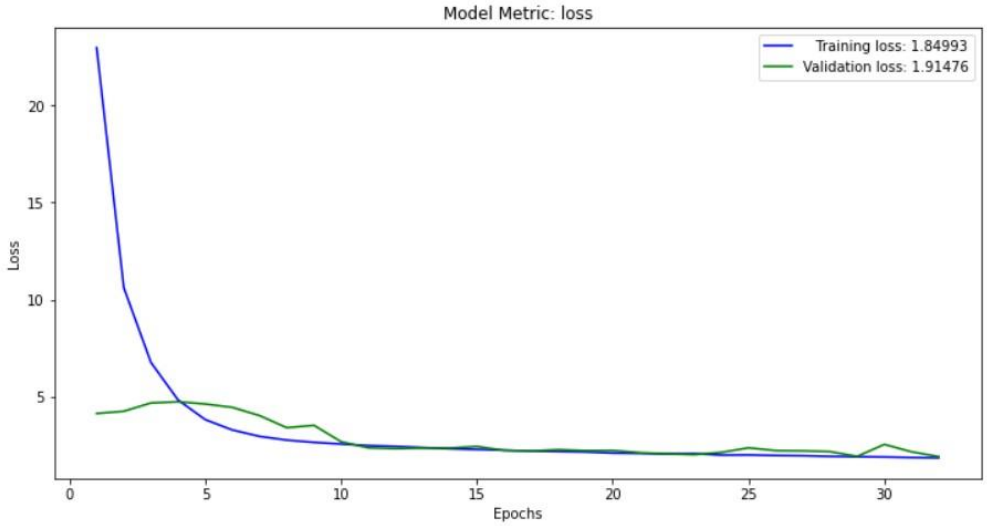
Model's Evaluation

For the model's evaluation, the one hot encoded validation data has been used for "ethnicity" and "gender" features. The "age" data have been rescaled between -1 and 1. All the evaluated metrics are in **Figure 2**.

```
Validation loss for age: 0.015
Validation loss for ethnicity: 0.724
Validation loss for gender: 0.273
Validation accuracy for age: 4.403 %
Validation accuracy for ethnicity: 74.453 %
Validation accuracy for gender: 88.505 %
Validation mean absolute error for age: 0.097
Validation mean absolute error for ethnicity: 0.137
Validation mean absolute error for gender: 0.245
```

***Figure 2**: Multioutput Model's (for gender, age and ethnicity prediction) Evaluation Metrics*

The plots for each of the multiple losses can be found in **Figure 3**.



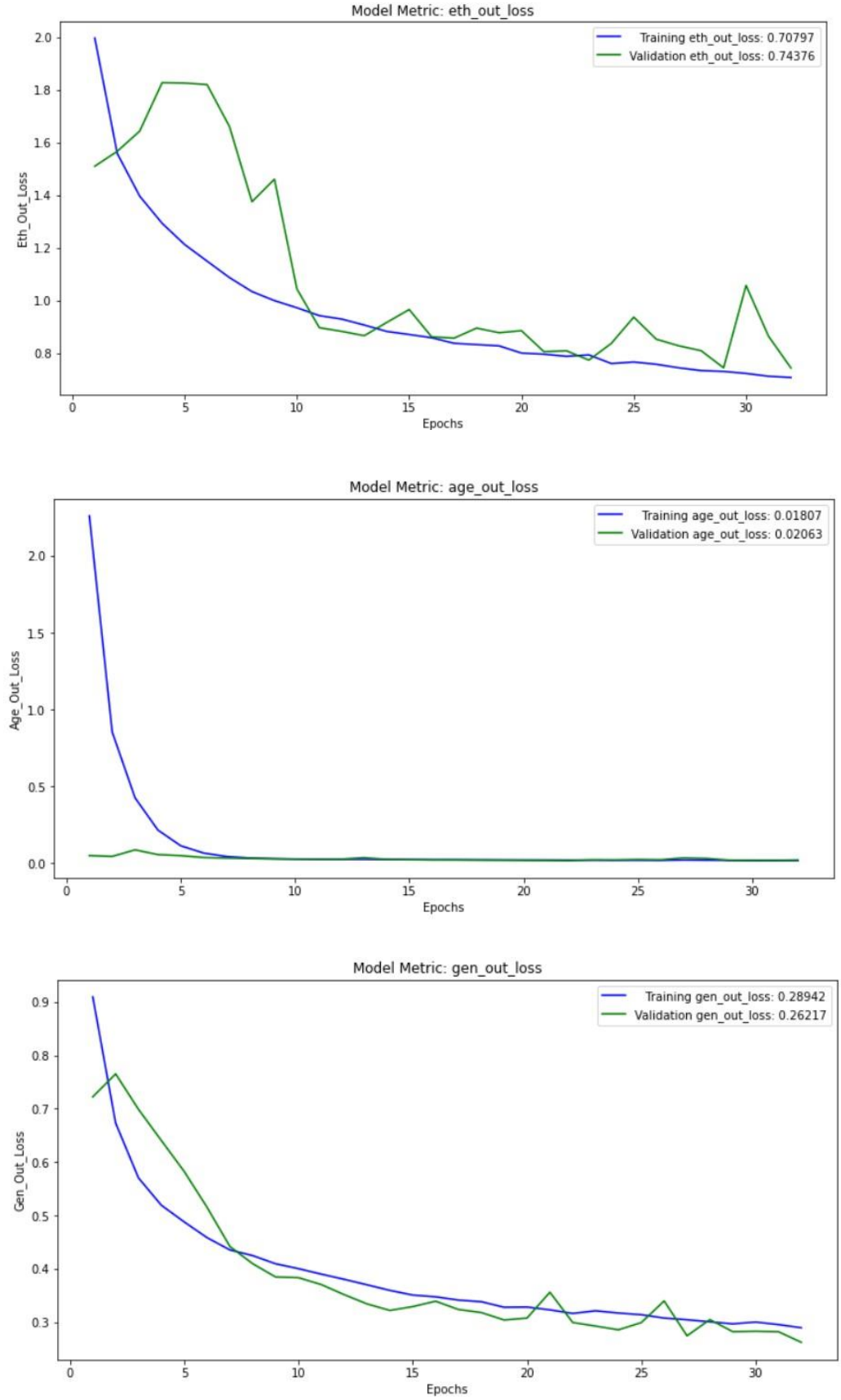


Figure 3: Multioutput Model– *Plots with Losses*

The plots with the respective accuracies in “ethnicity”, “age”, and “gender” can be found in **Figure 4** and the plots with their mean absolute errors in **Figure 5**.

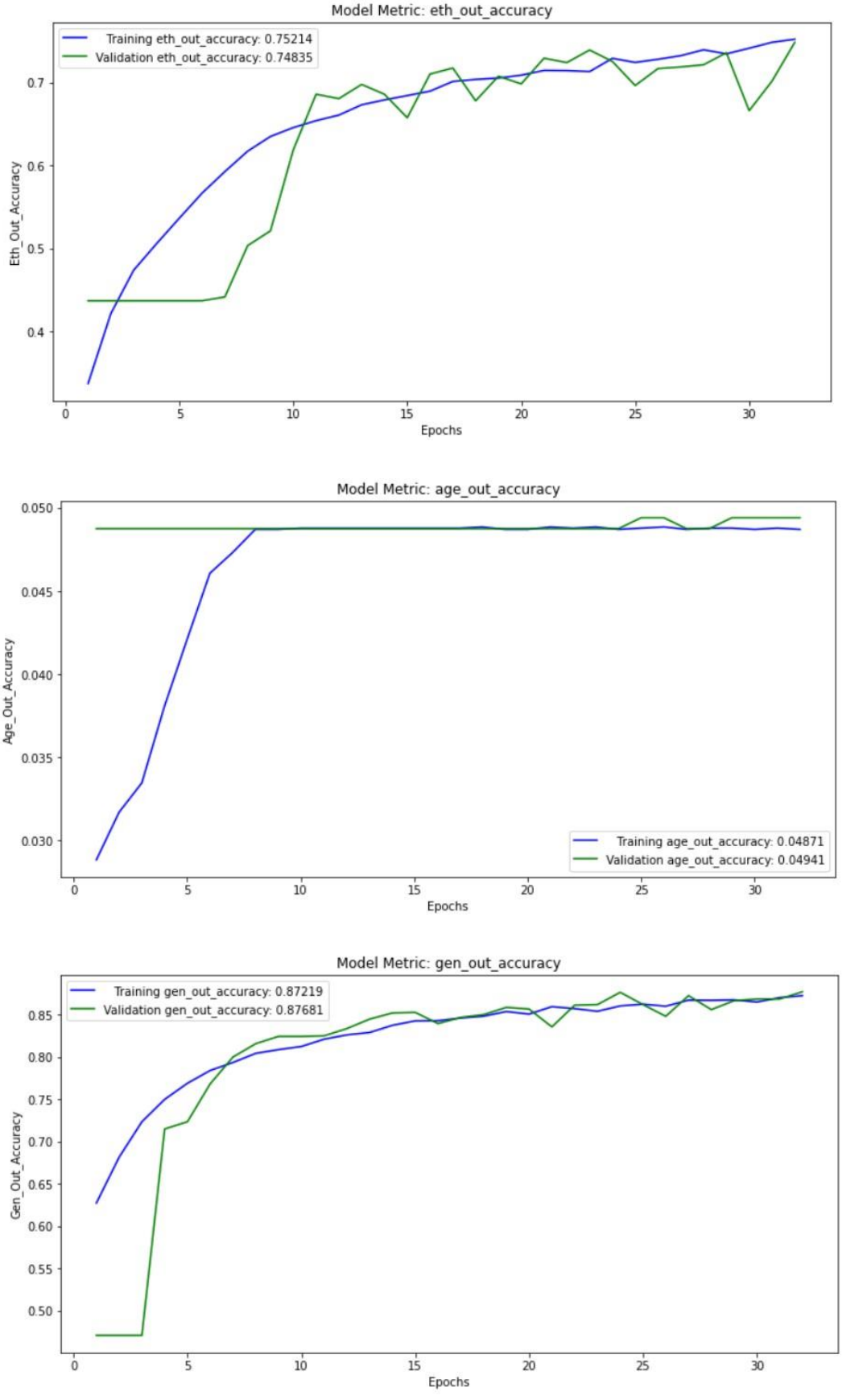
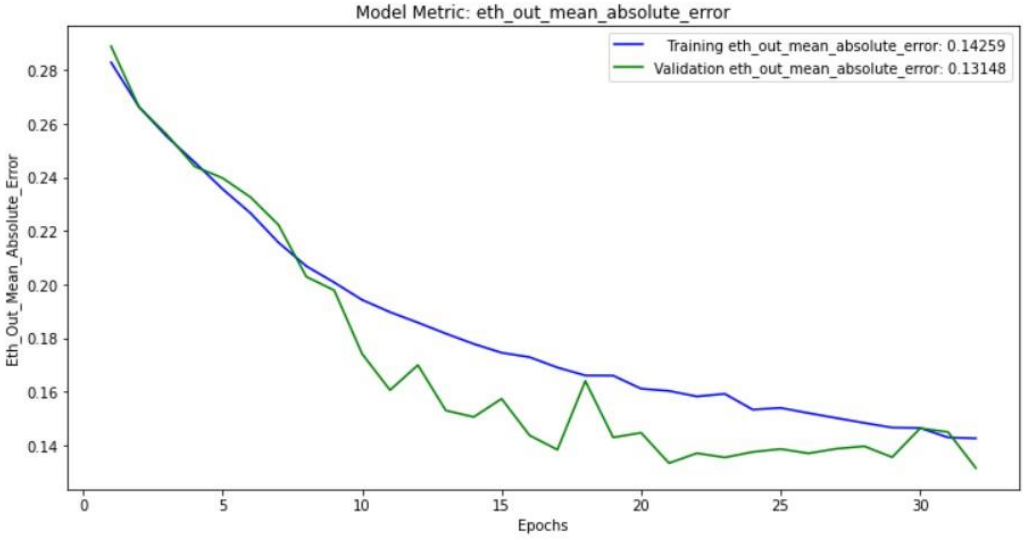
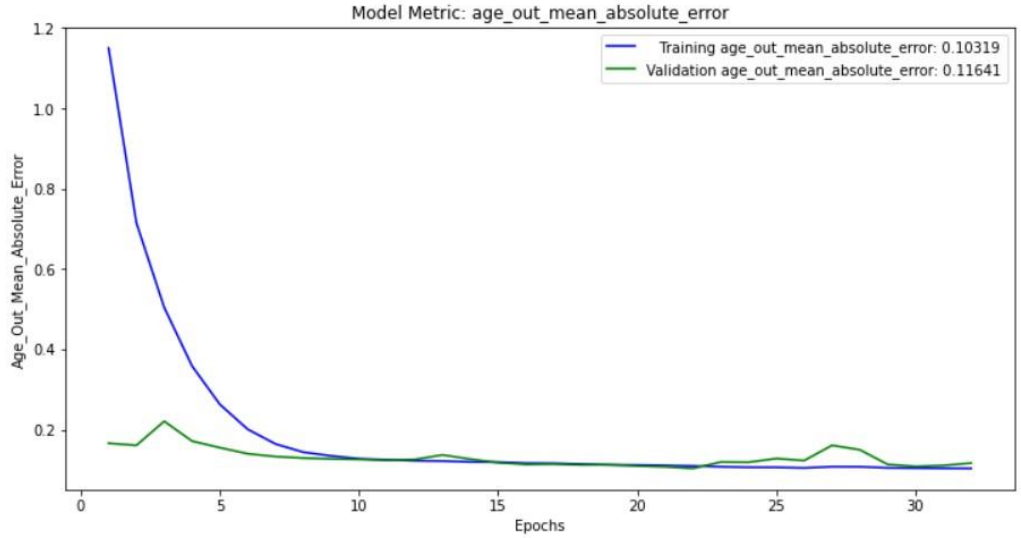


***Figure 4: Multioutput Model**– Plots with accuracies*





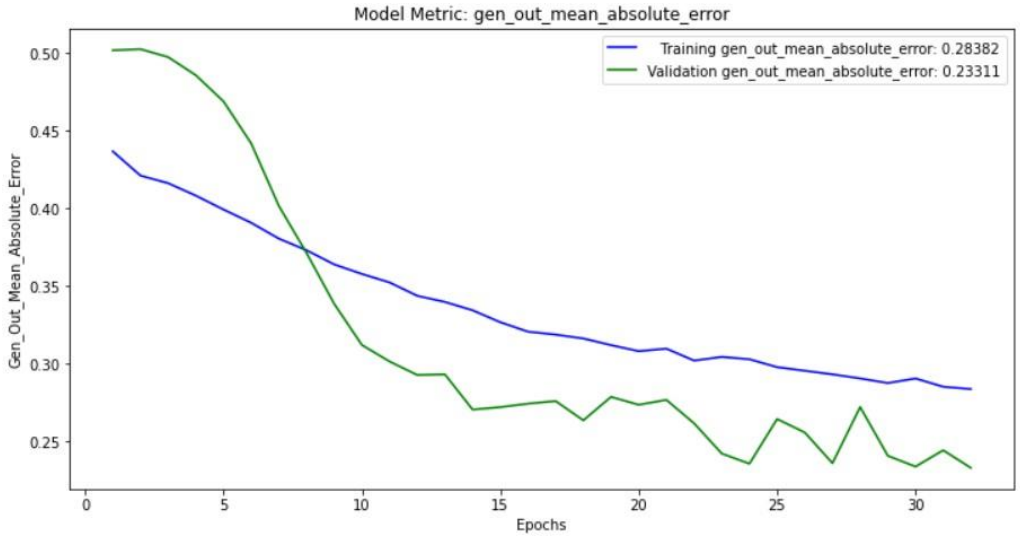


***Figure 5: Multioutput Model**– Plots for mean absolute errors*

From the plots above it comes out that:

- The model in “gender” feature seems to stabilize itself by epoch 15 on the validation set, with an accuracy of approximately 88%.
- The model in “ethnicity” feature does not stabilize itself and reach an accuracy of approximately 75%.
- The model in “age” feature seems unable to learn the patterns to properly predict the age. The accuracy does not exceed 5%.
- The validation loss in “age” feature is constant from the first epochs. So does the mean absolute error. This is an indication of overfitting. Maybe the data in “age” feature are imbalanced. In the next pages, we distribute the different ages to bins.

Qualitative & Error Analysis

Finally, the multioutput model can be used to predict features in the test dataset that has been split before.

In **Figures 6 and 7**, there are the confusion matrices for “gender” and “ethnicity” respectively. Each cell (i,j) in the confusion matrix contains the number of instances of class i that were predicted to be in class j. In **Figures 8 and 9,** the corresponding classification reports for these two features are displayed.

	0	1
0	2109	340
1	234	2058

***Figure 6**: **Multioutput Model** - Confusion Matrix for Gender*

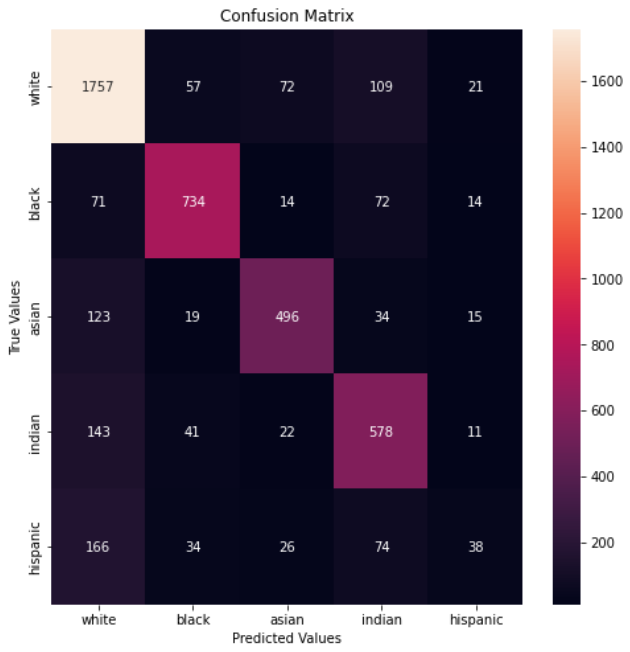


***Figure 7**: **Multioutput Model** - Confusion Matrix for Ethnicity*

	precision	recall	f1-score	support
male	0.90	0.86	0.88	2449
female	0.86	0.90	0.88	2292
accuracy			0.88	4741
macro avg	0.88	0.88	0.88	4741
weighted avg	0.88	0.88	0.88	4741

***Figure 8**: **Multioutput Model** - Classification Report for Gender*

	precision	recall	f1-score	support
white	0.78	0.87	0.82	2016
black	0.83	0.81	0.82	905
asian	0.79	0.72	0.75	687
indian	0.67	0.73	0.70	795
hispanic	0.38	0.11	0.17	338
accuracy			0.76	4741
macro avg	0.69	0.65	0.65	4741
weighted avg	0.74	0.76	0.74	4741

***Figure 9**: **Multioutput Model** - Classification Report for Ethnicity*

From the above confusion matrices and classification reports, some information come out:

- The model predicts better the males. However, in both genders the accuracy is high.
- The model is good at predicting Blacks, White and Asian people.
- Hispanic people are the hardest to predict.
- The most mistaken predictions happen between Hispanic and White people. The model confuses some Hispanic people with White.
- For the Black people, the most mistaken classification matches are with Indians.
- Also, the Asian and the Indian people are commonly mistaken for White by the model.

Some predictions made by the model can be seen below in **Figure 10**.

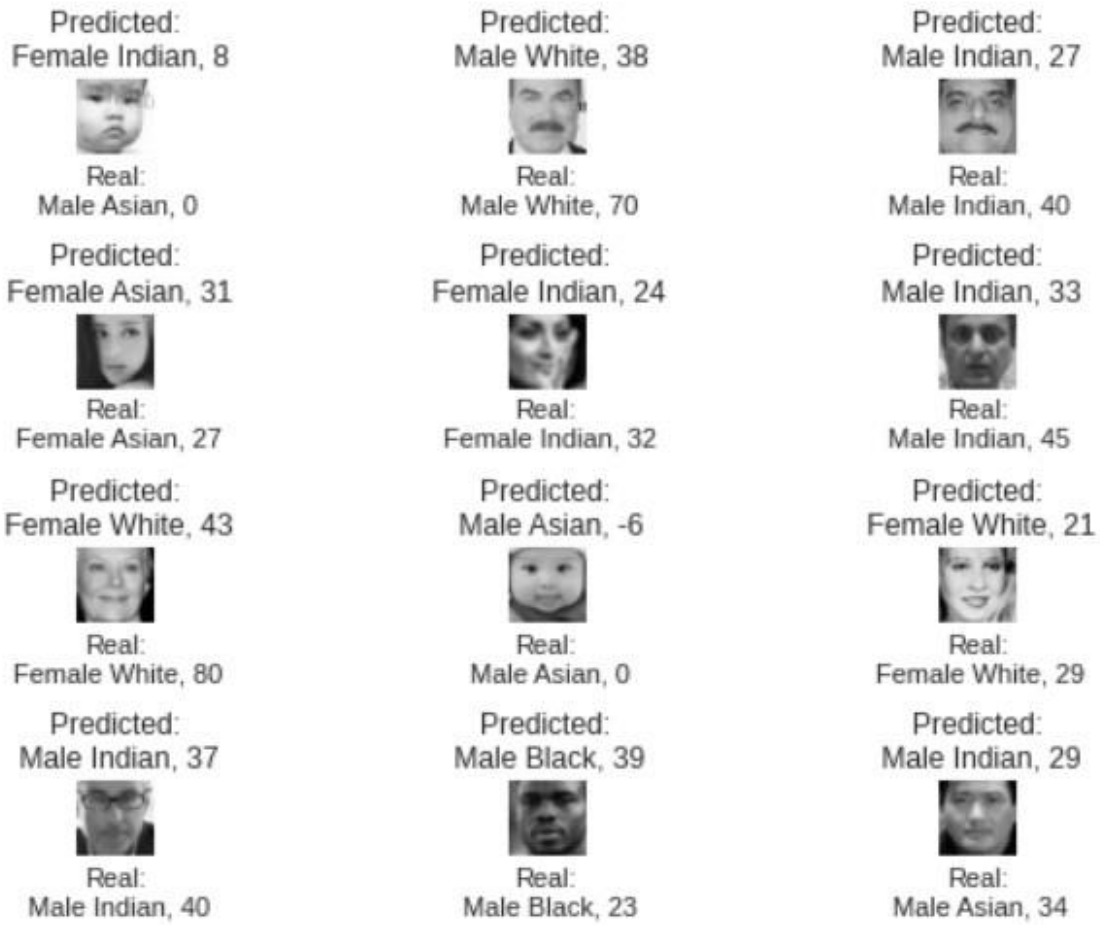


***Figure 10**: **Multioutput Model** – Predicted vs Real Values for Gender, Age and Ethnicity*

Single-output models

Nine of the single-output models implemented have been kept and compared, and more specifically, two CNNs and one RNN (GRU) for each predicted feature. For each feature (gender, age and ethnicity accordingly), the model with the lowest validation loss has been selected as the best and is the one on which the evaluation and prediction analyzes using an unseen test-set have been applied.

Gender Detection

Experiments – Setup, Configuration

As the analyzed data did not come equally from both gender classes (unbalanced), the weights used per class in the three following models, calculated by dividing each of the classes with the total, are the following **Weights per gender class**: `{0: 1.0, 1: 1.095152603231598}`

Those weights will lead the model to focus on the weaker class as well, as it will be penalized higher for wrong predictions concerning it.

❖ Model 1 - CNN

Concerning **gender** detection, the 1st model's final summary and architecture, are presented in ***Summary 2*** and plotted in ***Figure 11*** below.

```
Model: "Model1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 46, 46, 32)        320

 conv2d_10 (Conv2D)          (None, 44, 44, 64)        18496

 max_pooling2d_9 (MaxPooling  (None, 22, 22, 64)       0
 2D)

 dropout_2 (Dropout)         (None, 22, 22, 64)        0

 flatten_9 (Flatten)         (None, 30976)             0

 dense_20 (Dense)            (None, 128)               3965056

 dropout_3 (Dropout)         (None, 128)               0

 dense_21 (Dense)            (None, 2)                 258

=================================================================
Total params: 3,984,130
Trainable params: 3,984,130
Non-trainable params: 0
_________________________________________________________________
```

Summary 2: Model 1 *CNN model for gender detection*

Proceeding to the description of the model's architecture, as we can observe in **Summary 2** above and **Figure 11** below, the specific model consists of 8 layers, after inserting an input of size 48x48x1 (input layer).

- A 2D convolution: 32 is being initialized, using a 3x3x1 size kernel. After doing the calculation and making the convolution, the "Relu" function is being put on top, as activation function, in order to convert negative values to 0. The original images' shape is being set as input_shape. So, the original image will pass from 32 filters in the specific layer. The params are 320 (resulting from the act 3x3x32filters + 32biased terms).
- In the next 2D convolution, the filters put were 64 (using a 3x3x32 kernel and Relu activation function, as it is more robust comparing with others), making the depth 64. The params are 18496 (resulting from the act 3x3x32x64 + 64biased terms).
- The following MaxPooling (2,2) layer, halves its input to a 22x22x64 output, cutting the size of the conv-layers as reducing the image's dimensions.
- Adding dropout to regularize the model.
- Flatten the MaxPooling Layer, as image has been compressed.
- Adding another Dense layer of 128 neurons with Relu activation.
- Adding dropout to regularize the model (deleting filters, in a trial of all filters to be used for learning at the same time).
- As we have converted the problem to multi-class, with 2 classes, 2 neurons are used, as many as the genders, with the Softmax activation. In the fully connected layer, this final Dense will decide which is the gender of the person depicted in the image.
- The total system's params are 3984130, as the sum of all the system's weights (320+18496+3965056+258).

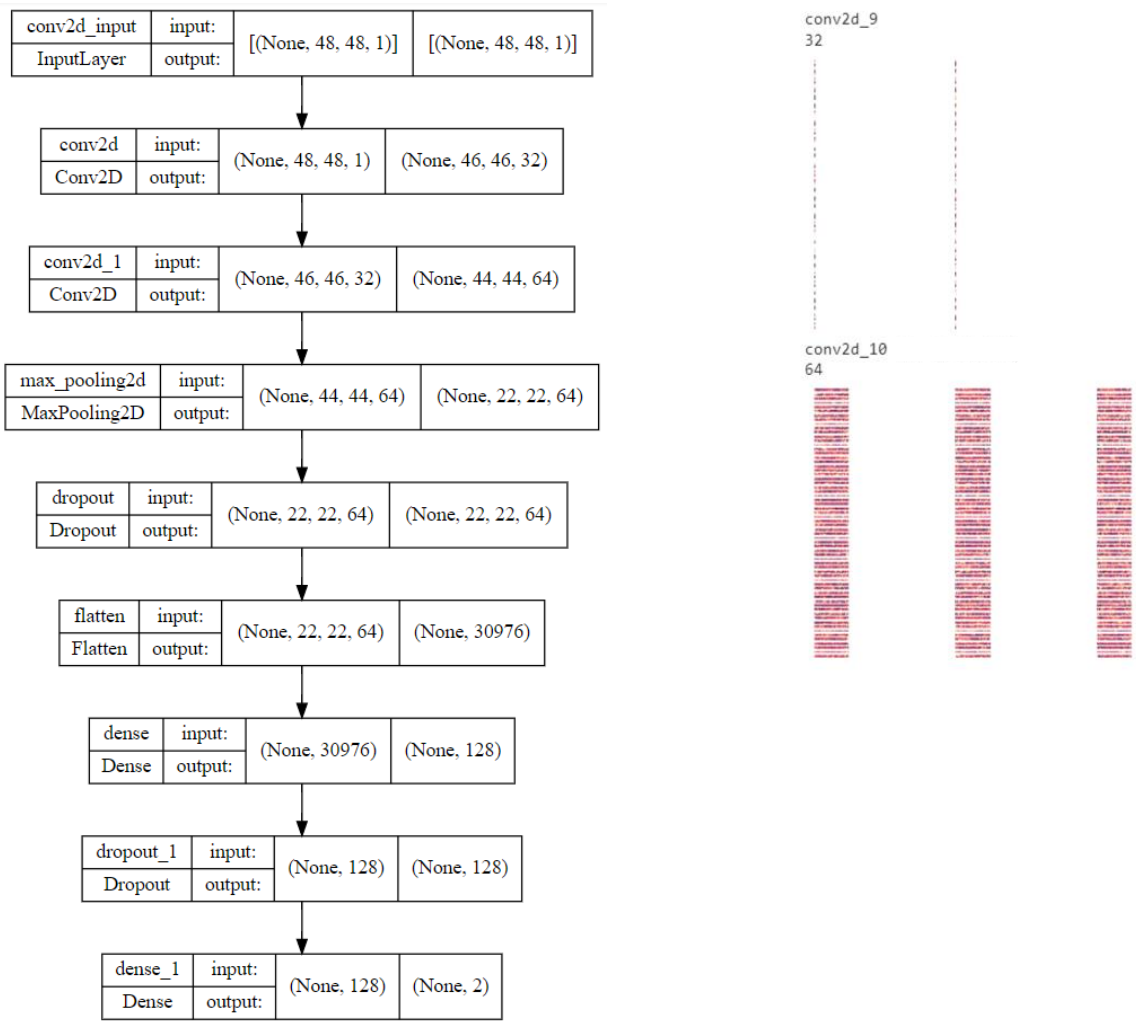


Figure 11: Model 1 *CNN for gender detection – Left: Plot of the model's architecture | Right: Convolutions' representation*

For the model's compile, categorical_crossentropy loss-function has been used, as a multi-class problem with 2 classes has to be solved and "Adam" was set as optimizer, with accuracy metrics.

The hyperparameters used for the model's training are:

- Batch_size, set as 200, indicating the number of subsamples in which the dataset will split.
- Epochs, set as 180, which is the number of times it will go through all the data.
- Early stoppings/callbacks: using 'val_loss' as a criterion to stop training, 0 as the minimum change in the monitored quantity to qualify as an improvement, and patience as 4 epochs before stopping.

After the model's fitting, the model's weights and the model were saved, so that they could be loaded if needed later on. The model's history and the metrics' evolution through epochs are displayed in **Appendix 1** of appendices.

Model weights

Some of the model layers' weights are presented in **Output 7.**

```
[array([[ 0.07341859,  0.18472378],
        [-0.24339661,  0.21725586],
        [-0.09734053,  0.1711123 ],
        [ 0.15626334, -0.1837635 ],
        [ 0.00265307,  0.14449635],
        [-0.02733894,  0.12774372],
        [-0.17772661, -0.1060657 ],
        [ 0.19424094,  0.06234702],
        [-0.1446404 , -0.0451085 ],
        [ 0.2477021 ,  0.00128091],
        [ 0.01421729,  0.2832157 ],
        [-0.00264425, -0.00391086],
        [-0.12199859,  0.13166428],
        [ 0.12292947,  0.1930362 ],
        [-0.2239344 , -0.11179363],
        [-0.2714638 ,  0.04820799],
        [ 0.00812295, -0.18344092],
        [-0.1876602 , -0.00460011],
        [ 0.19142725,  0.0144499 ],
        [-0.10754157,  0.18988581],
```

Output 7: Model 1 *CNN for gender detection – Weights*

❖ **Model 2 – CNN using hyperparameters tunning**

The 2nd model (Model 2) implemented with the use of hyperparameters tunning. The model's summary and architecture are presented in ***Summary 3*** and plotted in ***Figure 12*** below.

```
Model: "Model2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_9 (Flatten)         (None, 2304)              0

 dense_18 (Dense)            (None, 288)               663840

 dense_19 (Dense)            (None, 2)                 578

=================================================================
Total params: 664,418
Trainable params: 664,418
Non-trainable params: 0
_________________________________________________________________
```

Summary 3: Model 2 *CNN model for gender detection using hyperparameters tunning.*

For the specific model's deployment, a function called "model_builder" has been created, in which a Sequential model is built. As we can observe in **Figure 12,** a layer Flattens inputs of shape (48,48). The number of units in the first Dense layer is tuned and is set that an optimal value between 32-512 should be chosen. The activation function used is Relu. The final Dense layer will decide which is the gender of the person depicted in the image, using sigmoid activation function, as the classification is binary (the one hot encoded data have not used for the specific model's training). Last is the learning rate for the optimizer, which chooses an optimal value from 0.01, 0.001, or 0.0001.

For the model's compile "Adam" was used as an optimizer, sparse categorical crossentropy as a loss-function and "accuracy" with "binary accuracy" as metrics.

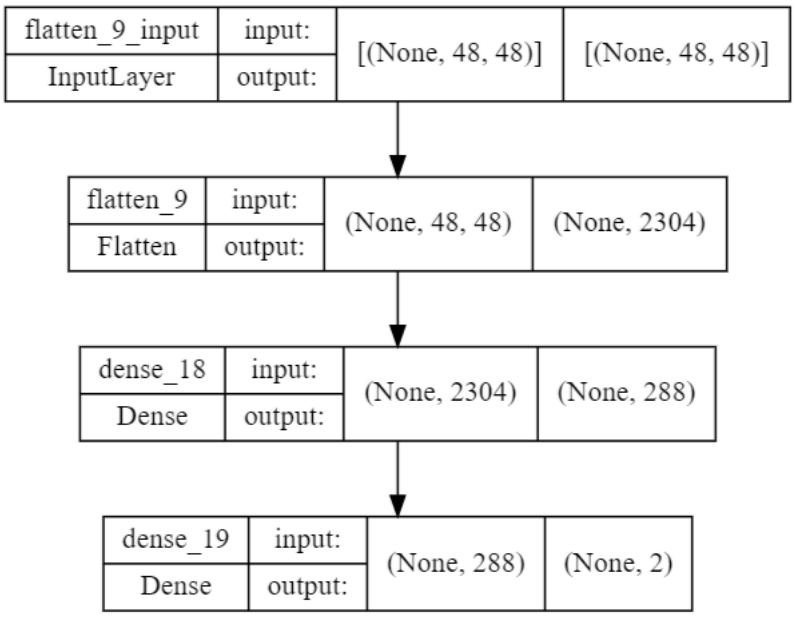


Figure 12: Model 2 *CNN for gender detection using hyperparameters tunning – Plot of the model's architecture*

Following a tuner was created, using val_loss as an objective, and an early stoppings/callbacks was set: using 'val_loss' as a criterion to stop training and 5 for patience.

The tuner searched for the ideal hyperparameters using the gender data for 50 epochs, in order to get the optimal hyperparameters, that were saved in “best_hps” variable. The output of tuner search is displayed in **Output 8**.

```
The hyperparameter search is complete. The optimal number of units in the first densely-connected
layer is 288 and the optimal learning rate for the optimizer
is 0.01.
```

Output 8: Model 2 *CNN for gender detection using hyperparameters tunning – Output of tuner search*

Following, those optimal hyperparameters had been used to build the model, which was trained in the data for 50 epochs (and used the class_weights found earlier).

The hypermodel was re-instantiate and trained with the optimal number of epochs from above (17). The model was retrained, and its weights were saved for being able to be loaded for later use.

The model’s tunning and training history, as well as the metrics’ evolution through epochs are displayed in **Appendix 2** of appendices.

❖ **Model 3 – RNN (GRU)**

The 3rd model is an RNN model using the GRU architecture, and its summary is seen in **Summary 4**.

```
Model: "Model3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_4 (GRU)                 (None, 64)                21888

 batch_normalization_8 (Batc  (None, 64)               256
 hNormalization)

 dense_63 (Dense)            (None, 2)                 130

=================================================================
Total params: 22,274
Trainable params: 22,146
Non-trainable params: 128
_________________________________________________________________
None
```

Summary 4: Model 3 *GRU model for gender detection*

The specific Sequential model displayed in **Figure 13** consists of three layers after the input layer receiving inputs of shape (48,48):

- GRU: 64, resulting to 21888 params.
- Batch normalization, resulting to 256 params.
- And the final Dense layer: 2.

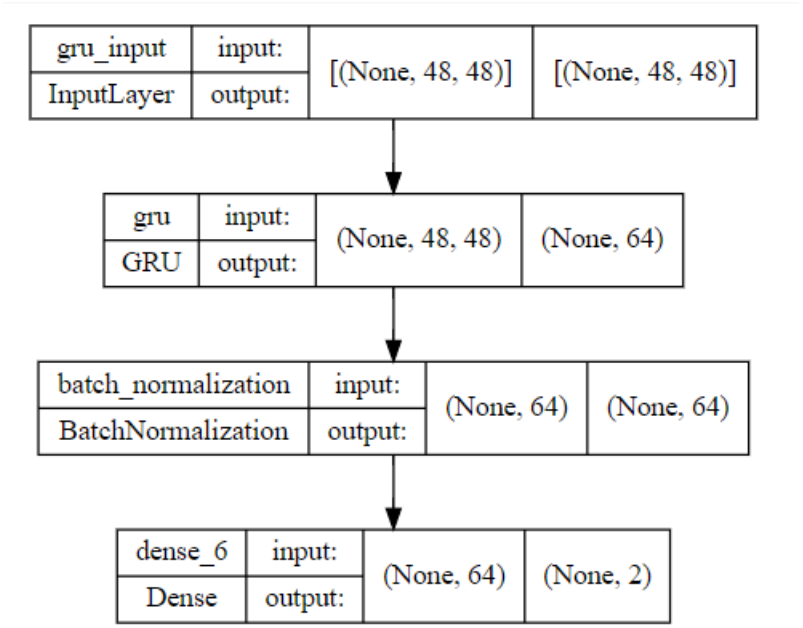


Figure 13: Model 3 *GRU model for gender detection - Plot of the model's architecture*

❖ Model 1

Model evaluation

For the model's evaluation, the one hot encoded validation data has been used. Validation loss scores 0.249 and validation accuracy 0.891.

```
119/119 [==============================] - 1s 5ms/step - loss: 0.2494 - accuracy: 0.8911 - binary_accuracy: 0.8911
[0.24944446980953217, 0.8911151885986328, 0.8911151885986328]
```

Output 9: Model 1 *CNN for gender detection – Evaluation on validation data*

Visualizations

As we can observe from the visualizations of **Figure 14**, after epoch 5 the training loss is lower than validation loss. In addition, after epoch 4, training accuracy is higher than validation accuracy, with a tendency to overfitting.

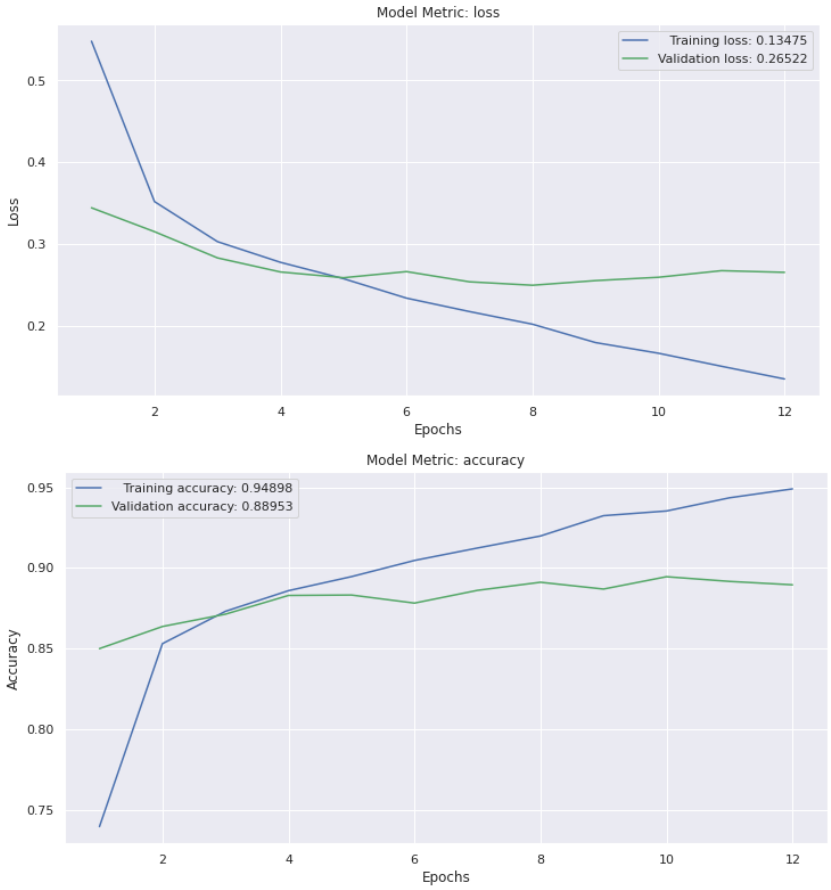


Figure 14: Model 1 *CNN for gender detection – Loss and accuracy plots*

❖ **Model 2**

Model evaluation

For the model's evaluation, the validation data has been used. Validation loss scores 0.349, validation accuracy 0.831 and validation binary accuracy 0.495.

```
119/119 [==============================] - 0s 2ms/step - loss: 0.3491 - accuracy: 0.8318 - binary_accuracy: 0.4951
[val loss, val accuracy]: [0.3491223454475403, 0.8317953944206238, 0.4951225817203522]
```

Output 10: Model 2 *CNN for gender detection using hyperparameters tunning – Evaluation on validation data*

Visualizations

As we can observe from the visualizations of **Figure 15**, although there are some strong fluctuations in the validation losses and accuracies, especially on the 5th epoch, the validation and training curves remain close after the 12th epoch for all metrics.

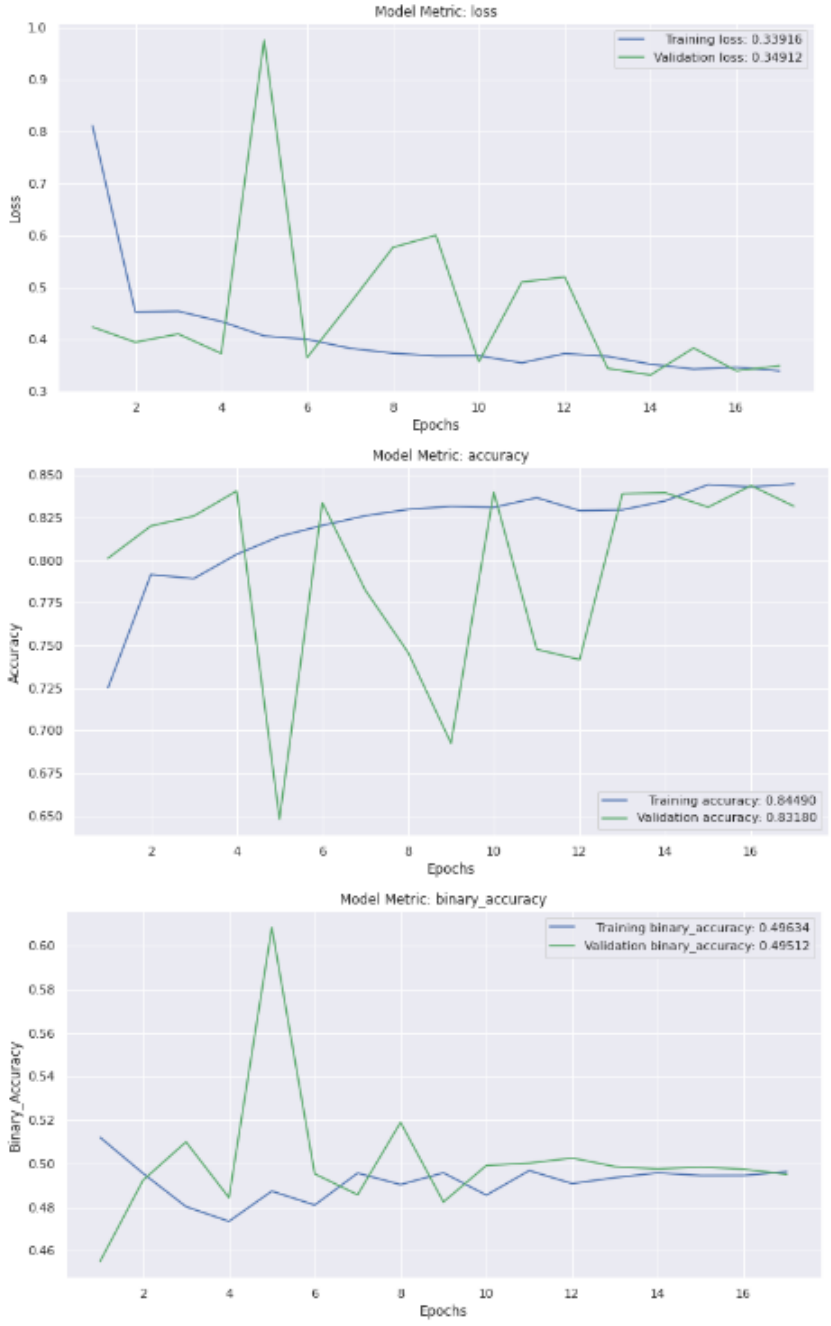


Figure 15: Model 2 *CNN for gender detection using hyperparameters tunning – Loss, accuracy and binary accuracy plots*

❖ **Model 3**

Model evaluation

For the model's evaluation, the validation data has been used. Validation loss scores 0.510, validation accuracy 0.748 and validation binary accuracy 0.548.

```
119/119 [==============================] - 0s 4ms/step - loss: 0.5107 - accuracy: 0.7482 - binary_accuracy: 0.5480
[0.5106934309005737, 0.7482203841209412, 0.5479831099510193]
```

Output 11: Model 3 *GRU for gender detection - Evaluation on validation data*

Visualizations

As we can observe from the visualizations of **Figure 16**, validation loss is higher than training loss after epoch 3, while the opposite comes with the accuracy metric, revealing model tendency to overfitting.

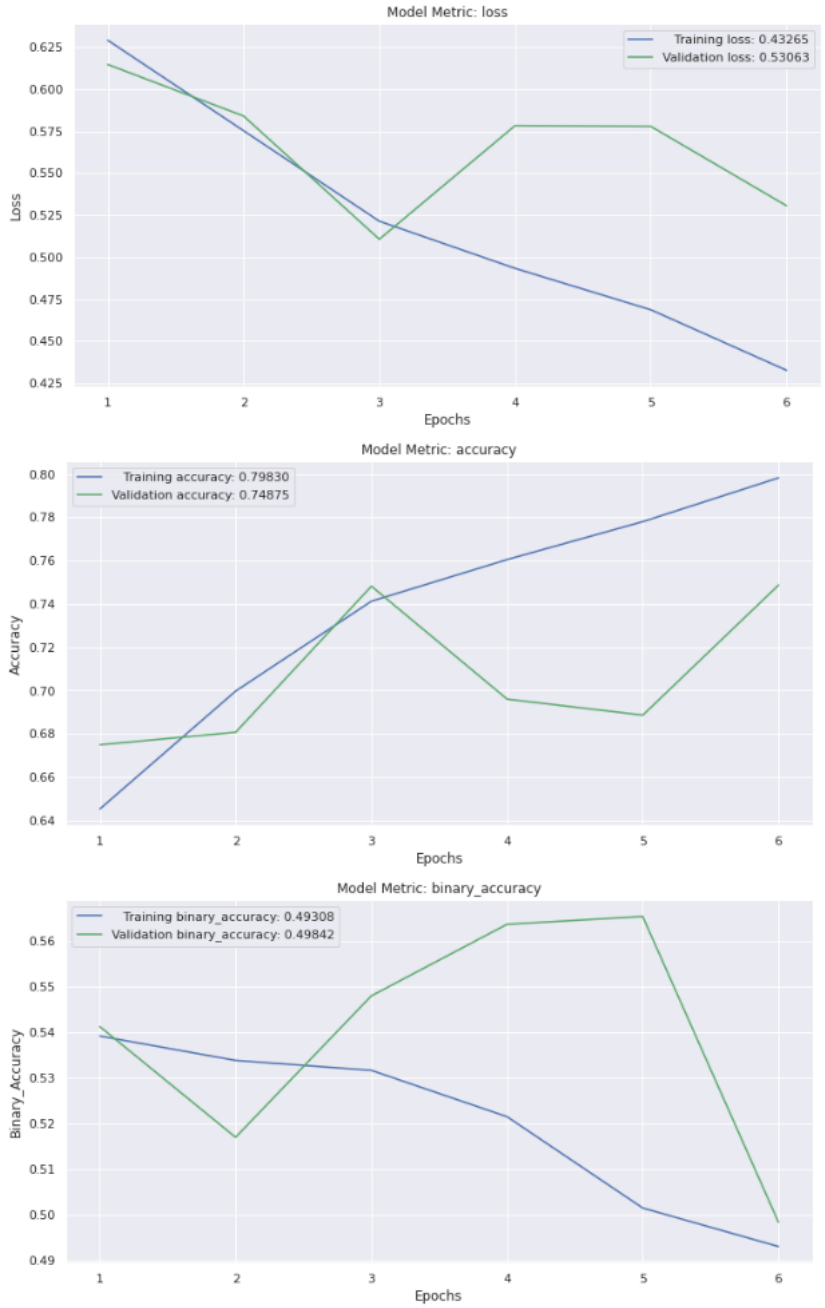


Figure 16: Model 3 *GRU for gender detection – Loss, accuracy and binary accuracy plots*

Comparing the three models of gender detection

As we can conclude from the above results, comparing the evaluation outputs of the three models, using validation data, the model that should be selected as the best is **Model 1**, as it presents the lowest validation loss from the three models deployed for gender prediction, with score 0.249.

	Model 1	Model 2	Model 3
accuracy	0.891	0.831	0.748
loss	0.249	0.349	0.510

Table 1: Gender detection *Comparison of the three models' results*

So, this is the model that will be used for the gender prediction and evaluation with the use of an unseen test set.

Model Evaluation and Prediction using unseen data

The data that will be used in the evaluation and prediction procedures are the test data, which have not been used at all during the models' training procedure. This is the reason why they represent unseen data that will be useful for an objective valuation of the best model's performance, concerning gender detection.

Model Evaluation using test data

```
24/24 - 0s - loss: 0.2564 - accuracy: 0.8947 - binary_accuracy: 0.8947 - 285ms/epoch - 12ms/step

Test categorical_crossentropy: 0.25635629892349243

Test accuracy: 89.475 %
```

Output 12: Model 1 *CNN for gender detection – Evaluation on test data*

As we can observe in **Output 12** above, the accuracy metric scores at 0.894 using the unseen test data, which is a high enough score and very close to the validation accuracy score of 0.891 (using the validation data in the previous evaluation process), revealing that the model has not underfitted or overfitted.

Prediction output (for 50 values)

In **Output 13**, the predicted values concerning the gender class of 50 images have been displayed.

```
array([0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1, 1, 0])
```

Output 13: Model 1 *CNN for gender detection – Predictions output*

Confusion matrix and Classification report

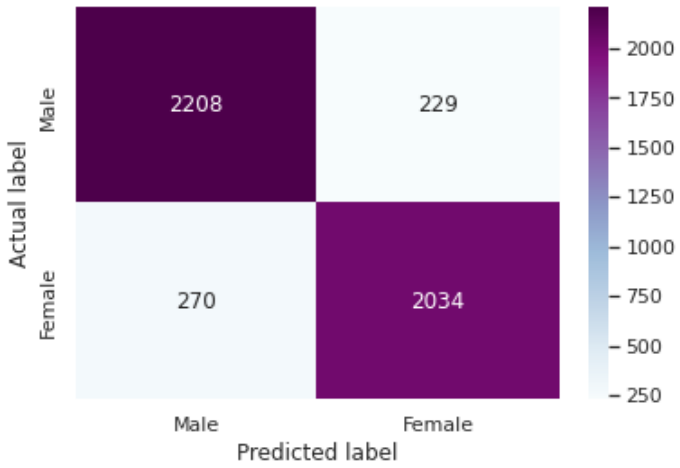


Figure 17: Model 1 *CNN for gender detection – Confusion matrix*

Each cell (i,j) in the confusion matrix contains the number of instances of class i that were predicted to be in class j. As presented in **Figure 17**, the accumulated values on the diagonal are very high, revealing that the classifier is good for both genders. This can be additionally ensured calculating the accuracy metric, which is high for both genders (Female: 2034/2263 Male:2208/2478).

	precision	recall	f1-score	support
0	0.91	0.89	0.90	2478
1	0.88	0.90	0.89	2263
accuracy			0.89	4741
macro avg	0.89	0.89	0.89	4741
weighted avg	0.89	0.89	0.89	4741

Figure 18: Model 1 *CNN for gender detection – Classification report*

As we can observe in **Figure 18**, all the metrics score very high, revealing that the predictive ability of the model is good. Based on the support metric, the Males form a larger number of samples comparing to Females.

Checking the averages, are all 89%, revealing an appropriate classification. As we can detect, the macro average (unweighted average) is equal to the weighted average (average of the values of each column weighted by their support).

Ethnicity Detection

Experiments – Setup, Configuration

As the analyzed data did not come equally from all ethnicity classes (unbalanced), the weights used per class in the three following models, calculated by dividing each of the classes with the total, are the following.

Weights per ethnicity class:

```
{0: 1.0,
 1: 2.2260959613393165,
 2: 2.934030937215651,
 3: 2.53498427672956,
 4: 5.954755309325947}
```

Those weights will lead the model to focus on the weaker classes as well, as it will be penalized higher for wrong predictions concerning them.

❖ Model 4 – CNN

```
Model: "Model4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_27 (Conv2D)          (None, 48, 48, 32)        320

 batch_normalization_9 (Batc  (None, 48, 48, 32)       128
 hNormalization)

 max_pooling2d_18 (MaxPoolin  (None, 24, 24, 32)       0
 g2D)

 conv2d_28 (Conv2D)          (None, 24, 24, 64)        18496

 batch_normalization_10 (Bat  (None, 24, 24, 64)       256
 chNormalization)

 max_pooling2d_19 (MaxPoolin  (None, 12, 12, 64)       0
 g2D)

 conv2d_29 (Conv2D)          (None, 12, 12, 64)        36928

 max_pooling2d_20 (MaxPoolin  (None, 6, 6, 64)         0
 g2D)

 flatten_30 (Flatten)        (None, 2304)              0

 dense_64 (Dense)            (None, 256)               590080

 dropout_20 (Dropout)        (None, 256)               0

 dense_65 (Dense)            (None, 5)                 1285

=================================================================
Total params: 647,493
Trainable params: 647,301
Non-trainable params: 192
```

Summary 5: Model 4 *CNN for ethnicity detection*

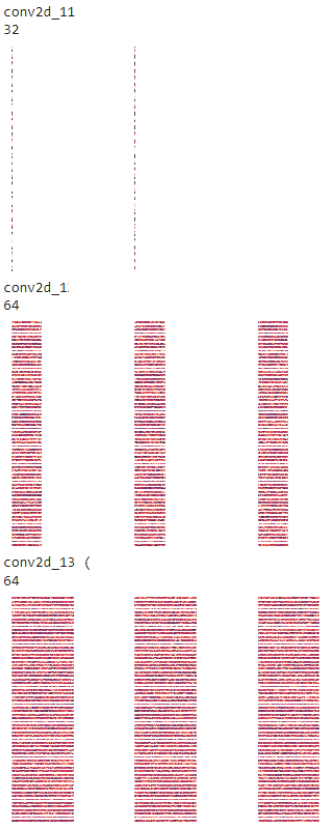


Figure 19: Model 4 *Convolutions' representation*

Proceeding to the description of the model's architecture, as we can observe in **Summary 5** above and **Figure 20** below, the specific model consists of 12 layers, after inserting an input of size 48x48x1 (input layer).

- A 2D convolution: 32 is being initialized, using a 3x3x1 size kernel. A "Relu" function is being set as activation function, in order to convert negative values to 0. Padding was set as "same", in order to maintain the same size. The original images' shape is being set as input_shape. So, the original image will pass from 32 filters in the specific layer, producing a 48x48x32 output, because of those 32 filters passed through. The params are 320 (resulting from the act 3x3x32filters + 32biased terms).
- The next layer is a BatchNormalization, followed by a MaxPooling (2,2) layer, halving its input to a 24x24x32 output, cutting the size of the conv-layers as reducing the image's dimensions.
- In the next 2D convolution, the filters put were 64 (using a 3x3x32 kernel, Relu activation function and padding "same"), making the depth 64. The params are 18496.
- Again, the next layer is a BatchNormalization, followed by a MaxPooling (2,2) layer, halving its input to a 12x12x64 output, cutting the size of the conv-layers as reducing the image's dimensions.
- Flatten the MaxPooling Layer, as image has been compressed.
- Adding a Dense layer of 256 neurons with Relu activation.
- Adding dropout to regularize the model (deleting filters).
- As we have converted the problem to multi-class, with 5 classes, 5 neurons are used, as many as the ethnicity categories, with the Softmax activation. In the fully connected layer, this final Dense will decide which is the ethnicity of the person depicted in the image.
- The total system's params are 647493, as the sum of all the system's params.

For the model's compile, categorical_crossentropy loss-function has been used, as a multi-class problem with 5 classes has to be solved, "Adam" was set as optimizer and "accuracy" as metric.

The hyperparameters used for the model's training are:

- Batch_size, set as 200, indicating the number of subsamples in which the dataset will split.
- Epochs, set as 180, which is the number of times it will go through all the data.
- Early stoppings/callbacks: using 'val_loss' as a criterion to stop training, 0 as the minimum change in the monitored quantity to qualify as an improvement, and patience as 4 epochs before stopping.

After the model's fitting, the model's weights and the model were saved, so that they could be loaded if needed. The model's history and the metrics' evolution through epochs are displayed in **Appendix 4** of appendices.

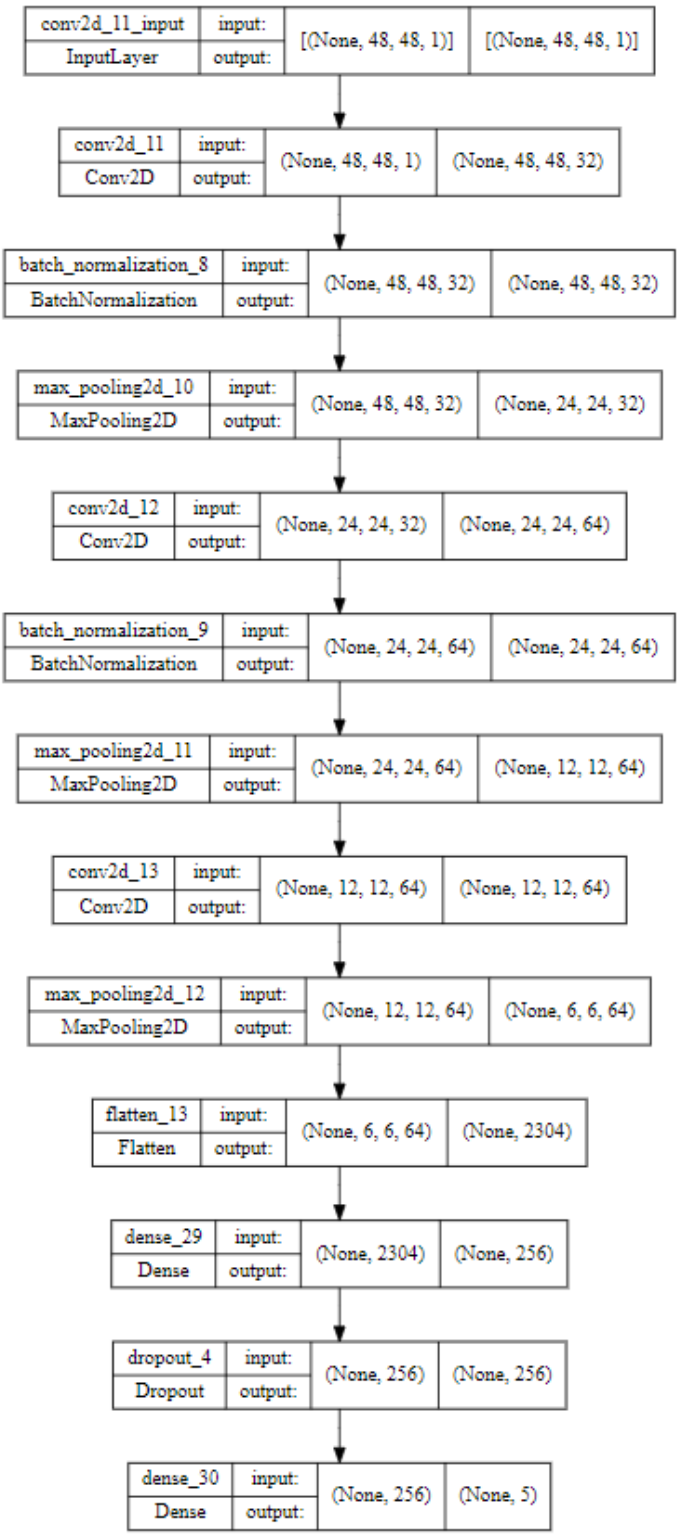


Figure 20: Model 4 *CNN for ethnicity detection – Plot of the model's architecture*

❖ **Model 5 – CNN using hyperparameters tunning**

The 5th model (Model 5) implemented with the use of hyperparameters tunning. The model's summary and architecture are presented in ***Summary 6*** and plotted in ***Figure 21*** below.

```
Model: "Model5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_8 (Flatten)         (None, 2304)              0

 dense_16 (Dense)            (None, 448)               1032640

 dense_17 (Dense)            (None, 5)                 2245

=================================================================
Total params: 1,034,885
Trainable params: 1,034,885
Non-trainable params: 0
_________________________________________________________________
```

Summary 6: Model 5 *CNN model for ethnicity detection using hyperparameters*

For the specific model's deployment, a function called "model_builder" has been created, in which a Sequential model is built. As we can observe in **Figure 21,** a layer Flattens inputs of shape (48,48). The number of units in the first Dense layer is tuned and is set that an optimal value between 40-800 should be chosen. The activation used is Relu. The final Dense: 5 will decide which is the ethnicity of the person depicted in the image. Last is the learning rate for the optimizer, which chooses an optimal value from 0.01, 0.001, or 0.0001.

For the model's compile "Adam" was used as an optimizer, sparse categorical crossentropy as a loss-function and "accuracy" as metric.

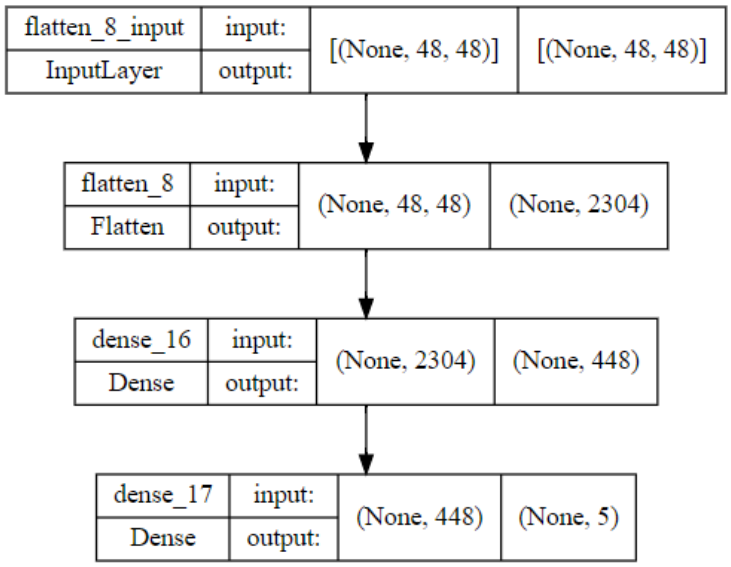


Figure 21: Model 5 *CNN for ethnicity detection using hyperparameters tunning – Plot of the model's architecture*

Following a tuner was created, using val_loss as an objective, and an early stoppings/callbacks was set: using 'val_loss' as a criterion to stop training, and 5 for patience.

The tuner searched for the ideal hyperparameters using the ethnicity data for 50 epochs, in order to get the optimal hyperparameters that were saved in “best_hps” variable. The output of tuner search is displayed in **Output 14**.

```
The hyperparameter search is complete. The optimal number of units in the first densely-connected
layer is 448 and the optimal learning rate for the optimizer
is 0.0001.
```

Output 14: Model 6 *CNN for ethnicity detection using hyperparameters tunning – Output of tuner search*

Following, those optimal hyperparameters had been used to build the model, which was trained in the data for 50 epochs (and used the class_weights found earlier).

The hypermodel was re-instantiate and trained with the optimal number of epochs from above (50). The model was retrained, and its weights were saved for being able to be loaded for later use.

The model’s tunning and training history, as well as the metrics’ evolution through epochs are displayed in **Appendix 6** of appendices.

❖ **Model 6**

The 6th model is an RNN model using the GRU architecture, and its summary is seen in **Summary 7**.

```
Model: "Model6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 gru_4 (GRU)                 (None, 64)                21888

 batch_normalization_10 (Bat  (None, 64)               256
 chNormalization)

 dense_37 (Dense)            (None, 5)                 325

=================================================================
Total params: 22,469
Trainable params: 22,341
Non-trainable params: 128
_________________________________________________________________
None
```

Summary 7: Model 6 *GRU model for ethnicity detection*

The specific Sequential model displayed in **Figure 22** consists of three layers after the input layer receiving inputs of shape (48,48):

- GRU: 64, resulting to 21888 params.
- Batch normalization, resulting to 256 params.
- And the final Dense layer: 5.

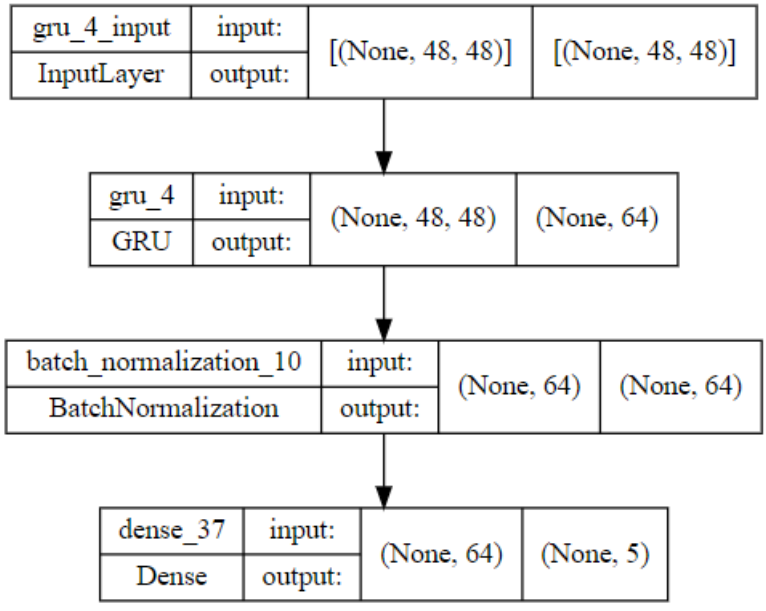


Figure 22: Model 6 *GRU model for ethnicity detection - Plot of the model's architecture*

❖ Model 4

Model evaluation

For the model's evaluation, the ethnicity validation data has been used. Validation loss scores 0.703 and validation accuracy 0.758.

```
119/119 [==============================] - 1s 4ms/step - loss: 0.7036 - accuracy: 0.7588
[0.7035977840423584, 0.7587661743164062]
```

Output 15: Model 4 *CNN for ethnicity detection – Evaluation on validation data*

Visualizations

As we can observe from the visualizations of **Figure 23**, during all epochs the training loss remains higher than the validation loss, with their difference decreasing as the epochs pass through. The same is true for the accuracy metric, but with intensely falls on validation loss in epochs 8,14 and 16.

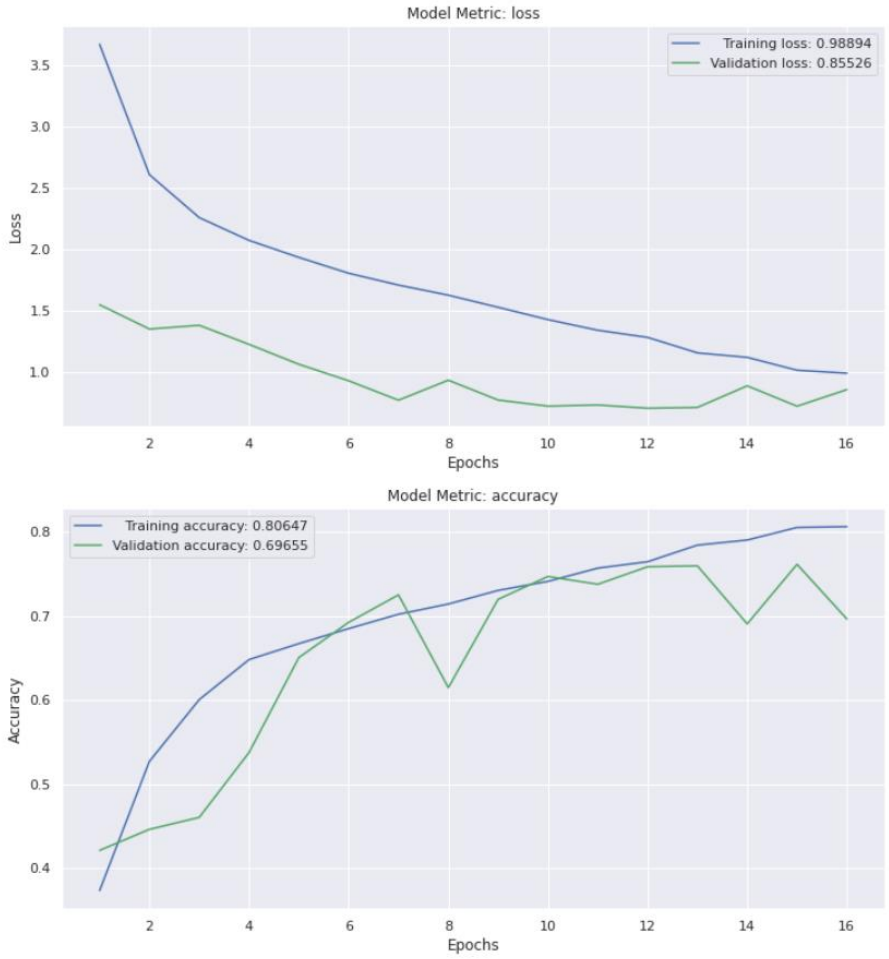


Figure 23: Model 4 *CNN for ethnicity detection – Loss and accuracy plots*

❖ **Model 5**

Model evaluation

For the model's evaluation, the ethnicity validation data has been used. Validation loss scores 0.752 and validation accuracy 0.735.

```
119/119 [==============================] - 0s 2ms/step - loss: 0.7528 - accuracy: 0.7353
[test loss, test accuracy]: [0.7528499364852905, 0.7353018522262573]
```

Output 16: Model 5 *CNN for ethnicity detection using hyperparameters tunning – Evaluation on validation data*

Visualizations

As we can observe from the visualizations of **Figure 24**, during all epochs the validation loss remains higher than the training loss, with their difference decreasing as the epochs pass. The opposite goes to accuracy, as training accuracy is higher, revealing model's overfitting.

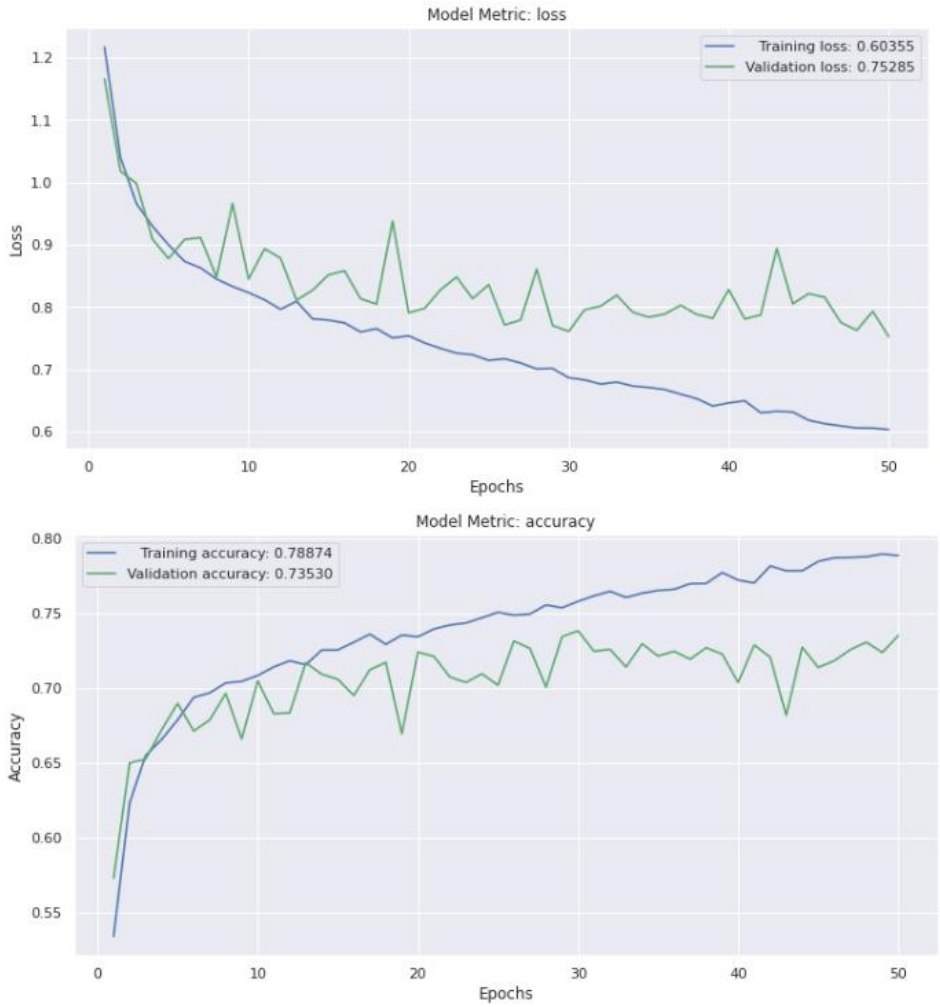


Figure 24: Model 5 *CNN for ethnicity detection using hyperparameters tunning – Loss and accuracy plots*

❖ **Model 6**

Model evaluation

For the model's evaluation, the ethnicity validation data has been used. Validation loss scores 0.960 and validation accuracy 0.642.

```
119/119 [==============================] - 0s 4ms/step - loss: 0.9602 - accuracy: 0.6422
[0.9602137207984924, 0.6422356963157654]
```

Output 17: Model 6 *GRU for ethnicity detection - Evaluation on validation data*

Visualizations

As we can observe from the visualizations of **Figure 25**, validation loss is slightly higher than training loss in all epochs and validation accuracy is slightly lower than training accuracy. Both metrics validation curves present an intense peak and drop accordingly on epochs 2 and 7.

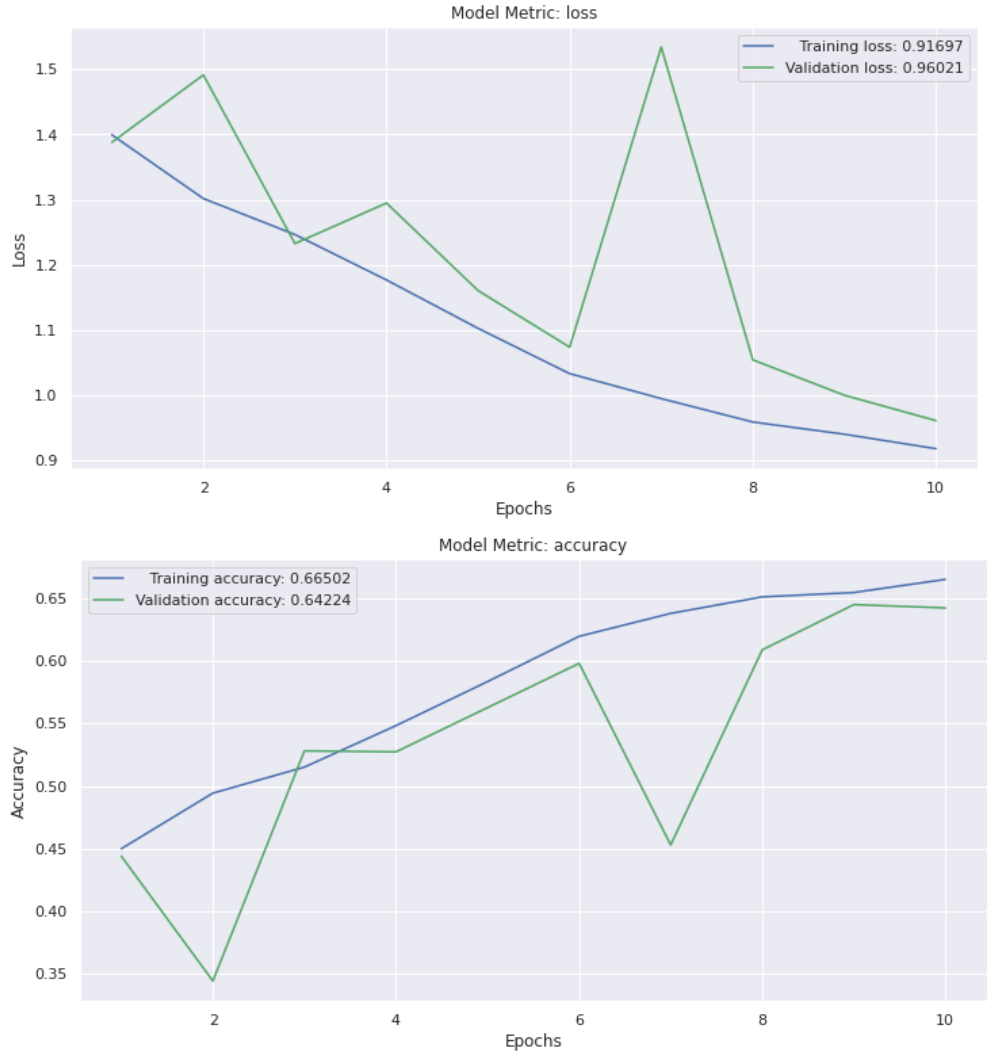


Figure 25: Model 5 *GRU for ethnicity detection – Loss and accuracy plots*

Comparing the three models of ethnicity detection

As we can conclude from the above results, comparing the evaluation outputs of the three models, using validation data, the model that should be selected as the best is **Model 4**, as it presents the lowest validation loss from the three models deployed for ethnicity prediction, with score 0.703.

	Model 4	Model 5	Model 6
accuracy	0.758	0.735	0.642
loss	0.703	0.753	0.960

Table 2: Ethnicity detection *Comparison of the three models' results*

So, this is the model that will be used for the ethnicity prediction and evaluation with the use of an unseen test set.

Model Evaluation and Prediction using unseen data

The data that will be used in the evaluation and prediction procedures are the test data, which have not been used at all during the models' training procedure. This is the reason why they represent unseen data that will be useful for an objective valuation of the best model's performance, concerning ethnicity detection.

Model Evaluation using test data

```
24/24 - 0s - loss: 0.7286 - accuracy: 0.7557 - 312ms/epoch - 13ms/step

Test categorical_crossentropy: 0.7286047339439392

Test accuracy: 75.575 %
```

Output 18: Model 4 *CNN for ethnicity detection – Evaluation on test data*

As we can observe in **Output 18** above, the accuracy metric scores at 0.755 using the unseen test data, which is a satisfactory high score and very close to the validation accuracy score of 0.758 (using the validation data in the previous evaluation process), revealing that the model has not overfitted.

Prediction output (for 50 values)

In **Output 19**, the predicted values concerning the ethnicity class of 50 images have been displayed.

```
array([3, 0, 4, 1, 1, 3, 1, 1, 0, 0, 0, 0, 3, 0, 0, 3, 2, 0, 1, 2, 3, 0,
       0, 4, 0, 0, 0, 0, 0, 4, 4, 0, 2, 2, 0, 2, 4, 3, 3, 0, 1, 3, 0, 3,
       3, 1, 0, 0, 1, 0])
```

Output 19: Model 4 *CNN for ethnicity detection – Predictions output*

Confusion matrix and Classification report

As presented in **Figure 26**, the number of accumulated values on the diagonal are high for some ethnicities and lower for others.

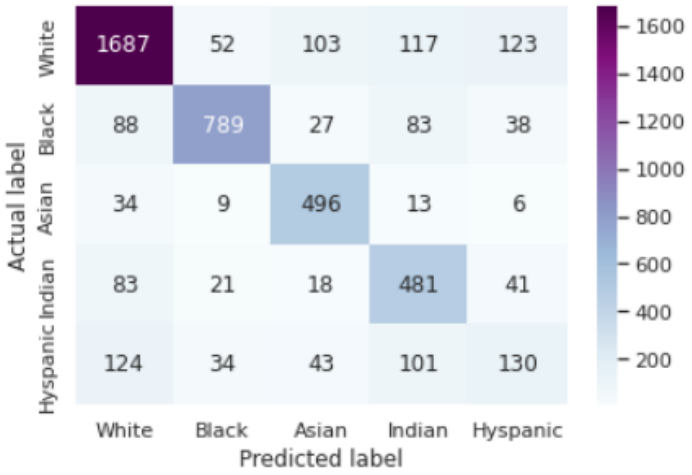


Figure 26: Model 4 *CNN for ethnicity detection – Confusion matrix*

Specifically, calculating the accuracy metric for each ethnicity class, the model seems to be:

- Extremely good in classifying Whites and Blacks, with accuracy scores 0.83 and 0.87 respectively.
- Very good in classifying Asians (accuracy score: 0.72), even though the data referring to this ethnicity are less comparing to other ethnicities.
- Poor in Indians classification (accuracy score 0.66). and very poor in classifying Hispanic people (accuracy score: 0.38), which are mostly classified as Whites or Indians.
- Also, the Asian and the Indian people are commonly mistaken for White by the model.

Precision and recall could be more informative, as the dataset is imbalanced in terms of ethnicity class, based on the support score seen in **Figure 27** below.

	precision	recall	f1-score	support
0	0.81	0.84	0.82	2016
1	0.77	0.87	0.82	905
2	0.89	0.72	0.80	687
3	0.75	0.61	0.67	795
4	0.30	0.38	0.34	338
accuracy			0.76	4741
macro avg	0.70	0.68	0.69	4741
weighted avg	0.77	0.76	0.76	4741

Figure 27: Model 4 *CNN for ethnicity detection – Classification report*

As we can observe in **Figure 27**, the metrics of the three top classes (Whites, Blacks and Asians) score very high, revealing that the predictive ability of the model is satisfactory. The f1 score for Indian and Hispanic class, are 0.67 and 0.34 respectively, probably due to the reduced number of images used in the model training process depicting people from those ethnicities. Checking the averages, are all around 69-76%, while macro average is lower than the weighted average for all metrics.

Age Group Detection

Experiments – Setup, Configuration

For the specific analysis, the grouped in buckets age data have been used. As those analyzed data did not come equally from all age-bucket classes (unbalanced), the weights used per class in the two out of three following models, calculated by dividing each of the classes with the total, are the following.

Weights per age bucket class:

```
{0: 2.3132588635259834,
 1: 5.822738386308068,
 2: 1.0,
 3: 1.4700617283950617,
 4: 2.42886282508924,
 5: 3.5203252032520327,
 6: 4.8751279426816785}
```

Those weights will lead the model to focus on the weaker classes as well, as it will be penalized higher for wrong predictions concerning them.

❖ **Model 7**

```
Model: "Model7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_14 (Conv2D)          (None, 46, 46, 32)        320

 max_pooling2d_13 (MaxPoolin  (None, 23, 23, 32)       0
 g2D)

 conv2d_15 (Conv2D)          (None, 21, 21, 64)        18496

 max_pooling2d_14 (MaxPoolin  (None, 10, 10, 64)       0
 g2D)

 conv2d_16 (Conv2D)          (None, 8, 8, 64)          36928

 max_pooling2d_15 (MaxPoolin  (None, 4, 4, 64)         0
 g2D)

 flatten_17 (Flatten)        (None, 1024)              0

 dense_38 (Dense)            (None, 7)                 7175

=================================================================
Total params: 62,919
Trainable params: 62,919
Non-trainable params: 0
_________________________________________________________________
```

Summary 8: Model 7 *CNN model for age detection*

Proceeding to the description of the model's architecture, as we can observe in **Summary 8** above and **Figure 28** below, the specific model consists of 8 layers, after inserting an input of size 48x48x1 (input layer).

- A 2D convolution: 32 is being initialized, using a 3x3x1 size kernel. A "Relu" function is being set as activation function. The original images' shape is being set as input_shape. So, the original image will pass from 32 filters in the specific layer. The params are 320 (resulting from the act 3x3x32filters + 32biased terms).
- A MaxPooling (2,2) layer follows, halving its input to a 23x23x32 output, cutting the size of the conv-layers as reducing the image's dimensions.
- A 2D convolution: 64 (using a 3x3 kernel and Relu activation function) followed by a MaxPooling (2,2) have been used twice.
- A Flatten layer is being followed by the final dense. As we have converted the problem to multi-class, with 7 classes (buckets), 7 neurons are used, as many as the age buckets, with the Softmax activation. In the fully connected layer, this final Dense will decide which is the age bucket of the person depicted in the image.
- The total system's params are 62919, as the sum of all the system's params.

For the model's compile, categorical crossentropy loss-function has been used, as a multi-class problem with 7 classes has to be solved, "Adam" was set as optimizer and "accuracy" as metric.

The hyperparameters used for the model's training are:

- Batch_size, set as 1024, indicating the number of subsamples in which the dataset will split.
- Epochs, set as 180, which is the number of times it will go through all the data.
- Early stoppings/callbacks: using 'val_loss' as a criterion to stop training, 0 as the minimum change in the monitored quantity to qualify as an improvement, and patience as 5 epochs before stopping.

After the model's fitting, the model's weights and the model were saved, so that they could be loaded if needed. The model's history and the metrics' evolution through epochs are displayed in **Appendix 7** of appendices.

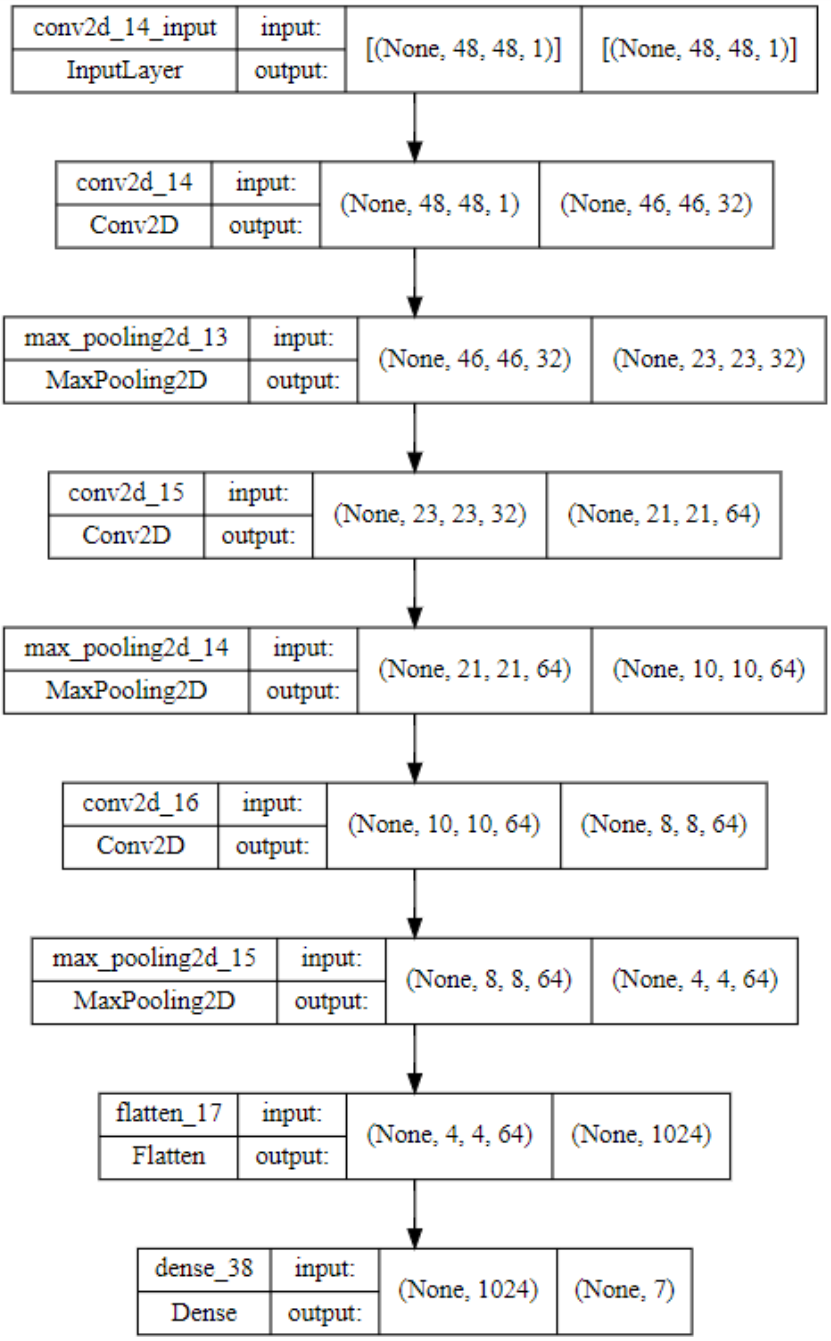


Figure 28: Model 7 *CNN for age detection – Plot of the model's architecture*

❖ **Model 8**

```
Model: "Model8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_17 (Conv2D)          (None, 48, 48, 32)        320

 batch_normalization_11 (Bat  (None, 48, 48, 32)       128
 chNormalization)

 max_pooling2d_16 (MaxPoolin  (None, 24, 24, 32)       0
 g2D)

 conv2d_18 (Conv2D)          (None, 24, 24, 64)        18496

 batch_normalization_12 (Bat  (None, 24, 24, 64)       256
 chNormalization)

 max_pooling2d_17 (MaxPoolin  (None, 12, 12, 64)       0
 g2D)

 conv2d_19 (Conv2D)          (None, 12, 12, 64)        36928

 max_pooling2d_18 (MaxPoolin  (None, 6, 6, 64)         0
 g2D)

 flatten_18 (Flatten)        (None, 2304)              0

 dense_39 (Dense)            (None, 256)               590080

 dropout_5 (Dropout)         (None, 256)               0

 dense_40 (Dense)            (None, 7)                 1799

=================================================================
Total params: 648,007
Trainable params: 647,815
Non-trainable params: 192
_________________________________________________________________
```

Summary 9: Model 8 *CNN model for age detection*

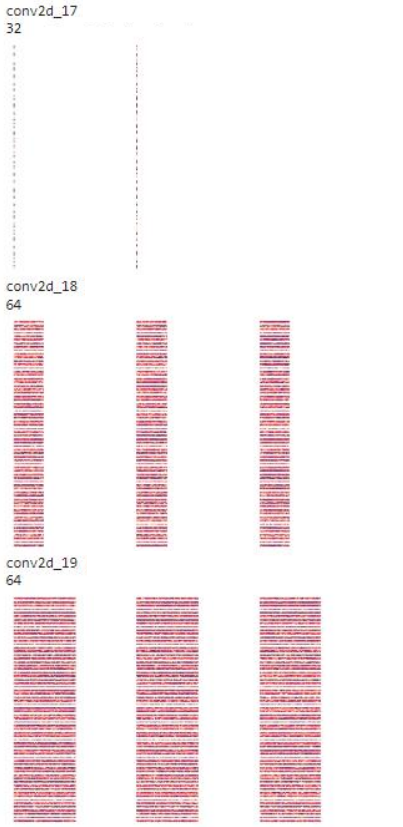


Figure 29: Model 8 *Convolutions' representation*

Proceeding to the description of the specific CNN model's architecture, as we can observe in **Summary 9** above and **Figure 30** below, the specific model consists of 12 layers, after inserting an input of size 48x48x1 (input layer).

- A 2D convolution: 32 is being initialized, using a 3x3x1 size kernel. A "Relu" function is being set as activation function. Padding was set as "same", in order to maintain the same size. The original images' shape has been set as input_shape. So, the original image will pass from 32 filters in the specific layer, producing a 48x48x32 output. The params are 320 (resulting from the act 3x3x32filters + 32biased terms).
- The next layer is a BatchNormalization, followed by a MaxPooling (2,2) layer, halving its input to a 24x24x32 output, cutting the size of the conv-layers.
- In the next 2D convolution, the filters put were 64 (using a 3x3 kernel, Relu activation function and padding "same"), making the depth 64.
- Again, the next layer is a BatchNormalization, followed by a MaxPooling (2,2) layer, halving its input to a 12x12x64 output.
- In the next 2D convolution, the filters put were 64 (using a 3x3 kernel, Relu activation function and padding "same"). A MaxPooling (2,2) layer followed, halving its input to a 6x6x64 output.
- Flatten the MaxPooling Layer, as image has been compressed.

- Adding a Dense layer of 256 neurons with Relu activation.
- Adding dropout to regularize the model (deleting filters).
- As we have converted the problem to multi-class, with 7 classes, 7 neurons are used, as many as the age buckets, with the Softmax activation. In the fully connected layer, this final Dense will decide which is the age category of the person depicted in the image.
- The system's total params are 648007.

For the model's compile, categorical_crossentropy loss-function has been used, as a multi-class problem with 7 classes has to be solved, "Adam" was set as optimizer and "accuracy" as metric. The classes weights have not been used for the training of the specific model.

The hyperparameters used for the model's training are:

- Batch_size, set as 400, indicating the number of subsamples in which the dataset will split.
- Epochs, set as 180, which is the number of times it will go through all the data.
- Early stoppings/callbacks: using 'val_loss' as a criterion to stop training, 0 as the minimum change in the monitored quantity to qualify as an improvement, and patience as 4 epochs before stopping.

After the model's fitting, the model's weights and the model were saved, so that they could be loaded if needed. The model's history and the metrics' evolution through epochs are displayed in **Appendix 8** of appendices.

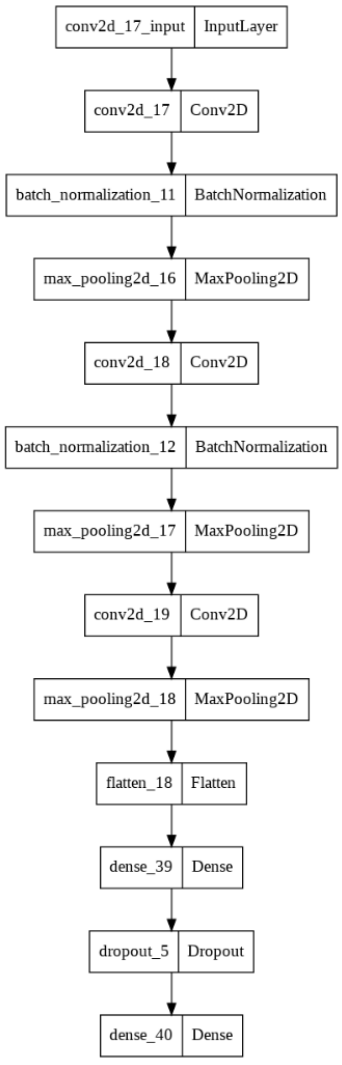


Figure 30: Model 8 *CNN for age detection – Plot of the model's architecture*

❖ **Model 9**

The 9th model is an RNN model using the GRU architecture, and its summary is seen in **Summary 10**.

```
Model: "Model9"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 gru_5 (GRU)                  (None, 64)                21888

 batch_normalization_13 (Bat  (None, 64)               256
 chNormalization)

 dense_41 (Dense)             (None, 7)                 455

=================================================================
Total params: 22,599
Trainable params: 22,471
Non-trainable params: 128
_________________________________________________________________
```

Summary 10: Model 9 *GRU model for age detection*

The specific Sequential model displayed in **Figure 31** consists of three layers after the input layer receiving inputs of shape (48,48):

- GRU: 64, resulting to 21888 params.
- Batch normalization, resulting to 256 params.
- And the final Dense layer: 7.

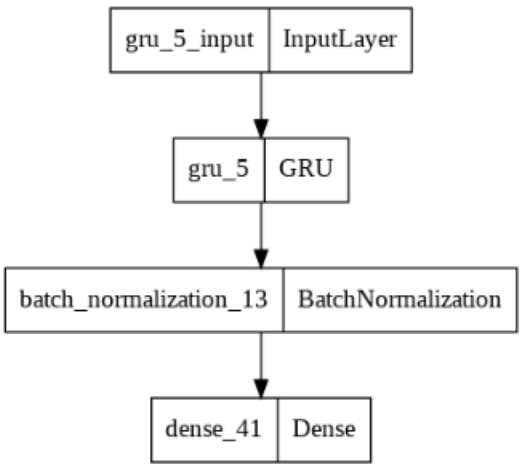


Figure 31: Model 9 *GRU for age detection – Plot of the model's architecture*

❖ Model 7

Model evaluation

For the model's evaluation, the ethnicity validation data has been used. Validation loss scores 1.306 and validation accuracy 0.465.

```
119/119 [==============================] - 0s 3ms/step - loss: 1.3066 - accuracy: 0.4653
[1.306612253189087, 0.4653308689594269]
```

Output 20: Model 7 *CNN for age prediction - Evaluation on validation data*

Visualizations

As we can observe from the visualizations of **Figure 32**, training loss is much higher than validation loss in all epochs, while both are decreasing. Training accuracy is steadily increasing while validation accuracy has fluctuations through the epochs.

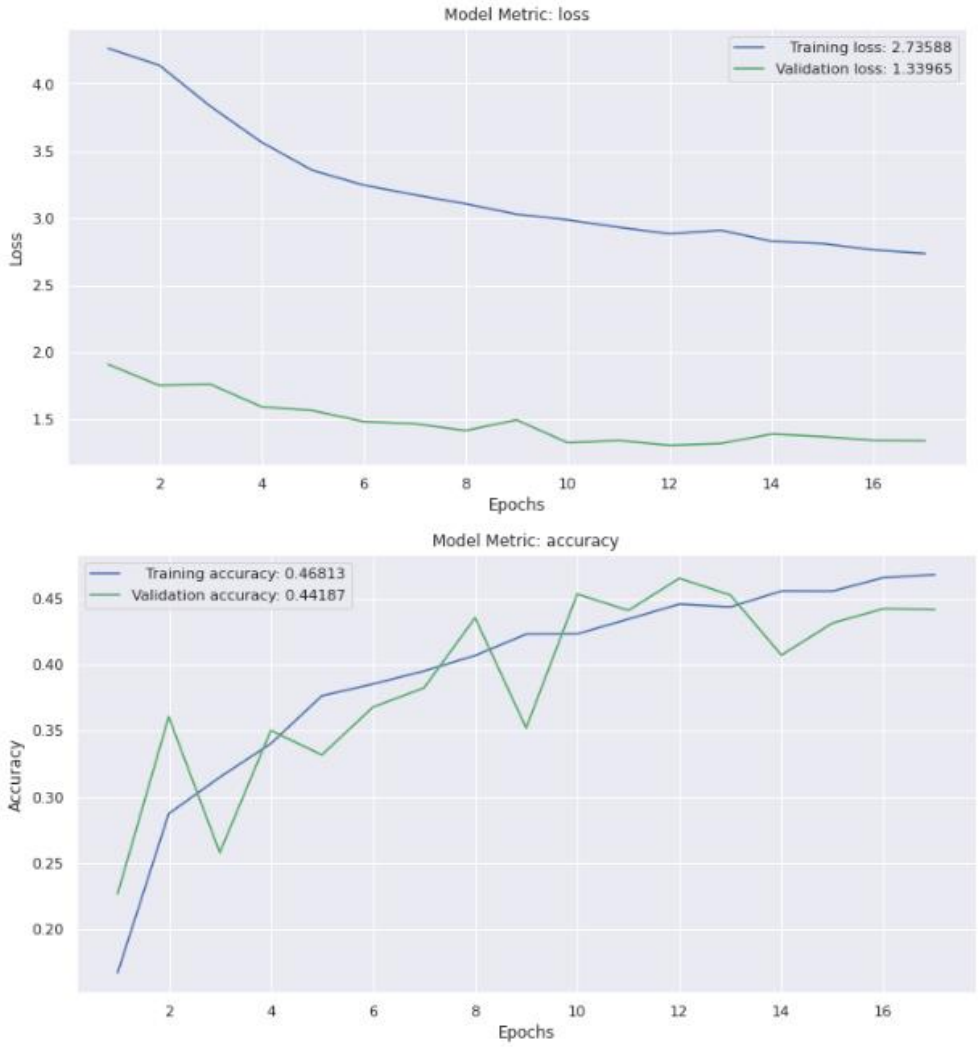


Figure 32: Model 7 *CNN for age detection – Loss and accuracy plots*

❖ **Model 8**

Model evaluation

For the model's evaluation, the ethnicity validation data has been used. Validation loss scores 1.069 and validation accuracy 0.565.

```
119/119 [==============================] - 0s 4ms/step - loss: 1.0698 - accuracy: 0.5658
[1.0698493719100952, 0.5657790899276733]
```

Output 21: Model 8 *CNN for age detection - Evaluation on validation data*

Visualizations

As we can observe from the visualizations of **Figure 33**, validation loss is higher than training loss in all epochs. In addition, after epoch 4, training accuracy is higher than validation accuracy.

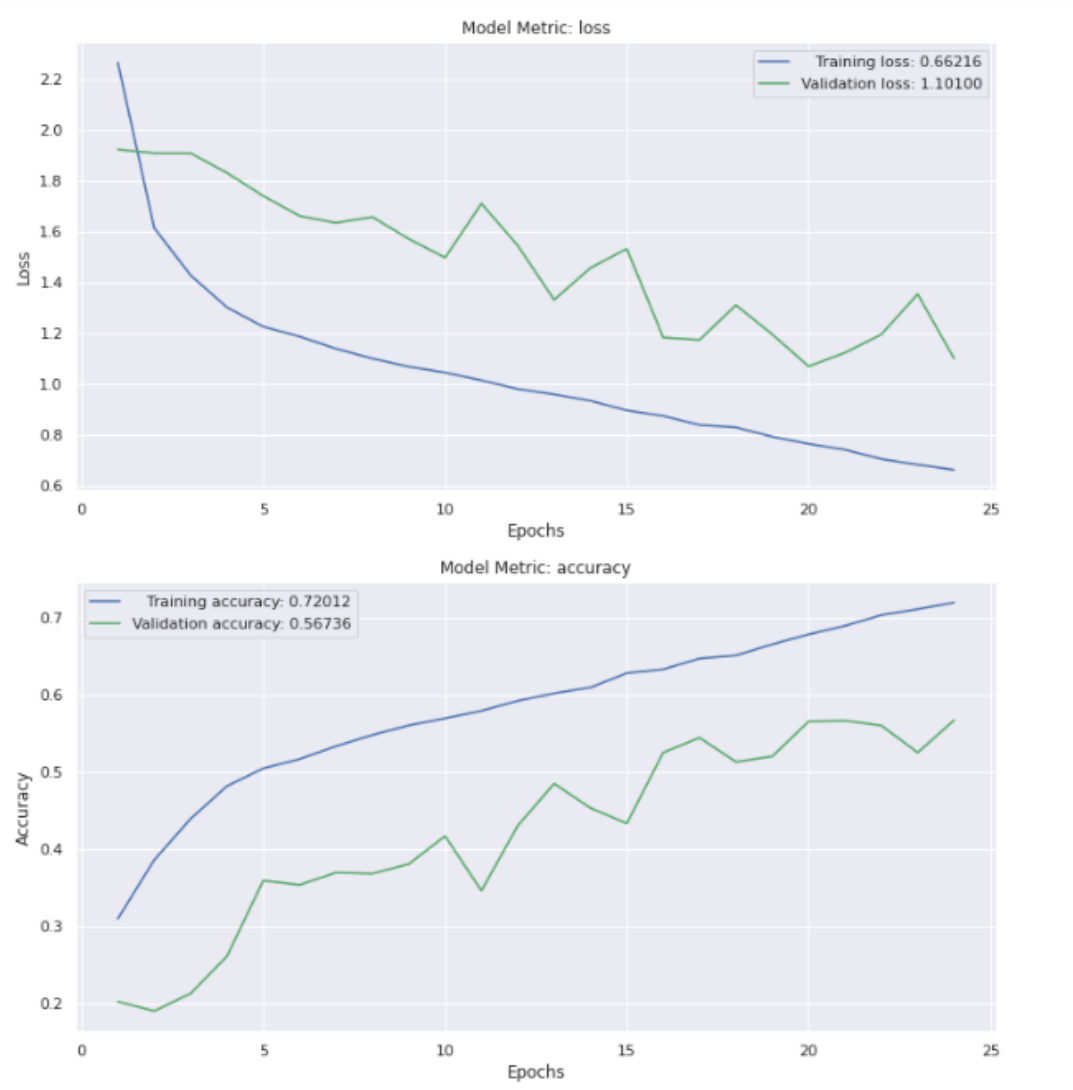


Figure 33: Model 8 *CNN for age detection – Loss and accuracy plots*

❖ **Model 9**

Model evaluation

For the model's evaluation, the ethnicity validation data has been used. Validation loss scores 1.398 and validation accuracy 0.424.

```
119/119 [==============================] - 0s 4ms/step - loss: 1.3982 - accuracy: 0.4242
[1.3982439041137695, 0.42420247197151184]
```

Output 22: Model 9 *GRU for age detection - Evaluation on validation data*

Visualizations

As we can observe from the visualizations of **Figure 34**, training loss is much higher than validation loss, while validation accuracy present significant fluctuations.

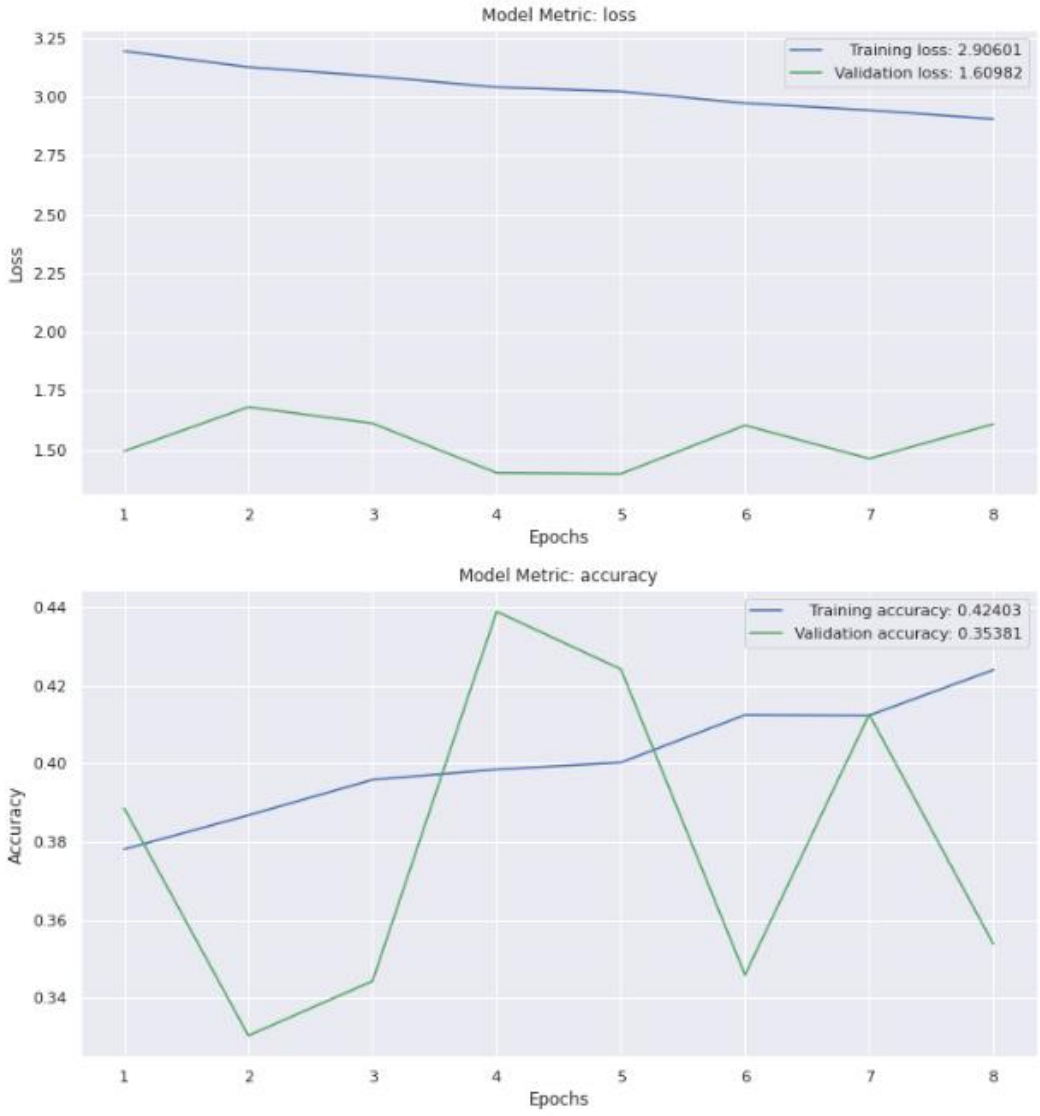


Figure 34: Model 9 *GRU for age detection – Loss and accuracy plots*

Comparing the three models of age detection

As we can conclude from the above results, summed up in **Table 3**, comparing the evaluation outputs of the three models, using validation data, the model that should be selected as the best is **Model 8**, as it presents the lowest validation loss from the three models deployed for ethnicity prediction, with score 1.069.

	Model 7	Model 8	Model 9
accuracy	0.465	0.656	0.424
loss	1.306	1.069	1.398

Table 3: Age detection *Comparison of the three models' results*

So, this is the model that will be used for the age prediction and evaluation with the use of an unseen test set.

Model Evaluation and Prediction using unseen data

The data that will be used in the evaluation and prediction procedures are the test data, which have not been used at all during the models' training procedure. This is the reason why they represent unseen data that will be useful for an objective valuation of the best model's performance, concerning age detection.

Model Evaluation using test data

```
5/5 - 0s - loss: 1.0891 - accuracy: 0.5609 - 154ms/epoch - 31ms/step

Test categorical_crossentropy: 1.0891221761703491

Test accuracy: 56.085 %
```

Output 23: Model 8 *CNN for age detection – Evaluation on test data*

As we can observe in **Output 23** above, the accuracy metric scores at 0.560 using the unseen test data, which is a not very high, but still satisfactory score and very close to the validation accuracy score of 0.565 (using the validation data in the previous evaluation process), revealing that the model has not overfitted.

Prediction output (for 50 values)

In **Output 24**, the predicted values concerning the age bucket classes of 50 images have been displayed.

```
array([0, 6, 0, 2, 4, 2, 2, 2, 2, 0, 2, 3, 3, 6, 2, 2, 3, 0, 3, 0, 3, 2,
       2, 2, 2, 5, 4, 3, 6, 2, 2, 0, 4, 2, 0, 2, 2, 2, 1, 2, 4, 4, 2, 2,
       0, 2, 1, 4, 2, 0])
```

Output 24: Model 8 *CNN for ethnicity detection – Predictions output*

Classification report

The age buckets classes refer to age groups of:

0: 0-10
1: 11-18
2: 19-29
3: 30-40
4: 41-54
5: 55-67
6: 68+

	precision	recall	f1-score	support
0	0.76	0.93	0.83	644
1	0.38	0.20	0.26	255
2	0.60	0.83	0.69	1488
3	0.43	0.26	0.32	1012
4	0.39	0.48	0.43	613
5	0.45	0.14	0.21	423
6	0.64	0.56	0.59	306
micro avg	0.56	0.56	0.56	4741
macro avg	0.52	0.48	0.48	4741
weighted avg	0.53	0.56	0.53	4741
samples avg	0.56	0.56	0.56	4741

Figure 35: Model 8 *CNN for ethnicity detection – Classification report*

As we can observe in **Figure 35**, most of the metrics score low, revealing that the predictive ability of the model is poor. This is probably caused because of wrong classification and due to the small amount of data referring to the ages included in the buckets, except of the age buckets of classes 0, 2, 3, which are the three classes with the highest support of all. Checking the averages, are all around 50-60%.

- The model has the best performance in the first age group, for which it also makes the best predictions.
- It is remarkable that age bucket 3, containing ages between 30-40, has very low precision score, referring to the fact that the people predicted truly as belonging to this age group was much less of those classified as this age group, although there were enough images for the training process depicting people on that age bucket. This probably happens because there are no particularly strong visual characteristics that can be identified to separate the individuals of the specific group from the immediately preceding and immediately following ones.

Sum up of the best models' performance for Gender, Ethnicity and Age Group

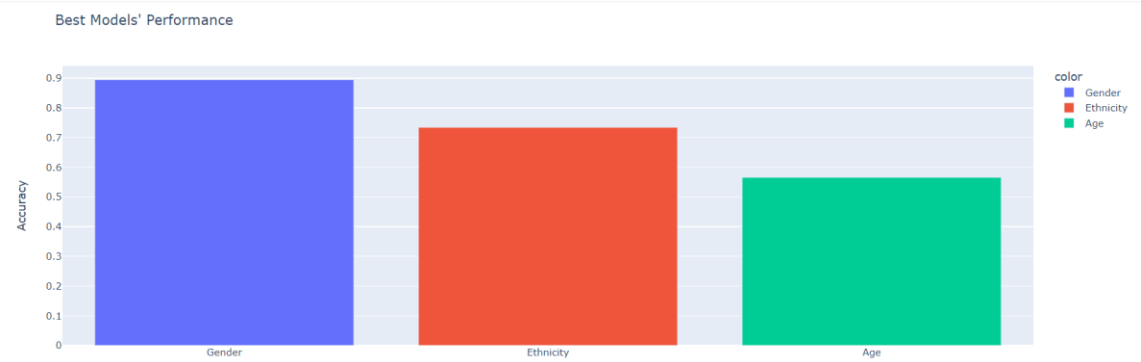


Figure 36: *Best Single-output models performance (accuracy on un-seen test data)*

	Gender	Ethnicity	Age
accuracy	0.894	0.755	0.560
loss	0.256	0.728	1.089

Table 4: Best models *- Comparison of the three models' results on the unseen test data*

From **Figure 36** and **Table 4,** comparing the accuracy metric for the three best models, using the un-seen test data, we can conclude that the gender classification model has the best performance with test accuracy score: 0.894. The model of ethnicity detection follows with test accuracy score of 0.755, while the poorest classification model refers to age group prediction, with test accuracy score of 0.560. So, even if the age broke into groups for the single-output models deployment, the accuracy score remained lowed, as this feature is probably more difficult to detect based on characteristics depicted on human images, comparing to features gender and ethnicity.

Colored dataset

As the “gender” was the feature for which the multioutput and single-output models had the best performance, in order to proceed more CNN architectures concerning gender detection, a new dataset consisting of **colored images** (RGB) has been used and presented in the following pages.

Dataset Overview

The data were uploaded in Google Drive and some folders were created. The first folder is called ‘Image Classification’ and it contains another one called ‘dataset’. Three more folders are contained in the ‘dataset’ one. These are named ‘train’, ‘validation’ and ‘test’. Finally, the folders’ contents are divided into ‘female’ and ‘male’.

The number of pictures in the original dataset was quite high, for all the images to be uploaded in google drive, so a subset of each category was selected. In the ‘train’ folder there are 3000 pictures of males and 3000 of females, which were taken from the training data in the original dataset. A sample of the validation data of the dataset was chosen, to create both validation data for the evaluation of the models, and test data for the predictions of the best model. The ‘validation’ folder consists of 1000 pictures. Half of them present men and the other half women. Finally, in the ‘test’ folder there are 200 pictures of males and 200 of females. Thus, all the created datasets are balanced.

Data Processing/Annotation/Normalization

Before the creation of the models, data need to be prepared. Since deep learning models receive an input of fixed size, all images in train, validation and test dataset are required to be rescaled to one specific size. Neural networks prefer to deal with small input values as they train faster on small images. So, all the values are divided by 255, in order to be normalized to range from 0 to 1.

Algorithms, NLP architectures/systems

All the models that were created for the second dataset are pre-trained and are based to the Convolutional Neural Network algorithm. The selected models were downloaded, and modified slightly to suit the specific task, while the layers of the pre-trained model were frozen.

The four models that were chosen are VGG-16, GoogleNet/Inception, ResNet50 and EfficientNetB0.

Some general information for each model and their architecture is presented below.

- **VGG-16 Model:** VGG is a convolutional neural network design, that has been around for a long time. It was based on a study on how to make such networks denser. Small 3x3 filters are used in the network. The network is otherwise defined by its simplicity, with simply pooling layers and a fully linked layer as additional components. In VGG16 there are thirteen convolutional layers, five Max Pooling layers, and three Dense layers which sum

up to 21 layers, but it has only sixteen weight layers i.e., learnable parameters layer. Thus, 16 in VGG16 refers to the 16 layers that have weights.

- **GoodleNet/Inception Model:** The GoogLeNet architecture solved most of the problems that large networks faced, mainly through the Inception module's utilisation. The Inception/GoogleNet design is made up of 9 inception modules stacked on top of each other, with max pooling layers between them. It is made up of 22 layers (27 with the pooling layers). After the last inception module, it employs global average pooling. There are also a dropout, a linear and a softmax layer.
- **ResNet50 Model:** The most important thing with ResNet, was that it allowed to train extremely deep neural networks with 150+layers successfully. In this project a ResNet50 will be coded, which is a smaller version of the original ResNet model, and frequently used as a starting point for transfer learning. The ResNet-50 model consists of 5 stages each with a convolution and Identity block. Each convolution block has 3 convolution layers and each identity block also has 3 convolution layers. The ResNet-50 has over 23 million trainable parameters.
- **EfficientNetB0:** The EfficientNet models belong in total 8 categories, from EfficientNetB0 to EfficientNetB7. These models consist of many layers. The chosen model for this project is the EfficientNetB0 which has in total 237 layers. Among them, there are convolutional, rescaling, batch normalization, global average pooling, zero padding layers and many others.

Experiments – Setup, Configuration

VGG16 Model

The first step is to flow training, validation and test images from the corresponding directories, in batches of 32, using the corresponding generators that were created in the rescaling step. Also, the images are resized to (224,224), because VGG-16 accepts only that image size.

The VGG-16 weights are downloaded by including the top layer parameter as false. In that way, the last fully connected layer is left out and in retraining, only the training classes need to be added. Also, the input shape of the images is set to be 224,224,3.

Since VGG-16 is already trained on huge data, there is no need to train all the layers. So, they are frozen by set to be non trainable.

As all layers of VGG-16 are frozen, the last fully-connected layer needs to be modified. There are added a max pooling, a dense layer with 512 hidden units and ReLU activation, a dropout rate of 0.5 and a final sigmoid layer with 2 nodes for classification output.

The next step, is to merge the original VGG-16 layers, with the custom layers that were created. Finally, before the model's training, it needs to be compiled. For the model's compile "Adam" was used as an optimizer, sparse categorical crossentropy as a loss-function and "accuracy" as metric.

The model's summary and architecture are presented and plotted below.

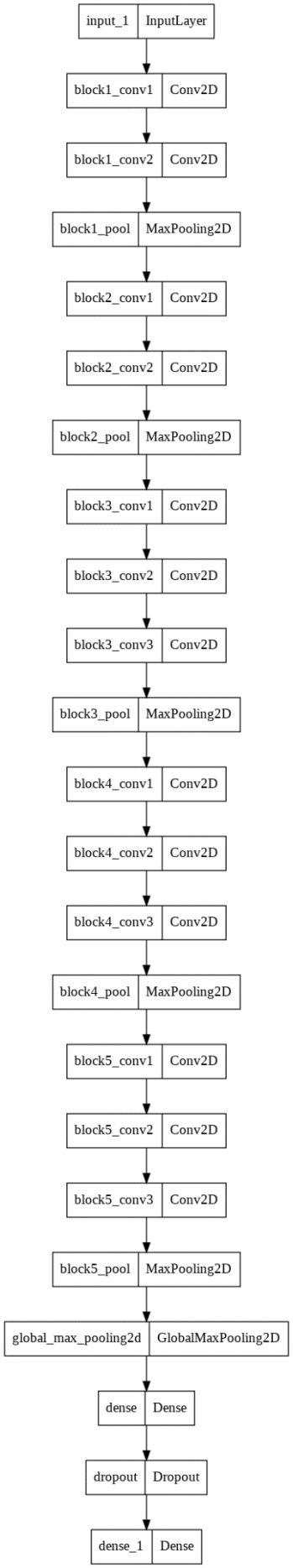


Figure 37: VGG16 Model *– Plot of the model's architecture*

Model: "model"

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
global_max_pooling2d (GlobalMaxPooling2D)	(None, 512)	0
dense (Dense)	(None, 512)	262656
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 2)	1026

Total params: 14,978,370
Trainable params: 263,682
Non-trainable params: 14,714,688

Summary 11: VGG16 Model *for gender detection*

In order to stop the training of the model when there is not an improvement in a selected metric, the callback named EarlyStopping was used. This callback was set by using the metric 'val_loss' as a criterion to stop training, patience as 5 epochs before stopping and 1 as the verbosity level. The exact same callback will be also used, for the rest of the models.

The final step, is the training of the model. The hyperparameters used for this process is the EarlyStopping callback that was created previously, a 32 batch size, 1 as the verbosity level and 30 epochs. Also, the steps per epoch are defined with the division of the total number of pictures in the train dataset by the batch size, and the validation steps from the division of the total number of pictures in the validation dataset by the batch size.

Results & Quantitative Analysis (incl. visualizations) - Qualitative & Error Analysis

For the model's evaluation, the validation data has been used. Validation loss scores 0.216 and validation accuracy 90.9% . The best epoch is the 21st.

The following plots depict the loss and the accuracy of the model, both on training and validation data. There are appeared to be some fluctuations around the 3rd and 8th epoch in both charts. Also, the training loss generally remains slightly higher than the validation loss, while training accuracy is lower than validation accuracy.

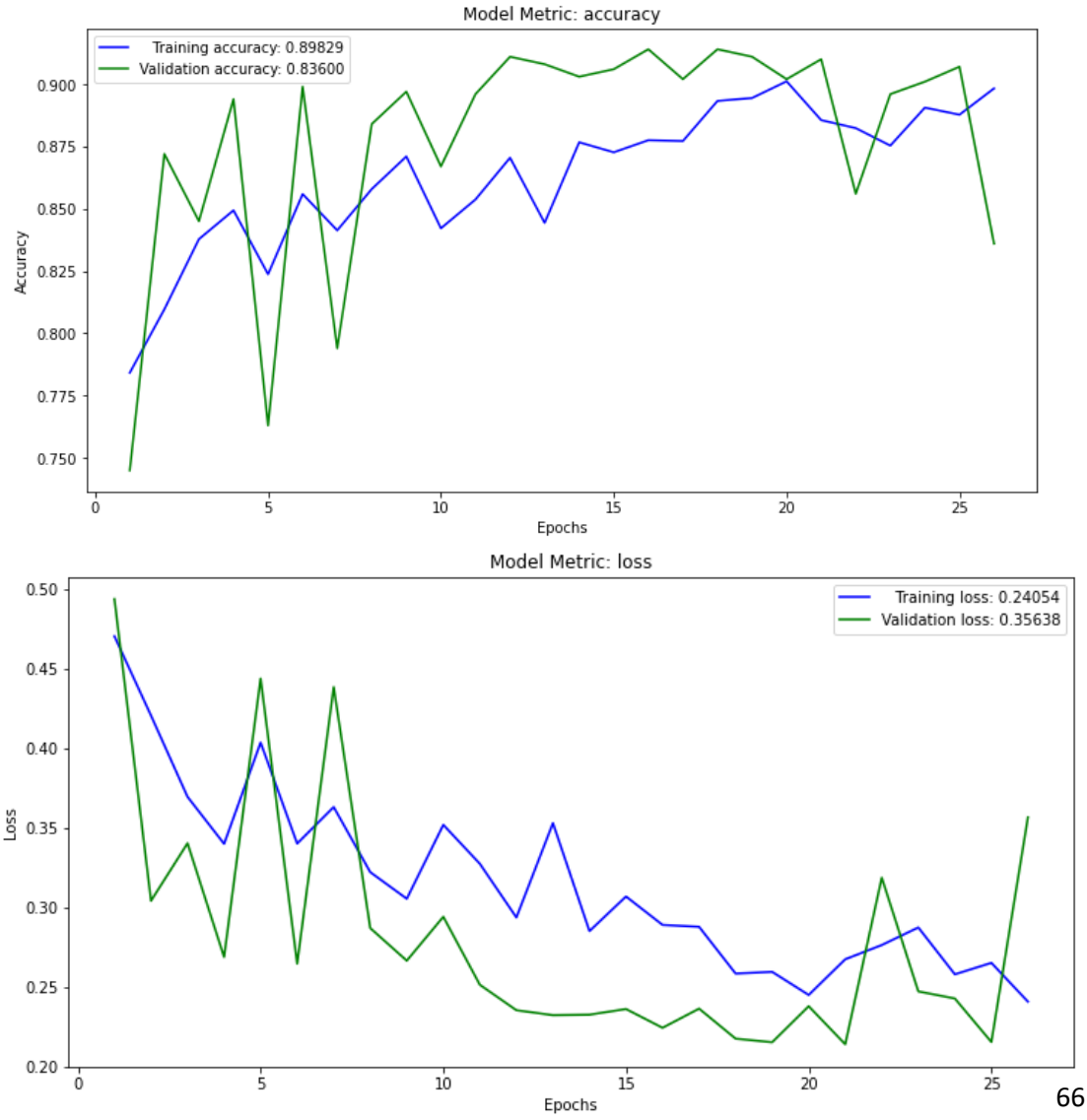


Figure 38: VGG16 Model *– Loss and accuracy plots*

GoogleNet/Inception Model

The generators that were created in the rescaling step are used to flow training, validation and test images in batches of 32. Also, for this model the images are resized to (150,150).

The models weights are downloaded, and the top layer parameter is included as false for the same reasons as in the previous model. The input shape of the images is set to be 150,150,3. Then, the layers of the pre-trained model are frozen by setting to be non trainable.

Thus, just like VGG-16, only the last layer will be changed. The performed operations were to flatten the output of the model, add a fully connected layer with 1024 hidden units and ReLU activation, a dropout rate of 0.2 and a final fully connected sigmoid layer with 2 nodes.

After that, the custom created layers are merged with the original GoogleNet layers. Finally, the model needs to be compiled before it's training. For the model's compile "Adam" was used as an optimizer, sparse categorical crossentropy as a loss-function and "accuracy" as metric.

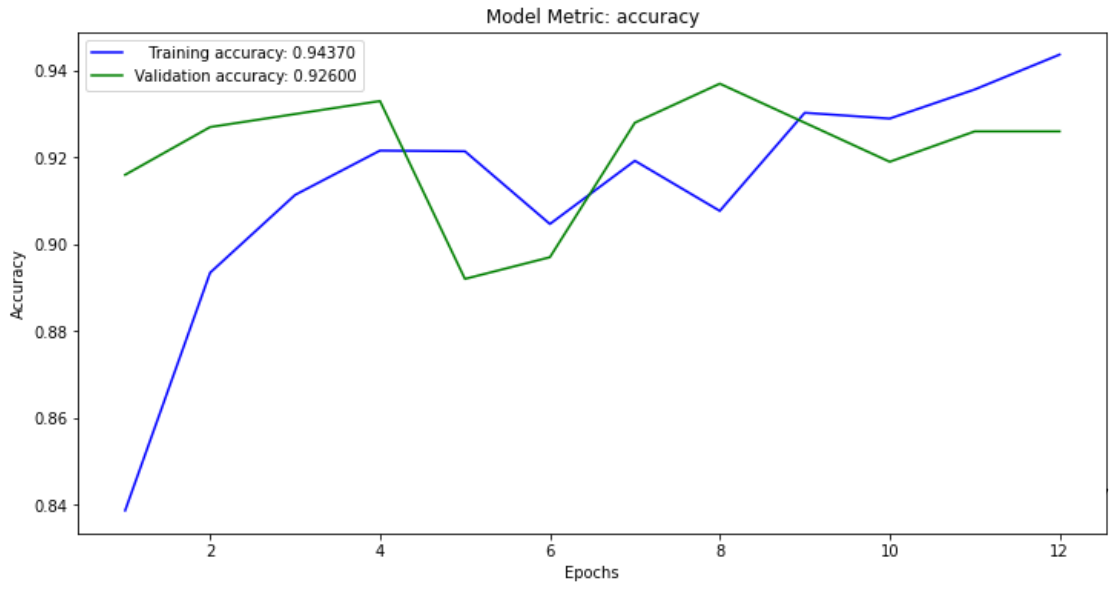
The model's summary and architecture take much space to be presented as the model has many layers. They can be found in the jupyter notebook, where the assignment was implemented.

The final step, is the training of the model. The hyperparameters used for this process is the EarlyStopping callback that was created before, a 32 batch size, 1 as the verbosity level and 30 epochs. Also, the steps per epoch are defined with the division of the total number of pictures in the train dataset by the batch size, and the validation steps from the division of the total number of pictures in the validation dataset by the batch size.

Results & Quantitative Analysis (incl. visualizations) - Qualitative & Error Analysis

For the model's evaluation, the validation data has been used. Validation loss scores 0.19 and validation accuracy 92.8% . The 7th epoch is the best.

The following plots depict the loss and the accuracy of the model, both on training and validation data. Regarding the accuracy plot, there are some fluctuations between the accuracy of the training and validation data. In the loss plot, after the 2nd epoch both training and validation loss seem to take similar values.



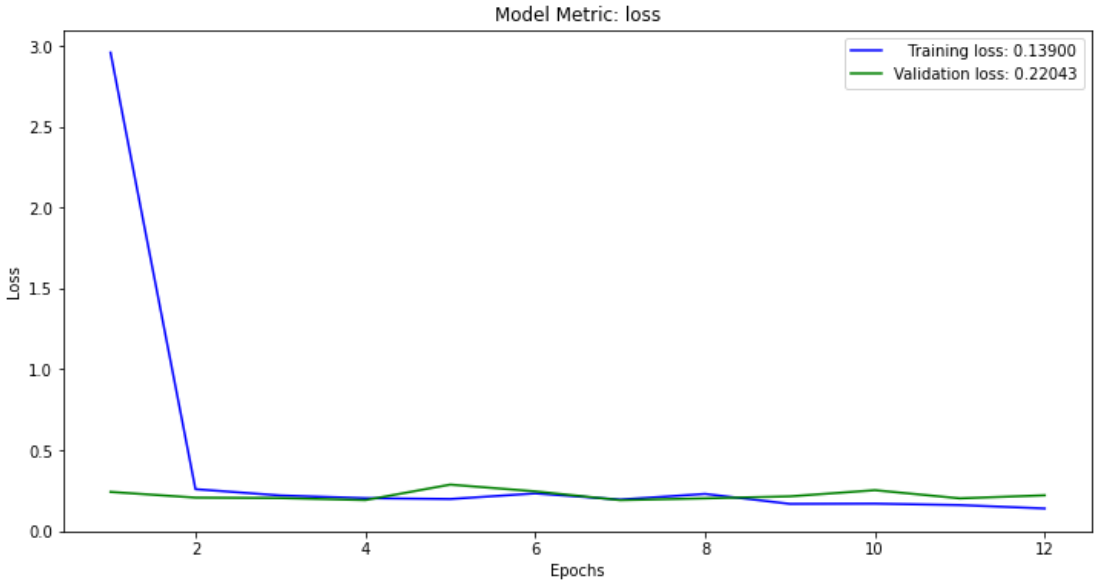


Figure 39: GoogleNet/Inception Model *– Loss and accuracy plots*

ResNet50 Model

Just like in the previous models, the training, validation and test images are read directly from the corresponding directories in batches of 32, with the use of the generators that were created in the rescaling step. The images are resized to (224,224).

The ResNet50 base model is imported and the weights are downloaded. The top layer is defined as false, in order to not load the fully connected output layer, and allow a new output layer to be added and trained. The input shape of the images is set to be 224,224,3.

The layers of the pre-trained model are frozen by setting to be non trainable. The basic architecture of the ResNet model is kept. A Dense layer is added at the end with sigmoid activation and 2 neurons, because of the 2 classes the images belong to.

Finally, before it's training, the model needs to be compiled. For the compile "Adam" was used as an optimizer, sparse categorical crossentropy as a loss-function and "accuracy" as metric.

The model's summary and architecture are presented and plotted below.

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712

 dense_2 (Dense)             (None, 2)                 4098

=================================================================
Total params: 23,591,810
Trainable params: 23,538,690
Non-trainable params: 53,120
```

Summary 12: ResNet50 Model *for gender detection*

The final step, is the training of the model. The hyperparameters used for this process is the EarlyStopping callback that was created before, a 32 batch size, 1 as the verbosity level and 30 epochs. Also, the steps per epoch are defined with the division of the total number of pictures in the train dataset by the batch size, and the validation steps from the division of the total number of pictures in the validation dataset by the batch size.

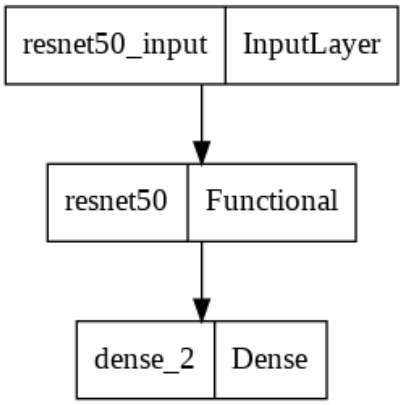
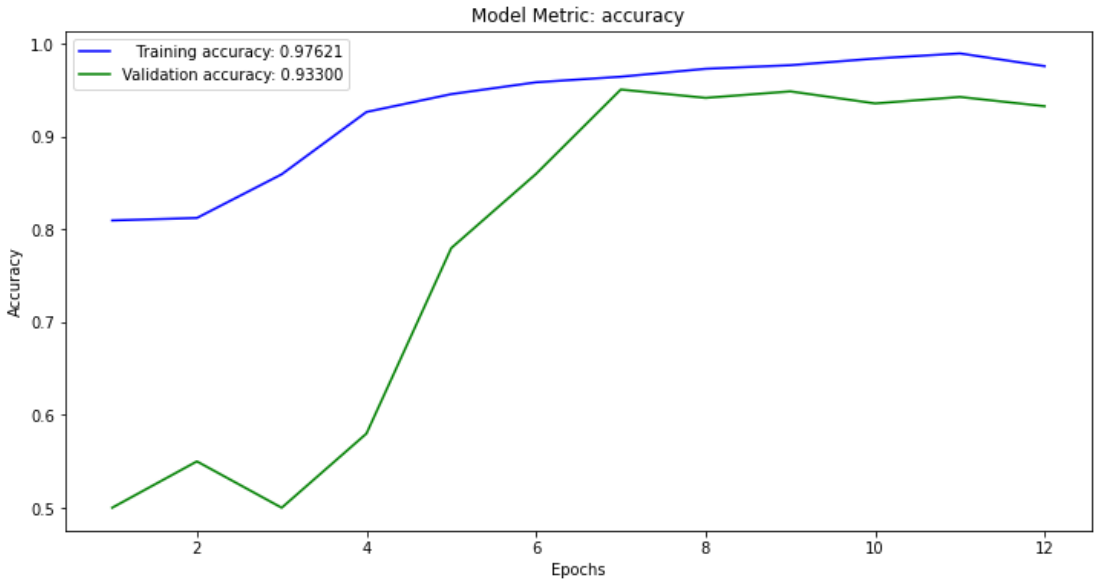


Figure 40: ResNet50 Model *- Plot of the model's architecture*

Results & Quantitative Analysis (incl. visualizations) - Qualitative & Error Analysis

For the model's evaluation, the validation data has been used. Validation loss scores 0.153 and validation accuracy 95.1% .

The following plots depict the loss and the accuracy of the model, both on training and validation data. Regarding the accuracy plot, the accuracy on the validation data is getting higher during the first epochs and takes it's max value around the 7th epoch. In the plot for the loss metric, both training and validation losses are higher at the beginning and take lower values as epochs go by.



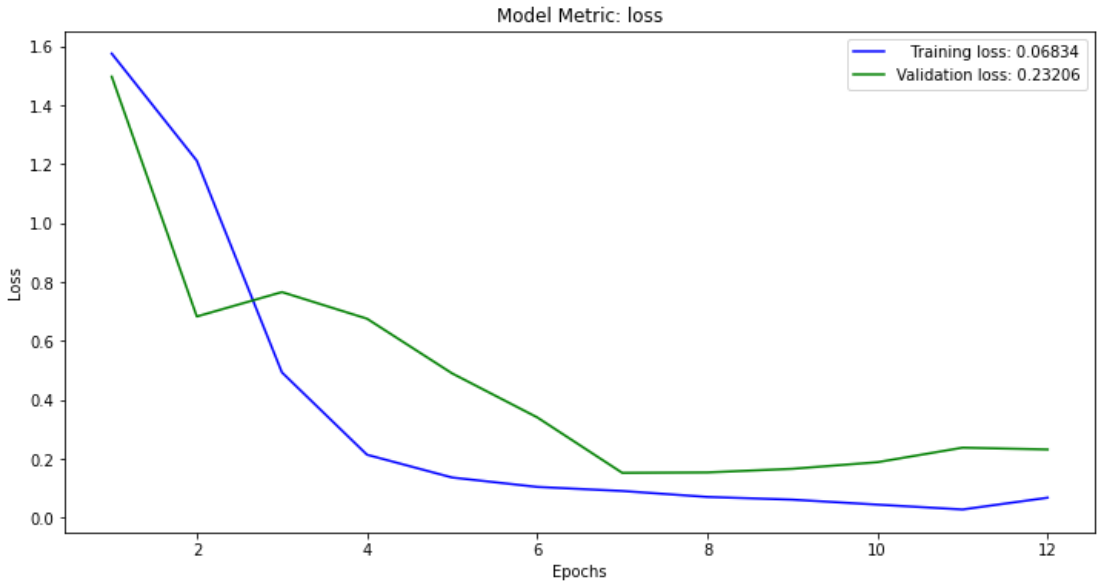


Figure 41: ResNet50 Model *– Loss and accuracy plots*

EfficientNetB0 Model

The generators that were created in the rescaling step, are used to flow training, validation and test images from the corresponding directories, in batches of 32. Also, for this model the images are resized to (224,224) like in the VGG-16 and the ResNet50 model

The models weights are downloaded, and the top layer parameter is included as false, like in all the previous models. The input shape of the images is set to be 224,224,3. Then, the layers of the pre-trained model are defined as non trainable, in order to freeze.

The last layer was modified. The performed steps were to add a global average pooling, a batch normalization layer, a dropout rate of 0.7 and a final fully connected sigmoid layer with 2 neurons.

After that, the custom created layers are merged with the original EfficientNet layers. Finally, the model needs to be compiled before it's training. For the model's compile "RMSProp" was used as an optimizer, sparse categorical crossentropy as a loss-function and "accuracy" as metric.

The model's summary and architecture take too much space to be presented, as the model has many layers. They can be found in the jupyter notebook, where the assignment was implemented.

The final step, is the training of the model. The hyperparameters used for this process is the EarlyStopping callback that was created before, a 32 batch size, 1 as the verbosity level and 30 epochs. Also, the steps per epoch are defined with the division of the total number of pictures in the train dataset by the batch size, and the validation steps from the division of the total number of pictures in the validation dataset by the batch size.

Results & Quantitative Analysis (incl. visualizations) - Qualitative & Error Analysis

For the model's evaluation, the validation data has been used. Validation loss scores 0.693 and validation accuracy 50%.

The following plots depict the loss and the accuracy of the model, both on training and validation data. In both plots, it can be seen that the validation accuracy and loss are stable and do not get better during the epochs.

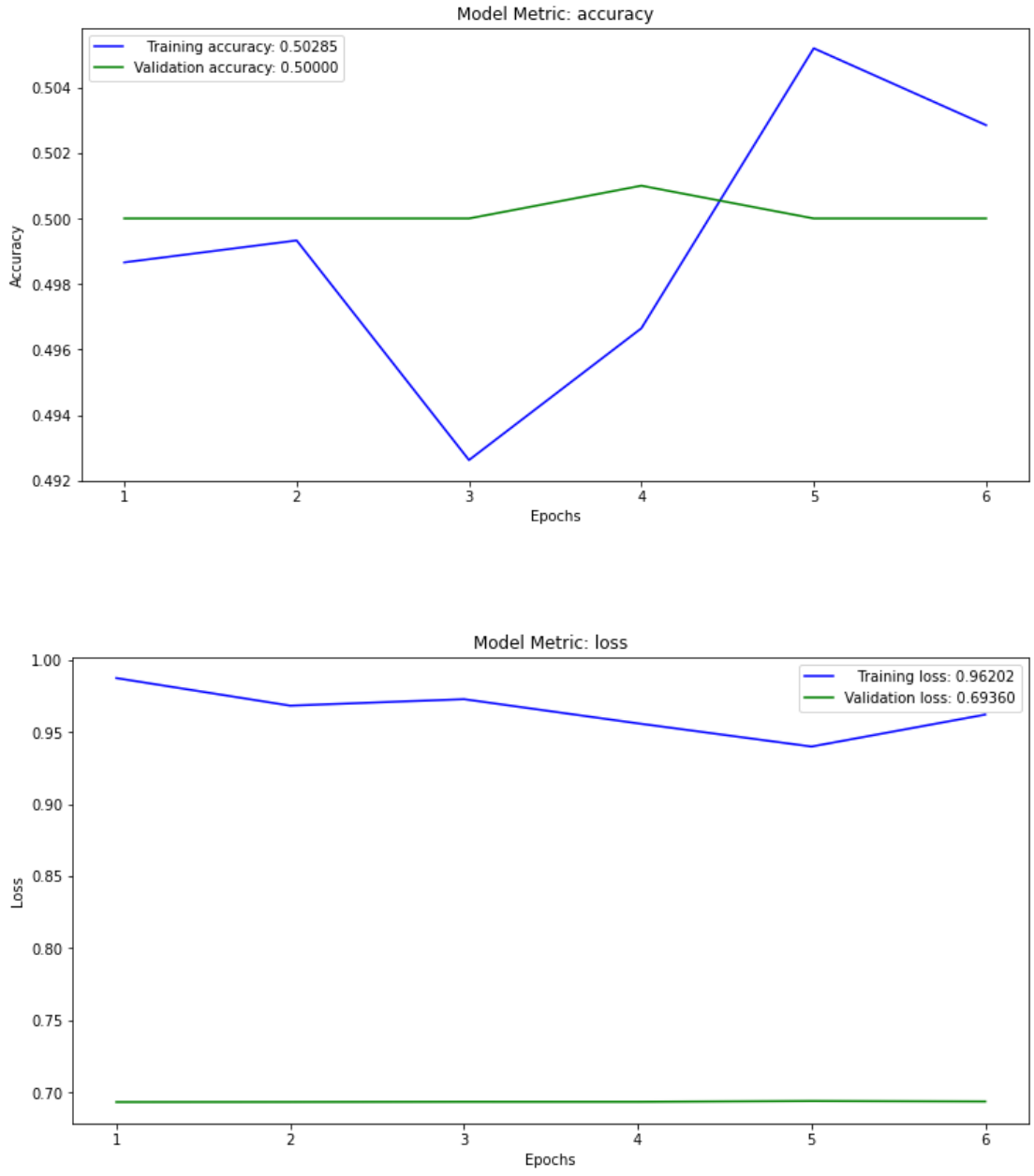


Figure 41: EfficientNetB0 Model *– Loss and accuracy plots*

Model Evaluation and Prediction using unseen data

The model with the best performance on the validation data, will be used to make predictions on new unseen images. This model will be chosen, via the comparison of the accuracy and the loss scores, of the four models. All the scores are presented on the following table.

	VGG-16	GoogleNet	ResNet50	EfficientNetB0
accuracy	90.9%	92.8%	95.1%	50%
loss	0.216	0.19	0.153	0.693

Table 5: Gender detection *Comparison of the three models' results*

The model with the lowest loss score (and highest score on the accuracy metric) is the **ResNet50**. Thus, this model will be used, to predict the people's gender, on the images that are included on the test dataset.

The predictions that the model made on the unseen data are presented in the following figure. The value 0 means that the ResNet model predicted the person on the specific image to be a male, while the value 1 corresponds to the females.

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1])
```

Output 25: ResNet50 Model *for gender detection – Predictions output*

On the classification report below, there can be seen the scores of some important metrics, for both males and females. The accuracy of the model on the new data is 94% .

```
Classification Report for ResNet50 model
              precision    recall  f1-score   support

        Male       0.95      0.94      0.94       200
      Female       0.94      0.95      0.94       200

    accuracy                           0.94       400
   macro avg       0.94      0.94      0.94       400
weighted avg       0.94      0.94      0.94       400
```

Figure 42: ResNet50 Model *for gender detection – Classification report*

The correct predictions of the selected model were 187+190=377 and the incorrect ones were 10+13=23. So, from the 400 images on the test dataset, the model classified correctly that 187 images depict a male person, and 190 a female. On the other hand, it incorrectly categorizes 13 persons to be females, whilst they are males. Similarly, 10 people were wrongly categorized as males.

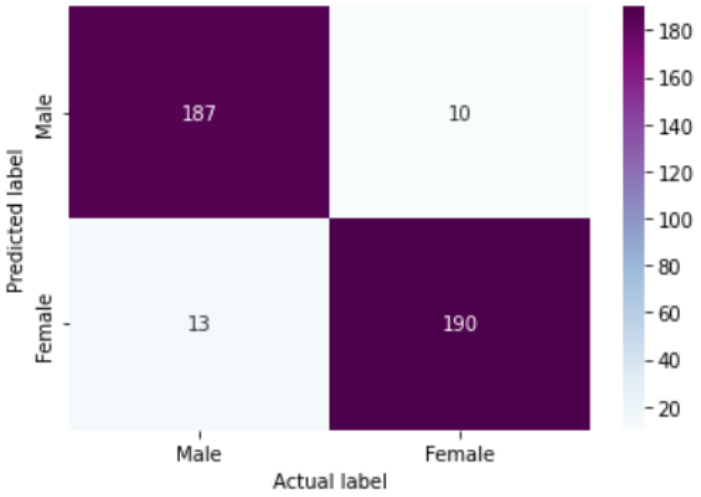


Figure 43: ResNet50 Model *for gender detection – Confusion matrix*

Discussion, Comments/Notes and Future Work

All in all, from all models deployed, the model with the lowest loss score (and highest score on the accuracy metric) for gender detection was the pre-trained model deployed using ResNet50. This additionally was the model with the most accurate predictions and the highest average metrics on classification report. Concerning ethnicity, the multioutput model (developed with the use of the uncolored dataset) had the best performance, as the loss score using the validation data was 0.724, very close to the best of the single-output validation evaluation's output (with loss score 0.728 for Model 4). Age detection models had the poorest performance and prediction ability, even with the use of age groups (in the single-output model selected - Model 8) instead of individual ages. Referring to the prediction outputs, ResNet50 is the model with the highest average accuracy score comparing to the rest best models (0.94).

As a conclusion, the pre-trained model has a significant advantage and much better performance comparing to other models, but the architectures used for deploying those models would not be easily implemented to an uncolored dataset.

Future Work

As a short-term goal for the project's development, would be to include object detection as a pre-step to image classification, in order to extract only the face from total photos and to detect whether there is more than one human depicted. Additionally, a long-term goal would be, that in case where more than one person is depicted to the image, a common demographic profile would be used, revealing relations between them (for example a mother with her child). This would enrich our vision, as it would be useful for recognizing who is the user that is actual the "shopper" and who is the user that will make the decision (the one who pays). In addition, this information would be useful as people shopping behavior and reaction to advertisement is being affected by their shopping buddies.

Members/Roles

The team that came up with the demographical characteristics' detection idea constists of four members. These are Mandyli Chrysoula-Charikleia, Papaloukas Ioannis, Papidaki Christina and Taklakoglou-Chidiroglou Argyrios. All the team members cooperated in the search of an idea that would be interesting and challenging for all of them, as well as for the datasets that were used. Argyrios was involved in the descriptive statistics analysis of the uncolored dataset and its pre-steps interpretation, while Christina prepared the data in the appropriate format that was required for the implementation of the algorithms that were presented in this project. Both of them built the single output models for age, gender and ethnicity, and Christina was responsible for their analyzes, evaluations and predictions. Ioannis constructed the multioutput model and took over its whole interpretation. Both Ioannis and Chrysoula worked on the colored dataset for the execution of the pre-trained algorithms, their evaluation and their analysis. Finally, Chrysoula and Christina were in charge for the business part of the project. They performed the market research and found possible use cases of the idea implementation in business units. During the whole project, all team members were in touch analyzing the arising problems and difficulties, using them to improve the next steps and approaches of the project.

Time Plan

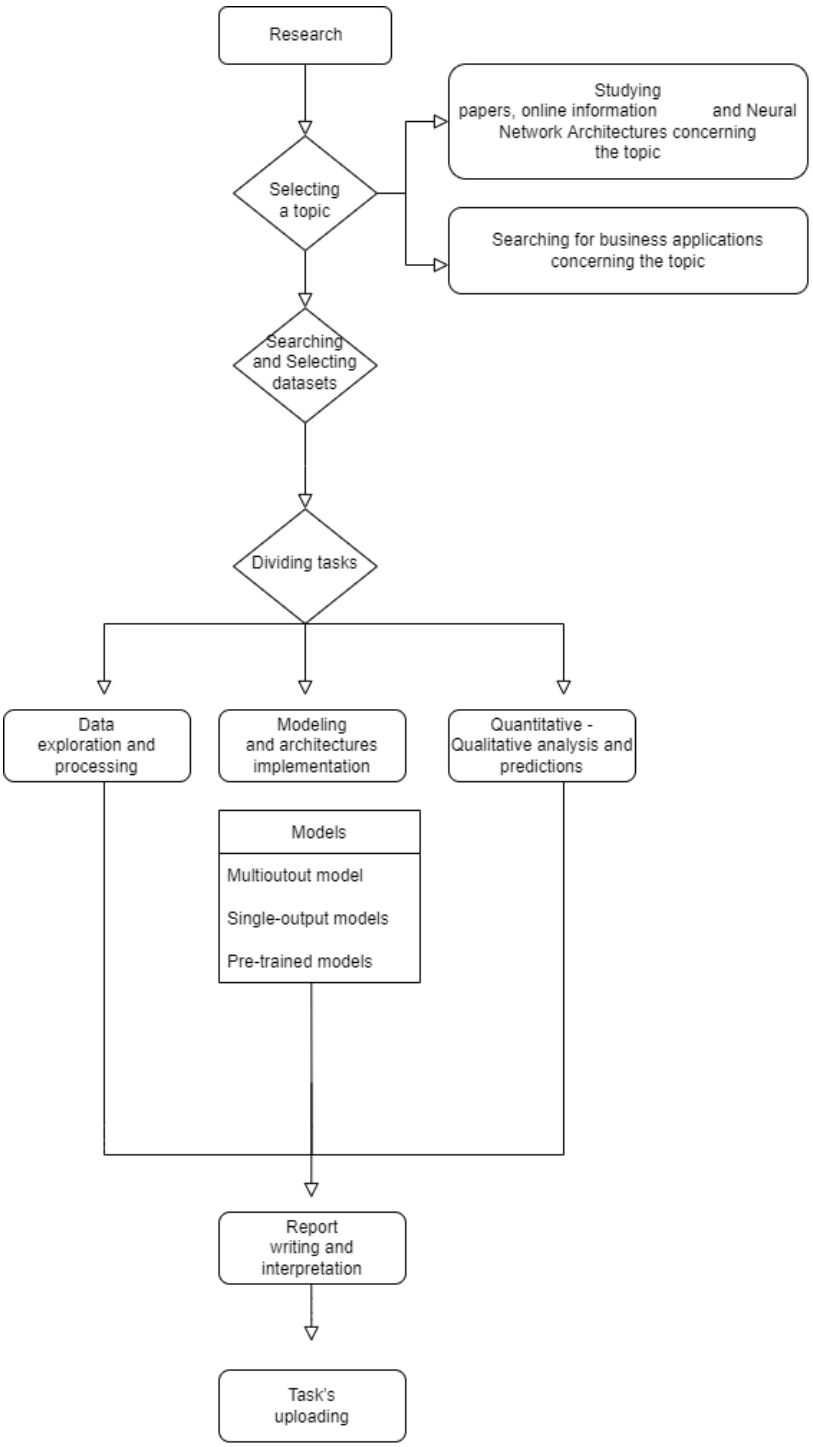


Figure 44: *Project's time plan*

Bibliography - References

Links containing the datasets used

Colored dataset: https://drive.google.com/drive/folders/1cvKxi7H1nE0RkW9D-a6CQB3YotUWGiSF?fbclid=IwAR06lsPNj87tgr2LL_afs1ke5IvErqgmFNQLTVREkTktydSjtb_q_4Z4vh8

Uncolored dataset: https://drive.google.com/file/d/1NT0DDymO46N3_J0Z5Bj211uK-ujMehLO/view?usp=sharing

Papers/Handbooks

1. https://www.researchgate.net/profile/Kuebra-Uyar/publication/330279739_Gender_Classification_with_A_Novel_Convolutional_Neural_Network_CNN_Model_and_Comparison_with_other_Machine_Learning_and_Deep_Learning_CNN_Models/links/5cc5f658299bf12097874264/Gender-Classification-with-A-Novel-Convolutional-Neural-Network-CNN-Model-and-Comparison-with-other-Machine-Learning-and-Deep-Learning-CNN-Models.pdf
2. https://iopscience.iop.org/article/10.1088/1757-899X/928/3/032039
3. https://link.springer.com/article/10.1007/s11042-021-11614-4
4. https://sbic.org.br/lnlm/wp-content/uploads/sites/4/2019/03/vol15-no1-art3.pdf
5. https://jakevdp.github.io/PythonDataScienceHandbook/

Websites' links

6. https://www.businessnewsdaily.com/15779-small-business-marketing-demographics.html
7. https://segment.com/recipes/personalize-landing-pages-to-visitor-interests/
8. https://www.wordstream.com/dynamic-landing-pages
9. https://ppc.co/dynamic-landing-pages/
10. https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4
11. https://levity.ai/blog/image-classification-in-ai-how-it-works
12. https://viso.ai/applications/computer-vision-applications/
13. https://viso.ai/computer-vision/image-classification/
14. https://www.analyticsvidhya.com/blog/2020/10/create-image-classification-model-python-keras/
15. https://www.geeksforgeeks.org/python-image-classification-using-keras/
16. https://shadab-hussain.medium.com/building-a-convolutional-neural-network-male-vs-female-50347e2fa88b
17. https://www.pluralsight.com/guides/data-visualization-deep-learning-model-using-matplotlib
18. https://towardsdatascience.com/error-analysis-in-neural-networks-6b0785858845
19. https://python.tutorialink.com/how-to-plot-the-accuracy-and-and-loss-from-this-keras-cnn-model/
20. https://towardsdatascience.com/convolutional-neural-network-feature-map-and-filter-visualization-f75012a5a49c
21. https://medium.com/@mygreatlearning/everything-you-need-to-know-about-vgg16-7315defb5918
22. https://towardsdatascience.com/deep-learning-googlenet-explained-de8861c82765
23. https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
24. https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142
25. **Other sources useful for code scripting:** Course notes and Lab notes: Machine Learning and Content Analytics, Tensorflow, Kaggle, Stackoverflow

Appendices

1. Model 1 (Gender detection): History, Metrics through epochs

```
Epoch 1/180
76/76 [==============================] - 3s 37ms/step - loss: 0.5481 - accuracy: 0.7397 - binary_accuracy: 0.7397 - val_loss: 0.3442 - val_accuracy: 0.8500 - val_binary_accuracy: 0.8500
Epoch 2/180
76/76 [==============================] - 3s 34ms/step - loss: 0.3518 - accuracy: 0.8531 - binary_accuracy: 0.8531 - val_loss: 0.3150 - val_accuracy: 0.8637 - val_binary_accuracy: 0.8637
Epoch 3/180
76/76 [==============================] - 3s 34ms/step - loss: 0.3028 - accuracy: 0.8731 - binary_accuracy: 0.8731 - val_loss: 0.2829 - val_accuracy: 0.8713 - val_binary_accuracy: 0.8713
Epoch 4/180
76/76 [==============================] - 2s 32ms/step - loss: 0.2775 - accuracy: 0.8859 - binary_accuracy: 0.8859 - val_loss: 0.2657 - val_accuracy: 0.8829 - val_binary_accuracy: 0.8829
Epoch 5/180
76/76 [==============================] - 3s 33ms/step - loss: 0.2575 - accuracy: 0.8946 - binary_accuracy: 0.8946 - val_loss: 0.2586 - val_accuracy: 0.8832 - val_binary_accuracy: 0.8832
Epoch 6/180
76/76 [==============================] - 2s 32ms/step - loss: 0.2337 - accuracy: 0.9046 - binary_accuracy: 0.9046 - val_loss: 0.2661 - val_accuracy: 0.8782 - val_binary_accuracy: 0.8782
Epoch 7/180
76/76 [==============================] - 3s 34ms/step - loss: 0.2172 - accuracy: 0.9123 - binary_accuracy: 0.9123 - val_loss: 0.2536 - val_accuracy: 0.8861 - val_binary_accuracy: 0.8861
Epoch 8/180
76/76 [==============================] - 3s 34ms/step - loss: 0.2018 - accuracy: 0.9198 - binary_accuracy: 0.9198 - val_loss: 0.2494 - val_accuracy: 0.8911 - val_binary_accuracy: 0.8911
Epoch 9/180
76/76 [==============================] - 2s 32ms/step - loss: 0.1793 - accuracy: 0.9324 - binary_accuracy: 0.9324 - val_loss: 0.2551 - val_accuracy: 0.8869 - val_binary_accuracy: 0.8869
Epoch 10/180
76/76 [==============================] - 2s 32ms/step - loss: 0.1662 - accuracy: 0.9353 - binary_accuracy: 0.9353 - val_loss: 0.2592 - val_accuracy: 0.8945 - val_binary_accuracy: 0.8945
Epoch 11/180
76/76 [==============================] - 3s 34ms/step - loss: 0.1503 - accuracy: 0.9434 - binary_accuracy: 0.9434 - val_loss: 0.2673 - val_accuracy: 0.8916 - val_binary_accuracy: 0.8916
Epoch 12/180
75/76 [============================>.] - ETA: 0s - loss: 0.1350 - accuracy: 0.9488 - binary_accuracy: 0.9488Restoring model weights from the end of the best epoch: 8.
76/76 [==============================] - 3s 34ms/step - loss: 0.1348 - accuracy: 0.9490 - binary_accuracy: 0.9490 - val_loss: 0.2652 - val_accuracy: 0.8895 - val_binary_accuracy: 0.8895
Epoch 12: early stopping
```

2. Model 2 (Gender detection): Metrics through epochs | Hyperparameters tunning

```
Epoch 12/50
475/475 [==============================] - 2s 4ms/step - loss: 0.4046 - accuracy: 0.8328 - binary_accuracy: 0.4809 - val_loss: 0.3661 - val_accuracy: 0.8442 - val_binary_accuracy: 0.4798
Epoch 13/50
475/475 [==============================] - 2s 4ms/step - loss: 0.4020 - accuracy: 0.8318 - binary_accuracy: 0.4835 - val_loss: 0.3865 - val_accuracy: 0.8334 - val_binary_accuracy: 0.4706
Epoch 14/50
475/475 [==============================] - 2s 3ms/step - loss: 0.4100 - accuracy: 0.8313 - binary_accuracy: 0.4755 - val_loss: 0.4008 - val_accuracy: 0.8212 - val_binary_accuracy: 0.4366
Epoch 15/50
475/475 [==============================] - 2s 4ms/step - loss: 0.3925 - accuracy: 0.8396 - binary_accuracy: 0.4798 - val_loss: 0.3598 - val_accuracy: 0.8458 - val_binary_accuracy: 0.4825
Epoch 16/50
475/475 [==============================] - 2s 4ms/step - loss: 0.3995 - accuracy: 0.8371 - binary_accuracy: 0.4771 - val_loss: 0.3657 - val_accuracy: 0.8402 - val_binary_accuracy: 0.5050
Epoch 17/50
475/475 [==============================] - 2s 4ms/step - loss: 0.3934 - accuracy: 0.8374 - binary_accuracy: 0.4771 - val_loss: 0.3565 - val_accuracy: 0.8474 - val_binary_accuracy: 0.4768
Epoch 18/50
475/475 [==============================] - 2s 4ms/step - loss: 0.3886 - accuracy: 0.8414 - binary_accuracy: 0.4757 - val_loss: 0.4884 - val_accuracy: 0.7859 - val_binary_accuracy: 0.5498
Epoch 19/50
475/475 [==============================] - 2s 4ms/step - loss: 0.3953 - accuracy: 0.8356 - binary_accuracy: 0.4769 - val_loss: 0.3768 - val_accuracy: 0.8334 - val_binary_accuracy: 0.5152
Epoch 20/50

Epoch 39/50
475/475 [==============================] - 2s 4ms/step - loss: 0.3721 - accuracy: 0.8500 - binary_accuracy: 0.4723 - val_loss: 0.4510 - val_accuracy: 0.7967 - val_binary_accuracy: 0.4249
Epoch 40/50
475/475 [==============================] - 2s 3ms/step - loss: 0.3708 - accuracy: 0.8510 - binary_accuracy: 0.4692 - val_loss: 0.4333 - val_accuracy: 0.8020 - val_binary_accuracy: 0.4081
Epoch 41/50
475/475 [==============================] - 2s 3ms/step - loss: 0.3655 - accuracy: 0.8508 - binary_accuracy: 0.4698 - val_loss: 0.3781 - val_accuracy: 0.8347 - val_binary_accuracy: 0.4347
Epoch 42/50
475/475 [==============================] - 2s 3ms/step - loss: 0.3797 - accuracy: 0.8459 - binary_accuracy: 0.4669 - val_loss: 0.3667 - val_accuracy: 0.8371 - val_binary_accuracy: 0.4947
Epoch 43/50
475/475 [==============================] - 2s 4ms/step - loss: 0.3633 - accuracy: 0.8542 - binary_accuracy: 0.4688 - val_loss: 0.3577 - val_accuracy: 0.8431 - val_binary_accuracy: 0.4681
Epoch 44/50
475/475 [==============================] - 2s 3ms/step - loss: 0.3657 - accuracy: 0.8521 - binary_accuracy: 0.4684 - val_loss: 0.3518 - val_accuracy: 0.8426 - val_binary_accuracy: 0.4844
Epoch 45/50
475/475 [==============================] - 2s 3ms/step - loss: 0.3623 - accuracy: 0.8538 - binary_accuracy: 0.4690 - val_loss: 0.3561 - val_accuracy: 0.8413 - val_binary_accuracy: 0.4938
Epoch 46/50
475/475 [==============================] - 3s 7ms/step - loss: 0.3694 - accuracy: 0.8482 - binary_accuracy: 0.4683 - val_loss: 0.3887 - val_accuracy: 0.8339 - val_binary_accuracy: 0.5173
Epoch 47/50
475/475 [==============================] - 2s 4ms/step - loss: 0.3684 - accuracy: 0.8493 - binary_accuracy: 0.4662 - val_loss: 0.3502 - val_accuracy: 0.8434 - val_binary_accuracy: 0.4438
Epoch 48/50
475/475 [==============================] - 2s 4ms/step - loss: 0.3645 - accuracy: 0.8526 - binary_accuracy: 0.4693 - val_loss: 0.4311 - val_accuracy: 0.7978 - val_binary_accuracy: 0.3765
Epoch 49/50
475/475 [==============================] - 2s 4ms/step - loss: 0.3674 - accuracy: 0.8489 - binary_accuracy: 0.4652 - val_loss: 0.3756 - val_accuracy: 0.8350 - val_binary_accuracy: 0.4291
Epoch 50/50
475/475 [==============================] - 2s 3ms/step - loss: 0.3700 - accuracy: 0.8508 - binary_accuracy: 0.4633 - val_loss: 0.4631 - val_accuracy: 0.8104 - val_binary_accuracy: 0.5397
Best epoch: 17
```

Re-instantiating the hypermodel and train it with the optimal number of epochs from above and retraining the model:

```
Epoch 1/17
475/475 [==============================] - 2s 4ms/step - loss: 0.8113 - accuracy: 0.7251 - binary_accuracy: 0.5121 - val_loss: 0.4237 - val_accuracy: 0.8012 - val_binary_accuracy: 0.4553
Epoch 2/17
475/475 [==============================] - 2s 3ms/step - loss: 0.4524 - accuracy: 0.7916 - binary_accuracy: 0.4956 - val_loss: 0.3943 - val_accuracy: 0.8202 - val_binary_accuracy: 0.4925
Epoch 3/17
475/475 [==============================] - 2s 3ms/step - loss: 0.4540 - accuracy: 0.7893 - binary_accuracy: 0.4803 - val_loss: 0.4103 - val_accuracy: 0.8260 - val_binary_accuracy: 0.5100
Epoch 4/17
475/475 [==============================] - 2s 3ms/step - loss: 0.4342 - accuracy: 0.8035 - binary_accuracy: 0.4736 - val_loss: 0.3728 - val_accuracy: 0.8408 - val_binary_accuracy: 0.4844
Epoch 5/17
475/475 [==============================] - 2s 3ms/step - loss: 0.4065 - accuracy: 0.8141 - binary_accuracy: 0.4874 - val_loss: 0.9753 - val_accuracy: 0.6480 - val_binary_accuracy: 0.6084
Epoch 6/17
475/475 [==============================] - 2s 3ms/step - loss: 0.3999 - accuracy: 0.8205 - binary_accuracy: 0.4810 - val_loss: 0.3649 - val_accuracy: 0.8339 - val_binary_accuracy: 0.4954
Epoch 7/17
475/475 [==============================] - 2s 3ms/step - loss: 0.3828 - accuracy: 0.8262 - binary_accuracy: 0.4957 - val_loss: 0.4692 - val_accuracy: 0.7825 - val_binary_accuracy: 0.4858
Epoch 8/17
475/475 [==============================] - 2s 3ms/step - loss: 0.3732 - accuracy: 0.8299 - binary_accuracy: 0.4904 - val_loss: 0.5766 - val_accuracy: 0.7458 - val_binary_accuracy: 0.5190
Epoch 9/17
475/475 [==============================] - 2s 3ms/step - loss: 0.3681 - accuracy: 0.8317 - binary_accuracy: 0.4958 - val_loss: 0.6001 - val_accuracy: 0.6926 - val_binary_accuracy: 0.4825
Epoch 10/17
475/475 [==============================] - 2s 3ms/step - loss: 0.3684 - accuracy: 0.8311 - binary_accuracy: 0.4856 - val_loss: 0.3568 - val_accuracy: 0.8400 - val_binary_accuracy: 0.4992
Epoch 11/17
475/475 [==============================] - 2s 4ms/step - loss: 0.3547 - accuracy: 0.8368 - binary_accuracy: 0.4969 - val_loss: 0.5100 - val_accuracy: 0.7477 - val_binary_accuracy: 0.5003
Epoch 12/17
475/475 [==============================] - 2s 4ms/step - loss: 0.3726 - accuracy: 0.8292 - binary_accuracy: 0.4909 - val_loss: 0.5198 - val_accuracy: 0.7416 - val_binary_accuracy: 0.5026
Epoch 13/17
475/475 [==============================] - 2s 3ms/step - loss: 0.3672 - accuracy: 0.8297 - binary_accuracy: 0.4936 - val_loss: 0.3440 - val_accuracy: 0.8392 - val_binary_accuracy: 0.4987
Epoch 14/17
475/475 [==============================] - 2s 3ms/step - loss: 0.3521 - accuracy: 0.8349 - binary_accuracy: 0.4958 - val_loss: 0.3314 - val_accuracy: 0.8397 - val_binary_accuracy: 0.4976
Epoch 15/17
475/475 [==============================] - 2s 3ms/step - loss: 0.3428 - accuracy: 0.8445 - binary_accuracy: 0.4946 - val_loss: 0.3832 - val_accuracy: 0.8313 - val_binary_accuracy: 0.4985
Epoch 16/17
475/475 [==============================] - 2s 3ms/step - loss: 0.3464 - accuracy: 0.8432 - binary_accuracy: 0.4946 - val_loss: 0.3393 - val_accuracy: 0.8439 - val_binary_accuracy: 0.4975
Epoch 17/17
475/475 [==============================] - 2s 3ms/step - loss: 0.3392 - accuracy: 0.8449 - binary_accuracy: 0.4963 - val_loss: 0.3491 - val_accuracy: 0.8318 - val_binary_accuracy: 0.4951
```

3. Model 3 (Gender detection): Metrics through epochs

```
Epoch 1/10
238/238 [==============================] - 4s 9ms/step - loss: 0.6292 - accuracy: 0.6453 - binary_accuracy: 0.5392 - val_loss: 0.6148 - val_accuracy: 0.6749 - val_binary_accuracy: 0.5413
Epoch 2/10
238/238 [==============================] - 2s 7ms/step - loss: 0.5753 - accuracy: 0.6998 - binary_accuracy: 0.5338 - val_loss: 0.5841 - val_accuracy: 0.6807 - val_binary_accuracy: 0.5170
Epoch 3/10
238/238 [==============================] - 2s 10ms/step - loss: 0.5214 - accuracy: 0.7413 - binary_accuracy: 0.5317 - val_loss: 0.5107 - val_accuracy: 0.7482 - val_binary_accuracy: 0.5480
Epoch 4/10
238/238 [==============================] - 2s 7ms/step - loss: 0.4934 - accuracy: 0.7605 - binary_accuracy: 0.5215 - val_loss: 0.5784 - val_accuracy: 0.6960 - val_binary_accuracy: 0.5637
Epoch 5/10
238/238 [==============================] - 2s 7ms/step - loss: 0.4687 - accuracy: 0.7781 - binary_accuracy: 0.5015 - val_loss: 0.5779 - val_accuracy: 0.6886 - val_binary_accuracy: 0.5654
Epoch 6/10
231/238 [============================>.] - ETA: 0s - loss: 0.4330 - accuracy: 0.7978 - binary_accuracy: 0.4932Restoring model weights from the end of the best epoch: 3.
238/238 [==============================] - 2s 6ms/step - loss: 0.4326 - accuracy: 0.7983 - binary_accuracy: 0.4931 - val_loss: 0.5306 - val_accuracy: 0.7487 - val_binary_accuracy: 0.4984
Epoch 6: early stopping
```

4. Model 4 (Ethnicity detection): Metrics through epochs

```
Epoch 1/180
76/76 [==============================] - 17s 31ms/step - loss: 3.6661 - accuracy: 0.3731 - val_loss: 1.5464 - val_accuracy: 0.4210
Epoch 2/180
76/76 [==============================] - 2s 23ms/step - loss: 2.6054 - accuracy: 0.5271 - val_loss: 1.3484 - val_accuracy: 0.4461
Epoch 3/180
76/76 [==============================] - 2s 23ms/step - loss: 2.2554 - accuracy: 0.6006 - val_loss: 1.3791 - val_accuracy: 0.4603
Epoch 4/180
76/76 [==============================] - 2s 23ms/step - loss: 2.0701 - accuracy: 0.6482 - val_loss: 1.2235 - val_accuracy: 0.5376
Epoch 5/180
76/76 [==============================] - 2s 23ms/step - loss: 1.9316 - accuracy: 0.6673 - val_loss: 1.0616 - val_accuracy: 0.6507
Epoch 6/180
76/76 [==============================] - 2s 23ms/step - loss: 1.8025 - accuracy: 0.6851 - val_loss: 0.9272 - val_accuracy: 0.6926
Epoch 7/180
76/76 [==============================] - 2s 23ms/step - loss: 1.7064 - accuracy: 0.7022 - val_loss: 0.7692 - val_accuracy: 0.7253
Epoch 8/180
76/76 [==============================] - 2s 22ms/step - loss: 1.6241 - accuracy: 0.7144 - val_loss: 0.9320 - val_accuracy: 0.6148
Epoch 9/180
76/76 [==============================] - 2s 23ms/step - loss: 1.5255 - accuracy: 0.7306 - val_loss: 0.7702 - val_accuracy: 0.7200
Epoch 10/180
76/76 [==============================] - 2s 23ms/step - loss: 1.4252 - accuracy: 0.7413 - val_loss: 0.7207 - val_accuracy: 0.7472
Epoch 11/180
76/76 [==============================] - 2s 23ms/step - loss: 1.3385 - accuracy: 0.7571 - val_loss: 0.7301 - val_accuracy: 0.7379
Epoch 12/180
76/76 [==============================] - 2s 23ms/step - loss: 1.2804 - accuracy: 0.7647 - val_loss: 0.7036 - val_accuracy: 0.7588
Epoch 13/180
76/76 [==============================] - 2s 25ms/step - loss: 1.1545 - accuracy: 0.7845 - val_loss: 0.7102 - val_accuracy: 0.7598
Epoch 14/180
76/76 [==============================] - 2s 23ms/step - loss: 1.1178 - accuracy: 0.7906 - val_loss: 0.8860 - val_accuracy: 0.6907
Epoch 15/180
76/76 [==============================] - 2s 23ms/step - loss: 1.0133 - accuracy: 0.8056 - val_loss: 0.7202 - val_accuracy: 0.7617
Epoch 16/180
76/76 [==============================] - ETA: 0s - loss: 0.9889 - accuracy: 0.8065Restoring model weights from the end of the best epoch: 12.
76/76 [==============================] - 2s 23ms/step - loss: 0.9889 - accuracy: 0.8065 - val_loss: 0.8553 - val_accuracy: 0.6965
Epoch 16: early stopping
```

5. Model 5 (Ethnicity detection): Metrics through epochs

```
475/475 [==============================] - 2s 3ms/step - loss: 1.6804 - accuracy: 0.7091 - val_loss: 0.8610 - val_accuracy: 0.6950
Epoch 36/50
475/475 [==============================] - 1s 3ms/step - loss: 1.6569 - accuracy: 0.7179 - val_loss: 0.9625 - val_accuracy: 0.6462
Epoch 37/50
475/475 [==============================] - 2s 3ms/step - loss: 1.6438 - accuracy: 0.7200 - val_loss: 0.8102 - val_accuracy: 0.7176
Epoch 38/50
475/475 [==============================] - 2s 3ms/step - loss: 1.6348 - accuracy: 0.7179 - val_loss: 0.9734 - val_accuracy: 0.6565
Epoch 39/50
475/475 [==============================] - 2s 3ms/step - loss: 1.6300 - accuracy: 0.7204 - val_loss: 0.8483 - val_accuracy: 0.6929
Epoch 40/50
475/475 [==============================] - 2s 4ms/step - loss: 1.5910 - accuracy: 0.7259 - val_loss: 0.8496 - val_accuracy: 0.7002
Epoch 41/50
475/475 [==============================] - 2s 4ms/step - loss: 1.6027 - accuracy: 0.7236 - val_loss: 0.9489 - val_accuracy: 0.6533
Epoch 42/50
475/475 [==============================] - 2s 3ms/step - loss: 1.5694 - accuracy: 0.7306 - val_loss: 0.9678 - val_accuracy: 0.6275
Epoch 43/50
475/475 [==============================] - 2s 3ms/step - loss: 1.5767 - accuracy: 0.7261 - val_loss: 0.8274 - val_accuracy: 0.7105
Epoch 44/50
475/475 [==============================] - 2s 3ms/step - loss: 1.5617 - accuracy: 0.7319 - val_loss: 1.0156 - val_accuracy: 0.6156
Epoch 45/50
475/475 [==============================] - 2s 3ms/step - loss: 1.5410 - accuracy: 0.7358 - val_loss: 0.9305 - val_accuracy: 0.6523
Epoch 46/50
475/475 [==============================] - 2s 3ms/step - loss: 1.5252 - accuracy: 0.7383 - val_loss: 0.9063 - val_accuracy: 0.6876
Epoch 47/50
475/475 [==============================] - 2s 3ms/step - loss: 1.5227 - accuracy: 0.7376 - val_loss: 0.8275 - val_accuracy: 0.7134
Epoch 48/50
475/475 [==============================] - 2s 3ms/step - loss: 1.5142 - accuracy: 0.7363 - val_loss: 0.9198 - val_accuracy: 0.6652
Epoch 49/50
475/475 [==============================] - 2s 3ms/step - loss: 1.5031 - accuracy: 0.7426 - val_loss: 0.9643 - val_accuracy: 0.6420
Epoch 50/50
475/475 [==============================] - 2s 3ms/step - loss: 1.4967 - accuracy: 0.7407 - val_loss: 0.8026 - val_accuracy: 0.7187
Best epoch: 50
```

Re-instantiating the hypermodel and train it with the optimal number of epochs from above and retraining the model:

```
Epoch 42/50
475/475 [==============================] - 1s 3ms/step - loss: 0.6306 - accuracy: 0.7818 - val_loss: 0.7875 - val_accuracy: 0.7208
Epoch 43/50
475/475 [==============================] - 2s 3ms/step - loss: 0.6329 - accuracy: 0.7785 - val_loss: 0.8943 - val_accuracy: 0.6820
Epoch 44/50
475/475 [==============================] - 1s 3ms/step - loss: 0.6318 - accuracy: 0.7786 - val_loss: 0.8054 - val_accuracy: 0.7274
Epoch 45/50
475/475 [==============================] - 1s 3ms/step - loss: 0.6186 - accuracy: 0.7848 - val_loss: 0.8215 - val_accuracy: 0.7139
Epoch 46/50
475/475 [==============================] - 1s 3ms/step - loss: 0.6130 - accuracy: 0.7872 - val_loss: 0.8158 - val_accuracy: 0.7184
Epoch 47/50
475/475 [==============================] - 2s 3ms/step - loss: 0.6095 - accuracy: 0.7874 - val_loss: 0.7756 - val_accuracy: 0.7258
Epoch 48/50
475/475 [==============================] - 2s 3ms/step - loss: 0.6061 - accuracy: 0.7879 - val_loss: 0.7627 - val_accuracy: 0.7308
Epoch 49/50
475/475 [==============================] - 2s 3ms/step - loss: 0.6059 - accuracy: 0.7897 - val_loss: 0.7930 - val_accuracy: 0.7240
Epoch 50/50
475/475 [==============================] - 2s 3ms/step - loss: 0.6035 - accuracy: 0.7887 - val_loss: 0.7528 - val_accuracy: 0.7353
```

6. Model 6 (Ethnicity detection): Metrics through epochs

```
Epoch 1/10
238/238 [==============================] - 4s 8ms/step - loss: 1.3995 - accuracy: 0.4499 - val_loss: 1.3880 - val_accuracy: 0.4440
Epoch 2/10
238/238 [==============================] - 2s 6ms/step - loss: 1.3015 - accuracy: 0.4943 - val_loss: 1.4915 - val_accuracy: 0.3443
Epoch 3/10
238/238 [==============================] - 2s 6ms/step - loss: 1.2462 - accuracy: 0.5151 - val_loss: 1.2323 - val_accuracy: 0.5281
Epoch 4/10
238/238 [==============================] - 2s 6ms/step - loss: 1.1763 - accuracy: 0.5482 - val_loss: 1.2945 - val_accuracy: 0.5273
Epoch 5/10
238/238 [==============================] - 2s 6ms/step - loss: 1.1020 - accuracy: 0.5834 - val_loss: 1.1600 - val_accuracy: 0.5626
Epoch 6/10
238/238 [==============================] - 1s 6ms/step - loss: 1.0324 - accuracy: 0.6195 - val_loss: 1.0731 - val_accuracy: 0.5979
Epoch 7/10
238/238 [==============================] - 2s 6ms/step - loss: 0.9940 - accuracy: 0.6379 - val_loss: 1.5343 - val_accuracy: 0.4529
Epoch 8/10
238/238 [==============================] - 2s 6ms/step - loss: 0.9580 - accuracy: 0.6510 - val_loss: 1.0535 - val_accuracy: 0.6088
Epoch 9/10
238/238 [==============================] - 2s 6ms/step - loss: 0.9392 - accuracy: 0.6545 - val_loss: 0.9991 - val_accuracy: 0.6449
Epoch 10/10
238/238 [==============================] - 2s 6ms/step - loss: 0.9170 - accuracy: 0.6650 - val_loss: 0.9602 - val_accuracy: 0.6422
```

7. Model 7 (Age detection): Metrics through epochs

```
Epoch 1/180
15/15 [==============================] - 3s 84ms/step - loss: 4.2642 - accuracy: 0.1673 - val_loss: 1.9104 - val_accuracy: 0.2270
Epoch 2/180
15/15 [==============================] - 1s 67ms/step - loss: 4.1390 - accuracy: 0.2875 - val_loss: 1.7527 - val_accuracy: 0.3607
Epoch 3/180
15/15 [==============================] - 1s 64ms/step - loss: 3.8331 - accuracy: 0.3151 - val_loss: 1.7617 - val_accuracy: 0.2581
Epoch 4/180
15/15 [==============================] - 1s 64ms/step - loss: 3.5662 - accuracy: 0.3405 - val_loss: 1.5947 - val_accuracy: 0.3504
Epoch 5/180
15/15 [==============================] - 1s 68ms/step - loss: 3.3555 - accuracy: 0.3766 - val_loss: 1.5661 - val_accuracy: 0.3319
Epoch 6/180
15/15 [==============================] - 1s 64ms/step - loss: 3.2460 - accuracy: 0.3855 - val_loss: 1.4836 - val_accuracy: 0.3680
Epoch 7/180
15/15 [==============================] - 1s 64ms/step - loss: 3.1735 - accuracy: 0.3953 - val_loss: 1.4678 - val_accuracy: 0.3825
Epoch 8/180
15/15 [==============================] - 1s 64ms/step - loss: 3.1062 - accuracy: 0.4070 - val_loss: 1.4165 - val_accuracy: 0.4355
Epoch 9/180
15/15 [==============================] - 1s 64ms/step - loss: 3.0287 - accuracy: 0.4231 - val_loss: 1.4960 - val_accuracy: 0.3522
Epoch 10/180
15/15 [==============================] - 1s 64ms/step - loss: 2.9865 - accuracy: 0.4233 - val_loss: 1.3259 - val_accuracy: 0.4535
Epoch 11/180
15/15 [==============================] - 1s 64ms/step - loss: 2.9313 - accuracy: 0.4345 - val_loss: 1.3421 - val_accuracy: 0.4411
Epoch 12/180
15/15 [==============================] - 1s 63ms/step - loss: 2.8828 - accuracy: 0.4459 - val_loss: 1.3066 - val_accuracy: 0.4653
Epoch 13/180
15/15 [==============================] - 1s 64ms/step - loss: 2.9093 - accuracy: 0.4437 - val_loss: 1.3204 - val_accuracy: 0.4527
Epoch 14/180
15/15 [==============================] - 1s 68ms/step - loss: 2.8270 - accuracy: 0.4557 - val_loss: 1.3910 - val_accuracy: 0.4073
Epoch 15/180
15/15 [==============================] - 1s 64ms/step - loss: 2.8096 - accuracy: 0.4557 - val_loss: 1.3713 - val_accuracy: 0.4316
Epoch 16/180
15/15 [==============================] - 1s 64ms/step - loss: 2.7626 - accuracy: 0.4659 - val_loss: 1.3436 - val_accuracy: 0.4424
Epoch 17/180
14/15 [===========================>..] - ETA: 0s - loss: 2.7371 - accuracy: 0.4703Restoring model weights from the end of the best epoch: 12.
15/15 [==============================] - 1s 65ms/step - loss: 2.7359 - accuracy: 0.4681 - val_loss: 1.3397 - val_accuracy: 0.4419
Epoch 17: early stopping
```

8. Model 8 (Age detection): Metrics through epochs

```
Epoch 1/180
38/38 [==============================] - 2s 44ms/step - loss: 2.2660 - accuracy: 0.3099 - val_loss: 1.9238 - val_accuracy: 0.2022
Epoch 2/180
38/38 [==============================] - 1s 38ms/step - loss: 1.6148 - accuracy: 0.3861 - val_loss: 1.9101 - val_accuracy: 0.1901
Epoch 3/180
38/38 [==============================] - 1s 38ms/step - loss: 1.4287 - accuracy: 0.4395 - val_loss: 1.9095 - val_accuracy: 0.2128
Epoch 4/180
38/38 [==============================] - 1s 38ms/step - loss: 1.3023 - accuracy: 0.4819 - val_loss: 1.8321 - val_accuracy: 0.2610
Epoch 5/180
38/38 [==============================] - 1s 38ms/step - loss: 1.2260 - accuracy: 0.5049 - val_loss: 1.7410 - val_accuracy: 0.3593
Epoch 6/180
38/38 [==============================] - 1s 38ms/step - loss: 1.1874 - accuracy: 0.5170 - val_loss: 1.6621 - val_accuracy: 0.3538
Epoch 7/180
38/38 [==============================] - 1s 38ms/step - loss: 1.1394 - accuracy: 0.5337 - val_loss: 1.6352 - val_accuracy: 0.3696
Epoch 8/180
38/38 [==============================] - 1s 38ms/step - loss: 1.1009 - accuracy: 0.5482 - val_loss: 1.6583 - val_accuracy: 0.3683
Epoch 9/180
38/38 [==============================] - 1s 38ms/step - loss: 1.0685 - accuracy: 0.5607 - val_loss: 1.5722 - val_accuracy: 0.3807
Epoch 10/180
38/38 [==============================] - 1s 38ms/step - loss: 1.0455 - accuracy: 0.5698 - val_loss: 1.4980 - val_accuracy: 0.4171
Epoch 11/180
38/38 [==============================] - 1s 38ms/step - loss: 1.0149 - accuracy: 0.5796 - val_loss: 1.7109 - val_accuracy: 0.3462
Epoch 12/180
38/38 [==============================] - 1s 38ms/step - loss: 0.9805 - accuracy: 0.5926 - val_loss: 1.5456 - val_accuracy: 0.4305
Epoch 13/180
38/38 [==============================] - 1s 39ms/step - loss: 0.9596 - accuracy: 0.6023 - val_loss: 1.3320 - val_accuracy: 0.4851
Epoch 14/180
38/38 [==============================] - 1s 38ms/step - loss: 0.9342 - accuracy: 0.6099 - val_loss: 1.4578 - val_accuracy: 0.4535
Epoch 15/180
38/38 [==============================] - 1s 38ms/step - loss: 0.8964 - accuracy: 0.6285 - val_loss: 1.5330 - val_accuracy: 0.4334
Epoch 16/180
38/38 [==============================] - 1s 38ms/step - loss: 0.8747 - accuracy: 0.6334 - val_loss: 1.1835 - val_accuracy: 0.5254
Epoch 17/180
38/38 [==============================] - 1s 38ms/step - loss: 0.8390 - accuracy: 0.6475 - val_loss: 1.1742 - val_accuracy: 0.5447
Epoch 18/180
38/38 [==============================] - 1s 38ms/step - loss: 0.8295 - accuracy: 0.6514 - val_loss: 1.3110 - val_accuracy: 0.5133
Epoch 19/180
38/38 [==============================] - 1s 38ms/step - loss: 0.7928 - accuracy: 0.6659 - val_loss: 1.1959 - val_accuracy: 0.5204
Epoch 20/180
38/38 [==============================] - 1s 38ms/step - loss: 0.7647 - accuracy: 0.6788 - val_loss: 1.0698 - val_accuracy: 0.5658
Epoch 21/180
38/38 [==============================] - 1s 38ms/step - loss: 0.7416 - accuracy: 0.6899 - val_loss: 1.1237 - val_accuracy: 0.5668
Epoch 22/180
38/38 [==============================] - 1s 39ms/step - loss: 0.7048 - accuracy: 0.7038 - val_loss: 1.1956 - val_accuracy: 0.5605
Epoch 23/180
38/38 [==============================] - 1s 38ms/step - loss: 0.6827 - accuracy: 0.7116 - val_loss: 1.3542 - val_accuracy: 0.5254
Epoch 24/180
37/38 [============================>.] - ETA: 0s - loss: 0.6618 - accuracy: 0.7202Restoring model weights from the end of the best epoch: 20.
38/38 [==============================] - 1s 38ms/step - loss: 0.6622 - accuracy: 0.7201 - val_loss: 1.1010 - val_accuracy: 0.5674
Epoch 24: early stopping
```

9. Model 9 (Age detection): Metrics through epochs

```
Epoch 1/10
238/238 [==============================] - 4s 9ms/step - loss: 3.1943 - accuracy: 0.3781 - val_loss: 1.4946 - val_accuracy: 0.3886
Epoch 2/10
238/238 [==============================] - 2s 7ms/step - loss: 3.1264 - accuracy: 0.3868 - val_loss: 1.6822 - val_accuracy: 0.3303
Epoch 3/10
238/238 [==============================] - 2s 7ms/step - loss: 3.0878 - accuracy: 0.3959 - val_loss: 1.6129 - val_accuracy: 0.3443
Epoch 4/10
238/238 [==============================] - 2s 7ms/step - loss: 3.0409 - accuracy: 0.3985 - val_loss: 1.4025 - val_accuracy: 0.4390
Epoch 5/10
238/238 [==============================] - 2s 7ms/step - loss: 3.0230 - accuracy: 0.4003 - val_loss: 1.3982 - val_accuracy: 0.4242
Epoch 6/10
238/238 [==============================] - 2s 7ms/step - loss: 2.9731 - accuracy: 0.4125 - val_loss: 1.6044 - val_accuracy: 0.3459
Epoch 7/10
238/238 [==============================] - 2s 7ms/step - loss: 2.9434 - accuracy: 0.4124 - val_loss: 1.4630 - val_accuracy: 0.4126
Epoch 8/10
234/238 [============================>.] - ETA: 0s - loss: 2.9068 - accuracy: 0.4241Restoring model weights from the end of the best epoch: 5.
238/238 [==============================] - 2s 7ms/step - loss: 2.9060 - accuracy: 0.4240 - val_loss: 1.6098 - val_accuracy: 0.3538
Epoch 8: early stopping
```

10. VGG16 Model (Gender detection): Metrics through epochs

```
Epoch 1/30
187/187 [==============================] - 377s 2s/step - loss: 0.4701 - accuracy: 0.7842 - val_loss: 0.4935 - val_accuracy: 0.7450
Epoch 2/30
187/187 [==============================] - 35s 186ms/step - loss: 0.4203 - accuracy: 0.8097 - val_loss: 0.3039 - val_accuracy: 0.8720
Epoch 3/30
187/187 [==============================] - 34s 184ms/step - loss: 0.3692 - accuracy: 0.8378 - val_loss: 0.3401 - val_accuracy: 0.8450
Epoch 4/30
187/187 [==============================] - 34s 182ms/step - loss: 0.3397 - accuracy: 0.8494 - val_loss: 0.2686 - val_accuracy: 0.8940
Epoch 5/30
187/187 [==============================] - 35s 188ms/step - loss: 0.4033 - accuracy: 0.8237 - val_loss: 0.4436 - val_accuracy: 0.7630
Epoch 6/30
187/187 [==============================] - 36s 191ms/step - loss: 0.3399 - accuracy: 0.8559 - val_loss: 0.2644 - val_accuracy: 0.8990
Epoch 7/30
187/187 [==============================] - 35s 185ms/step - loss: 0.3628 - accuracy: 0.8413 - val_loss: 0.4382 - val_accuracy: 0.7940
Epoch 8/30
187/187 [==============================] - 36s 190ms/step - loss: 0.3219 - accuracy: 0.8579 - val_loss: 0.2868 - val_accuracy: 0.8840
Epoch 9/30
187/187 [==============================] - 35s 184ms/step - loss: 0.3052 - accuracy: 0.8710 - val_loss: 0.2663 - val_accuracy: 0.8970
Epoch 10/30
187/187 [==============================] - 35s 186ms/step - loss: 0.3516 - accuracy: 0.8422 - val_loss: 0.2939 - val_accuracy: 0.8670
Epoch 11/30
187/187 [==============================] - 35s 185ms/step - loss: 0.3272 - accuracy: 0.8537 - val_loss: 0.2511 - val_accuracy: 0.8960
Epoch 12/30
187/187 [==============================] - 35s 186ms/step - loss: 0.2934 - accuracy: 0.8705 - val_loss: 0.2352 - val_accuracy: 0.9110
Epoch 13/30
187/187 [==============================] - 35s 185ms/step - loss: 0.3527 - accuracy: 0.8443 - val_loss: 0.2321 - val_accuracy: 0.9080
Epoch 14/30
187/187 [==============================] - 34s 184ms/step - loss: 0.2849 - accuracy: 0.8767 - val_loss: 0.2324 - val_accuracy: 0.9030
Epoch 15/30
187/187 [==============================] - 35s 185ms/step - loss: 0.3066 - accuracy: 0.8727 - val_loss: 0.2359 - val_accuracy: 0.9060
Epoch 16/30
187/187 [==============================] - 34s 184ms/step - loss: 0.2888 - accuracy: 0.8775 - val_loss: 0.2241 - val_accuracy: 0.9140
Epoch 17/30
187/187 [==============================] - 34s 183ms/step - loss: 0.2876 - accuracy: 0.8772 - val_loss: 0.2362 - val_accuracy: 0.9020
Epoch 18/30
187/187 [==============================] - 35s 185ms/step - loss: 0.2582 - accuracy: 0.8933 - val_loss: 0.2173 - val_accuracy: 0.9140
Epoch 19/30
187/187 [==============================] - 34s 184ms/step - loss: 0.2593 - accuracy: 0.8944 - val_loss: 0.2150 - val_accuracy: 0.9110
Epoch 20/30
187/187 [==============================] - 34s 183ms/step - loss: 0.2448 - accuracy: 0.9011 - val_loss: 0.2377 - val_accuracy: 0.9020
Epoch 21/30
187/187 [==============================] - 35s 185ms/step - loss: 0.2672 - accuracy: 0.8856 - val_loss: 0.2138 - val_accuracy: 0.9100
Epoch 22/30
187/187 [==============================] - 35s 184ms/step - loss: 0.2761 - accuracy: 0.8824 - val_loss: 0.3184 - val_accuracy: 0.8560
Epoch 23/30
187/187 [==============================] - 34s 183ms/step - loss: 0.2872 - accuracy: 0.8753 - val_loss: 0.2470 - val_accuracy: 0.8960
Epoch 24/30
187/187 [==============================] - 35s 185ms/step - loss: 0.2577 - accuracy: 0.8906 - val_loss: 0.2425 - val_accuracy: 0.9010
Epoch 25/30
187/187 [==============================] - 34s 184ms/step - loss: 0.2649 - accuracy: 0.8877 - val_loss: 0.2152 - val_accuracy: 0.9070
Epoch 26/30
187/187 [==============================] - ETA: 0s - loss: 0.2405 - accuracy: 0.8983Restoring model weights from the end of the best epoch: 21.
187/187 [==============================] - 34s 184ms/step - loss: 0.2405 - accuracy: 0.8983 - val_loss: 0.3564 - val_accuracy: 0.8360
Epoch 26: early stopping
```

11. GoogleNet/Inception Model (Gender detection): Metrics through epochs

```
Epoch 1/30
187/187 [==============================] - 910s 5s/step - loss: 2.9579 - accuracy: 0.8386 - val_loss: 0.2414 - val_accuracy: 0.9160
Epoch 2/30
187/187 [==============================] - 302s 2s/step - loss: 0.2583 - accuracy: 0.8934 - val_loss: 0.2061 - val_accuracy: 0.9270
Epoch 3/30
187/187 [==============================] - 302s 2s/step - loss: 0.2195 - accuracy: 0.9114 - val_loss: 0.2034 - val_accuracy: 0.9300
Epoch 4/30
187/187 [==============================] - 300s 2s/step - loss: 0.2027 - accuracy: 0.9216 - val_loss: 0.1908 - val_accuracy: 0.9330
Epoch 5/30
187/187 [==============================] - 300s 2s/step - loss: 0.1974 - accuracy: 0.9214 - val_loss: 0.2866 - val_accuracy: 0.8920
Epoch 6/30
187/187 [==============================] - 301s 2s/step - loss: 0.2322 - accuracy: 0.9047 - val_loss: 0.2448 - val_accuracy: 0.8970
Epoch 7/30
187/187 [==============================] - 301s 2s/step - loss: 0.1939 - accuracy: 0.9192 - val_loss: 0.1905 - val_accuracy: 0.9280
Epoch 8/30
187/187 [==============================] - 300s 2s/step - loss: 0.2291 - accuracy: 0.9077 - val_loss: 0.2016 - val_accuracy: 0.9370
Epoch 9/30
187/187 [==============================] - 300s 2s/step - loss: 0.1674 - accuracy: 0.9303 - val_loss: 0.2139 - val_accuracy: 0.9280
Epoch 10/30
187/187 [==============================] - 300s 2s/step - loss: 0.1689 - accuracy: 0.9290 - val_loss: 0.2521 - val_accuracy: 0.9190
Epoch 11/30
187/187 [==============================] - 300s 2s/step - loss: 0.1600 - accuracy: 0.9357 - val_loss: 0.2016 - val_accuracy: 0.9260
Epoch 12/30
187/187 [==============================] - ETA: 0s - loss: 0.1390 - accuracy: 0.9437Restoring model weights from the end of the best epoch: 7.
187/187 [==============================] - 300s 2s/step - loss: 0.1390 - accuracy: 0.9437 - val_loss: 0.2204 - val_accuracy: 0.9260
Epoch 12: early stopping
```

12. ResNet50 Model (Gender detection): Metrics through epochs

```
Epoch 1/30
187/187 [==============================] - 72s 357ms/step - loss: 1.5757 - accuracy: 0.8098 - val_loss: 1.4975 - val_accuracy: 0.5000
Epoch 2/30
187/187 [==============================] - 67s 356ms/step - loss: 1.2121 - accuracy: 0.8125 - val_loss: 0.6834 - val_accuracy: 0.5500
Epoch 3/30
187/187 [==============================] - 67s 356ms/step - loss: 0.4937 - accuracy: 0.8596 - val_loss: 0.7663 - val_accuracy: 0.5000
Epoch 4/30
187/187 [==============================] - 67s 360ms/step - loss: 0.2142 - accuracy: 0.9268 - val_loss: 0.6759 - val_accuracy: 0.5800
Epoch 5/30
187/187 [==============================] - 68s 363ms/step - loss: 0.1369 - accuracy: 0.9460 - val_loss: 0.4904 - val_accuracy: 0.7800
Epoch 6/30
187/187 [==============================] - 68s 364ms/step - loss: 0.1049 - accuracy: 0.9588 - val_loss: 0.3411 - val_accuracy: 0.8600
Epoch 7/30
187/187 [==============================] - 68s 365ms/step - loss: 0.0912 - accuracy: 0.9648 - val_loss: 0.1527 - val_accuracy: 0.9510
Epoch 8/30
187/187 [==============================] - 68s 363ms/step - loss: 0.0709 - accuracy: 0.9734 - val_loss: 0.1542 - val_accuracy: 0.9420
Epoch 9/30
187/187 [==============================] - 68s 363ms/step - loss: 0.0617 - accuracy: 0.9772 - val_loss: 0.1664 - val_accuracy: 0.9490
Epoch 10/30
187/187 [==============================] - 67s 360ms/step - loss: 0.0448 - accuracy: 0.9844 - val_loss: 0.1889 - val_accuracy: 0.9360
Epoch 11/30
187/187 [==============================] - 68s 362ms/step - loss: 0.0284 - accuracy: 0.9899 - val_loss: 0.2382 - val_accuracy: 0.9430
Epoch 12/30
187/187 [==============================] - ETA: 0s - loss: 0.0683 - accuracy: 0.9762Restoring model weights from the end of the best epoch: 7.
187/187 [==============================] - 68s 363ms/step - loss: 0.0683 - accuracy: 0.9762 - val_loss: 0.2321 - val_accuracy: 0.9330
Epoch 12: early stopping
```

13. EfficientNetB0 Model (Gender detection): Metrics through epochs

```
Epoch 1/30
187/187 [==============================] - 26s 110ms/step - loss: 0.9873 - accuracy: 0.4987 - val_loss: 0.6932 - val_accuracy: 0.5000
Epoch 2/30
187/187 [==============================] - 20s 106ms/step - loss: 0.9682 - accuracy: 0.4993 - val_loss: 0.6933 - val_accuracy: 0.5000
Epoch 3/30
187/187 [==============================] - 19s 103ms/step - loss: 0.9727 - accuracy: 0.4926 - val_loss: 0.6934 - val_accuracy: 0.5000
Epoch 4/30
187/187 [==============================] - 19s 103ms/step - loss: 0.9559 - accuracy: 0.4966 - val_loss: 0.6934 - val_accuracy: 0.5010
Epoch 5/30
187/187 [==============================] - 19s 101ms/step - loss: 0.9398 - accuracy: 0.5052 - val_loss: 0.6939 - val_accuracy: 0.5000
Epoch 6/30
187/187 [==============================] - ETA: 0s - loss: 0.9620 - accuracy: 0.5028Restoring model weights from the end of the best epoch: 1.
187/187 [==============================] - 19s 102ms/step - loss: 0.9620 - accuracy: 0.5028 - val_loss: 0.6936 - val_accuracy: 0.5000
Epoch 6: early stopping
```