



UNIVERSITÀ DEGLI STUDI DI PERUGIA
Dipartimento di Matematica e Informatica



Progetto di Machine Learning

What's My Age Again?

Laureandi:

Fabio Bocchini(340600)

Fabrizio Fagiolo(349370)

Professori:

Prof. Valentina Poggioni

Dott. Alina Elena Baia

Anno accademico 2021/2022

Indice

1 Obiettivo	4
2 Dataset	5
3 Modello	7
3.1 Ambiente di sviluppo	7
3.2 Installazione librerie	8
3.3 Preparazione dei dati	9
3.4 Creazione Classi	10
3.4.1 Estrazione Label	11
3.4.2 Generazione Data Frame	12
3.4.3 Visualizzazione dei dati	13
3.4.4 Costruzione dati per il training	17
3.5 Creazione del modello	19
3.5.1 Efficienza	26
3.6 Risultati	28
3.6.1 Genere	28
3.6.2 Età	30
3.6.3 Etnia	32
3.6.4 Matrici di confusione	34
3.7 Salvataggio Modello	38
3.8 Esportazione del modello	39
4 Applicativo Mobile	40
4.1 Ambiente di Sviluppo	40
4.2 Linguaggi e Framework	40
4.3 Installazione	41

4.3.1	Expo	42
4.3.2	React Native	42
4.4	Struttura del progetto	43
4.5	Assets	44
4.6	Src	45
4.6.1	Componets	46
4.6.2	screens/homepage	46
4.6.3	ui	55
4.6.4	constants	57
4.6.5	contexts	58
4.6.6	enums	60
4.6.7	types	61
4.6.8	utils	62
4.7	Risultato finale dell'app	67
5	Conclusioni	71
5.1	Classificazioni Corrette	72
5.2	Problemi	77

Capitolo 1

Obiettivo

L'obiettivo del progetto è quello di realizzare l'implementazione di un applicativo **mobile** sul quale è presente un **modello di Machine Learning** in grado di, tramite una foto passata in input dalla parte "nativa" dell'applicativo, riconoscere età, genere ed etnia di un soggetto.

Capitolo 2

Dataset

Il set di dati **UTKFace** è un dataset rappresentante volti su larga scala con un lungo intervallo di età (intervallo da 0 a 116 anni). Il set di dati è costituito da oltre 20.000 immagini con annotazioni di età, sesso ed etnia. Le immagini coprono grandi variazioni di posa, espressione facciale, illuminazione, occlusione, risoluzione, ecc.

Abbiamo utilizzato questo set per apprendere dalle label e dai volti 3 attributi:

1. Genere

2. Età

3. Etnia

Alcuni campioni delle immagini sono mostrate come segue:



Successivamente siamo andati ad unire il dataset composto da 3 file in uno solo. Inizialmente abbiamo optato per il dataset dove le immagini non erano ritagliate ("cropped"). Questo però non era molto performante ed ha anche portato a molti problemi durante la fase di test a causa del "rumore" presente nelle foto.



Figura 2.1: Foto dataset "cropped"



Figura 2.2: Foto dataset non "cropped"

Capitolo 3

Modello

3.1 Ambiente di sviluppo

Per la creazione del modello abbiamo utilizzato **Google Colab**, o "Colaboratory", che permette di scrivere ed eseguire codice Python nel tuo browser senza far andare in conflitto le librerie Python con:

- Nessuna configurazione necessaria
- Accesso gratuito alle GPU
- Condivisione semplificata

Il codice viene compilato in un ambiente interattivo chiamato blocco note Colab che ti permette di scrivere ed eseguire codice.

Per la risoluzione del nostro problema le risorse messe a disposizione dalla versione gratuita non erano sufficienti, quindi siamo passati alla versione pro.

3.2 Installazione librerie

Per prima cosa abbiamo installato le varie librerie necessarie per l'esecuzione delle "funzioni" ed i metodi derivanti da esse.

```
1 #import library
2 import numpy as np
3 import pandas as pd
4 import os
5 from google.colab import drive
6 from PIL import Image
7 import matplotlib.pyplot as plt
8 import random
9 from sklearn.model_selection import train_test_split
10 from keras.preprocessing.image import ImageDataGenerator
11 import seaborn as sns
12 from keras.models import Sequential, Model
13 from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D, Input
14 from sklearn.metrics import classification_report
15 from keras.models import Sequential, Model
16 from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D, Input
17 from keras import callbacks
18 from tensorflow.keras.utils import plot_model
19 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
20 import plotly.graph_objects as go
21 import plotly.express as px
22 from matplotlib import rcParams
```

3.3 Preparazione dei dati

Il dataset, essendo un file di grandi dimensioni, verrà scaricato dal nostro drive, in modo da non dover essere caricato sulla piattaforma ogni volta.

```
1 #mount dataset import from the google drive
2 drive.mount('/content/drive')
3 !tar -xvf "/content/drive/MyDrive/utk-faces-cropped.tar.gz" -C "/content/"
4 BASE_DIR = '/content/UTKFace'
```

Se tutto è andato a buon fine avremmo il seguente output:

```
Output streaming troncato alle ultime 5000 righe
UTKFace/33_1_0_20170111182452825.jpg.chip.jpg
UTKFace/26_1_1_20170116024053194.jpg.chip.jpg
UTKFace/25_1_0_20170109213232182.jpg.chip.jpg
UTKFace/28_1_1_20170112234526480.jpg.chip.jpg
UTKFace/57_0_0_20170117191704100.jpg.chip.jpg
UTKFace/27_1_1_20170117193211345.jpg.chip.jpg
UTKFace/46_1_1_20170116161237892.jpg.chip.jpg
UTKFace/24_0_2_20170116171647508.jpg.chip.jpg
UTKFace/42_1_1_20170113005712902.jpg.chip.jpg
UTKFace/25_1_1_20170116001337504.jpg.chip.jpg
UTKFace/40_1_0_20170116222110661.jpg.chip.jpg
UTKFace/50_1_0_20170109012257664.jpg.chip.jpg
UTKFace/26_1_3_20170104235421282.jpg.chip.jpg
UTKFace/7_0_0_20170110215711115.jpg.chip.jpg
UTKFace/1_1_2_20161219155353413.jpg.chip.jpg
UTKFace/26_0_1_20170117195651493.jpg.chip.jpg
UTKFace/7_0_0_20170110215648859.jpg.chip.jpg
UTKFace/64_0_0_20170117155540137.jpg.chip.jpg
UTKFace/48_0_0_20170109004813150.jpg.chip.jpg
UTKFace/39_0_0_20170103183230555.jpg.chip.jpg
UTKFace/45_0_1_20170117190019363.jpg.chip.jpg
UTKFace/38_1_0_20170104192820567.jpg.chip.jpg
UTKFace/36_1_0_20170117135852762.jpg.chip.jpg
UTKFace/30_1_1_20170117133829099.jpg.chip.jpg
```

Il dataset poi verrà estratto e passato alla directory **BASE_DIR**

```
1 # directory of the dataset
2 BASE_DIR = '/content/UTKFace'
```

3.4 Creazione Classi

Per categorizzare i problemi di **etnia** e **genere**, siamo andati a creare il seguente dizionario con le seguenti classi.

```
1 # map labels for gender and ethnicity
2 gender_dict = {
3     0: 'Male',
4     1: 'Female'
5 }
6 ethnicity_dict = {
7     0: 'European',
8     1: 'African',
9     2: 'Asian',
10    3: 'Ocean',
11    4: 'Other'
12 }
```

Come è possibile notare, la classe che definisce il genere è binaria, mentre quella che definisce la classificazione dell'etnia è **categorica** (multi-classe).

3.4.1 Estrazione Label

Per il processo di classificazione abbiamo estratto le feature presenti nel dataset e le abbiamo messe all'interno di 4 liste a seconda del problema:

- **image_paths**

E' la lista contenente il percorso delle foto.

- **age_labels**

E' la lista contenente le label dell'età

- **gender_labels**

E' la lista contenente le label del genere

- **ethnicity_labels**

E' la lista contenente le label dell'etnia

```
1 # list labels: age, gender, ethnicity
2 image_paths = []
3 age_labels = []
4 gender_labels = []
5 ethnicity_labels = []
6
7
8 # take the labels by the directory
9 for filename in os.listdir(BASE_DIR):
10     image_path = os.path.join(BASE_DIR, filename)
11     temp = filename.split('_')
12
13     #skip ds_store file
14     if(filename == '.DS_Store'):
15         continue
16
17     image_path = os.path.join(BASE_DIR, filename)
18     temp = filename.split('_')
19
20     #skip doblue underscore
21     if(temp[1]== ''):
22         continue
23
24     #if the ethnicity not define assign class 'other'
25     try:
26         ethnicity = int(temp[2])
27     except ValueError:
28         ethnicity = 4
29
30     #append labels in the list
31     age = int(temp[0])
32     gender = int(temp[1])
33     image_paths.append(image_path)
34     age_labels.append(age)
35     gender_labels.append(gender)
36     ethnicity_labels.append(ethnicity)
37
```

3.4.2 Generazione Data Frame

Successivamente andiamo a convertire le liste create al passo precedente in **dataframe** che andremo poi ad utilizzare per il processo di addestramento.

```
1 # convert to dataframe
2 df = pd.DataFrame()
3 # create dataframe for age, gender, ethnicity
4 df[['image'], df['age'], df['gender'], df['ethnicity']] = image_paths, age_labels, gender_labels, ethnicity_labels
5 # print first 5 element df
6 df.head()
```

Se tutto è andato a buon fine possiamo visualizzare la composizione del dataframe come nel seguente output.

	image	age	gender	ethnicity
0	/content/UTKFace/58_0_2_20170112205242580.jpg....	58	0	2
1	/content/UTKFace/63_0_1_20170116235836887.jpg....	63	0	1
2	/content/UTKFace/26_0_0_20170113210127408.jpg....	26	0	0
3	/content/UTKFace/73_0_3_20170119212057504.jpg....	73	0	3
4	/content/UTKFace/52_0_0_20170111204457405.jpg....	52	0	0

Dividiamo poi a dividere il **dataframe** in 2 parti:

1. Training

Parte del dataframe usato per il training del modello.

2. Test

Parte del dataframe usato per il preocesso di test del modello.

```
1 # take 10% of the data for validation
2 TEST_SPLIT = 0.1
3
4 # calculate array lenght test
5 test_df_length = int(len(df) * TEST_SPLIT)
6
7 # shuffle df
8 df = df.sample(frac=1)
9
10 # take 2 part and split in test and train
11 test_df = df.head(test_df_length)
12 df = df.tail(len(df) - test_df_length)
```

3.4.3 Visualizzazione dei dati

Successivamente andiamo a controllare come sono fatti i nostri dati. Come è possibile vedere oltre all'immagine avremo:

- **image number**

che corrisponde al numero dell'immagine.

- **age**

che corrisponde all'età della persona in foto.

- **gender**

che corrisponde al genere della persona in foto.

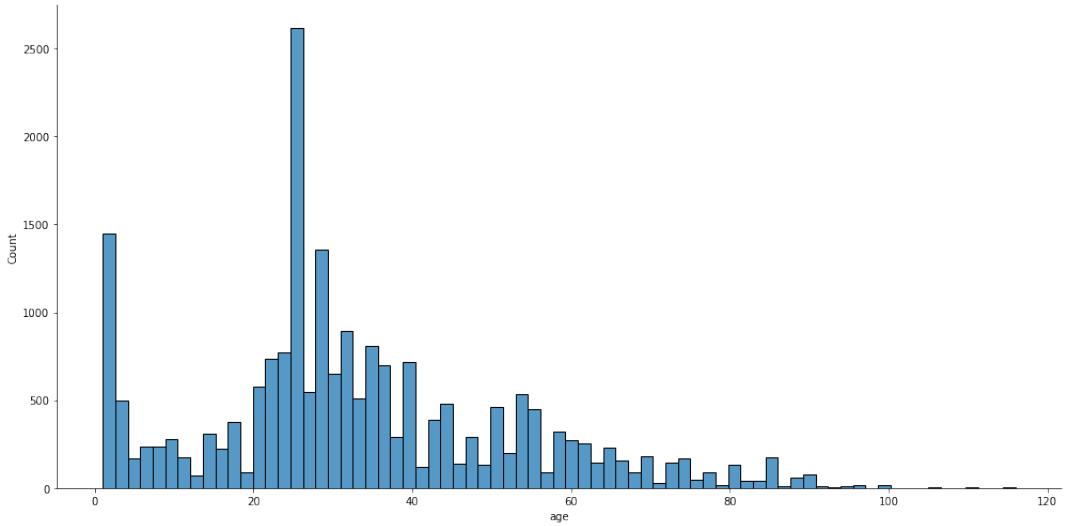
```
1 # take random index for print photo of dataset
2 i = random.randint(0, len(df.index))
3
4 print('image number:\t', i)
5 print('age:\t\t', df['age'][i])
6 print('gender:\t\t', gender_dict[df['gender'][i]])
7
8 # show image
9 img = Image.open(df['image'][i])
10 plt.axis('off')
11 plt.imshow(img);
```



image number: 8471
age: 60
gender: Female

Dall'output successivo possiamo verificare che è andato tutto a buon fine.
In seguito andremo a visualizzare la distribuzione dei dati rispetto all'età

```
1 # Graph age  
2 sns.displot(df['age'], height=7, aspect=2)
```



La distribuzione delle classi rispetto all'etnia

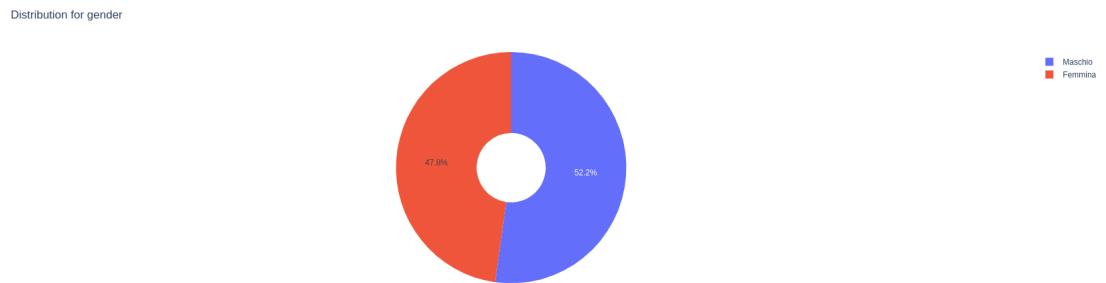
```
1 # Graph ethnicity
2 labels = ['Bianco', 'Nero', 'Asiatico', 'Indiano', 'Altro']
3 counts = df['ethnicity'].value_counts().values.tolist()
4
5 pie_plot = go.Pie(labels=labels, values=counts, hole=.3)
6 fig = go.Figure(data=[pie_plot])
7 fig.update_layout(title_text='Distribution for %s' % df['ethnicity'].name)
8
9 fig.show()
```

Distribution for ethnicity



Ed infine visualizzeremo la distribuzione binaria per il genere.

```
1 #Graph Gender
2 labels = ['Maschio', 'Femmina']
3 counts = df['gender'].value_counts().values.tolist()
4
5 pie_plot = go.Pie(labels=labels, values=counts, hole=.3)
6 fig = go.Figure(data=[pie_plot])
7 fig.update_layout(title_text='Distribution for %s' % df['gender'].name)
8
9 fig.show()
```



3.4.4 Costruzione dati per il training

Dall'ultimo dataset creato andiamo a prendere:

80%

per la costruzione del modello.

20%

per la convalida.

```
1 # for the last dataset split data for validation and train
2 training_data, validation_data = train_test_split(df, test_size=0.2)
```

Successivamente applichiamo del preprocessing alle immagini

- **normalizzazione**

In origine l'immagine è una lista di numeri interi da 0 a 255 rappresentanti i valori RGB per ogni pixel. Dividiamo questi valori per 255 così che diventino numeri a virgola mobile in un range da 0 a 1.

- **ridimensionamento**

Qualunque sia la dimensione dell'immagine originale vengono tutte scalate alla dimensione fissa di 128x128 pixel. Così facendo tutte le immagini avranno la stessa forma, ovvero una lista di 128x128x3 valori float compresi tra 0 e 1.

Assegniamo poi il valore di batch e andiamo a creare il nostro tensore.

```

1 # normalization of image (pixel from 0,1)
2 train_datagen = ImageDataGenerator(rescale=1./255)
3 val_datagen = ImageDataGenerator(rescale=1./255)
4
5 # assign dimension of batch
6 batch_size = 64
7
8 # take an image df and transform to tensor list by parameter for train and test
9 train_generator = train_datagen.flow_from_dataframe(training_data,
10                                         x_col = 'image',
11                                         y_col = ['age', 'gender', 'ethnicity'],
12                                         target_size = (128, 128),
13                                         class_mode = 'multi_output',
14                                         batch_size = batch_size)
15
16 val_generator = val_datagen.flow_from_dataframe(validation_data,
17                                         x_col = 'image',
18                                         y_col = ['age', 'gender', 'ethnicity'],
19                                         target_size = (128, 128),
20                                         class_mode = 'multi_output',
21                                         batch_size = batch_size)

```

Vado poi a trasformare il tensore in un numpy array così che possa effettuare l'allenamento sui dataframe definiti precedentemente. Creiamo la variabile di input da passare al

```

1 # transform df to numpy array
2 y_gender = np.array(df['gender'])
3 y_age = np.array(df['age'])
4 y_ethnicity = np.array(df['ethnicity'])
5

```

modello dove avremo un'immagine di grandezza 128 x 128 con 3 colori RGB

```

1 # create image shape 128x128x3 colors
2 input_shape = (128, 128, 3)

```

3.5 Creazione del modello

```
1 # take input
2 inputs = Input((input_shape))
3
4 # create model sequential
5 model = Sequential()
6
7 # create base model for all problems
8 base_model = Conv2D(32, (3, 3), activation = 'relu')(inputs)
9 base_model = MaxPooling2D((2, 2))(base_model)
10 base_model = Dropout(0.25)(base_model)
11 base_model = Conv2D(64, (3, 3), activation = 'relu')(base_model)
12 base_model = MaxPooling2D((2, 2))(base_model)
13 base_model = Dropout(0.25)(base_model)
14
15 # split model for age problem
16 age_model = base_model
17 age_model = Conv2D(128, (3, 3), activation = 'relu')(age_model)
18 age_model = MaxPooling2D((2, 2))(age_model)
19 age_model = Dropout(0.3)(age_model)
20 age_model = Dropout(0.3)(age_model)
21 age_model = Flatten()(age_model)
22 age_model = Dense(128, activation = 'relu')(age_model)
23 age_model = Dense(64, activation = 'relu')(age_model)
24 age_model = Dense(32, activation = 'relu')(age_model)
25 age_model = Dense(1, activation = 'relu', name='age_output')(age_model)
26
27 # split model for gender problem
28 gender_model = base_model
29 gender_model = Conv2D(128, (3, 3), activation = 'relu')(gender_model)
30 gender_model = MaxPooling2D((2, 2))(gender_model)
31 gender_model = Dropout(0.3)(gender_model)
32 gender_model = Conv2D(64, (3, 3), activation = 'relu')(gender_model)
33 gender_model = MaxPooling2D((2, 2))(gender_model)
34 gender_model = Dropout(0.3)(gender_model)
35 gender_model = Flatten()(gender_model)
36 gender_model = Dense(32, activation = 'relu')(gender_model)
37 gender_model = Dense(16, activation = 'relu')(gender_model)
38 gender_model = Dense(8, activation = 'relu')(gender_model)
39 gender_model = Dense(1, activation = 'sigmoid', name='gender_output')(gender_model)
```

```

41 # split model for ethnicity problem
42 ethnicity_model = base_model
43 ethnicity_model = Conv2D(128, (3, 3), activation = 'relu')(ethnicity_model)
44 ethnicity_model = MaxPooling2D((2, 2))(ethnicity_model)
45 ethnicity_model = Dropout(0.2)(ethnicity_model)
46 ethnicity_model = Conv2D(64, (3, 3), activation = 'relu')(ethnicity_model)
47 ethnicity_model = MaxPooling2D((2, 2))(ethnicity_model)
48 ethnicity_model = Dropout(0.2)(ethnicity_model)
49 ethnicity_model = Flatten()(ethnicity_model)
50 ethnicity_model = Dense(64, activation = 'relu')(ethnicity_model)
51 ethnicity_model = Dropout(0.2)(ethnicity_model)
52 ethnicity_model = Dense(32, activation = 'relu')(ethnicity_model)
53 ethnicity_model = Dropout(0.2)(ethnicity_model)
54 ethnicity_model = Dense(16, activation = 'relu')(ethnicity_model)
55 ethnicity_model = Dropout(0.2)(ethnicity_model)
56 ethnicity_model = Dense(8, activation = 'relu')(ethnicity_model)
57 ethnicity_model = Dropout(0.2)(ethnicity_model)
58 ethnicity_model = Dense(5, activation = 'softmax', name='ethnicity_output')(ethnicity_model)
59
60 # generate output
61 model = Model(inputs=inputs, outputs=[age_model, gender_model, ethnicity_model])
62
63 # stop training when val_loss does not increase
64 earlystopping = callbacks.EarlyStopping(monitor ="val_loss",
65                                         mode ="min", patience = 5,
66                                         restore_best_weights = True)

```

```

60 # generate output
61 model = Model(inputs=inputs, outputs=[age_model, gender_model, ethnicity_model])
62
63 # stop training when val_loss does not increase
64 earlystopping = callbacks.EarlyStopping(monitor ="val_loss",
65                                         mode ="min", patience = 5,
66                                         restore_best_weights = True)
67 # compile model
68 model.compile(
69     loss = {
70         'age_output': 'mse',
71         'gender_output': 'binary_crossentropy',
72         'ethnicity_output': 'sparse_categorical_crossentropy'
73     },
74     optimizer = 'adam',
75     metrics = {
76         'age_output': 'mae',
77         'gender_output': 'accuracy',
78         'ethnicity_output': 'sparse_categorical_accuracy'
79     },
80     loss_weights= {
81         'age_output': 4.,
82         'ethnicity_output': 1.5,
83         'gender_output': 0.3
84     }
85 )

```

Nel modello abbiamo utilizzato i seguenti livelli:

- **Convolutivi**

Applicano una serie di filtri all'immagine in input.

In questo caso applichiamo i seguenti parametri:

- **filters**

Rappresenta il numero di filtri applicati all'input.

- **kernel_size**

Rappresenta la grandezza del filtro (altezza e larghezza)

- **activation**

Rappresenta la funzione di attivazione da applicare. Nel caso dei livelli convolutivi andiamo ad applicare sempre la funzione di attivazione **relu** perché è una buone funzione generica e comporta meno problemi di **vanishing gradient** rispetto ad altre funzioni di attivazione.

- **MaxPooling2D**

Vanno a ridurre la dimensione dell'input in modo da rendere piu' piccolo l'insieme di dati che vengono passati ai livelli successivi.

In questo caso andiamo ad utilizzare i seguenti parametri:

- **pool_size**

Rappresenta una tupla che indica le dimensioni della finestra di **downsample** (altezza e larghezza).

- **Dropout**

Produce in output un **sotto-insieme** dei dati in input in modo da avere meno dati e ridurre l'overfitting.

In questo caso andiamo ad utilizzare i seguenti parametri:

- **rate**

Rappresenta la percentuale di dati da **trattenere**

- **Flatten** Trasforma i dati in input facendogli prendere la forma di un unico vettore.

Al seguente livello **non passiamo nessun parametro**

- **Dense**

I livelli densi vanno ad applicare la funzione di attivazione all'input tramite la seguente formula:

$$a_i^l = f(z_i^l) = f(\sum_j w_{ij}^l a_j^{l-1} + b_i^l)$$

Dove:

- a_i^l
è il valore di attivazione del **nodo i** al **layer l**
- f
è la **funzione di attivazione**
- w^l
matrice dei pesi al **layer l**
- b^l
vettore dei **bias** al layer l

Al livello denso andiamo a passare i seguenti parametri:

- **units**
Rappresenta la dimensionalità dello spazio di output
- **activation**
Rappresenta la funzione di attivazione applicata. Utilizziamo nella maggior parte dei casi **Relu** tranne nei livelli di output del modello in cui andiamo ad utilizzare funzione di attivazione specifiche per il tipo di problema.
- **name**
Rappresenta il **nome** dell'output, anche qua andiamo ad utilizzare il seguente parametro esclusivamente nei livelli di output.

Componiamo così il seguente modello:

1. Modello base

In questa sezione abbiamo:

- **2 livelli convolutivi**
- **2 livelli di maxpooling**
- **2 livelli di dropout**

Da qui il modello si divide in **3 rami** che rappresentano i 3 output.

2. Age

Composto da:

- **1 livello convolutivo**
- **1 livello di maxpooling**
- **2 livelli di dropout**
- **1 livello di flatten**
- **4 livelli densi**

Qui utilizziamo la funzione di attivazione **Relu** (REctified Linear Unit) perché essendo una funzione lineare si adatta bene alla risoluzione del problema dell'età.

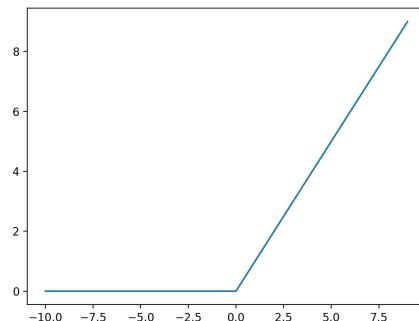


Figura 3.1: Grafico funzione di attivazione **relu**

Per la funzione di loss andiamo ad utilizzare **mean squared error** perché da un peso maggiore agli errori **più significativi** rispetto a gli errori più piccoli.

Applichiamo una **loss_weight** pari a 4 per dare un **peso maggiore** alla loss del ramo rispetto alla loss generale del modello.

In combinazione con questa funzione di **loss** utilizziamo come metrica **mean absolute error** che va a calcolare la media del valore assoluto dell'errore (più è alto il valore meno è performante il modello).

3. Gender

Composto da:

- **2 livelli convolutivi**
- **2 livelli di maxpooling**
- **2 livelli di dropout**
- **1 livello di flatten**
- **4 livelli densi**

Qui utilizziamo la funzione di attivazione **Sigmoide** perché essendo una funzione compresa tra 0 ed 1 si adatta bene alla risoluzione del problema del genere, in quanto può avere solo valori **binari**.

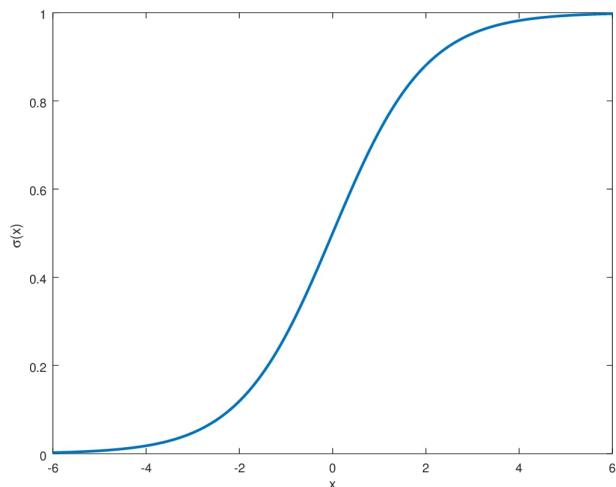


Figura 3.2: Grafico funzione di attivazione **sigmoide**

Per la funzione di **loss** andiamo ad utilizzare **binary cross entropy** perché è in grado di risolvere problemi binari.

Applichiamo una **loss_weight** pari a 1.5 per dare un **peso intermedio** sul calcolo della loss generale.

In combinazione con questa funzione di **loss** utilizziamo come metrica **accuracy** che va a calcolare il **rapporto** tra le predizione **corrette** e le **predizioni totali**.

4. Etnia

Composto da:

- **2** livelli convolutivi
- **2** livelli di maxpooling
- **6** livelli di dropout
- **1** livello di flatten
- **5** livelli densi

Qui utilizziamo la funzione di attivazione **Softmax** che prendendo in input un **vettore di 5 valori** restituisce le **probabilità** che il valore in input appartenga ad ogni classe.

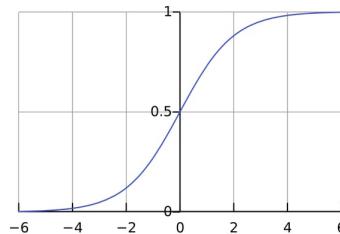


Figura 3.3: Grafico funzione di attivazione **softmax**

Per la funzione di **loss** andiamo ad utilizzare **sparse categorical cross entropy** perché rappresenta l'adattamento della **binary cross entropy** per problemi categorici. Applichiamo una **loss_weight** pari a 0.3 per dare un **peso basso** sul calcolo della loss totale.

In combinazione con questa funzione di **loss** utilizziamo come metrica **sparse categorical accuracy** che va a calcolare il **rapporto** tra le predizione **corrette** e le **predizioni totali** in problemi multi-classe.

3.5.1 Efficienza

Il nostro problema principale riguardava la **dimensione** del modello in termini di **byte** dato che il nostro obiettivo è quello di utilizzarlo all'interno di un **applicativo mobile**. Dato che lo smartphone possiede poca potenza computazionale rispetto ad un server dove di solito risiedono i modelli di ML, un modello relativamente pesante avrebbe comportato lunghi tempi di caricamento.

Quindi siamo arrivati ad un compromesso tra **peso (del modello)** e performance.

Questo compromesso è emerso da una serie di prove dove abbiamo notato che il peso:

- **Aumenta**

Con l'aumento dei livelli **convolutivi**, perché per ogni livello convolutivo aggiungiamo una matrice dei pesi.

- **Diminuisce**

Con l'aumento dei livelli di **maxpool**, perché aggiungendo questi livelli andiamo a diminuire la dimensione dei livelli successivi e quindi anche la dimensione delle matrici dei pesi.

Dunque il modello trovato sembra un buon compromesso anche se le perfomance non sono ottime.

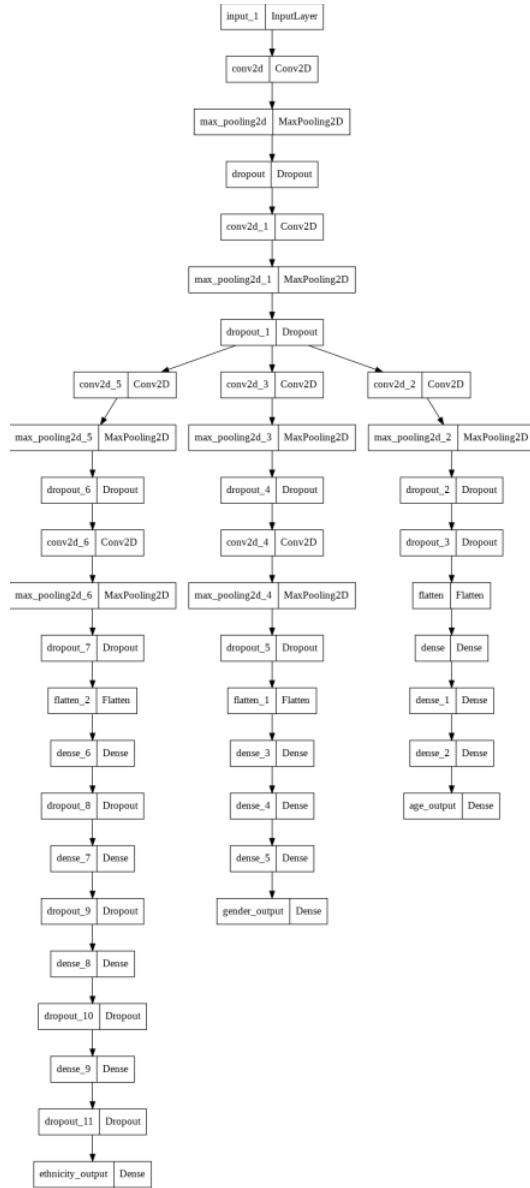


Figura 3.4: Immagine modello completo

3.6 Risultati

Dopo l'allenamento possiamo andare ad analizzare i risultati ottenuti valutando le metriche di errore e accuracy sui dati di train e sui dati di validazione.

3.6.1 Genere

Per prima cosa andiamo a costruire il grafico colorando di blu i valori di train sia dell'accuratezza che dell'errore, mentre di rosso andremo ad evidenziare i valori di validazione per l'accuracy e per il loss

```
1 #Gender Graph
2 acc = history.history['gender_output_accuracy']
3 val_acc = history.history['val_gender_output_accuracy']
4 epochs = range(len(acc))
5
6 plt.plot(epochs, acc, 'b', label = 'Training Accouracy')
7 plt.plot(epochs, val_acc, 'r', label = 'Validation Accouracy')
8 plt.title('Gender Accouracy Graph')
9 plt.legend()
10 plt.figure()
11
12 loss = history.history['gender_output_loss']
13 val_loss = history.history['val_gender_output_loss']
14
15 plt.plot(epochs, loss, 'b', label = 'Training Loss')
16 plt.plot(epochs, val_loss, 'r', label = 'Validation Loss')
17 plt.title('Gender Loss Graph')
18 plt.legend()
19 plt.show()
```

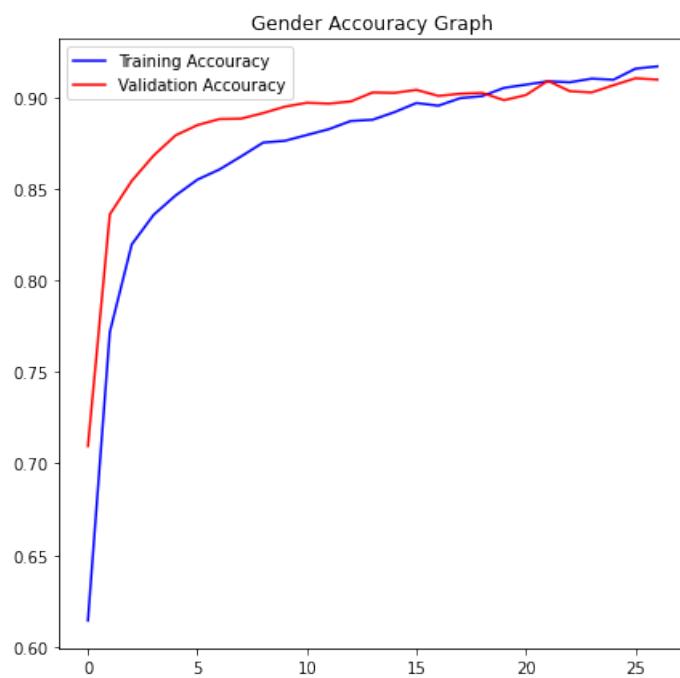


Figura 3.5: Grafico accuratezza genere

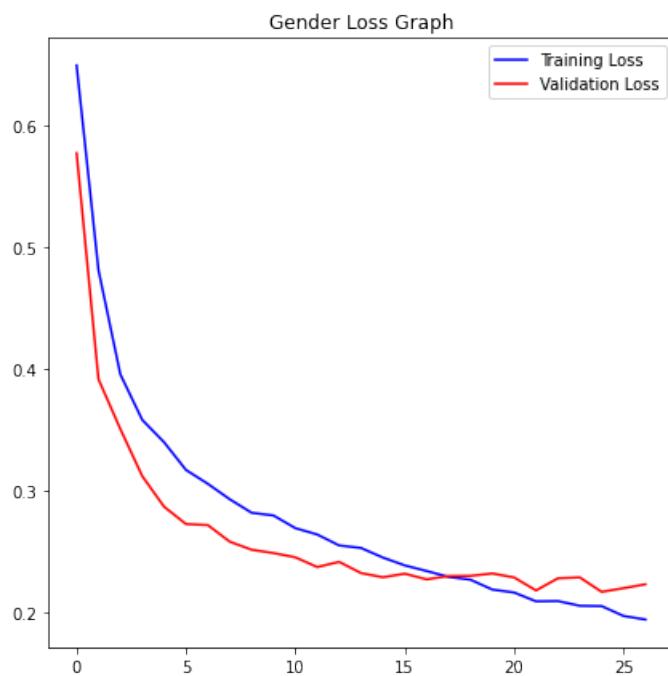
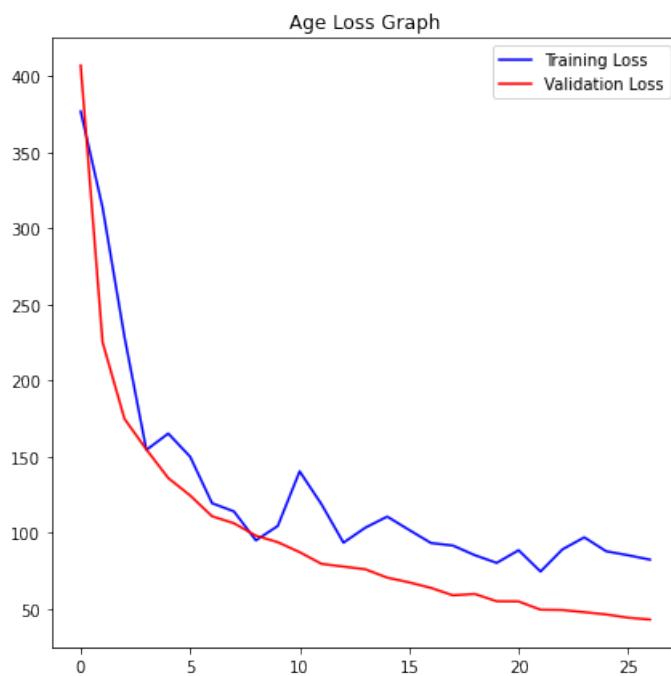
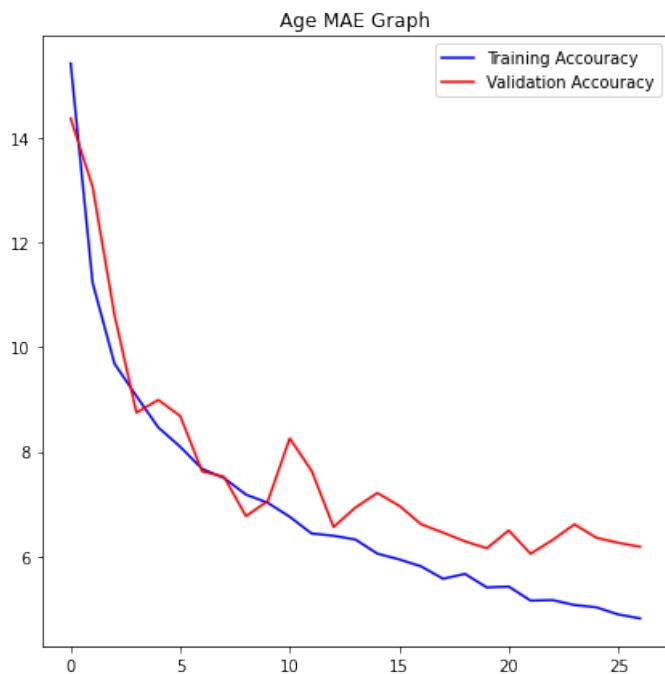


Figura 3.6: Grafico loss genere

3.6.2 Età

Costruiamo anche qui il grafico colorando di blu i valori di train sia dell'accuratezza che dell'errore, mentre di rosso andremo ad evidenziare i valori di validazione per l'accuracy e per il loss.

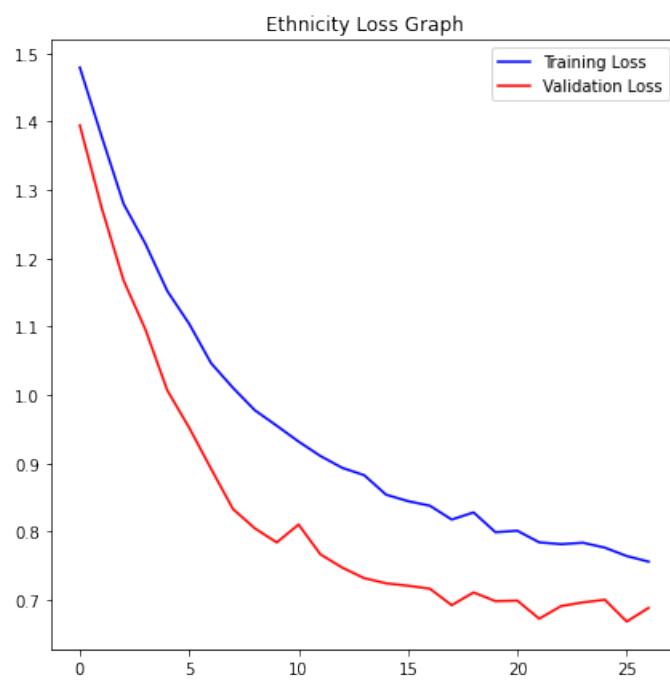
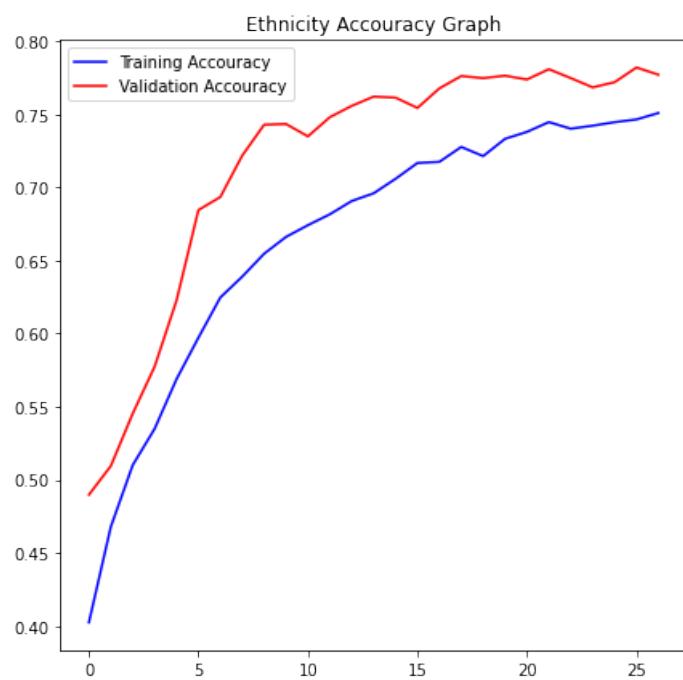
```
1 #Gender Graph
2 acc = history.history['gender_output_accuracy']
3 val_acc = history.history['val_gender_output_accuracy']
4 epochs = range(len(acc))
5
6 plt.plot(epochs, acc, 'b', label = 'Training Accouracy')
7 plt.plot(epochs, val_acc, 'r', label = 'Validation Accouracy')
8 plt.title('Gender Accouracy Graph')
9 plt.legend()
10 plt.figure()
11
12 loss = history.history['gender_output_loss']
13 val_loss = history.history['val_gender_output_loss']
14
15 plt.plot(epochs, loss, 'b', label = 'Training Loss')
16 plt.plot(epochs, val_loss, 'r', label = 'Validation Loss')
17 plt.title('Gender Loss Graph')
18 plt.legend()
19 plt.show()
```



3.6.3 Etnia

Facciamo per l'etnia la stessa cosa fatta per il genere e l'età

```
1 #Ethnicity Graph
2 acc = history.history['ethnicity_output_sparse_categorical_accuracy']
3 val_acc = history.history['val_ethnicity_output_sparse_categorical_accuracy']
4 epochs = range(len(acc))
5
6 plt.plot(epochs, acc, 'b', label = 'Training Accouracy')
7 plt.plot(epochs, val_acc, 'r', label = 'Validation Accouracy')
8 plt.title('Ethnicity Accouracy Graph')
9 plt.legend()
10 plt.figure()
11
12 loss = history.history['ethnicity_output_loss']
13 val_loss = history.history['val_ethnicity_output_loss']
14
15 plt.plot(epochs, loss, 'b', label = 'Training Loss')
16 plt.plot(epochs, val_loss, 'r', label = 'Validation Loss')
17 plt.title('Ethnicity Loss Graph')
18 plt.legend()
19 plt.show()
```



3.6.4 Matrici di confusione

Oltre al plot dei grafici andiamo ad analizzare le matrici di confusione per capire il comportamento del modello in maniera più approfondita. Prima di andare a fare il plot delle matrici abbiamo però bisogno dei dati di test, di validazione e delle predizioni.

```
1 # normalization dataset of test
2 test_datagen = ImageDataGenerator(rescale=1./255)
3
4 # generator for dataset test initial
5 test_generator = test_datagen.flow_from_dataframe(test_df,
6                                                 x_col = 'image',
7                                                 y_col = ['age', 'gender', 'ethnicity'],
8                                                 target_size = (128, 128),
9                                                 class_mode = 'multi_output',
10                                            shuffle=False,
11                                            batch_size = 64)
```

Andiamo quindi a normalizzare i dati e ad estrarre i dati di test dal dataset iniziale.

```
1 # create predict for age, gender, ethnicity
2 age_preds, gender_preds, ethnicity_preds = model.predict_generator(test_generator)
```

Successivamente estraiamo le predizioni per **età**, **genere** ed **etnia**.

```
1 # create true labels and prediction label for the confusion matrix for this transforms the array to make them have the same shape.
2 y_true_age = np.array(test_df['age'])
3 y_pred_age = age_preds.flatten().astype(int)
4
5 y_true_gender = np.array(test_df['gender'])
6 y_pred_gender = np.argmax(np.array(gender_preds.flatten())).astype(int)
7
8 y_true_ethnicity = np.array(test_df['ethnicity'])
9
10 y_pred_ethnicity = []
11 for i in ethnicity_preds:
12     y_pred_ethnicity.append(np.argmax(i))
13
```

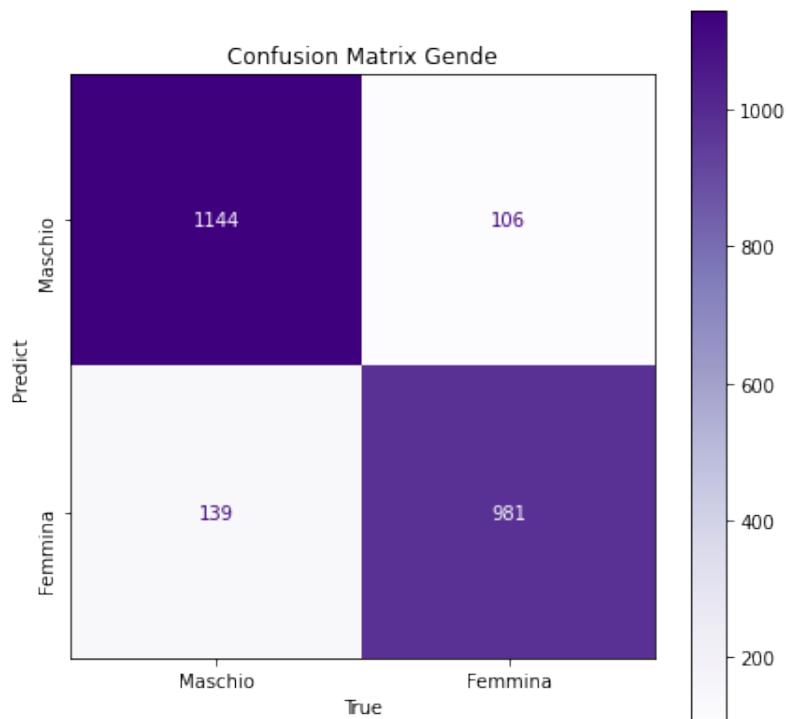
Creiamo quindi le liste per le etichette vere e per le etichette di previsione necessarie alla matrice di confusione, trasformando la matrice in modo che abbiano la stessa forma.

Facciamo questo per i 3 attributi definiti(età, genere, etnia)

Matrice confusione per il genere

```
1 # plot confusion matrix gender
2 cm = confusion_matrix(y_true_gender, y_pred_gender)
3 disp = ConfusionMatrixDisplay(confusion_matrix=cm)
4 disp.plot(cmap=plt.cm.Purples)
5 plt.title('Confusion Matrix Gende')
6 plt.yticks(np.arange(2), ['Maschio', 'Femmina'], rotation=90)
7 plt.xticks(np.arange(2), ['Maschio', 'Femmina'])
8 plt.xlabel('True')
9 plt.ylabel('Predict')
10 plt.show()
11
```

Creiamo la matrice, andiamo a posizionare le label vere sulle ascisse quelle di predizione sulle ordinate. In questo caso dato che la classe è binaria avremmo solo due valori (maschio, femmina) per ogni asse. Visualizziamo quindi il numero di predizioni corrette sulla diagonale da sinistra a destra e quelle errate sul resto della matrice.



Matrice confusione per l'etnia

```
1 # plot confusion matrix ethnicity
2 cm = confusion_matrix(y_true_ethnicity, y_pred_ethnicity)
3 disp = ConfusionMatrixDisplay(confusion_matrix=cm)
4 disp.plot(cmap=plt.cm.BuPu)
5 plt.title('Confusion Matrix Ethnicity')
6 plt.yticks(np.arange(5), ['Bianco', 'Nero', 'Asiatico', 'Indiano', 'Altro'], rotation=90)
7 plt.xticks(np.arange(5), ['Bianco', 'Nero', 'Asiatico', 'Indiano', 'Altro'])
8 plt.xlabel('True')
9 plt.ylabel('Predict')
10 plt.show()
11 plt.show()
```

Anche qua creiamo la matrice, posizionando le label vere e quelle di predizione rispetto alle classi di etnia definite in precedenza e facciamo il plot.

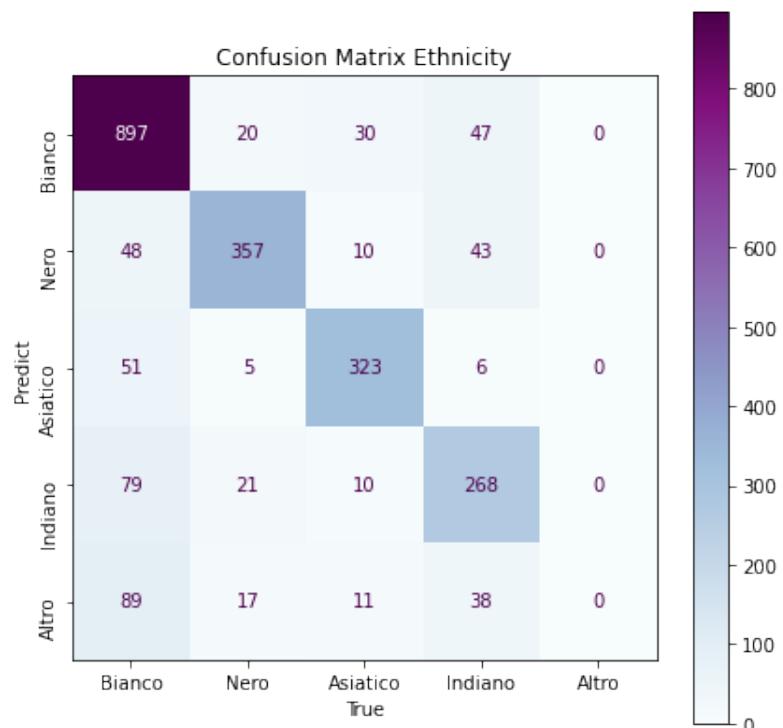
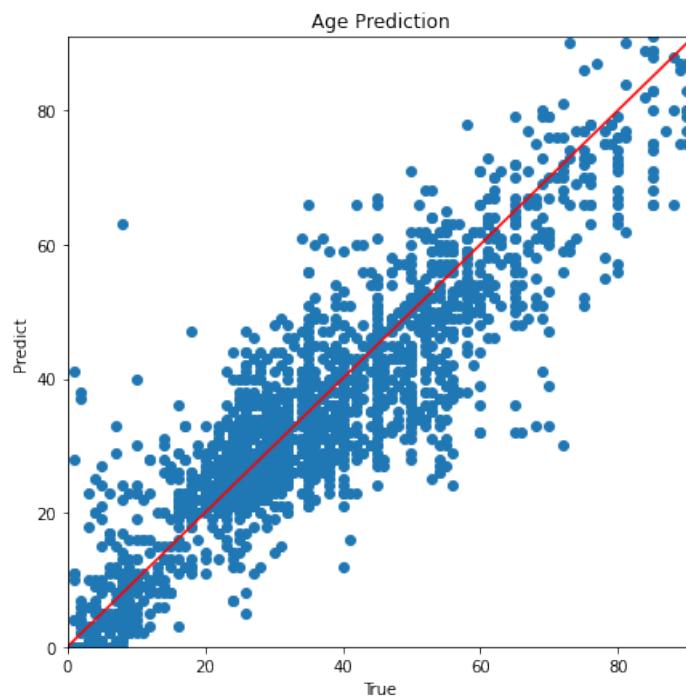


Grafico predizione età

```
1 # plot graph for the age
2 lineStart = y_pred_age.min()
3 lineEnd = y_pred_age.max()
4
5 plt.rcParams["figure.figsize"] = (7,7)
6 plt.figure()
7 plt.title('Age Prediction')
8 plt.scatter(y_true_age, y_pred_age)
9 plt.plot([lineStart, lineEnd], [lineStart, lineEnd], 'k-', color = 'r')
10 plt.xlim(lineStart, lineEnd)
11 plt.ylim(lineStart, lineEnd)
12 plt.xlabel('True')
13 plt.ylabel('Predict')
14 plt.show()
```

In questo caso andiamo a generare i punti che rappresentano le predizioni e la linea che rappresenta "l'andamento" delle predizioni corrette (ovvero quando $x = y$, quindi la retta diagonale). Ovviamente i più i punti sono vicini alla linea più il modello si comporta in modo corretto.



3.7 Salvataggio Modello

Dopo aver allenato il modello e aver plottato i grafici per verificarne l'efficienza, lo andiamo a salvare.

```
1 # save the model to h5 format by date time
2 from datetime import datetime
3
4 now = datetime.now()
5
6 date_time = now.strftime("%Y%m%d%H%M%S")
7
8 path = '/content/models/' + date_time + '.h5'
9 print(path)
10
11 model.save(path, save_format='h5')
```

Il modello viene salvato in base alla data e all'ora dentro la cartella **models**, in formato H5.

3.8 Esportazione del modello

Per andare a caricare il nostro modello creato in precedenza, dobbiamo portarlo in un formato leggibile da **tensorflowjs**, la libreria che permette l'utilizzo di tensorflow su javascript.

Per prima cosa quindi installiamo **tensorflowjs**

```
1 !pip install tensorflowjs
```

Successivamente convertiamo il modello in modo che tensorflowjs possa leggerlo. Il modello verrà salvato nel seguente modo:

- **model.json**

Rappresenta lo scheletro del modello con i livelli ecc...

- **file.bin**

Che rappresentano i pesi dati dall'allenamento.

```
1 !tensorflowjs_converter --input_format=keras --weight_shard_size_bytes 60000000 '/content/models/20220707210236.h5' '/content/modelJson'
```

Prendiamo in input quindi l'h5 e lo salviamo dentro la cartella modelJson, verrà salvato con il formato di keras e la dimensione massima del file .bin è 60000000 bytes.

La dimensione massima viene impostata così alta perché **tensorflowjs** supporta il caricamento di un solo file rappresentante i pesi, se lasciassimo il valore di default ne verrebbero creati più di uno.

Capitolo 4

Applicativo Mobile

4.1 Ambiente di Sviluppo

Come ambiente di sviluppo abbiamo utilizzato l'IDE **WebStorm**. Un ambiente di sviluppo integrato per JavaScript e tecnologie correlate. E' simile ad altri IDE come JetBrains, rende lo sviluppo più semplice andando ad automatizzare il lavoro e aiutandoti a gestire facilmente attività complesse.

4.2 Linguaggi e Framework

Per la realizzazione dell'applicativo mobile abbiamo scelto di utilizzare:

- **React Native**

che è un framework per applicazioni mobili open-source creato da Meta. Viene utilizzato per sviluppare applicazioni per Android, Android TV, iOS, macOS, tvOS, Web, Windows e UWP consentendo agli sviluppatori di utilizzare React framework insieme alle funzionalità della piattaforma nativa.

- **Expo**

Invece è sempre un framework che estende React Native, una libreria software nata in Meta (Facebook) per sviluppare app native iOS e Android con un'unica codebase scritta in JavaScript. Il suo SDK rende accessibili sia funzionalità native del dispositivo (per esempio la fotocamera e i sensori hardware) sia librerie di utilità standard (come ad esempio la predisposizione per l'autenticazione tramite i principali provider come Apple e Google).

Entrambi i framework utilizzano come linguaggio preponderante **Javascript** o più precisamente **TypeScript**. TypeScript è un linguaggio di programmazione open source sviluppato da Microsoft. Il linguaggio estende la sintassi di **JavaScript** (seguendo le specifiche ECMAScript 6 e aggiungendo un sistema di tipizzazione esplicita) in modo che qualunque programma scritto in JavaScript sia anche in grado di funzionare con TypeScript senza nessuna modifica. È stato progettato per lo sviluppo di grandi applicazioni ed è destinato a essere compilato in JavaScript per poter essere interpretato da qualunque web browser o app.

4.3 Installazione

Per l'installazione sia di **Expo** che di **React Native** c'è bisogno prima dell'installazione di nodejs, che sia una versione superiore alla 14, siccome entrambi i framework si basano su di esso. L'installazione di nodejs cambia a seconda del sistema operativo utilizzato, nel nostro caso abbiamo usato ubuntu, e possiamo installarlo con i seguenti comandi:

```
sudo apt install nodejs
```

Verificare la versione di Node.js installata:

```
nodejs -v
```

Inoltre abbiamo bisogno di **NPM** è un Package Manager per Node.js (Node Package Manager) necessario per installare moduli e pacchetti da utilizzare con Node.js.

```
sudo apt install npm
```

Verificare la versione di NPM installata:

```
npm -v
```

4.3.1 Expo

Andremo poi ad installare expo, in cui prima di effettuare l'installazione c'è bisogno di iscriversi al sito ufficiale (gratuitamente) ed creare il progetto.

Per installare expo:

```
npm install -g expo-cli
```

Per verificare che sia andato tutto a buon fine.

```
expo whoami
```

Per registrarsi al sito digitare

```
expo register
```

Dopo essersi registrati possiamo fare il log con il comando:

```
expo login
```

4.3.2 React Native

React Native viene installato insieme ad expo, quindi ci basta utilizzare la cli per inizializzare il progetto

```
expo init "nome progetto"
```

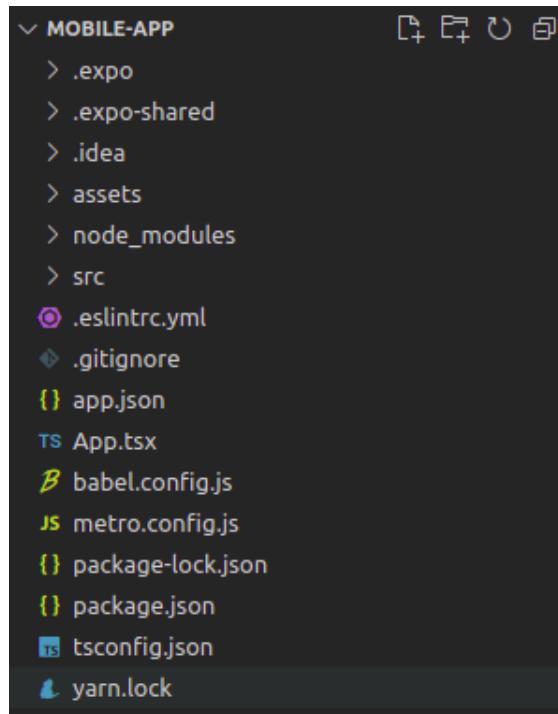
ed il progetto verrà compilato. Dopo l'esecuzione del comando verrà generato un qr code che se inquadrato potrà far eseguire il nostro progetto sia su applicativi **IOS** che **Android**, dopo aver scaricato l'app expo sul suddetto dispositivo.

Per avviare l'ambiente di sviluppo in un secondo momento basta trovarsi all'interno della directory e digitare

```
expo start
```

4.4 Struttura del progetto

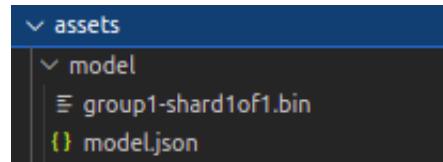
Il nostro progetto dopo l'installazione sarà diviso in molte cartelle e sotto cartelle, com'è possibile vedere nel seguente screen:



Per ora prenderemo in considerazione solo la parte chiamata **src** dove sarà presente la maggior parte del codice sorgente definito all'interno dell'app. Le altre cartelle solo in prevalenza file di sistema che servono per la build dell'app oppure per i controlli sul compilatore.

4.5 Assets

All'interno della cartella assets andiamo a caricare il nostro modello.



Il modello viene salvato nella cartella model, dove abbiamo 2 file:

- **file.bin**

che rappresenta i pesi del nostro modello.

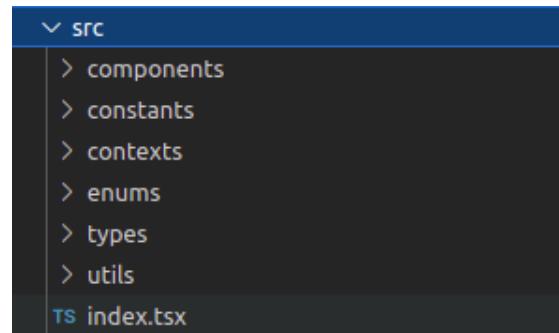
- **model.json**

che rappresenta la struttura del nostro modello.

Andremo ad esportare entrambi dentro 2 cartelle chiamate **MODEL_PATH** per il file.json e **WEIGHTS_PATH** per il file.bin definite dentro la cartella **contexts**

4.6 Src

Come detto in precedenza la seguente cartella contiene tutto il codice **TypeScript** per il funzionamento dell'app. La cartella **src** ha la seguente struttura:



Dove ogni cartella sviluppa una funzionalità diversa. Tutte le funzionalità verranno riassunte ed invocate all'interno del file **index.tsx**

4.6.1 Componets

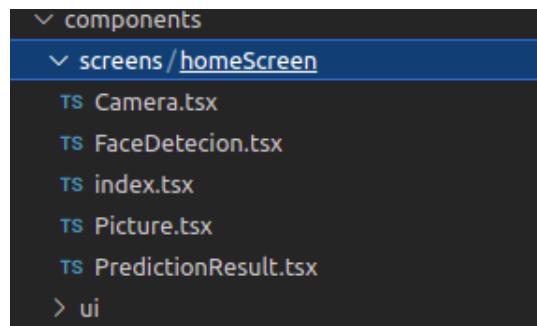
All'interno della cartella **components** troveremo altre 2 cartelle chiamate:

- **screens**

- **ui**

Andiamo ad analizzare il contenuto.

4.6.2 screens/homepage



Per prima cosa analizzeremo il file **Camera.tsx**. All'interno del file andiamo appunto ad interagire con la parte nativa della fotocamera del device. Per garantire anche il funzionamento della camera su expo è necessario per prima cosa installare i pacchetti con il comando:

```
npm i expo-camera
```

E appunto andiamo a definire la camera e le varie funzioni all'interno del file.

Camera.tsx

```
import React, {useState, useEffect, Dispatch, SetStateAction, useCallback, useRef} from 'react'
import {StyleSheet, Text, TouchableOpacity, View} from 'react-native'
import {Camera, CameraCapturedPicture, CameraType, FaceDetectionResult} from 'expo-camera'
import {Face} from 'expo-camera/build/Camera.types'
import * as FaceDetector from 'expo-face-detector'
import {Icon} from '@rneui/base'
import {FaceDeteacion} from './FaceDeteacion'
import {CameraCapturedPictureWithError} from '../../../../../types/cameraCapturedPictureWithError'
import {cropFaceFromImage} from '../../../../../utils/images'
import {ICON_COLOR} from '../../../../../constants/colors'
```

Qui abbiamo i vari import da librerie esterne e da file interni al progetto che andremo ad usare all'interno del file.

```

export const CustomCamera = ({setPicture, setOpenCamera}: TProps) => {
  const [hasPermission, setHasPermission] = useState<boolean | null>(null)
  const [type, setType] = useState(CameraType.front)
  const [cameraReady, setCameraReady] = useState(false)
  const [detectedFace, setDetectedFace] = useState<Face | undefined>(undefined)

  const cameraRef = useRef<Camera>(null)

  const handleFlip = useCallback(() => {
    setType(type === CameraType.back ? CameraType.front : CameraType.back)
  }, [type])

  const handleCancel = useCallback(() => {
    setOpenCamera(false)
  }, [setOpenCamera])

  const handleCapture = useCallback(async () => {
    if (!cameraReady || !cameraRef) {
      return
    }

    await cameraRef.current?.takePictureAsync({
      onPictureSaved: async (cameraCapturedPicture: CameraCapturedPicture) =>
        setPicture(
          await cropFaceFromImage(cameraCapturedPicture, detectedFace, type)
        )
    })
  }, [cameraReady, detectedFace])

  const handleFacesDetected = (faces: FaceDetectionResult) => {
    const face = faces.faces[0]
    if (face && typeof face !== 'undefined') {
      setDetectedFace(face)
      return
    }
    setDetectedFace(undefined)
  }

  useEffect(() => {
    (async () => {
      const {status} = await Camera.requestCameraPermissionsAsync()
      setHasPermission(status === 'granted')
    })()
  }, [()])
}

```

Nel seguente screen è possibile vedere invece sia le varie funzionalità della fotocamera ovvero quella di "fare" la foto direttamente dalla parte nativa, che possiamo vedere anche che prima di utilizzare la fotocamera è richiesto l'accesso da parte dell'utente.

```

return (
  <View style={styles.container}>
    <Camera
      style={styles.camera}
      type={type}
      onCameraReady={() => setCameraReady(true)}
      ref={cameraRef}
      ratio={'1:1'}
      onFacesDetected={handleFacesDetected}
      faceDetectorSettings={{
        mode: FaceDetector.FaceDetectorMode.fast,
        detectLandmarks: FaceDetector.FaceDetectorLandmarks.none,
        runClassifications: FaceDetector.FaceDetectorClassifications.none,
        minDetectionInterval: 100,
        tracking: true,
      }}
    >
      <View style={styles.topRowContainer}>
        <TouchableOpacity style={styles.abortButton} onPress={handleCancel}>
          <Icon type="antdesign" name="close" size={30} color={ICON_COLOR}/>
        </TouchableOpacity>
      </View>

      <View style={styles.bottomRowContainer}>
        <View style={styles.placeholder}/>

        <TouchableOpacity style={styles.captureButton} onPress={handleCapture}>
          <Icon type="entypo" name="circle" size={70} color={ICON_COLOR}/>
        </TouchableOpacity>

        <TouchableOpacity style={styles.flipButton} onPress={handleFlip}>
          <Icon type="antdesign" name="sync" size={30} color={ICON_COLOR}/>
        </TouchableOpacity>
      </View>

      </Camera>
      {detectedFace && <FaceDetection face={detectedFace}/>}
    </View>
  )
)

```

Successivamente andiamo a definire i componenti jsx (molto simile all'html) per l'utilizzo della fotocamera (con buttoni ecc..) per catturare la foto.

```
const styles = StyleSheet.create({
  container: {
    height: '100%',
  },
  camera: {
    display: 'flex',
    height: '100%',
    justifyContent: 'space-between',
  },
  topRowContainer: {
    backgroundColor: 'black',
    justifyContent: 'flex-end',
    height: 120,
    opacity: 0.8,
    padding: 20
  },
  bottomRowContainer: {
    backgroundColor: 'black',
    display: 'flex',
    flexDirection: 'row',
    justifyContent: 'space-between',
    height: 150,
    opacity: 0.8,
    padding: 20
  },
  abortButton: {
    alignSelf: 'flex-start',
    justifyContent: 'center'
  },
  flipButton: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  captureButton: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  placeholder: {
    flex: 1,
  },
  text: {
    fontSize: 18,
    color: 'white',
  },
})
```

Qui invece andiamo a lavorare con i css per modellare lo style dei componenti jsx definiti in precedenza.

FaceDetection.tsx

Successivamente abbiamo implementato un detector per tagliare la foto dopo lo scatto in modo da andare a rimuovere del "rumore" intorno al volto.

```
import * as React from 'react'
import {Face} from 'expo-camera/build/Camera.types'
import {StyleSheet, View} from 'react-native'
import {SECONDARY_COLOR} from '../../../../../constants/colors'

type TProps = {
  face: Face
}

export const FaceDeteccion = ({face}: TProps) => {
  const {size, origin} = face.bounds

  return (
    <View
      style={[
        styles.face,
        {
          width: size.width,
          height: size.height,
          left: origin.x,
          top: origin.y,
        },
      ]}
    >
      </View>
    )
}

const styles = StyleSheet.create({
  facesContainer: {
    position: 'absolute',
    bottom: 0,
    right: 0,
    left: 0,
    top: 0,
  },
  face: {
    padding: 10,
    borderWidth: 2,
    borderRadius: 1,
    position: 'absolute',
    borderColor: SECONDARY_COLOR,
    justifyContent: 'center',
    backgroundColor: 'transparent',
  },
})
```

Dove il detector è un container con lo sfondo trasparente in modo da vedere la faccia ed ha i bordi colorati di verde.

Picture.tsx

In questo file andiamo a trasformare l'immagine in un uri che poi useremo successivamente, in più andiamo a definire la larghezza e l'altezza della foto da visualizzare.

```
import React, {useEffect} from 'react'
import {View, StyleSheet, Image} from 'react-native'
import {CameraCapturedPictureWithError} from '../../types/cameraCapturedPictureWithError'
import {FaceDetectionError} from '../../enums/faceDetectionError'
import {faceDetectionErrorMessages} from '../../constants/faceDetectionErrorMessages'
import {ErrorMessage} from '../../ui/ErrorMessage'

type TProps = {
  picture: CameraCapturedPictureWithError,
  error: FaceDetectionError | undefined
}

export const Picture = ({picture, error}: TProps) => {
  useEffect(() => {
    if (picture.error) {
      console.error(FaceDetectionError[picture.error])
    }
    if (error) {
      console.error(error)
    }
  }, [picture])

  return (
    <View>
      <Image
        source={{uri: picture.uri}}
        style={styles.image}
      />

      <ErrorMessage show={typeof picture.error !== 'undefined'}>
        {faceDetectionErrorMessages[picture.error!]}
      </ErrorMessage>

      <ErrorMessage show={typeof error !== 'undefined'}>
        {faceDetectionErrorMessages[error!]}
      </ErrorMessage>

    </View>
  )
}

const styles = StyleSheet.create({
  image: {
    width: 350,
    height: 350,
    borderRadius: 4
  },
})
```

Prediction.tsx

Nel seguente file invece andiamo a prendere le predizioni fatte dal modello e le andiamo a stampare a video.

```
import React from 'react'
import {Text, View, StyleSheet} from 'react-native'
import {Prediction} from '../../../../../types/prediction'

type Tprops = {
  prediction: Prediction
}

export const PredictionResult = ({prediction}: Tprops) => {
  return (
    <View style={styles.container}>
      <Text style={styles.name}>Sesso:</Text><Text style={styles.prediction}>{prediction.gender}</Text>
      <Text style={styles.name}>Età:</Text><Text style={styles.prediction}>{prediction.age}</Text>
      <Text style={styles.name}>Etnia:</Text><Text style={styles.prediction}>{prediction.ethnicity}</Text>
    </View>
  )
}

const styles = StyleSheet.create({
  container: {
    display: 'flex',
    flexDirection: 'row',
    flexWrap: 'wrap',
    justifyContent: 'center',
    paddingBottom: 20,
    height: 50
  },
  name: {
    flexBasis: '40%',
    fontWeight: 'bold',
    textAlign: 'right',
    padding: 4
  },
  prediction: {
    flexBasis: '40%',
    padding: 4
  }
})
```

index.tsx

Questo file utilizza tutti i componenti definiti dagli altri file nella cartella **homepage**, li combina aggiungendo logica e li espone come un unico componente.

Andiamo prima ad importare sempre i metodi e le librerie sia esterne che interne di cui abbiamo bisogno.

```
import React, {useCallback, useState} from 'react'
import {StyleSheet, View} from 'react-native'
import {CustomCamera as Camera} from './Camera'
import {Picture} from './Picture'
import {Button} from '../ui/Button'
import {PredictionResult} from './PredictionResult'
import {Prediction} from '../../../../../types/prediction'
import {useTensorflow} from '../../../../../contexts/tensorflow'
import {Loading} from '../ui>Loading'
import {BACKGROUND_COLOR} from '../../../../../constants/colors'
import {CameraCapturedPictureWithError} from '../../../../../types/cameraCapturedPictureWithError'
import {FaceDetectionError} from '../../../../../enums/faceDetectionError'
```

Andiamo poi a definire la nostra home con i vari bottoni e gli spinner di caricamento.

```
export const HomeScreen = () => [
  const [pic, setPic] = useState<CameraCapturedPictureWithError | null>(null)
  const [openCamera, setOpenCamera] = useState(false)
  const [prediction, setPrediction] = useState<Prediction | null>(null)
  const [error, setError] = useState<FaceDetectionError | undefined>(undefined)

  const {predict, isModelLoading} = useTensorflow()

  const handleShoot = useCallback(() => {
    setPic(null)
    setPrediction(null)
    setOpenCamera(true)
  }, [])

  const handlePredict = useCallback(async () => {
    if (!pic) {
      setError(FaceDetectionError.NO_PICTURE_TAKEN)
      return
    }

    const res = await predict(pic)
    setPrediction(res)
  }, [pic])

  if (isModelLoading) {
    return (
      <View style={styles.container}>
        <Loading/>
      </View>
    )
  }

  if (!pic) {
    return (
      openCamera ?
        <Camera setOpenCamera={setOpenCamera} setPicture={setPic}/>
        :
        <View style={styles.container}>
          <Button onPress={handleShoot}>Scatta una foto</Button>
        </View>
    )
  }

  return (
    <View style={styles.container}>
      <Button onPress={handleShoot}>Riscatta la foto</Button>
      <Picture picture={pic} error={error}/>
      {!prediction ?
        <Button onPress={handlePredict}>Fai una predizione</Button>
        :
        <PredictionResult prediction={prediction}/>
      }
    </View>
  )
]
```

Infine andiamo ad aggiungere il css.

```
const styles = StyleSheet.create({
  container: {
    backgroundColor: BACKGROUND_COLOR,
    flex: 1,
    justifyContent: 'space-evenly',
    alignItems: 'center'
  }
})
```

4.6.3 ui

All'interno della directory **ui** andiamo a definire qui componenti grafici riutilizzabili con cui appunto andrà ad interagire l'utente.

Button.tsx

```
import {GestureResponderEvent, StyleSheet, Text, TouchableOpacity} from 'react-native'
import React from 'react'
import {PRIMARY_COLOR} from '../../../../../constants/colors'

type TProps = {
  children: string
  onPress: ((event: GestureResponderEvent) => void) | undefined
}
export const Button = ( {children, onPress}: TProps ) => {
  return(
    <TouchableOpacity style={styles.button} onPress={onPress}>
      <Text style={styles.buttonText}>{children}</Text>
    </TouchableOpacity>
  )
}

const styles = StyleSheet.create({
  button:{
    width: '90%',
    borderRadius: 4,
    backgroundColor: PRIMARY_COLOR,
    flexDirection: 'row',
    justifyContent: 'center',
    alignItems: 'center',
    height: 50
  },
  buttonText:{
    color: 'white',
    fontWeight: 'bold',
  }
})
```

In questa sezione andiamo a creare i bottoni con i quali andrà ad interagire l'utente.

ErrorMessage.tsx

```
import {StyleSheet, Text, View} from 'react-native'
import React from 'react'
import {ERROR_COLOR, ICON_COLOR} from '../../constants/colors'
import {Icon} from '@rneui/base'

type TProps = {
  children: string
  show: boolean
}
export const ErrorMessage = ({children, show}: TProps) => {
  if (!show) {
    return <></>
  }

  return (
    <View style={styles.error}>
      <Icon type="antdesign" name="warning" size={20} color={ICON_COLOR} style={styles.icon}/>
      <Text style={styles.errorText}>{children}</Text>
    </View>
  )
}

const styles = StyleSheet.create({
  error: {
    borderRadius: 4,
    backgroundColor: ERROR_COLOR,
    flexDirection: 'row',
    justifyContent: 'center',
    alignItems: 'center',
    height: 50,
    marginTop: -50
  },
  icon: {
    paddingRight: 20
  },
  errorText: {
    color: 'white',
    fontWeight: 'bold',
  }
})
```

Qui andiamo a creare il messaggio d'errore nel caso in cui non venga trovato nessun volto nella foto.

Loading

```
import * as React from 'react'
import {ActivityIndicator, Text, View} from 'react-native'
import {PRIMARY_COLOR} from '../../constants/colors'

export const Loading = () => {
  return (
    <View>
      <ActivityIndicator size="large" color={PRIMARY_COLOR}/>
      <Text>Caricamento del modello...</Text>
    </View>
  )
}
```

In questa sezione invece carichiamo lo spinner definito precedentemente per il caricamento del modello.

4.6.4 constants

In questa cartella abbiamo la definizione di alcuni valori costanti come le stringhe di errore ed i colori.

colors.ts

```
export const BACKGROUND_COLOR = '#DDD5D0'
export const PRIMARY_COLOR = '#586F6B'
export const SECONDARY_COLOR = '#1cbe8b'
export const ICON_COLOR = '#FFFFFF'
export const ERROR_COLOR = '#a2411e'
```

Definizione dei colori.

faceDetectionErrorMessages.ts

```
import {FaceDetectionError} from '../enums/faceDetectionError'

const faceDetectionErrorMessages: string[] = []

faceDetectionErrorMessages[FaceDetectionError.NO_PICTURE_TAKEN] = 'La foto non è stata scattata'
faceDetectionErrorMessages[FaceDetectionError.NO_FACE_DETECTED] = 'Non è stato riconosciuto alcun volto'

export {
  faceDetectionErrorMessages
}
```

Definizione messaggi d'errore.

4.6.5 contexts

tensorflow.tsx

All'interno di questo file abbiamo tutto le iterazioni che avvengono con la libreria di **tensorflow** come per esempio il caricamento del modello e la predizione.

```
import React, {createContext, useCallback, useContext, useEffect, useState} from 'react'
import {Prediction} from '../types/prediction'
import * as tf from '@tensorflow/tfjs'
import '@tensorflow/tfjs-react-native'
import {bundleResourceIO, decodeJpeg} from '@tensorflow/tfjs-react-native'
import {CameraCapturedPicture} from 'expo-camera'
import {preprocessImageForTensorflow} from '../utils/images'
import {interpretPrediction} from '../utils/prediction'

const MODEL_DIR = '../../assets/model'
const MODEL_PATH = `${MODEL_DIR}/model.json`
const WEIGHTS_PATH = `${MODEL_DIR}/group1-shard0of1.bin`

type TProps = {
  children: JSX.Element
}

type TContext = {
  isModelLoading: boolean
  predict: (T: CameraCapturedPicture) => Promise<Prediction | null>
}

const TensorflowContext = createContext({} as TContext)

export const TensorflowProvider = ({children}: TProps) => {
  const [isModelLoading, setIsModelLoading] = useState(false)
  const [model, setModel] = useState<tf.LayersModel | null>(null)

  /**
   * Loads the model from the `assets` directory, while doing so puts the app in a loading state.
   */
  const loadModel = useCallback(async () => {
    setIsModelLoading(true)
    await tf.ready()
    try {
      // eslint-disable-next-line @typescript-eslint/no-var-requires
      const modelJson = require(MODEL_PATH)
      // eslint-disable-next-line @typescript-eslint/no-var-requires
      const modelWeights = require(WEIGHTS_PATH)

      const m = await tf.loadLayersModel(bundleResourceIO(modelJson, modelWeights))
      setModel(m)
    } catch (e) {
      console.error(e)
    }
    setIsModelLoading(false)
  }, [])
  return (
    <TensorflowContext.Provider value={{isModelLoading, predict: loadModel}}>
      {children}
    </TensorflowContext.Provider>
  )
}
```

Qui oltre a definire i tipi per la predizione, le varie cartelle dov'è caricato il modello, abbiamo la funzione **loadModel** che si occupa appunto di andare a caricare effettivamente il modello sul device.

```

/**
 * Processes the picture, normalizes it and calls model.predict().
 *
 * @param {CameraCapturedPicture} picture Full picture taken from camera
 * @return {Prediction | null} Human-readable prediction, null if there is an error
 */
const predict = useCallback(async (picture: CameraCapturedPicture) => {
  if (model === null) {
    console.error('model is not ready yet')
    return null
  }

  try {
    const processedPicture = await preprocessImageForTensorflow(picture)
    const imageTensor = decodeJpeg(processedPicture.data, 3)
    const reshaped = tf.reshape(imageTensor, [1, 128, 128, 3])
    const normalized = reshaped.div(255)
    const result = model.predict(normalized)

    return interpretPrediction(result)
  } catch (e) {
    console.error(e)
    return null
  }
}, [model])

// the model is loaded as soon as the app starts
useEffect(() => {
  (async () => loadModel())()

  return () => {
    setModel(null)
  }
}, [])

return (
  <TensorflowContext.Provider value={{
    isModelLoading,
    predict,
  }}>
    {children}
  </TensorflowContext.Provider>
)
}

export const useTensorflow = () => useContext<TContext>(TensorflowContext)

```

Qua invece andiamo a processare l'immagine ed a fare la predizione dopo aver passato l'immagine al modello.

4.6.6 enums

Qui sono presenti una serie di valori che chiameremo in base ad i risultati ottenuti.

ethnicity.ts

```
// export enum Ethnicity {  
//   'White' = 0,  
//   'Black' = 1,  
//   'Asian' = 2,  
//   'Indian' = 3,  
//   'Other' = 4  
// }  
  
export enum Ethnicity {  
  'Bianco' = 0,  
  'Nero' = 1,  
  'Asiatico' = 2,  
  'Indiano' = 3,  
  'Altro' = 4  
}
```

Valori classi di etnia.

faceDetectionError.ts

```
export enum FaceDetectionError {  
  'NO_FACE_DETECTED',  
  'NO_PICTURE_TAKEN'  
}
```

Errori per il face detection.

gender.ts

```
// export enum Gender {  
//   'Male' = 0,  
//   'Female' = 1,  
// }  
  
export enum Gender {  
  'Maschio' = 0,  
  'Femmina' = 1,  
}
```

Valori per il genere.

4.6.7 types

In questa cartella abbiamo la definizione dei tipi usati all'interno delle altre cartelle.

cameraCapturedPictureWithError

```
import {CameraCapturedPicture} from 'expo-camera'
import {FaceDetectionError} from '../enums/faceDetectionError'

export type CameraCapturedPictureWithError = CameraCapturedPicture & { error?: FaceDetectionError }
```

Rappresenta il tipo CameraCapturedPicture che contiene l'uri dell'immagine e le sue dimensioni, a cui andiamo ad aggiungere un possibile errore, così da essere gestito da componenti diversi.

predictions.ts

```
export type Prediction = {
  age: number
  ethnicity: string
  gender: string
}
```

Rappresenta il tipo della predizione già interpretata e leggibile dall'utente.

uninterpretedPrediction

```
import * as tf from '@tensorflow/tfjs'

export type UninterpretedPrediction = tf.Tensor<tf.Rank> | tf.Tensor<tf.Rank>[]
```

Rappresenta il tipo della predizione non interpretata, quindi solamente il tensore che viene restituito come risultato da **model.predict()**

4.6.8 utils

In questa cartella andiamo ad effettuare tutte le operazioni e gli aggiustamenti per rendere il lavoro più efficiente. Rappresenta come una cassetta degli attrezzi dove possiamo andare a prendere quello di cui più abbiamo bisogno.

array.ts

```
/**  
 * Retrieve the array key corresponding to the largest element in the array.  
 *  
 * @param {Array.<number>} array Input array  
 * @return {number} Index of array element with the largest value  
 */  
export const argMax = (array: never[]) => {  
    return array.map((x, i) => [x, i]).reduce((r, a) => (a[0] > r[0] ? a : r))[1]  
}
```

In questo file è presente una funzione che si occupa di recuperare il valore nell'array corrispondente all'elemento più grande.

images.ts

```
import {FlipType, manipulateAsync} from 'expo-image-manipulator'
import {fetch} from '@tensorflow/tfjs-react-native'
import {CameraCapturedPicture, CameraType} from 'expo-camera'
import {Face} from 'expo-camera/build/Camera.types'
import {FaceDetectionError} from '../enums/faceDetectionError'
import {Dimensions} from 'react-native'

/**
 * Resizes an image to be 128x128 pixels.
 *
 * @param {CameraCapturedPicture} image Input image
 * @return {CameraCapturedPicture} Resized image
 */
const resizeImage = async (image: CameraCapturedPicture) => {
  return manipulateAsync(image.uri, [{resize: {height: 128, width: 128}}])
}

/**
 * Updates a CameraCapturedPicture `data` field to store a Uint8Array instead of the image uri.
 *
 * @param {CameraCapturedPicture} image Input image
 * @return {CameraCapturedPicture} Image object with update `data` field
 */
const imageToUint8Array = async (image: CameraCapturedPicture) => {
  const response = await fetch(image.uri, {}, {isBinary: true})
  const imageDataArrayBuffer = await response.arrayBuffer()
  const data = new Uint8Array(imagedataArrayBuffer)

  return {
    data,
    height: image.height,
    width: image.width
  }
}

/**
 * Process the image passed to be used with the tensorflow model. It applies resizeImage() and imageToUint8Array().
 *
 * @param {CameraCapturedPicture} image Input image
 * @return {CameraCapturedPicture} Processed image
 */
export const preprocessImageForTensorflow = async (image: CameraCapturedPicture) => {
  const r = await resizeImage(image)
  return imageToUint8Array(r)
}

/**
 * Crops the image using the coordinates returned from the face recognition.
 *
 * @param {CameraCapturedPicture} image Input image
 * @param {Face | undefined} detectedFaceFeatures Coordinates of the face recognition square
 * @param {CameraType} cameraType Front or Back camera
 * @return {CameraCapturedPicture} Processed image
 */
*/
```

Nel seguente file invece andiamo a fare il resize dell'immagine a 128x128 in modo da passarla al modello con lo stesso formato in cui viene allenato. Poi andiamo a trasformare l'immagine da Uint8Array in uri. Successivamente si elabora l'immagine passata per essere utilizzata con il modello tensorflow. L'immagine successivamente verrà ritagliata in modo che il modello funzioni in maniera più efficente.

```

export const cropFaceFromImage = async (
  image: CameraCapturedPicture,
  detectedFaceFeatures: Face | undefined,
  cameraType: CameraType
) => {
  // if face recognition does not find a face returns an error
  if (typeof detectedFaceFeatures === 'undefined') {
    return {
      ...image,
      error: FaceDetectionError.NO_FACE_DETECTED
    }
  }

  // flipping the image only if it's taken from front cameras
  const flippedImage = cameraType === CameraType.front ?
    await manipulateAsync(image.uri, [{flip: FlipType.Horizontal}]) :
    image

  const {origin, size} = detectedFaceFeatures.bounds

  // calculating coordinates based on the screen dimensions
  const windowHeight = Dimensions.get('screen').width
  const windowHeight = Dimensions.get('screen').height

  const wRatio = image.width / (windowWidth)
  const hRatio = image.height / (windowHeight)

  const originX = origin.x * wRatio
  const originY = origin.y * hRatio
  const width = size.width * wRatio
  const height = size.height * hRatio

  // reduce the image by 8% from the original coordinates to better fit the face
  const shift = 8
  const orizontalShift = originX / 100 * shift
  const verticalShift = originY / 100 * shift

  return manipulateAsync(flippedImage.uri, [
    {
      crop: {
        originX: originX + orizontalShift,
        originY: originY + verticalShift,
        width: width - 2 * orizontalShift,
        height: height - 2 * verticalShift
      }
    }
  ])
}

```

Anche qua vengono effettuate una serie di operazioni sull'immagine in modo da renderla più "adatta" per essere passata al modello. l'immagine viene capovolta se è stata scattata dalla fotocamera anteriore poi vengono calcolate le coordinate in base alle dimensioni dello schermo e si va a ridurre l'immagine dell'8% rispetto alle coordinate originali per adattarla meglio al volto.

prediction.ts

```
import {UninterpretedPrediction} from '../types/uninterpretedPrediction'
import {Gender} from '../enums/gender'
import {Ethnicity} from '../enums/ethnicity'
import {argMax} from './array'

/**
 * Takes a model.predict() result and maps it to fit the human-readable Prediction type.
 *
 * - age is a float number, returned as a rounded int.
 * - gender is a float number between 0 and 1, closer to 0 is male and closer to 1 is female, mapped with the Gender enum.
 * - ethnicity is an array of 5 float numbers between 0 and 1, the index with the largest value is the index of the ethnicity with the best probability, it is mapped with the Ethnicity enum.
 *
 * @param {UninterpretedPrediction} prediction Input image
 * @return {Prediction} Image object with update `data` field
 */
export const interpretPrediction = async (prediction: UninterpretedPrediction) => {
    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-ignore
    const age = await prediction[0].arraySync()[0][0]
    console.log('age:', age)

    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-ignore
    const gender = await prediction[1].arraySync()[0]
    console.log('gender:', gender[0])

    // eslint-disable-next-line @typescript-eslint/ban-ts-comment
    // @ts-ignore
    const ethnicityArray = await prediction[2].arraySync()[0]
    const ethnicity = argMax(ethnicityArray)
    console.log('ethnicity:', ethnicityArray.map((value: number, index: number) => `${Ethnicity[index]}: ${value}`))

    return {
        gender: Gender[Math.round(gender)],
        age: Math.round(age),
        ethnicity: Ethnicity[ethnicity]
    }
}
```

Nel seguente file si va a stampare sulla console i valori di quanto la foto appartenga ad i valori nelle classi descritte in precedenza. Quindi andiamo a prendere il risultato di **model.predict()** e lo mappiamo in modo che diventi leggibile.

I valori per le predizioni stampati in console sono i seguenti.

1. Età

è un numero float, restituito come un int arrotondato.

2. Genere

è un numero float compreso tra 0 e 1, più vicino a 0 se è maschio e più vicino a 1 se è femmina, mappato con l'enum Gender.

3. Etnia

è un array di 5 numeri float compresi tra 0 e 1, l'indice con il valore più grande è l'indice dell'etnia con la migliore probabilità. etnia con la migliore probabilità, è mappato con l'enum Ethnicity.

4.7 Risultato finale dell'app



Figura 4.1: Schermata Iniziale



Figura 4.2: Caricamento del modello



Figura 4.3: Foto presa dalla camera dello smartphone



Figura 4.4: Messaggio d'errore



Figura 4.5: Bottone per fare la predizione dopo aver scattato la foto



Figura 4.6: Bottone per fare la predizione

Capitolo 5

Conclusioni

Abbiamo fatto svariati test e l'app sembra che funzioni bene solo in alcune circostanze, nonostante il modello sembri abbastanza accurato. Si tiene a specificare che i soggetti nelle foto sono stati informati che le loro foto verranno riportate per scopi scientifici e che hanno dato tutti il **CONSENSO** al fine dell'utilizzo. Inoltre non abbiamo potuto provare tutte le classi definite all'interno dell'etnia per mancanza di campioni.

Riporto di seguito alcuni degli esempi classificati correttamente.

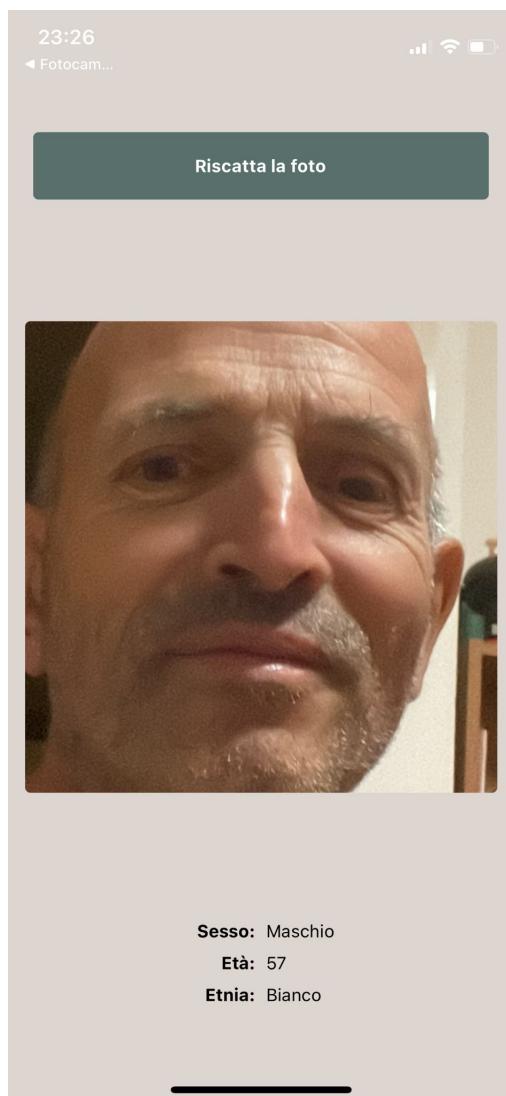
5.1 Classificazioni Corrette



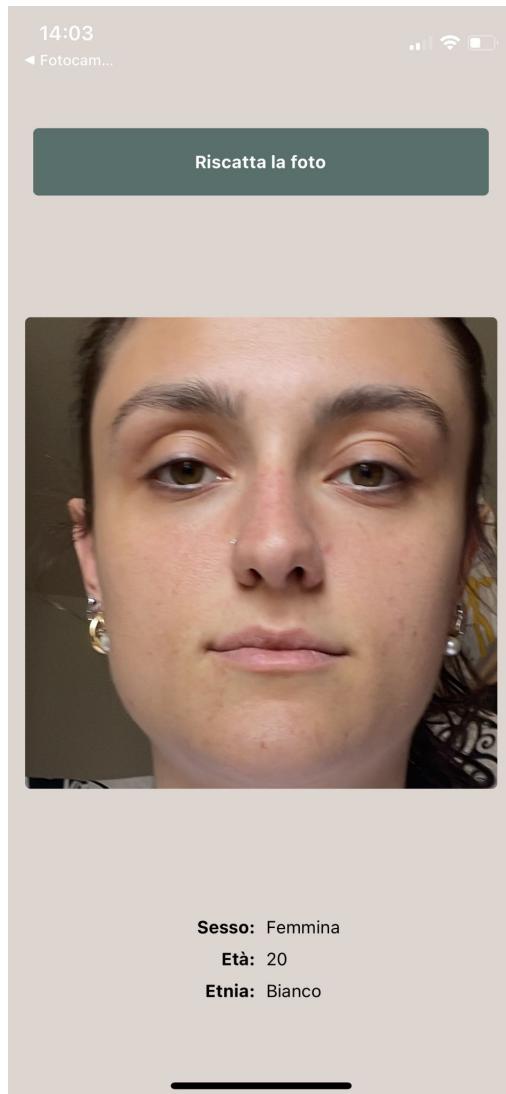
In questo caso possiamo vedere che i campi di sesso ed etnia sono corretti mentre l'età reale corrisponde a 23 anni.



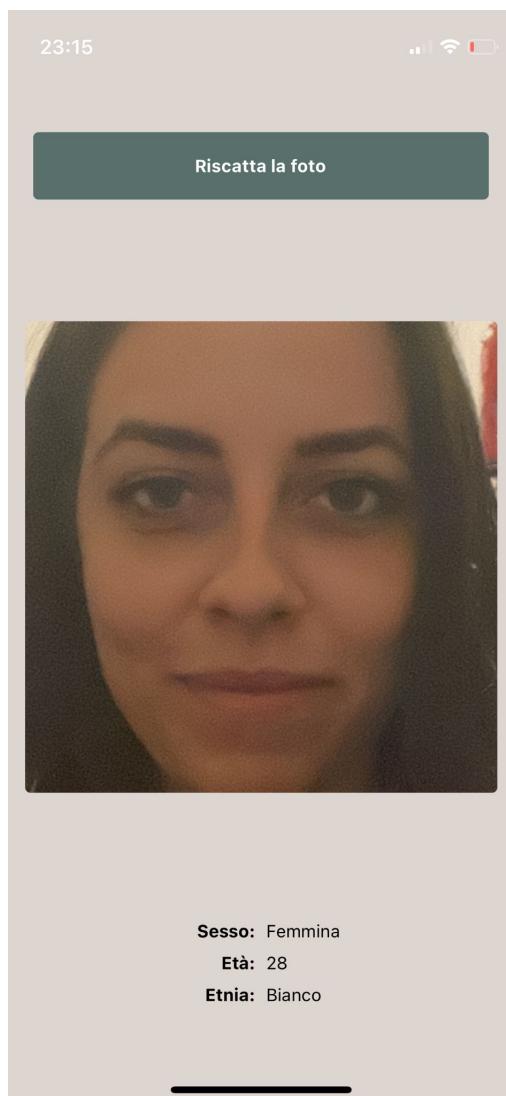
In questo caso il modello va a classificare tutti i campi in **modo corretto**.



In questo caso possiamo vedere che i campi di sesso ed etnia sono corretti mentre l'età reale corrisponde a **58 anni**.



In questo caso possiamo vedere che i campi di sesso ed etnia sono corretti mentre l'età reale corrisponde a **21 anni**.

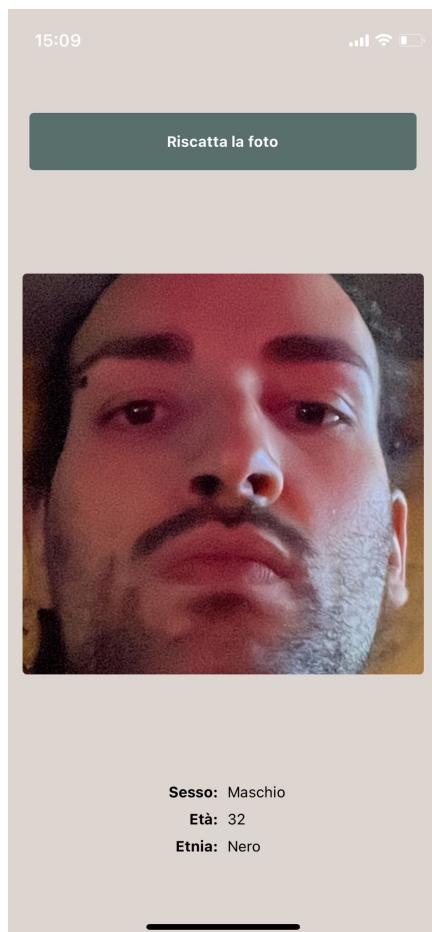


In questo caso possiamo vedere che i campi di sesso ed etnia sono corretti mentre l'età reale corrisponde a **22 anni**.

5.2 Problemi

Facendo vari test abbiamo notato come l'esposizione alla luce nelle foto, va cambiare i risultati dell'app e come per esempio facendo delle foto allo schermo, il rumore prodotto dallo schermo va ad influire sulle prestazioni del modello. In più abbiamo notato che è anche molto importante la posizione in cui viene fatta la foto. Se la luce è corretta e la foto viene fatta **dall'alto** il modello classificherà sicuramente meglio l'età mentre potrebbe fallire sul genere. Se la foto viene fatta **dal basso** il genere classificherà meglio, e questo andrà a discapito dell'età che sarà leggermente sbagliata. Sul genere invece avremo un comportamento abbastanza regolare, molto resistente a qualsiasi tipo di rumore.

Riporto alcuni esempi.



In questo caso la foto è in condizioni di luminosità scarse e possiamo vedere che l'etnia non è corretta mentre l'età reale corrisponde a **23 anni**.



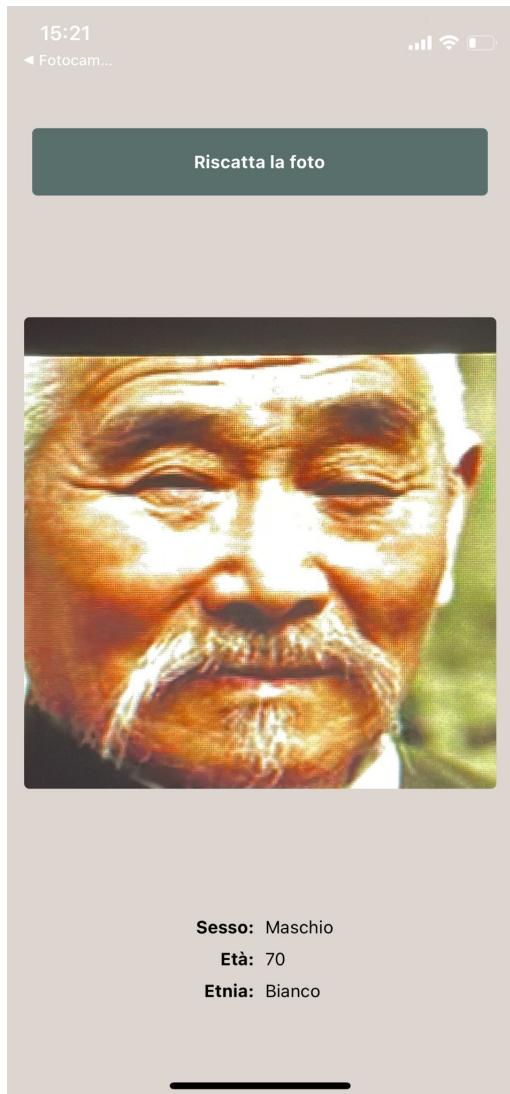
In questo caso la foto è in condizioni di luminosità scarse e possiamo vedere che l'etnia non è corretta mentre l'età reale corrisponde a **25 anni**.



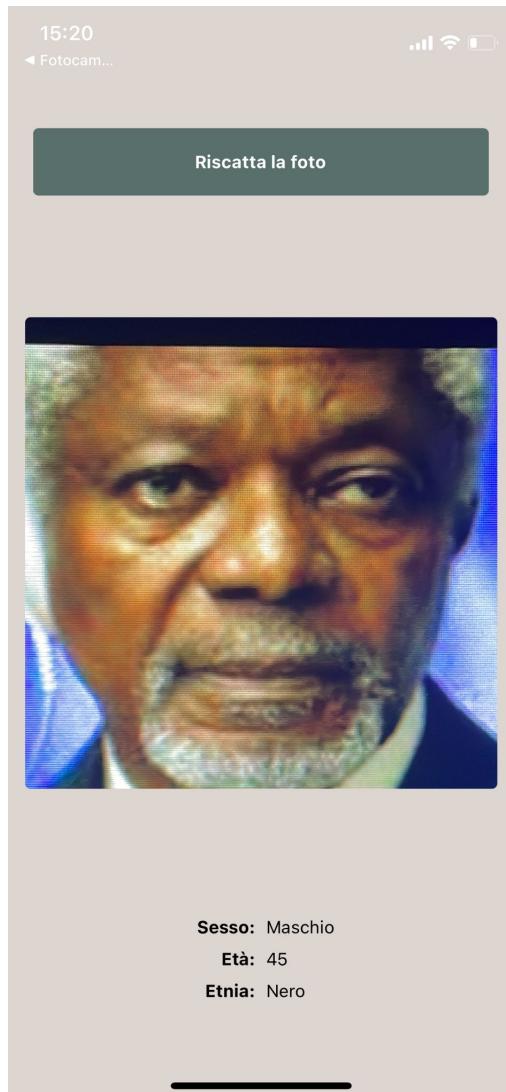
In questo caso la foto è fatta allo schermo del pc e possiamo vedere che il campo del sesso è sbagliato, l'etnia è corretta mentre l'età reale corrisponde a **65 anni**.



In questo caso la foto è fatta allo schermo del pc e possiamo vedere che il campo del sesso è corretto, l'etnia è corretta mentre l'età reale corrisponde a **72 anni**.



In questo caso la foto è fatta allo schermo del pc e possiamo vedere che il campo del sesso è corretto, l'etnia non è corretta mentre l'età reale corrisponde a **82 anni**.



In questo caso la foto è fatta allo schermo del pc e possiamo vedere che il campo del sesso è corretto, l'etnia è corretta mentre l'età reale corrisponde a **70 anni**.