# Schönheitserkennung von Gesichtern mit einem Convolutional Neural Network

**Jonathan Helmond**

**Universität: TU-Sofia**

**Kurs: Neuro-Fuzzy Systems**

**Dozent: Professor I. Topalova**

## ABSTRACT

Das Ziel dieser Arbeit ist es ein convolutionales neuronales Netzwerk aufzubauen, das in der Lage ist menschliche Gesichter nach ihrer objektiven Schönheit zu bewerten.

Keywords: Schönheit, KI, Convolutional

## EINLEITUNG

Für lange Zeit war die Klassifizierung von Schönheit eine Aufgabe, die nur Menschen bewältigen konnten. Mit dem fortschreiten der Technik und neuen Methoden der Bilderkennung ergeben sich nun neue Möglichkeiten. Heutzutage sind künstliche Intelligenzen in der Lage Gesichter zuverlässig und effizient einer Person zu zuordnen. Mithilfe der gleichen Technik sollte es möglich sein Gesichter nach ihrer objektiven Schönheit zu bewerten. Auch wenn Schönheit subjektiv ist und im Auge des Betrachters liegt kann man dennoch eine Einstimmigkeit bei der Bewertung der Schönheit festzustellen.

## METHODEN UND MATERIAL

Die Daten, die in dieser Arbeit genutzt werden stammen aus dem Datensatz SCUT-FBP5500 von [1]. Der Datensatz hierfür besteht aus Gesichtern von 5500 Personen unterschiedlichen Geschlechts. Darunter 2000 asiatische Frauen, 2000 asiatische Männer, 750 kaukasische Frauen und 750 kaukasischen Männern. Für eine möglichst objektive Bewertung der Gesichter wurden 70 Leute herangezogen um die Gesichter zu bewerten. Schlussendlich hat sich dabei gezeigt, dass die Schönheit eines Gesichts, wie erwartet, nicht eindeutig ist. Jedoch zeigt sich dass für jedes Gesicht eine Gauß-Verteilung der Bewertung entsteht , wobei die Standwartabweichung klein genug ist um von einer korrekten Zuordnung zu sprechen. Der Mittelwert der Gauß-Verteilung entspricht damit der objektiven Schönheit des Gesichts. Da Ausreißer den Mittelwert stark verzerren, wurde diese im Vorfeld aus den Daten entfernt.

Die Verteilung aller Bewertungen der Gesichter zeigt ebenfalls annähernd eine Normalverteilung, wie in Bild 2 zu sehen ist.

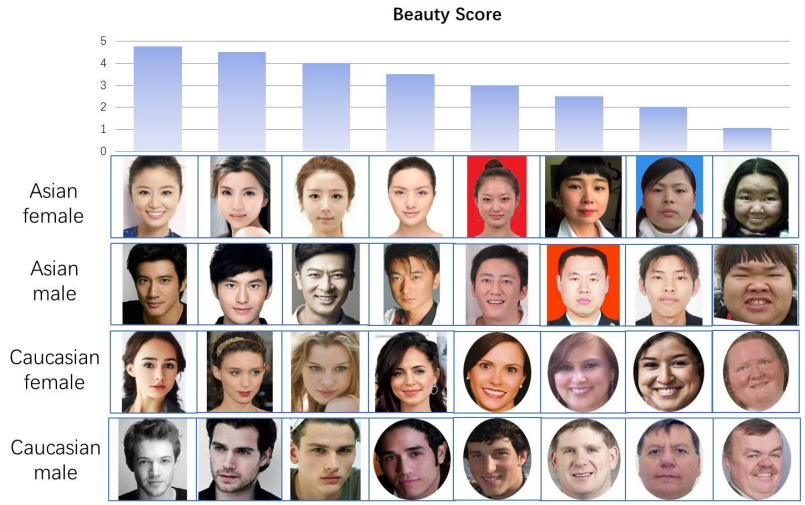



**Figure 1.** Dieses Bild zeigt die verschieden Gruppen von Gesichtern mit unterschiedlichen Bewertungen. ([1])

## IMPLEMENTIERUNG

Die Implementierung der Schönheitsklassifizierung wurde mit Google Colab umgesetzt. Google Colab bietet die Möglichkeit online Python Notebooks zu erstellen. Diese Notebooks beinhalten bereits alle Maschine-learning Bibliotheken und besitzen GPU-Beschleunigung und eignen sich somit hervorragend zur Umsetzung von Maschine-learning Modellen. Der bearbeitete Datensatz mit allen Bildern und den zugehörigen Klassifikationen wurde in der Google Drive abgespeichert und kann von dort einfach in das Colab-notebook integriert werden. Von den Daten bis zum finalen Neuronalen Netz wird wie folgt vorgegangen:

- Importieren des Datensatzes
- Konvertieren der Bilder in numerische Daten
- Speichern des Datensatzes als extrahierter Datensatz

Diese drei Schritte müssen nur einmalig ausgeführt werden damit die Daten strukturiert als Rohdaten für das neuronale Netz zur Verfügung stehen. Beim **Konvertieren der Daten** werden diese als int16 der numpy Bibliothek([2] abgespeichert. Dies ist erforderlich, da es der größte Speichertyp ist, bei dem der gesamte Datensatz auf den RAM-Speicher passt und der kleinste Speichertyp bei dem keine Informationen verloren gehen.

- Manipulation der Daten mit dem Keras Daten Generator
- Erstellen des Convolutional Neuronalem Netz
- Splitten des Datensatzes in Training und Test Datensatz
- Trainieren des Neuronalen Netzes mit dem Training-Datensatz
- Evaluation des Neuronalen Netzes

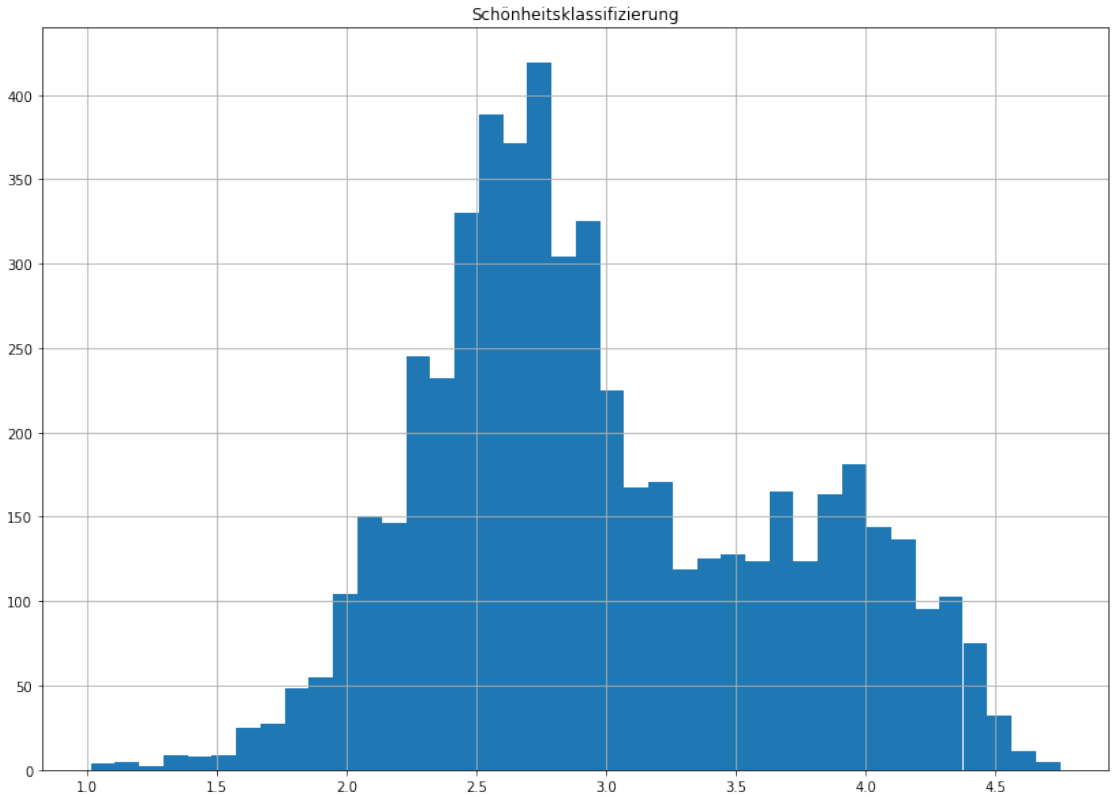



**Figure 2.** Das Histogramm zeigt die Verteilung der Bewertungen für alle Gesichter im Datensatz.

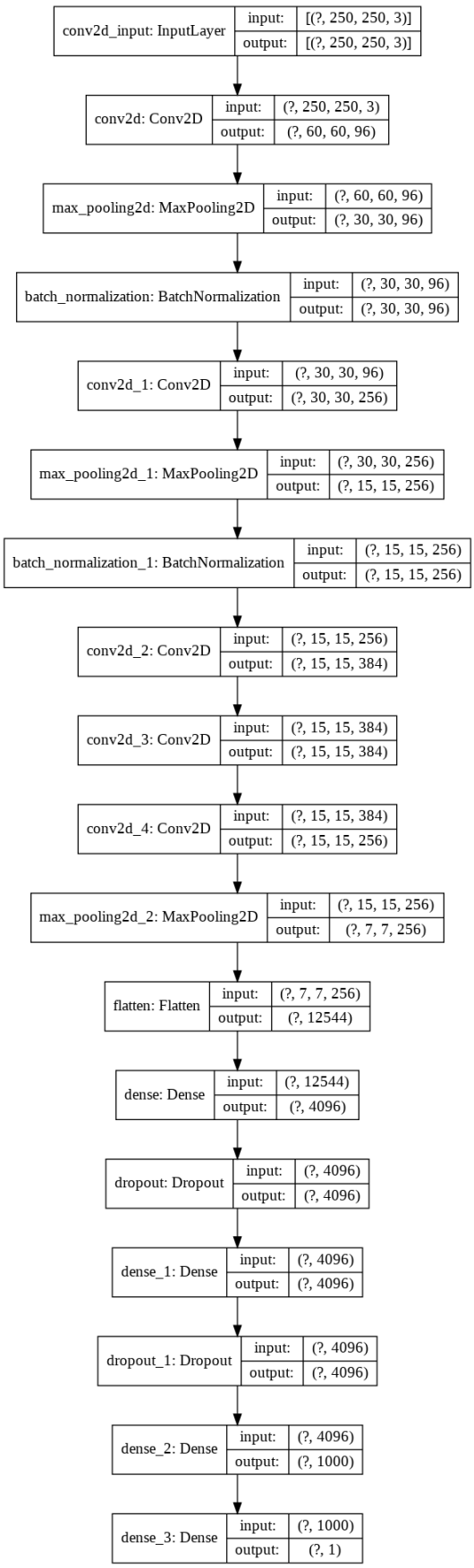



**Figure 3.** Der Graph zeigt den Aufbau des neuronalen Netzes.

### Daten

Bei der **Manipulation der Daten** werden die Farbwerte der einzelnen Pixel von einem Wertebereich o bis 255 auf den Wertebereich 0 bis 1 normalisiert. Dies wird gemacht um dem mathematischem Model des neuronalem Netzes zu entsprechen. Zusätzlich besteht die Möglichkeit die Gesichter zufällig leicht zu rotieren, um leicht rotierte Gesichter klassifizieren zu können. Ebenso besteht die Möglichkeit Bilder vertikal zu spiegeln um den Datensatz insgesamt zu vergrößern. Neben diesen Möglichkeiten gibt es noch andere Möglichkeiten, die jedoch den Rahmen dieser Arbeit übersteigen.

### Aufbau des neuronalen Netzes

Das **Neuronale Netz** besitzt insgesamt 17 Schichten. Hiervon entsprechen fünf Schichten Convolutional-Schichten. Diese befinden sich am Anfang des Netzes um Muster. Die Architektur der Convolutional-Schichten stammt von Alexnet einer effektiven Architektur zur Gesichtserkennung [3]. Zwischen den Convolutional-Schichten befinden sich Pooling-Schichten welche mit Max-pooling arbeiten. Diese verkleinern die Matrix indem sie den größten Werte aus einer Submatrix an lokalen Stellen des Outputs der vorherigen Schicht nehmen. Hierdurch kann die Komplexität des gesamten Netzes verkleinert werden ohne, dass Informationen verloren gehen. In dem Fall der Schönheitserkennung werden die nachfolgenden Schichten verändert im Gegensatz zum klassischen Alexnet [3]. Den Convolutional-Schichten folgen Dense-Schichten. Diese entsprechen den Standart Schichten eines neuronalen Netzes uns sorgen dafür, dass normalerweise ein Gesicht einer Person zugeordnet werden kann anhand der durch die ersten Ebenen erkannten Gesichtsmerkmale wie Kanten, Positionen, Größe und Farbe von Gesichtselementen wie Mund, Nase, Augen, usw. Für die Schönheistsklassifizierung wurde im Anschluss an die letzte Schicht noch eine Dense-Schicht mit einem einzigen Neuron als Ausgangsneuron ergänzt. Der Output des Ausgangsneuron ist letztendlich der Wert der Klassifizierung, welcher ebenso auf den Geschichtsmerkmalen beruht. Zwischen den verschiedenen Dense-Schichten befinden sich noch Dropout-Schichten, welche Overfitting verhindern, indem sie zufällig eine kleine Anzahl an Neuronen deaktivieren.

### Training und Evaluation des neuronalen Netzes

Für das Training und die spätere Evaluation des neuronalen Netzes wird der Datensatz in Training, Validation und Test Datensatz gesplittet. Dabei enthalten Test und Validation jeweils 15% des gesamten Datensatzes. Für das Training selbst werden der Training und Validation Datensatz verwendet. Es wurde mit einer Standard Batchsize von 16 trainiert und für 40 Epochen trainiert. Wie im Graphen 4 erkennbar sinkt die Loss Funktion etwa konstant wohingegen die Loss Funktion für die Validation ab etwa 30 Epochen konstant bleibt. Die absolute gemittelte Abweichung des neuronales Netzes für unbekannte Gesichter beträgt 0.4.

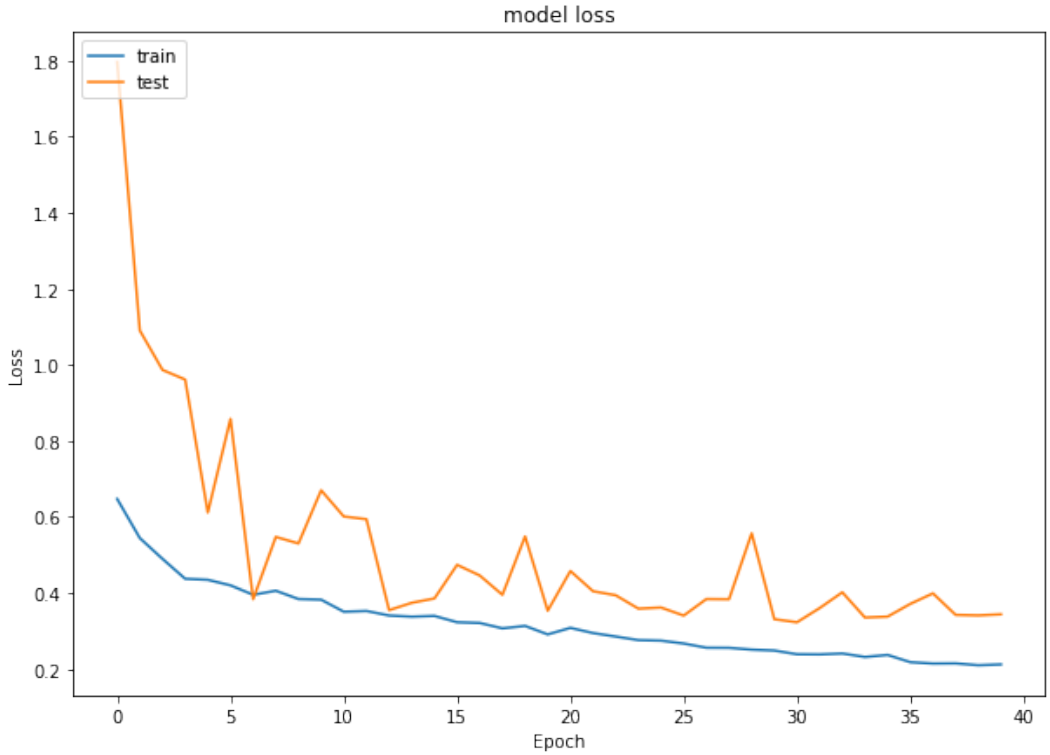



**Figure 4.** Der Graph zeigt den Verlauf der Loss Funktion für den Trainings und Validation Datensatz.

## ERGEBNISSE

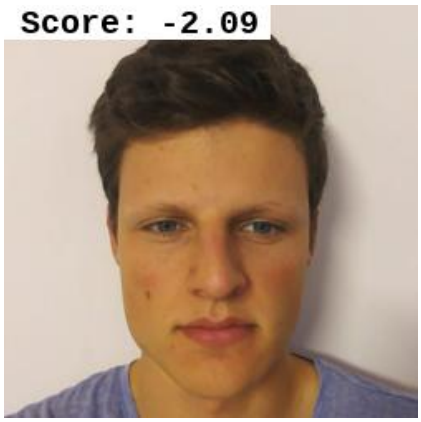
Die Funktionalität des neuronalen Netzes wurde für den vorbereiteten Datensatz gezeigt. Nun sollen komplett unbekannte Bilder klassifiziert werden, welche zufällig aus dem Internet ausgewählt worden. Um die Klassifizierung erkenntlicher zu machen wurden die Ergebnisse standardisiert. Das bedeutet, dass ein durchschnittliches Gesicht einen Score von 0 bekommt. Je negativer der Score ist desto schlechter wurde das Gesicht bewertet und je positiver der Wert ist, desto besser wurde das Gesicht bewertet.

Genauso interessant war es natürlich mein eigenes Gesicht zu klassifizieren. Interessanterweise wurde mein Gesicht besser klassifiziert als ich meine Augen weit geöffnet habe und meine Lippen größer gemacht habe siehe Grafik 6b. Diese Manipulation des Score ist logisch. Selbst wir Menschen nehmen Bilder von anderen Personen sehr unterschiedlich war je nachdem wie die Person auf dem Bild schaut, ob sie lächelt, aus welcher Perspektive das Bild aufgenommen wurde oder viele weitere Aspekte. (Alle Rechte der Bilder in diesem Abschnitt gehören zu deren ursprünglichen Erstellern.)

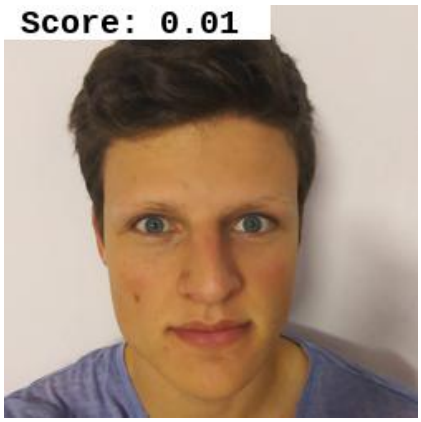

**(a)** Ein Gesicht mit einer schlechten Bewertung von -5.22.


**(b)** Ein Gesicht mit einer guten Bewertung von 5.07.

**Figure 5.** Diese zwei Gesichter zeigen, dass das neuronale Netz in der Lage ist zufällig ausgewählte Gesichter richtig zu klassifizieren. Das Gesicht eines bekannten gut aussehenden Schauspielers wurde besser bewertet als das Gesicht eines Mannes mit Doppelkinn.




**(a)** Mein normales Gesicht wird unterdurchschnittlich bewertet.




**(b)** Mein Gesicht mit großen Augen wird durchschnittlich bewertet.

**Figure 6.** Es zeigt sich, dass sich das Model leicht beeinflussen lässt, allein dadurch wie man sich selbst darstellt.


**Figure 7.** Das Gesicht von Kim Jong Un wurde am schlechtesten bewertet.


**Figure 8.** Steve Buscemi eine Person, die gefunden wird wenn man auf Google nach unschönen Schauspielern sucht, wird schlecht bewertet.

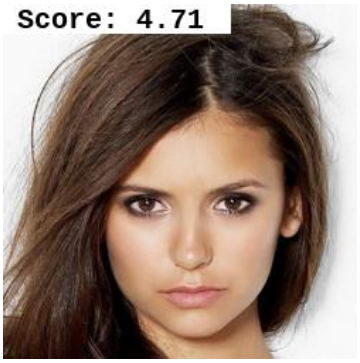



**Figure 9.** Nina Dobrov, eine bulgarische Schauspielerin wird gut bewertet

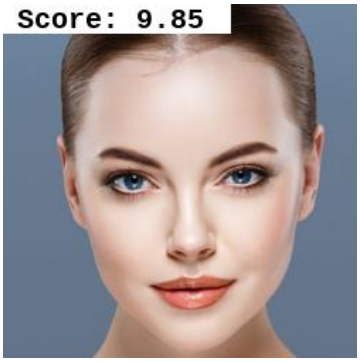



**Figure 10.** Am Besten wird ein Model mit Make-up und perfekten Lichteinstellungen bewertet.

## FAZIT

Es hat sich gezeigt, dass die Klassifikation von Schönheit mithilfe einer künstlichen Intelligenz funktioniert. Die Abweichung der Klassifizierung beträgt im Schnitt etwa 0.4. Im Gegensatz zu der Standardabweichung der menschlichen Klassifizierung verhält sich die künstliche Intelligenz sogar besser und ist damit in der Lage Gesichter objektiv besser zu bewerten als ein einzelner Mensch. Die Standardabweichung der menschlichen Bewertung beträgt etwa 0.7, siehe Paper [1], Seite 3, Figure 3. Der Grund für die bessere objektive Bewertung ist, das die KI verallgemeinern kann, währenddessen ein Mensch immer Vorkenntnisse, Erfahrungen und auch persönliche Vorlieben hat. Dass eine künstliche Intelligenz den Menschen in einer weiteren Disziplin das Wasser reichen kann oder diese Disziplin sogar besser erledigen kann ist ein weiterer Schritt in eine Richtung in der künstliche Intelligenzen den Menschen ersetzen können. Vor allem in einem Feld, der Schönheit, dass von dem Menschen geglaubt zu dominiert zu sein. Ob man dies erstaunlich findet oder beängstigend entscheidet jeder für sich selbst. In der Zukunft werden mit KI's noch weitaus mehr Disziplinen beherrscht werden.

## REFERENCES


[1] Lingyu Liang et al. “SCUT-FBP5500: a diverse benchmark dataset for multi-paradigm facial beauty prediction”. In: *2018 24th International Conference on Pattern Recognition (ICPR)*. IEEE. 2018, pp. 1598–1603.

[2] none. *Data types*. `https://numpy.org/doc/stable/user/basics.types.html`. 2020.

[3] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman. “Deep face recognition”. In: (2015).


## CODE

In diesem Abschnitt befindet sich der verwendete Code

# Beauty Recognition

**This notebook creates a beauty classifier on the database - SCUT-FBP5500-Database**

In [ ]:

```
from google.colab import drive

drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

In [ ]:

```
import pandas as pd
import pickle
import os
import datetime
import cv2
import matplotlib.pyplot as plt
import numpy as np

#ai stuff
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.models import model_from_json
from keras.optimizers import Adam
from keras.layers import (Dense, Dropout, Flatten, Activation, Conv2D, MaxPooling2D, Bat
chNormalization)
from keras.layers.core import Activation
from keras import backend as K
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
```

In [ ]:

```
# Data processing

def dataset(fromid,toid):
    images = []
    labels = []
    count=1
    df = pd.read_csv(LABELS, sep=" ")
    for index, row in df.iterrows():

      if((index>=fromid) and (index<toid)):
        #print("is index: ",index, "between: ", fromid, " and ",toid)
        if(index%10 == 1):
          print("[process] ", index)
        filepath = IMAGES_DIR + row['filename'].replace(" ","")
        label = row['score']
        image = cv2.imread(filepath)
        image = cv2.resize(image, (250, 250))
        images_numpy = np.asarray(image, np.int16)
        labels_numpy = np.asarray(label, np.float16)
        images.append(images_numpy)
        #images.append(filepath)
        labels.append(labels_numpy)
        #if(count>20):
        # break
        count +=1
    return images, labels
```

In [ ]:

```
def pickle_save(object, path):
```

```
    with open(path, "wb") as f:
        return pickle.dump(object, f)


def pickle_load(path):
    with open(path, "rb") as f:
        return pickle.load(f)

def path(filename):
    return os.path.join(DIR, filename)

def show_image(images, number_of_image):
  #image = cv2.cvtColor(images[number_of_image], cv2.COLOR_BGR2RGB) for use with cv2
  image_bgr = images[number_of_image]
  image_rgb = image_bgr[...,::-1].copy()
  imgplot = plt.imshow(image_rgb)
  plt.show()
```

In [ ]:

```
# Define paths
DIR = "/content/gdrive/My Drive/Beauty_recognition/"
IMAGES_DIR = DIR + "Images/"
LABELS = DIR + "All_labels.txt"
pickeled_label_path = "labels_pickeled.pkl"
pickeled_image_path = "images_pickeled.pkl"
#only needed in case i run out of memory
pickeled_label_path_set2 = "labels_pickeled_set2.pkl"
pickeled_image_path_set2 = "images_pickeled_set2.pkl"
pickeled_label_path_set3 = "labels_pickeled_set3.pkl"
pickeled_image_path_set3 = "images_pickeled_set3.pkl"
pickeled_label_path_set4 = "labels_pickeled_set4.pkl"
pickeled_image_path_set4 = "images_pickeled_set4.pkl"

MY_IMAGES_DIR = DIR + "New_faces"
pickeled_my_images = "images_pickeled_mine.pkl"

model_path = path('model.json')
model_weight_path = path('model_weights.h5')
history_path = path('history_training.pkl')
BATCH_SIZE = 16
EPOCH = 2
TEST_SIZE = 0.15
VAL_SIZE = 0.15
FIG_SIZE = (10,7)
```

# Prepare the dataset

In [ ]:

```
#load images
#only needed once

images,labels = dataset(0,6000)
print(len(images))

#saving the data so they can be reloaded easier
pickle_save(images, path(pickeled_image_path))
pickle_save(labels, path(pickeled_label_path))
print("Parsing images done set 1")

del images
del labels

print("finished with parsing")
```

In [ ]:

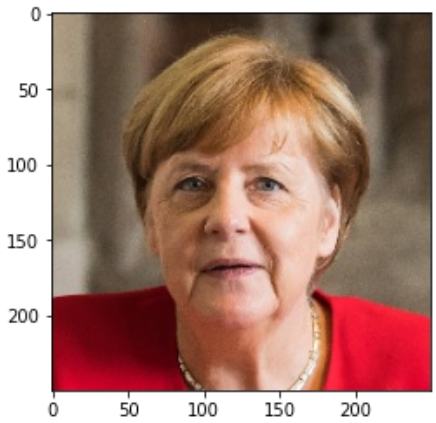
```
#Prepare the dataset for my own images
```

```
import glob

my_images_list = []
for filename in glob.glob(MY_IMAGES_DIR+'/*.jpg'): #assuming jpg's
  print("processing: "+ filename)
  image = cv2.imread(filename)
  image = cv2.resize(image, (250, 250))
  images_numpy = np.asarray(image, np.int16)
  my_images_list.append(images_numpy)

my_images_list = np.array(my_images_list)
```

```
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/Rumen_Radev.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/merkel.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni1.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni2.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni3.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni4.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni5.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni6.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/miko1.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/miko2.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/ninadobrov.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/Kim_Jong.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/messi.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/messi_young.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/muller_happy.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/Zac-Efron.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/ugly2.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/ugly1.jpg
processing: /content/gdrive/My Drive/Beauty_recognition/New_faces/beauty1.jpg
```

In [ ]:

```
show_image(my_images_list,1)
```




In [ ]:

```
#delete the data to provide crashing because of ram overflow
del images
del labels
```

# Load data from GoogleDrive

In [ ]:

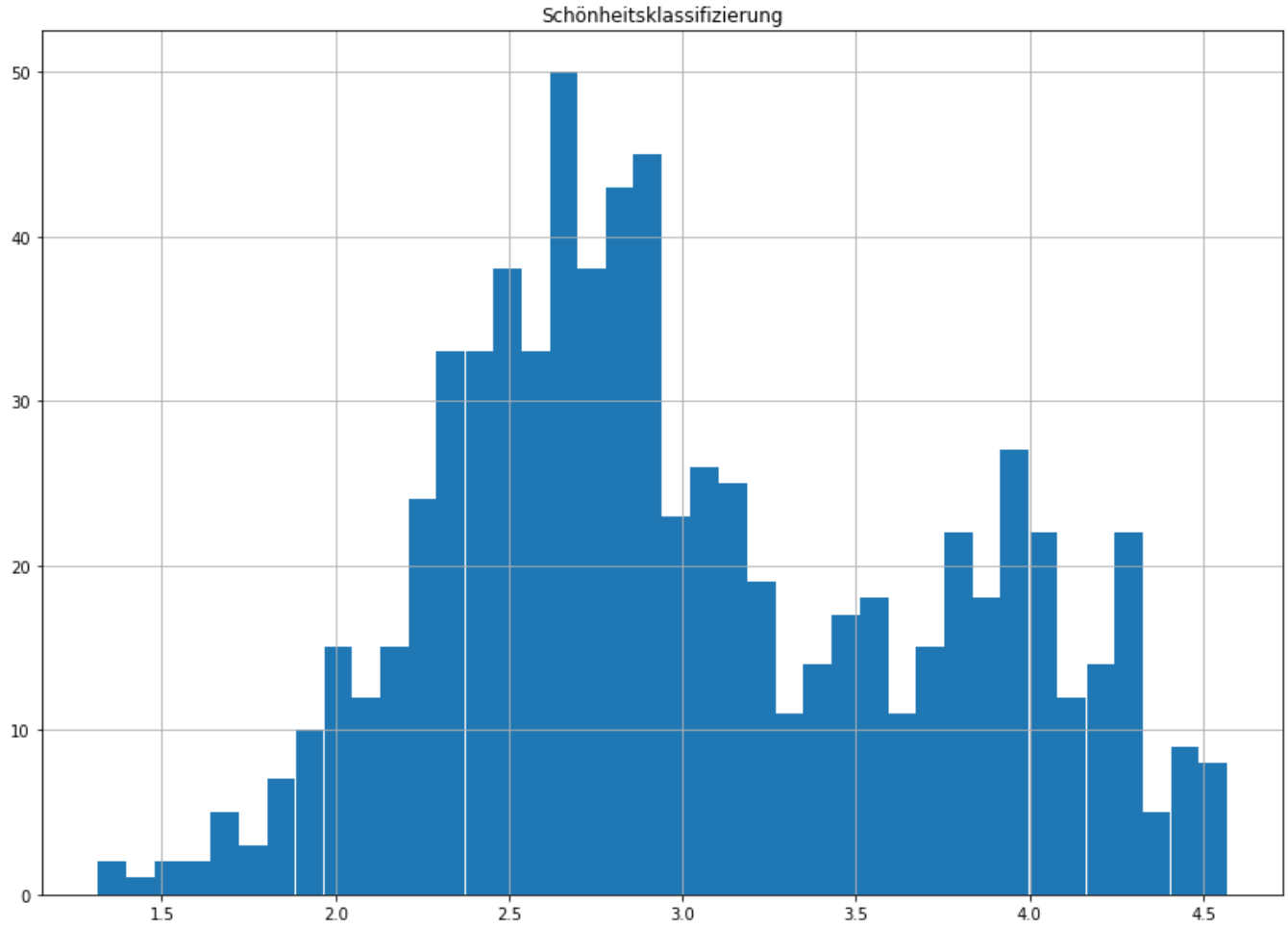
```
labels_numpy = pickle_load(os.path.join(DIR, pickeled_label_path))
```

In [ ]:

```
printdf = pd.DataFrame(labels_numpy, columns = ["Schönheitsklassifizierung"])
printdf[4750:5499].hist(bins = 40, figsize = (14,10))
```

Out[ ]:

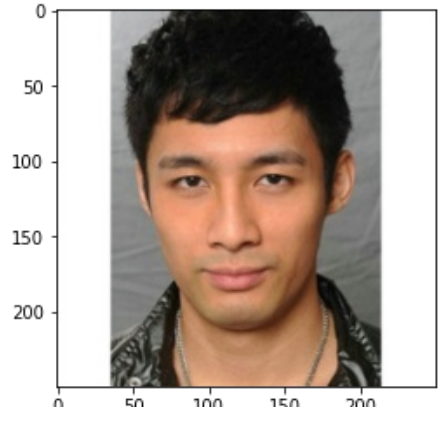
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f1c94a33668>]],
      dtype=object)
```




In [ ]:

```
# Load numpy images from drive
labels_numpy = pickle_load(os.path.join(DIR, pickeled_label_path))
images_numpy = pickle_load(os.path.join(DIR, pickeled_image_path))

print("Ectracted images: ",len(images_numpy))

#Convert to numpy arrays
images_numpy = np.asarray(images_numpy, np.int16)
labels_numpy = np.asarray(labels_numpy, dtype=np.float16)

show_image(images_numpy,50)
print("Data is a type of: ", type(images_numpy), "and ", type(labels_numpy))


print("Loading images done")
```

```
Ectracted images:  5500
```



```
Data is a type of:  <class 'numpy.ndarray'> and  <class 'numpy.ndarray'>
Loading images done
```

In [ ]:

```
del labels_numpy
del images_numpy
```

# Manipulate data

In [ ]:

```
# Create data generator
train_datagen = ImageDataGenerator(rescale=1./255,            #rescale, value between 0-255 is too big, instead use a value 0-1
                                   vertical_flip=False,       #flips the image horizontally or vertically
                                   rotation_range=0,          #rotation range
                                   )
test_datagen = ImageDataGenerator(rescale=1./255,
                                   vertical_flip=False,
                                   rotation_range=0
                                   )
#zoom_range=[0.0,1.0],width_shift_range=0.1,height_shift_range=0.1,brightness_range=[0.9,1.0],
```

In [ ]:

```
def init_model():
    # initialize alexNet model
    model = Sequential()
    # Convolutional layers
    model.add(Conv2D(kernel_size=(11, 11),
                     activation='relu',
                     strides=(4, 4),
                     filters=96,
                     padding='valid',
                     input_shape=(250,250,3)))
    model.add(MaxPooling2D(pool_size=(2, 2),
                           strides=(2, 2),
                           padding='valid'))
    model.add(BatchNormalization())
    model.add(Conv2D(filters=256,
                     kernel_size=(5, 5),
                     strides=(1, 1),
                     padding='same',
                     activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2),
                           strides=(2, 2),
                           padding='valid'))
    model.add(BatchNormalization())
    model.add(Conv2D(filters=384,
                     kernel_size=(3, 3),
                     strides=(1, 1),
                     padding='same',
                     activation='relu'))
    model.add(Conv2D(filters=384,
                     kernel_size=(3, 3),
                     strides=(1, 1),
                     padding='same',
                     activation='relu'))
    model.add(Conv2D(filters=256,
                     kernel_size=(3, 3),
                     strides=(1, 1),
                     padding='same',
                     activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2),
                           strides=(2, 2),
```

```
                          padding='valid'))
    model.add(Flatten())
    # fully connected
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(4096, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1000, activation='relu'))
    # model.add(Dropout(0.5))
    # output
    model.add(Dense(1, activation='linear'))
    model.compile(loss='mean_absolute_error',
                  optimizer=Adam(lr=1e-4),
                  metrics=['mse'])

    return model

#save model weights and history to Gdrive
def save_model(model, history):
    print('Saving the model ...')
    # save model structure
    with open(model_path, "w") as f:
        f.write(model.to_json())
    # save the weights and history
    model.save_weights(model_weight_path)
    pickle_save(history, history_path)
    print('Saving done')

#load model from Gdrive
def load_model():
    json_file = open(model_path, 'r')
    model = json_file.read()
    json_file.close()
    model = model_from_json(model)
    # load weights into new model
    model.load_weights(model_weight_path)
    history = pickle_load(history_path)
    return model,history

def plot_history(model_history_info, save=False):
    plt.figure(figsize=FIG_SIZE)
    plt.plot(model_history_info['loss'])
    plt.plot(model_history_info['val_loss'])
    plt.title('model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epochen')
    plt.legend(['train', 'validation'], loc='upper left')
    #saves png to file
    if(save):
        plt.savefig(DIR + 'history_of_loss.png')
    plt.show()
```

In [ ]:

```
model = init_model()
print(model.summary())
```

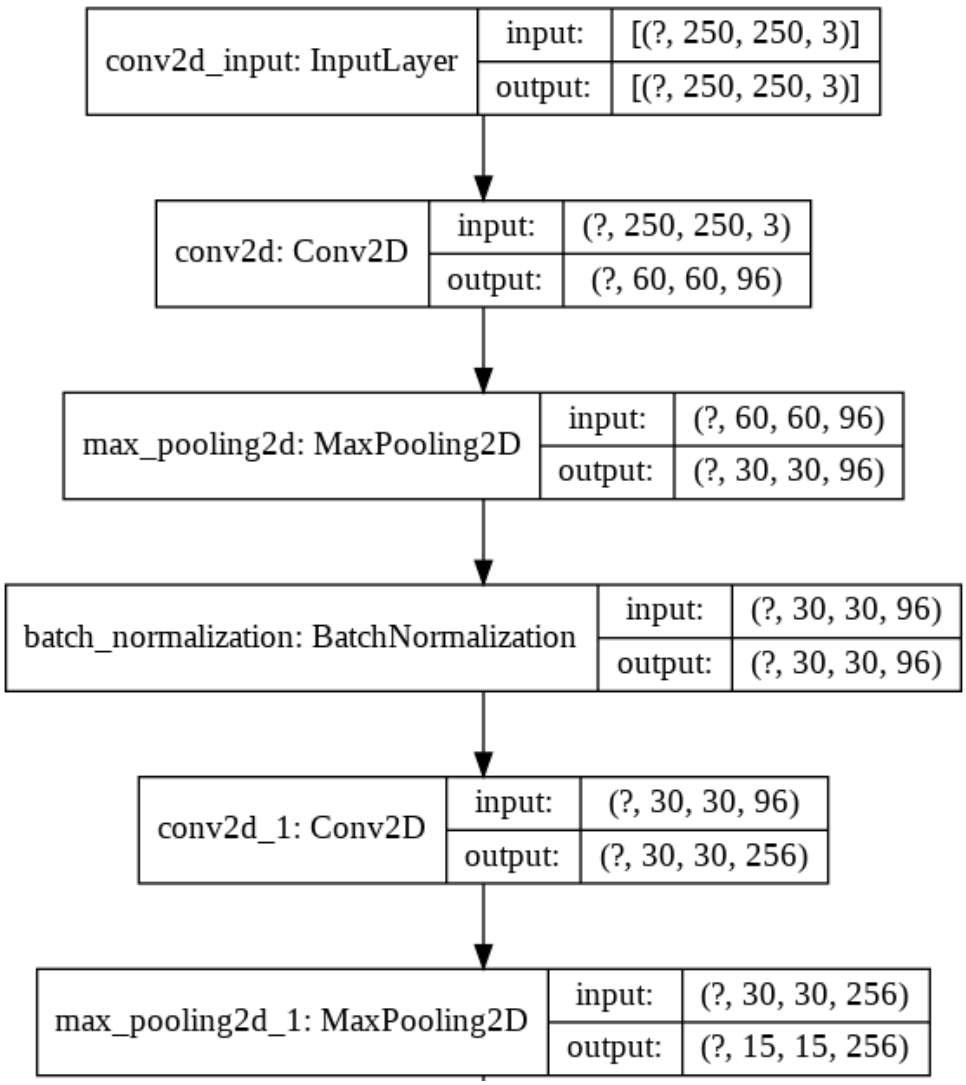
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 60, 60, 96)        34944
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 30, 30, 96)        0
_________________________________________________________________
batch_normalization (BatchNo (None, 30, 30, 96)        384
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 30, 30, 256)       614656
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 256)       0
_________________________________________________________________
batch_normalization_1 (Batch (None, 15, 15, 256)       1024
```

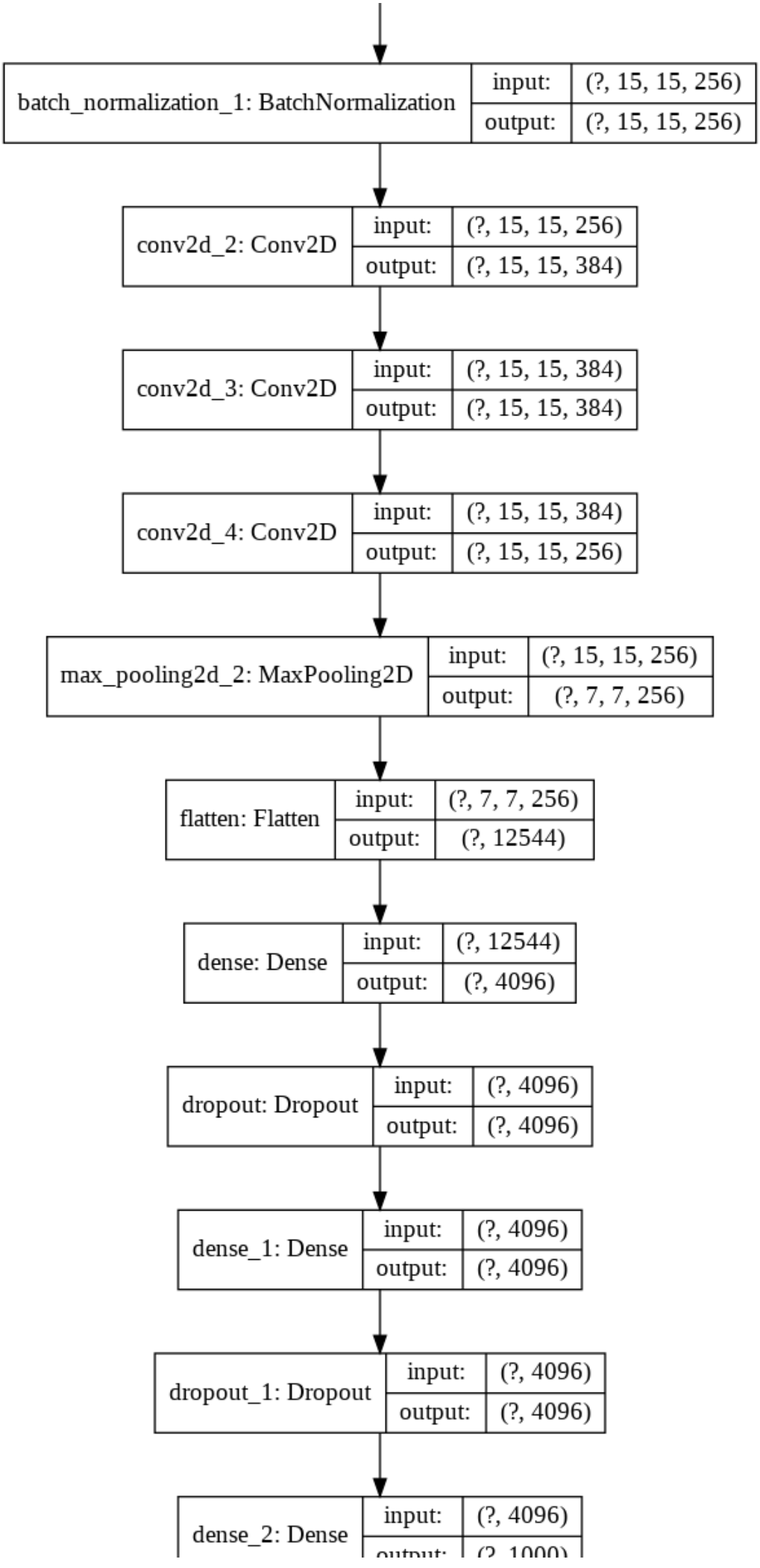
```
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 15, 15, 384)       885120
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 384)       1327488
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 15, 15, 256)       884992
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 7, 7, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 12544)             0
_________________________________________________________________
dense (Dense)                (None, 4096)              51384320
_________________________________________________________________
dropout (Dropout)            (None, 4096)              0
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dropout_1 (Dropout)          (None, 4096)              0
_________________________________________________________________
dense_2 (Dense)              (None, 1000)              4097000
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 1001
=================================================================
Total params: 76,012,241
Trainable params: 76,011,537
Non-trainable params: 704
_________________________________________________________________
None
```

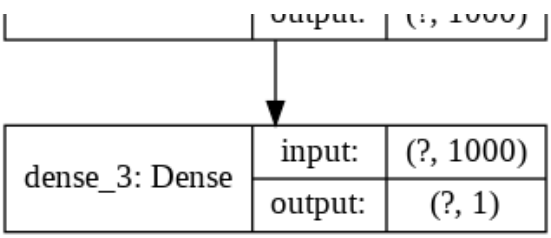
In [ ]:

```
keras.utils.plot_model(model, show_shapes=True)
```

Out[ ]:




batch_normalization_1: BatchNormalization
input: (?, 15, 15, 256)
output: (?, 15, 15, 256)
conv2d_2: Conv2D
input: (?, 15, 15, 256)
output: (?, 15, 15, 384)
conv2d_3: Conv2D
input: (?, 15, 15, 384)
output: (?, 15, 15, 384)
conv2d_4: Conv2D
input: (?, 15, 15, 384)
output: (?, 15, 15, 256)
max_pooling2d_2: MaxPooling2D
input: (?, 15, 15, 256)
output: (?, 7, 7, 256)
flatten: Flatten
input: (?, 7, 7, 256)
output: (?, 12544)
dense: Dense
input: (?, 12544)
output: (?, 4096)
dropout: Dropout
input: (?, 4096)
output: (?, 4096)
dense_1: Dense
input: (?, 4096)
output: (?, 4096)
dropout_1: Dropout
input: (?, 4096)
output: (?, 4096)
dense_2: Dense
input: (?, 4096)




In [ ]:

```
# split train/val
(trainX, valX, trainY, valY) = train_test_split(images_numpy,
 labels_numpy, test_size=VAL_SIZE, random_state=42)
# split train/test
(trainX, testX, trainY, testY) = train_test_split(trainX,
 trainY, test_size=TEST_SIZE, random_state=41)

print('Train ', len(trainX), ' samples', 'Test ', len(testX), ' samples')
```

```
Train  3973  samples Test  702  samples
```

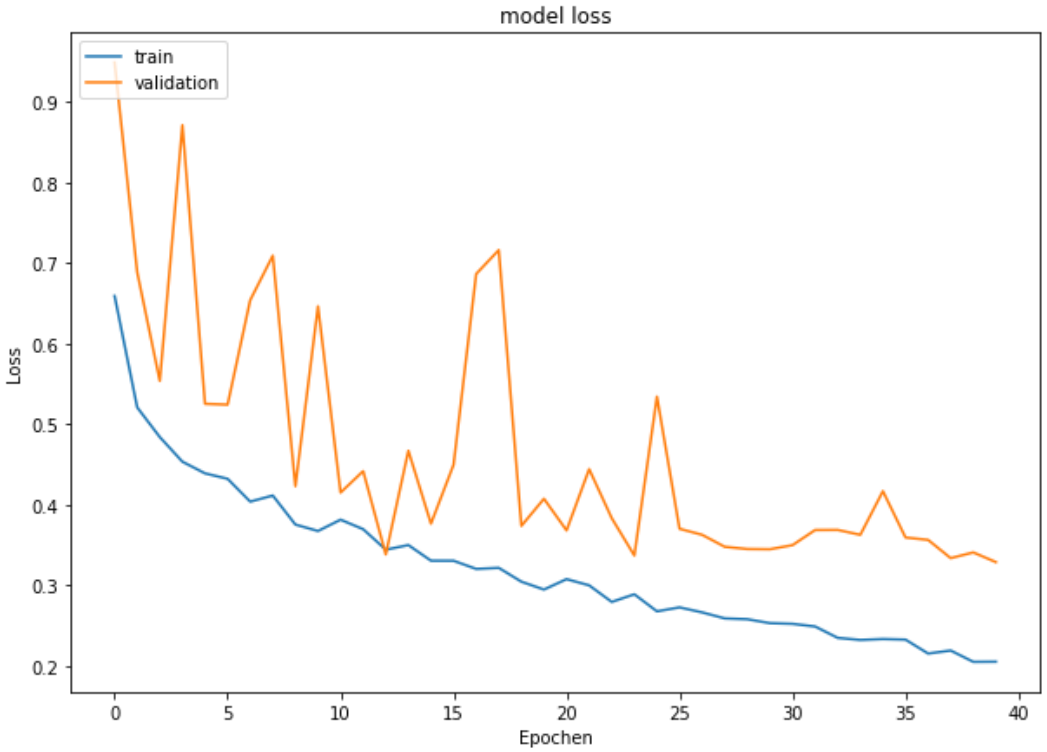
In [ ]:

```
EPOCH = 40
H = model.fit_generator(
            train_datagen.flow(trainX, trainY, batch_size=BATCH_SIZE),
            steps_per_epoch=len(trainX) // BATCH_SIZE,
            epochs=EPOCH,
            validation_data=test_datagen.flow(valX, valY)
            )

model.evaluate(testX/255.0, testY)
plot_history(H.history)
save_model(model, H.history)
```

```
WARNING:tensorflow:From <ipython-input-10-3c47a3e55a57>:6: Model.fit_generator (from tens
orflow.python.keras.engine.training) is deprecated and will be removed in a future versio
n.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/40
248/248 [==============================] - 12s 47ms/step - loss: 0.6592 - mse: 0.7918 - v
al_loss: 0.9493 - val_mse: 1.2875
Epoch 2/40
248/248 [==============================] - 11s 44ms/step - loss: 0.5207 - mse: 0.4402 - v
al_loss: 0.6886 - val_mse: 0.6845
Epoch 3/40
248/248 [==============================] - 11s 44ms/step - loss: 0.4839 - mse: 0.3853 - v
al_loss: 0.5537 - val_mse: 0.4854
Epoch 4/40
248/248 [==============================] - 11s 44ms/step - loss: 0.4533 - mse: 0.3340 - v
al_loss: 0.8715 - val_mse: 0.9932
Epoch 5/40
248/248 [==============================] - 11s 44ms/step - loss: 0.4388 - mse: 0.3109 - v
al_loss: 0.5252 - val_mse: 0.4314
Epoch 6/40
248/248 [==============================] - 11s 44ms/step - loss: 0.4320 - mse: 0.2989 - v
al_loss: 0.5242 - val_mse: 0.4400
Epoch 7/40
248/248 [==============================] - 11s 44ms/step - loss: 0.4036 - mse: 0.2679 - v
al_loss: 0.6541 - val_mse: 0.6023
Epoch 8/40
248/248 [==============================] - 11s 44ms/step - loss: 0.4112 - mse: 0.2758 - v
al_loss: 0.7093 - val_mse: 0.6794
Epoch 9/40
248/248 [==============================] - 11s 44ms/step - loss: 0.3752 - mse: 0.2302 - v
al_loss: 0.4228 - val_mse: 0.2842
Epoch 10/40
248/248 [==============================] - 11s 44ms/step - loss: 0.3672 - mse: 0.2234 - v
al_loss: 0.6464 - val_mse: 0.5758
Epoch 11/40
```

```
248/248 [==============================] - 11s 44ms/step - loss: 0.3812 - mse: 0.2411 - v
al_loss: 0.4150 - val_mse: 0.2824
Epoch 12/40
248/248 [==============================] - 11s 44ms/step - loss: 0.3694 - mse: 0.2261 - v
al_loss: 0.4415 - val_mse: 0.3084
Epoch 13/40
248/248 [==============================] - 11s 44ms/step - loss: 0.3439 - mse: 0.1969 - v
al_loss: 0.3380 - val_mse: 0.2013
Epoch 14/40
248/248 [==============================] - 11s 44ms/step - loss: 0.3497 - mse: 0.1975 - v
al_loss: 0.4669 - val_mse: 0.3332
Epoch 15/40
248/248 [==============================] - 11s 44ms/step - loss: 0.3303 - mse: 0.1776 - v
al_loss: 0.3763 - val_mse: 0.2503
Epoch 16/40
248/248 [==============================] - 11s 44ms/step - loss: 0.3302 - mse: 0.1774 - v
al_loss: 0.4495 - val_mse: 0.3278
Epoch 17/40
248/248 [==============================] - 11s 44ms/step - loss: 0.3201 - mse: 0.1689 - v
al_loss: 0.6866 - val_mse: 0.6730
Epoch 18/40
248/248 [==============================] - 11s 44ms/step - loss: 0.3214 - mse: 0.1686 - v
al_loss: 0.7163 - val_mse: 0.6947
Epoch 19/40
248/248 [==============================] - 11s 44ms/step - loss: 0.3042 - mse: 0.1558 - v
al_loss: 0.3735 - val_mse: 0.2377
Epoch 20/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2944 - mse: 0.1432 - v
al_loss: 0.4072 - val_mse: 0.2766
Epoch 21/40
248/248 [==============================] - 11s 44ms/step - loss: 0.3074 - mse: 0.1561 - v
al_loss: 0.3679 - val_mse: 0.2279
Epoch 22/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2996 - mse: 0.1506 - v
al_loss: 0.4441 - val_mse: 0.3169
Epoch 23/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2790 - mse: 0.1296 - v
al_loss: 0.3831 - val_mse: 0.2397
Epoch 24/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2885 - mse: 0.1369 - v
al_loss: 0.3367 - val_mse: 0.1990
Epoch 25/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2673 - mse: 0.1191 - v
al_loss: 0.5342 - val_mse: 0.4353
Epoch 26/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2722 - mse: 0.1253 - v
al_loss: 0.3699 - val_mse: 0.2377
Epoch 27/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2660 - mse: 0.1183 - v
al_loss: 0.3624 - val_mse: 0.2214
Epoch 28/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2585 - mse: 0.1111 - v
al_loss: 0.3475 - val_mse: 0.2088
Epoch 29/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2575 - mse: 0.1099 - v
al_loss: 0.3448 - val_mse: 0.2084
Epoch 30/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2526 - mse: 0.1063 - v
al_loss: 0.3445 - val_mse: 0.2015
Epoch 31/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2518 - mse: 0.1051 - v
al_loss: 0.3496 - val_mse: 0.2099
Epoch 32/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2485 - mse: 0.1031 - v
al_loss: 0.3684 - val_mse: 0.2435
Epoch 33/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2344 - mse: 0.0928 - v
al_loss: 0.3685 - val_mse: 0.2341
Epoch 34/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2318 - mse: 0.0905 - v
al_loss: 0.3624 - val_mse: 0.2216
Epoch 35/40
```

```
248/248 [==============================] - 11s 44ms/step - loss: 0.2329 - mse: 0.0936 - v
al_loss: 0.4167 - val_mse: 0.2800
Epoch 36/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2322 - mse: 0.0906 - v
al_loss: 0.3591 - val_mse: 0.2160
Epoch 37/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2149 - mse: 0.0774 - v
al_loss: 0.3561 - val_mse: 0.2199
Epoch 38/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2186 - mse: 0.0797 - v
al_loss: 0.3334 - val_mse: 0.2006
Epoch 39/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2046 - mse: 0.0701 - v
al_loss: 0.3405 - val_mse: 0.2019
Epoch 40/40
248/248 [==============================] - 11s 44ms/step - loss: 0.2047 - mse: 0.0705 - v
al_loss: 0.3285 - val_mse: 0.1883
22/22 [==============================] - 1s 33ms/step - loss: 0.3438 - mse: 0.1897
```




```
Saving the model ...
Saving done
```

In [ ]:

```
model,history = load_model()
model.compile(loss='mean_absolute_error',
                optimizer=Adam(lr=1e-4),
                metrics=['mse','accuracy'])


#r = model.evaluate(testX / 255.0, testY)
print(model.metrics_names)
#print(r)
```

```
[]
```

# Get info from model

In [ ]:

```
#Get info from model
#average score
#predictforscores
```

In [ ]:

```
#This function adds the score to the image by painting a white rectangle into the corner
of the image and writing the score into the box

from PIL import ImageDraw
from PIL import Image
from PIL import ImageFont
def print_score_on_image(image_be_printed,score_to_print):
  image = Image.open(image_be_printed)
  image = image.resize((250, 250))
  draw = ImageDraw.Draw(image)
  font = ImageFont.truetype("/usr/share/fonts/truetype/liberation/LiberationMono-Bold.ttf
", 20)
  #the score is rounded to fill in the box
  final_scoretoprint = "Score: "+str(round(score_to_print[0],2))
  print(final_scoretoprint)
  draw.rectangle(((0, 00), (160, 20)), fill="white")
  draw.text((10, 0),final_scoretoprint,(0,0,0),font=font  ) # this will draw the text

  print("saving...")
  image.save("/content/gdrive/MyDrive/Beauty_recognition/rated_new_faces/"+str(score_to_p
rint)+'.jpg')
```

In [ ]:

```
print_score_on_image(MY_IMAGES_DIR+"/beauty1.jpg",predicts[0])
```

```
Score: 3.49
saving...
```

In [ ]:

```
#check model
predicts = model.predict(my_images_list/255.0)
```

In [ ]:

```
#saving images with their corresponding score printed on them

images_dirs = glob.glob(MY_IMAGES_DIR+'/*.jpg')
for r,dir_image in enumerate(images_dirs):
  print("going: "+dir_image)
  predicts = predicts - predicts.mean()
  print_score_on_image(dir_image,predicts[r]*10)
```

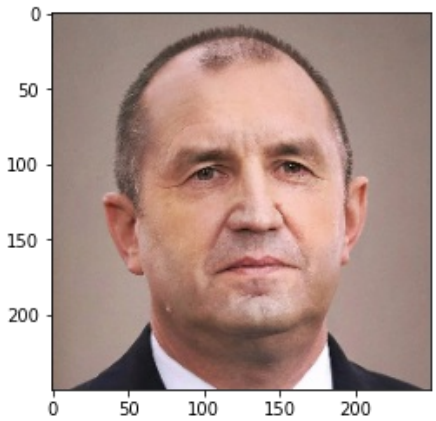
```
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/Rumen_Radev.jpg
Score: 4.61
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/merkel.jpg
Score: -0.76
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni1.jpg
Score: -3.71
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni2.jpg
Score: -5.35
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni3.jpg
Score: -2.09
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni4.jpg
Score: -1.16
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni5.jpg
Score: 0.01
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/joni6.jpg
Score: -2.02
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/miko1.jpg
```

```
Score: 5.45
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/miko2.jpg
Score: 7.55
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/ninadobrov.jpg
Score: 4.71
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/Kim_Jong.jpg
Score: -5.38
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/messi.jpg
Score: -2.08
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/messi_young.jpg
Score: -2.66
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/muller_happy.jpg
Score: -2.89
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/Zac-Efron.jpg
Score: 5.07
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/ugly2.jpg
Score: -5.22
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/ugly1.jpg
Score: -3.95
saving...
going: /content/gdrive/My Drive/Beauty_recognition/New_faces/beauty1.jpg
Score: 9.85
saving...
```

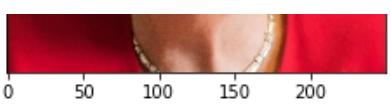
In [ ]:

```
#showing images with their corresponding score
for index,score in enumerate(predicts):
  if score<5 or score>3.5:
    print(index," score: ",(score-2)*2)
    show_image(my_images_list,index)
```

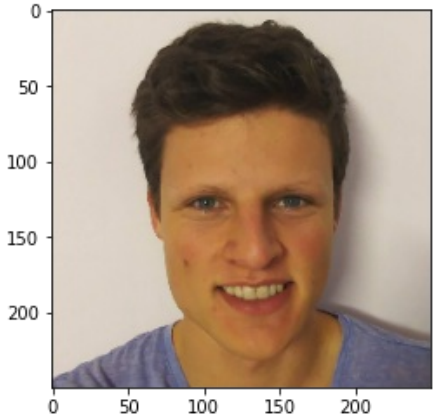
```
0  score:  [2.9815593]
```



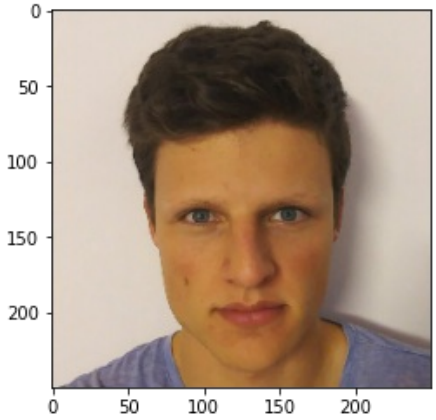

```
1  score:  [1.9067311]
```



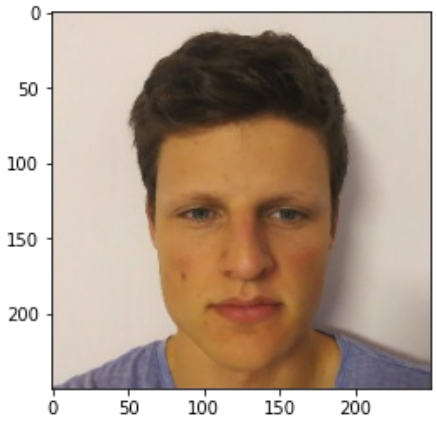

2 score: [1.3172631]



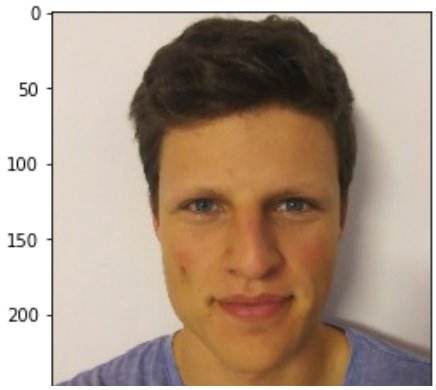

3 score: [0.9888959]

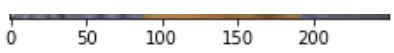



4 score: [1.6407247]

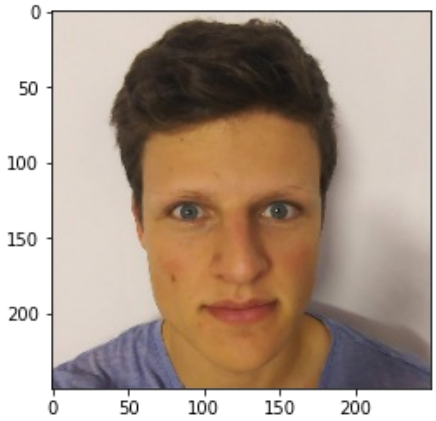



5 score: [1.8264546]

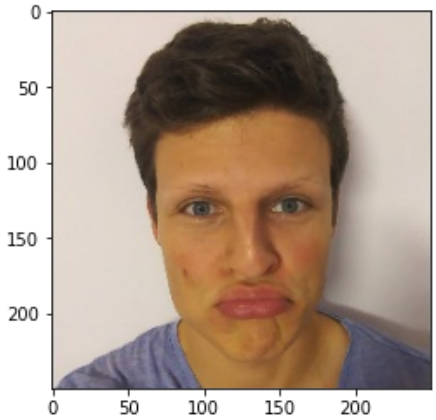




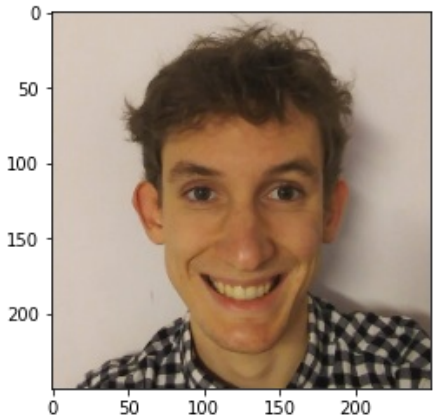

6 score: [2.0617018]



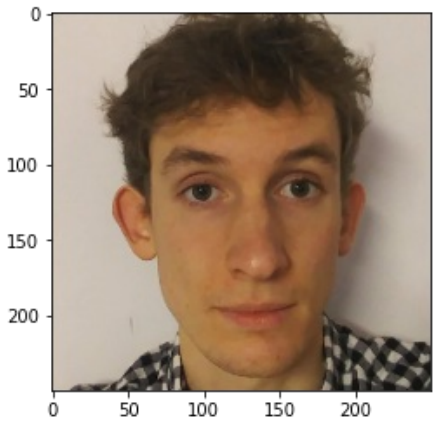

7 score: [1.6555777]



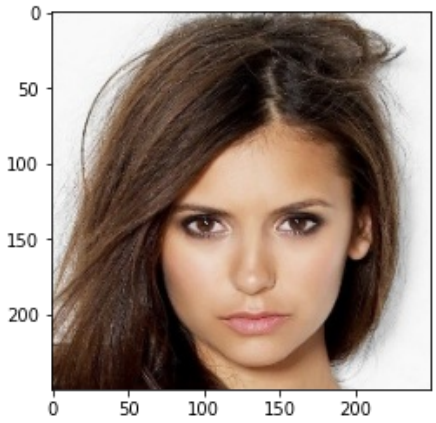

8 score: [3.149078]



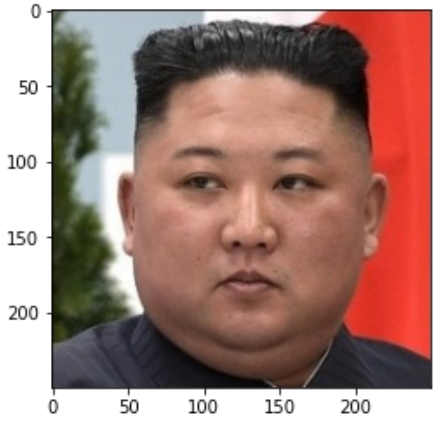

9 score: [3.568695]



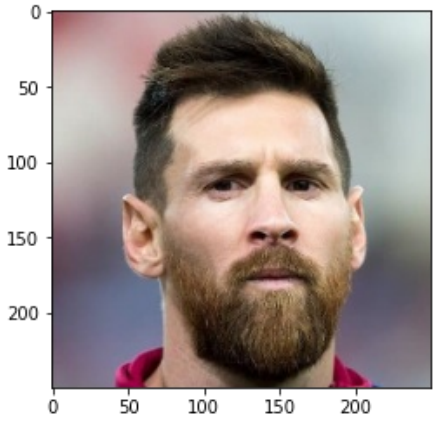
10 score: [3.0011015]



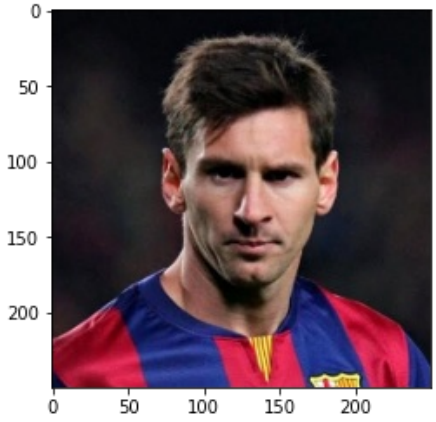

11 score: [0.9838314]



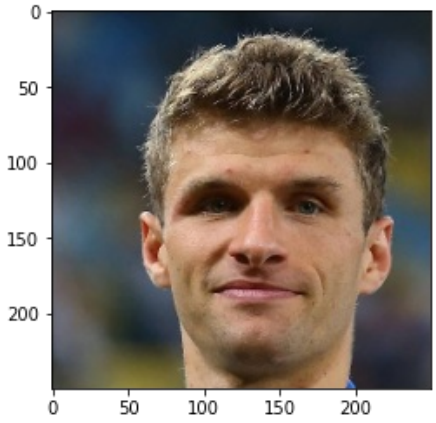

12 score: [1.643086]



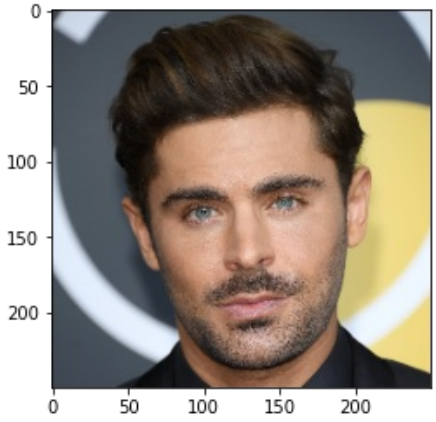

13 score: [1.5264721]



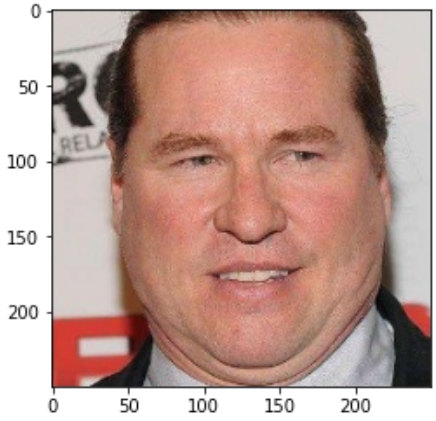

14 score: [1.481822]



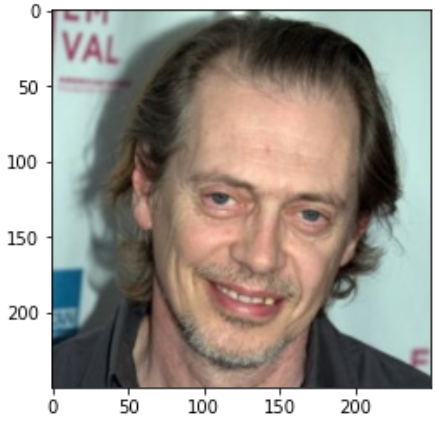
15 score: [3.0735645]



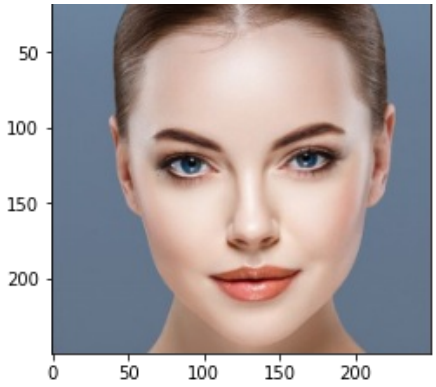
16 score: [1.0156522]



17 score: [1.2687798]



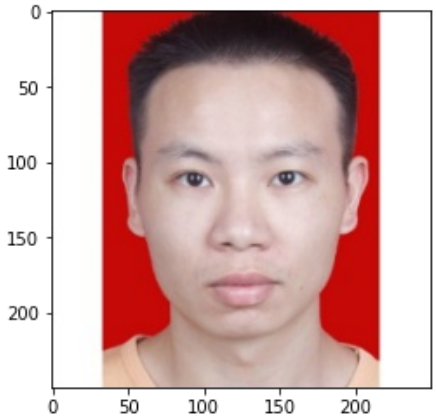
18 score: [4.028925]



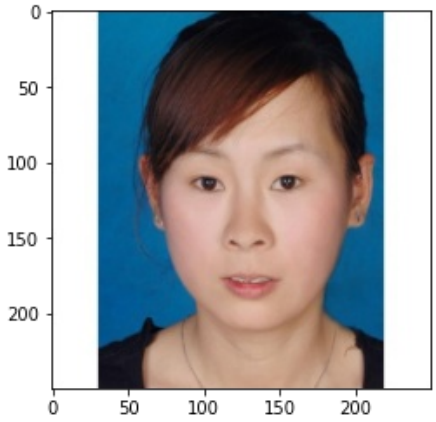

In [ ]:

```
predicts = model.predict(someTestX/255.0)
offset = 0
for number in range(len(predicts)):
  print("Predicted beauty score: ", predicts[number+offset])
  show_image(someTestX,number+offset)
```

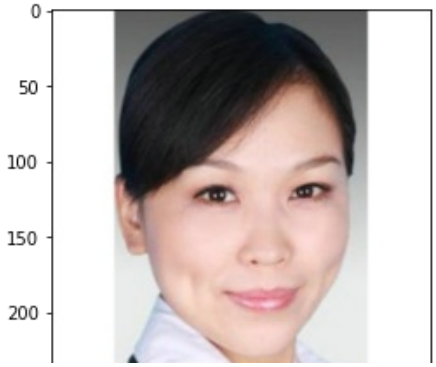
```
Predicted beauty score:  [2.6792898]
```

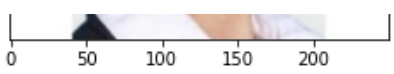



```
Predicted beauty score:  [2.7405338]
```

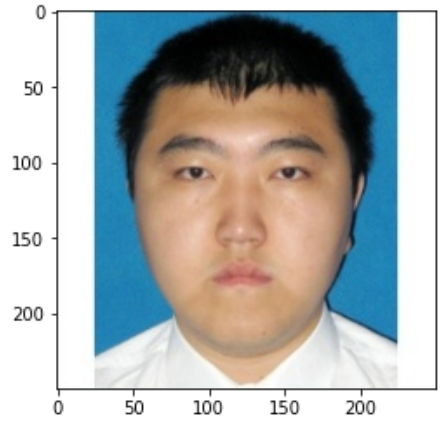



```
Predicted beauty score:  [3.5763016]
```

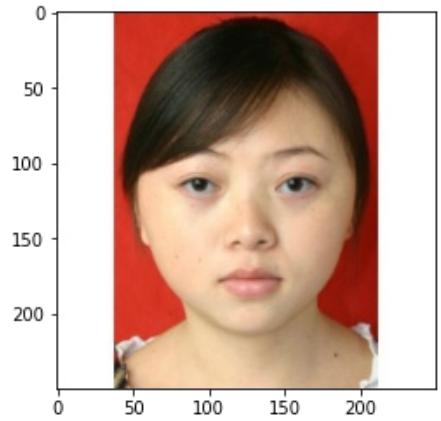




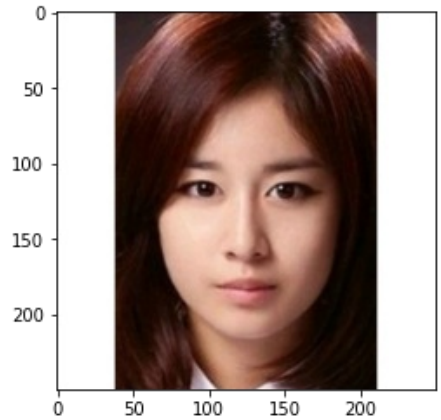

Predicted beauty score:  [2.3512235]



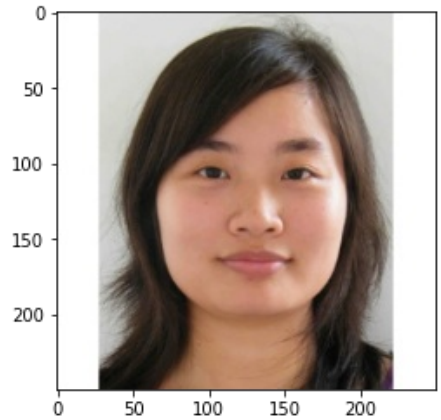

Predicted beauty score:  [2.931859]



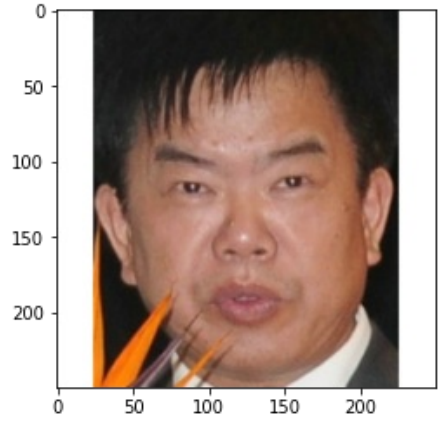

Predicted beauty score:  [3.873482]



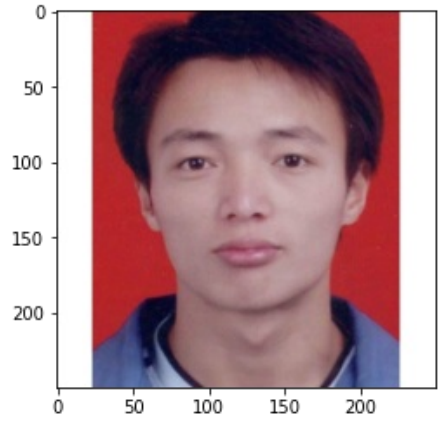

Predicted beauty score:  [2.9481704]



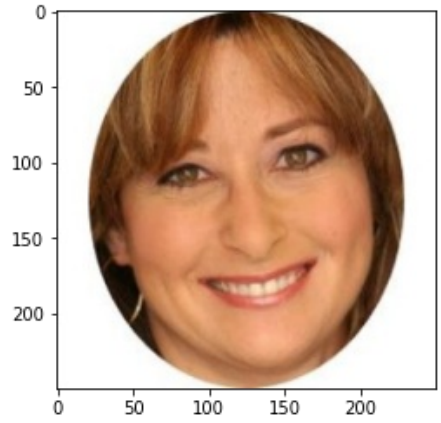
Predicted beauty score:  [2.6429071]



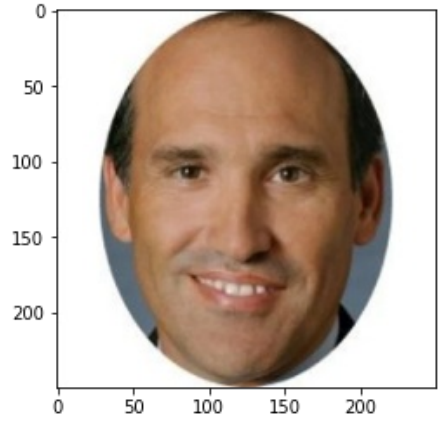

Predicted beauty score:  [2.2799594]



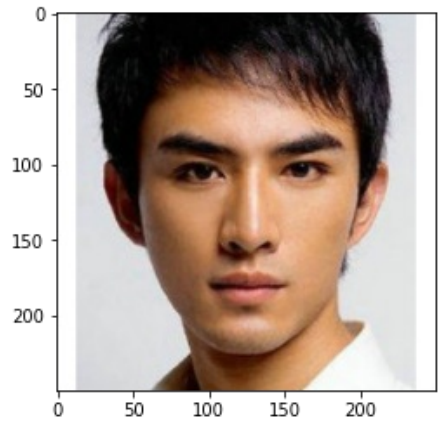

Predicted beauty score:  [2.6239514]



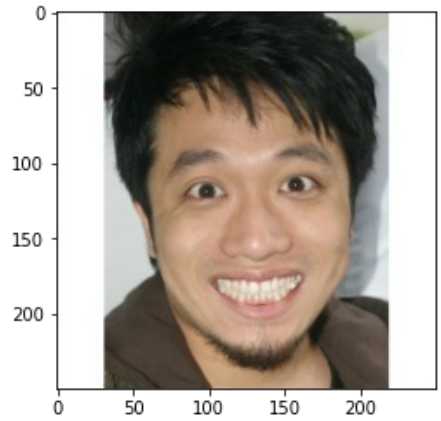

Predicted beauty score:  [2.7663817]



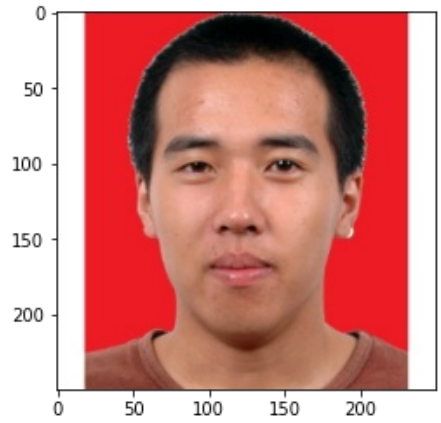

Predicted beauty score:  [3.90770961]



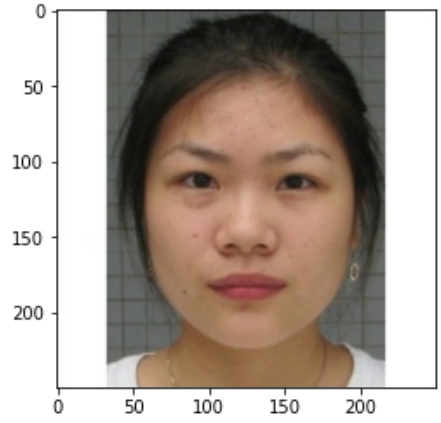
Predicted beauty score:  [3.4591532]



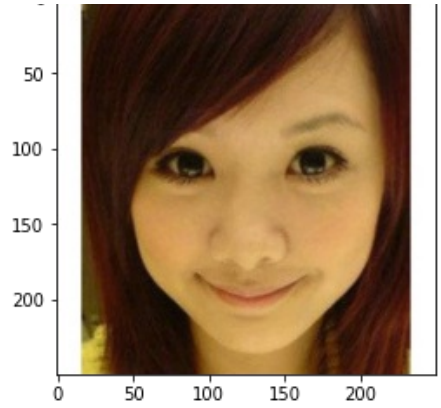
Predicted beauty score:  [2.2067833]



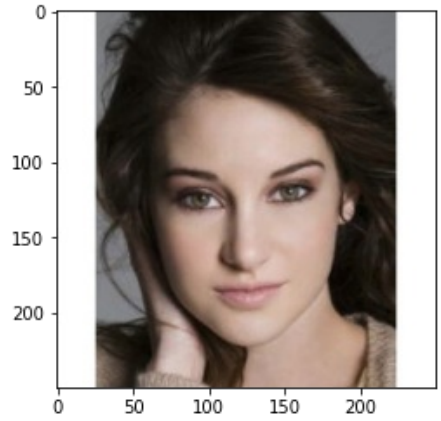
Predicted beauty score:  [3.0423756]



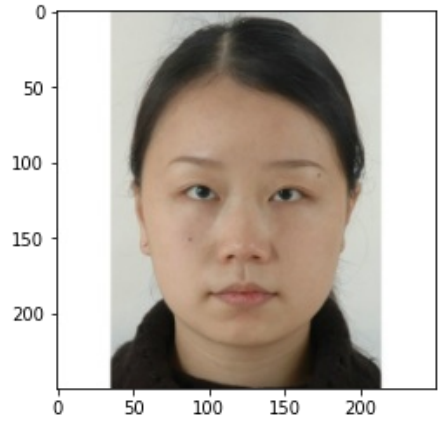
Predicted beauty score:  [3.5269063]



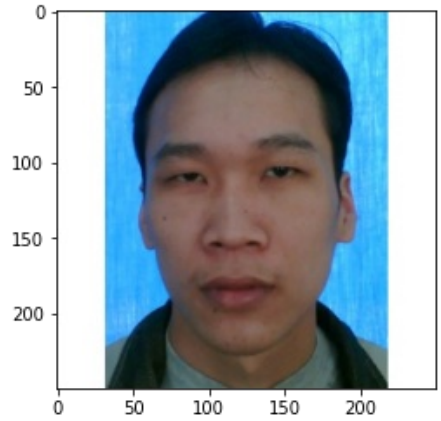

```
Predicted beauty score:  [3.997865]
```



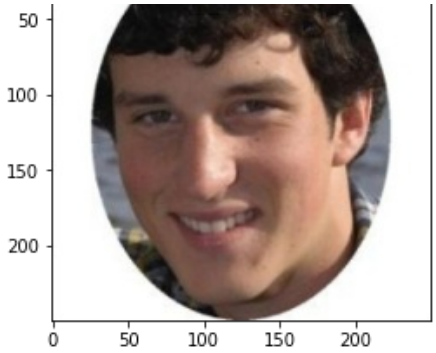

```
Predicted beauty score:  [2.8951976]
```




```
Predicted beauty score:  [2.1443956]
```




```
Predicted beauty score:  [2.9452329]
```




In [ ]:

```
all_predictions = model.predict(images_numpy/255.0)
```

In [ ]:

```
#make histogram of predictions
printdf = pd.DataFrame(all_predictions, columns = ["Schönheitsklassifizierung"])
printdf[0:5499].hist(bins = 40, figsize = (14,10))
```