# Race and Ethnicity Prediction in Keras

November 11, 2019 / Machine Learning

We've mentioned how to predict the identity, emotion, age and gender with deep learning in previous posts. Ethnicity and race are facial attributes as well similar to previous ones and we can predict it, too. Recognizing ethnicity from face photos could contribute a huge contribution to missing children, search investigations, refugee crisis and genealogy research. We've previously mentioned the ethnicity prediction topic in the perspective of AI Ethics.

Ethnicity diversity

### Data set

I’ve found two different public data sets including ethnicity labeled face pictures.



The first one is FairFace. This one is a large scale data set and it consists of 86K train and 11K test instances. Its labels are East Asian, Southeast Asian, Indian, Black, White, Middle-Eastern and Latino-Hispanic. Merging both east and southeast Asian races into a single Asian race would be better.

```
train_df = pd.read_csv(&amp;amp;amp;quot;fairface_label_train.csv&a
test_df = pd.read_csv(&amp;amp;amp;quot;fairface_label_val.csv&amp;
```

The second one is UTKFace. This one is a small scale data set. It has 10K instances. Besides, its labels are Asian, Indian, Black, White and Others (Latino and Middle Eastern).

Merging two data sets increased the accuracy in my experiments from 68% to 72% but I had to replace Latino and Middle Eastern races to Others. In other words, UTKFace would not increase the accuracy as expected. That's why, I prefer to train my model with just FairFace data set.

## Ethnicity distribution

The number of instances for each race is homogeneous in FairFace data set.

```
100*train_df.groupby([&amp;#039;race&amp;#039;]).count()[[&amp;#039
```

| | file |
|---|---|
| **race** | |
| **Black** | 14.102416 |
| **East Asian** | 14.164668 |
| **Indian** | 14.201559 |
| **Latino_Hispanic** | 15.409711 |
| **Middle Eastern** | 10.624366 |
| **Southeast Asian** | 12.444665 |
| **White** | 19.052615 |

Distribution in FairFace

I've merged two Asian races into a single Asian race.

```
idx = train_df[(train_df[&amp;#039;race&amp;#039;] == &amp;#039;Eas
train_df.loc[idx, &amp;#039;race&amp;#039;] = &amp;#039;Asian&amp;#

idx = test_df[(test_df[&amp;#039;race&amp;#039;] == &amp;#039;East
test_df.loc[idx, &amp;#039;race&amp;#039;] = &amp;#039;Asian&amp;#0
```

Thus, distribution becomes as illustrated below after data manipulations.

| race | file |
|---|---|
| **Asian** | 26.609333 |
| **Black** | 14.102416 |
| **Indian** | 14.201559 |
| **Latino_Hispanic** | 15.409711 |
| **Middle Eastern** | 10.624366 |
| **White** | 19.052615 |

Distribution after data manipulation

## Reading image pixels

The original data set includes just base image names and its race.

```
train_df.head()
```

| | file | race |
|---|---|---|
| **0** | FairFace/train/1.jpg | East Asian |
| **1** | FairFace/train/2.jpg | Indian |
| **2** | FairFace/train/3.jpg | Black |
| **3** | FairFace/train/4.jpg | Indian |
| **4** | FairFace/train/5.jpg | Indian |

FairFace head

We will read image pixels based on the file names.

```
target_size = (224, 224)
def getImagePixels(file):
    img = image.load_img(file, grayscale=False, target_size=target_
    x = image.img_to_array(img).reshape(1, -1)[0]
    return x

train_df[&amp;#039;pixels&amp;#039;] = train_df[&amp;#039;file&amp;
test_df[&amp;#039;pixels&amp;#039;] = test_df[&amp;#039;file&amp;#0
```

Now, images pixels are stored as a column

```
train_df.head()
```

| | file | race | pixels |
|---|---|---|---|
| **0** | FairFace/train/1.jpg | Asian | [8.0, 8.0, 10.0, 9.0, 9.0, 11.0, 10.0, 8.0, 11... |
| **1** | FairFace/train/2.jpg | Indian | [129.0, 127.0, 104.0, 127.0, 125.0, 102.0, 123... |
| **2** | FairFace/train/3.jpg | Black | [216.0, 171.0, 174.0, 212.0, 167.0, 170.0, 206... |
| **3** | FairFace/train/4.jpg | Indian | [42.0, 47.0, 50.0, 42.0, 47.0, 50.0, 41.0, 46.... |
| **4** | FairFace/train/5.jpg | Indian | [44.0, 39.0, 35.0, 44.0, 39.0, 35.0, 43.0, 40.... |

Image pixels added as a column

## Input features

Pixels are stored as a list. We need to reshape each line to (224, 224, 3). Besides, inputs should be normalized in neural networks because of activation functions. This is going to be input feature we will pass to the network as input.

```
train_features = []; test_features = []

for i in range(0, train_df.shape[0]):
    train_features.append(train_df[&amp;#039;pixels&amp;#039;].val

for i in range(0, test_df.shape[0]):
    test_features.append(test_df[&amp;#039;pixels&amp;#039;].value

train_features = np.array(train_features)
train_features = train_features.reshape(train_features.shape[0], 2

```

```
test_features = np.array(test_features)
test_features = test_features.reshape(test_features.shape[0], 224,

train_features = train_features / 255
test_features = test_features / 255
```

### Target

Race column is the target value we will predict. However, we need to apply it to one hot encoding. Network will have 6 outputs – this is the number of races in the data set.

```
train_label = train_df[[&amp;#039;race&amp;#039;]]
test_label = test_df[[&amp;#039;race&amp;#039;]]

races = train_df[&amp;#039;race&amp;#039;].unique()

for j in range(len(races)): #label encoding
    current_race = races[j]
    print(&amp;amp;amp;quot;replacing &amp;amp;amp;quot;,current_r
    train_label[&amp;#039;race&amp;#039;] = train_label[&amp;#039;
    test_label[&amp;#039;race&amp;#039;] = test_label[&amp;#039;ra

train_label = train_label.astype({&amp;#039;race&amp;#039;: &amp;#
test_label = test_label.astype({&amp;#039;race&amp;#039;: &amp;#03

train_target = pd.get_dummies(train_label[&amp;#039;race&amp;#039;
test_target = pd.get_dummies(test_label[&amp;#039;race&amp;#039;],
```

### Train and validation split

Train and test sets are separate. We will predict the test set at the end of this study. We should split train set into train and validation to avoid overfitting. In this way, we can apply early stopping.

```
train_x, val_x, train_y, val_y = train_test_split(
    train_features, train_target.values
    , test_size=0.12, random_state=17
)
```

### Base Model

We will use VGG-Face for transfer learning. Let’s construct it first.

```
model = Sequential()
model.add(ZeroPadding2D((1,1),input_shape=(224,224, 3)))
model.add(Convolution2D(64, (3, 3), activation=&amp;#039;relu&amp;
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, (3, 3), activation=&amp;#039;relu&amp;
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation=&amp;#039;relu&amp
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, (3, 3), activation=&amp;#039;relu&amp
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation=&amp;#039;relu&amp
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation=&amp;#039;relu&amp
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, (3, 3), activation=&amp;#039;relu&amp
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation=&amp;#039;relu&amp
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation=&amp;#039;relu&amp
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation=&amp;#039;relu&amp
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation=&amp;#039;relu&amp
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation=&amp;#039;relu&amp
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, (3, 3), activation=&amp;#039;relu&amp
model.add(MaxPooling2D((2,2), strides=(2,2)))

model.add(Convolution2D(4096, (7, 7), activation=&amp;#039;relu&am
model.add(Dropout(0.5))
model.add(Convolution2D(4096, (1, 1), activation=&amp;#039;relu&am
model.add(Dropout(0.5))
model.add(Convolution2D(2622, (1, 1)))
model.add(Flatten())
model.add(Activation(&amp;#039;softmax&amp;#039;))

#related blog post: https://sefiks.com/2018/08/06/deep-face-recogn
model.load_weights(&amp;#039;vgg_face_weights.h5&amp;#039;)
```

## Transfer Learning

Its early layers can detect some facial patterns already. We do not have to train it from scratch. Because we do not have millions of train set instances. We can lock its early layers and expect the late layers to learn.

```
for layer in model.layers[:-7]:
    layer.trainable = False
```

In this way, its all layers except the last 7 one are locked and its weights will not be updated. We expect its last 7 layers to learn something.

The original VGG-Face network has 2622 outputs but here we need just 6 outputs related to races. We will customize the VGG-Face here and it is going to be VGG-Race now.

```
base_model_output = Sequential()
base_model_output = Convolution2D(num_of_classes, (1, 1), name=&amp
base_model_output = Flatten()(base_model_output)
base_model_output = Activation(&amp;#039;softmax&amp;#039;)(base_mo

race_model = Model(inputs=model.input, outputs=base_model_output)
```

## Training

Instead of feeding all train data, I prefer to feed it as batches. I got the best result for 16.384 (2^14) batch size. I feed randomly selected 16K instances in every epoch. If validation loss would not decrease for 50 rounds, then training should be terminated to avoid overfitting.

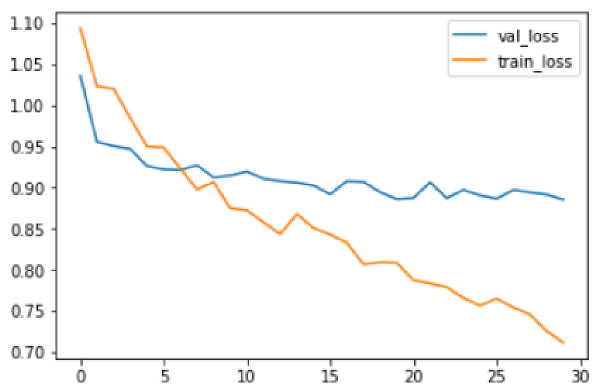
```
race_model.compile(loss=&amp;#039;categorical_crossentropy&amp;#03
, optimizer=keras.optimizers.Adam(), metrics=[&amp;#039;accuracy&a

checkpointer = ModelCheckpoint(filepath=&amp;#039;race_model_singl
, monitor = &amp;amp;amp;quot;val_loss&amp;amp;amp;quot;, verbose=

atch_size = pow(2, 14); patience = 50
ast_improvement = 0; best_iteration = 0
loss = 1000000 #initialize as a large value

for i in range(0, epochs):
    print(&amp;amp;amp;quot;Epoch &amp;amp;amp;quot;, i, &amp;amp;
    ix_train = np.random.choice(train_x.shape[0], size=batch_size)

    score = race_model.fit(
```

```
            train_x[ix_train], train_y[ix_train]
            , epochs=1
            , validation_data=(val_x, val_y)
            , callbacks=[checkpointer]
        )

        val_loss = score.history[&amp;#039;val_loss&amp;#039;][0]; tra
        val_scores.append(val_loss); train_scores.append(train_loss)

        if val_loss &amp;amp;amp;amp;amp;amp;lt; loss:
            loss = val_loss * 1
            last_improvement = 0
            best_iteration = i * 1
        else:
            last_improvement = last_improvement + 1
            print(&amp;amp;amp;quot;try to decrease val loss for &amp;

        if last_improvement == patience:
            print(&amp;amp;amp;quot;there is no loss decrease in valid
            break
```

## Loss

The best epoch was 29. I train the network for 80 rounds but train loss decreased while validation loss increased when epoch > 30 in the following steps. That's exactly overfitting.

```
plt.plot(val_scores[0:best_iteration+1], label=&amp;#039;val_loss&a
plt.plot(train_scores[0:best_iteration+1], label=&amp;#039;train_lc
plt.legend(loc=&amp;#039;upper right&amp;#039;)
plt.show()
```



Train and validation loss

That’s why, I loaded the weights for the best iteration

```
from keras.models import load_model
race_model = load_model(&amp;amp;amp;quot;race_model_single_batch.h
race_model.save_weights(&amp;#039;race_model_single_batch.h5&amp;#0
```

## Evaluation

We train the network with train data set and use validation set to apply early stop. Epoch is the best iteration for validation set actually. However, network could memorize the validation set and it could still be overfitted. That’s why, we haven’t feed test set to the network yet. We expect that test and validation loss should be close if the model is robust.

```
test_perf = race_model.evaluate(test_features, test_target.values,
print(test_perf)

validation_perf = race_model.evaluate(val_x, val_y, verbose=1)
print(validation_perf)

abs(validation_perf[0] - test_perf[0])
```

The both test and validation loss are 0.88 and accuracy are 68%. We can say that the model is robust.

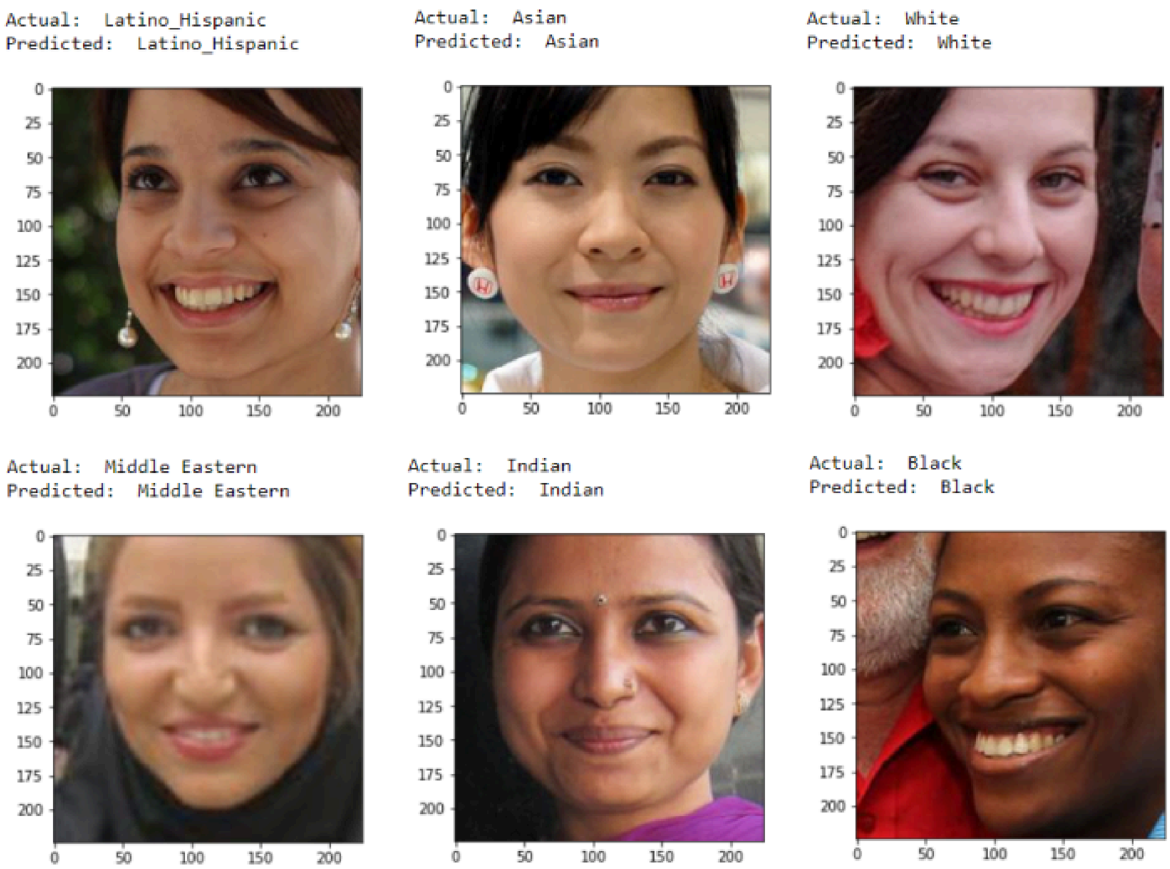
## Prediction

We can make predictions for the test set.

```
predictions = race_model.predict(test_features)
```

Also, we can print prediction and actual values and plot the original image as well.

```
predictions = race_model.predict(test_features)
for i in range(0, predictions.shape[0]):
    prediction = np.argmax(predictions[i])
    prediction_classes.append(races[prediction])

    actual = np.argmax(test_target.values[i])
    actual_classes.append(races[actual])

    if i == 10:
        print(&amp;amp;amp;quot;Actual: &amp;amp;amp;quot;,races[a
        print(&amp;amp;amp;quot;Predicted: &amp;amp;amp;quot;,race
        img = (test_df.iloc[i][&amp;#039;pixels&amp;#039;].reshape
        plt.imshow(img); plt.show()
```



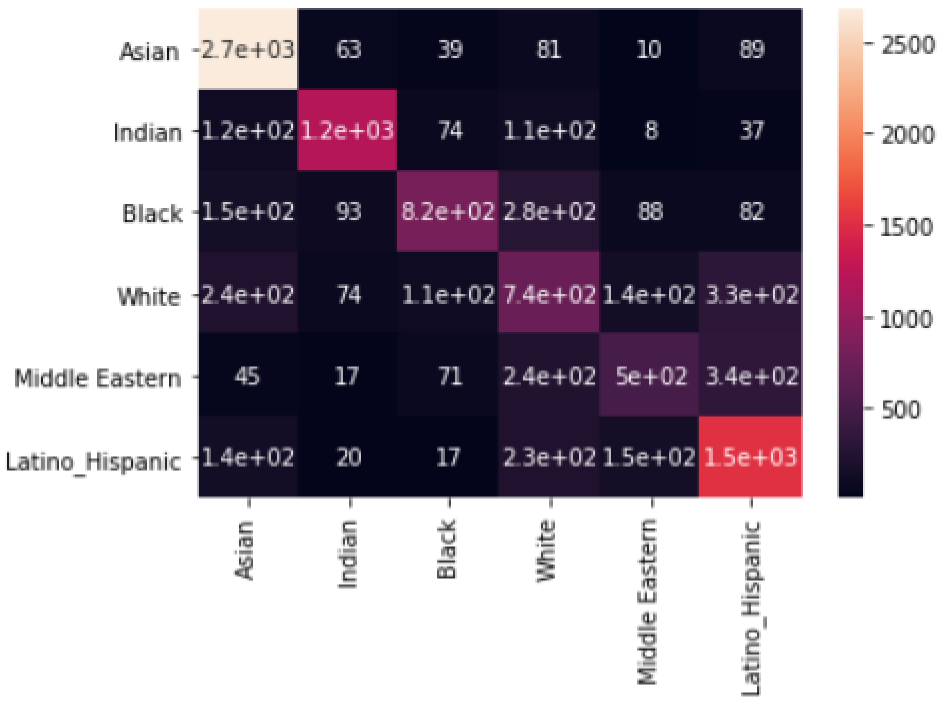

Predictions for randomly selected instances in the test set

## Confusion matrix

Accuracy does not mean anything for classification problems. We need precision and recall values. Confusion matrix is the best way to monitor the success of your model.

```python
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sn

cm = confusion_matrix(actual_classes, prediction_classes)
df_cm = pd.DataFrame(cm, index=races, columns=races)
sn.heatmap(df_cm, annot=True,annot_kws={&amp;amp;amp;quot;size&amp;
```

The following heat map explains everything.




Heat map

## Predicting custom images

We can predict the ethnicity for custom images as well.

```python
demo_set = [&amp;#039;fei-fei-li.jpg&amp;#039;, &amp;#039;sundar-p

for file in demo_set:
    path = &amp;#039;demo/%s&amp;#039; % (file)
    img = image.load_img(path, grayscale=False, target_size=target
    img = image.img_to_array(img).reshape(1, -1)[0]
    img = img.reshape(224, 224, 3)
    img = img / 255

    plt.imshow(img)
    plt.show()

```

```
    img = np.expand_dims(img, axis=0)

    prediction_proba = race_model.predict(img)

    print(&amp;amp;amp;quot;Prediction: &amp;amp;amp;quot;,races[n
    print(&amp;amp;amp;quot;--------------------------&amp;amp;an
```

I've applied prediction for the characters of Silicon Valley. Results are really satisfactory.


Ethnicity of silicon valley characters

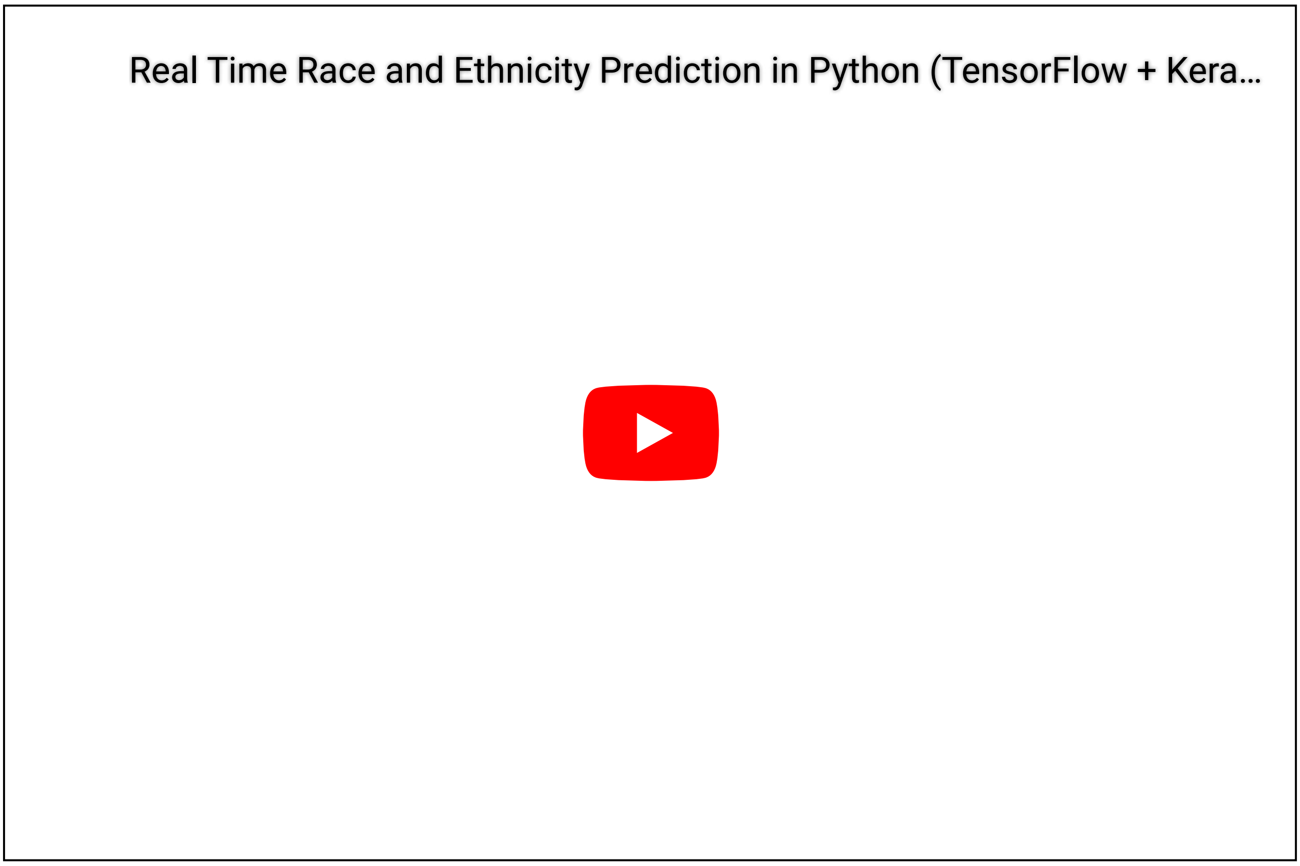
## Loading pre-trained network

I shared the pre-trained network weights on Google Drive. You can skip training step and load the weight when our race model is built.

```
race_model.load_weights(&amp;#039;race_model_weights_full_v2.h5&amp
```

### Real Time Ethnicity Prediction

We can apply race prediction in real time as well. Its source code is pushed to GitHub already. Additionally OpenCV's haar cascade module detects the face and we pass the detected face to the model.




BTW, have you subscribe my youtube channel 🙄

### Conclusion

So, we've mentioned how to build a race and ethnicity classifier from scratch in this post. I pushed the source code of this post as a notebook to GitHub. Besides, its real time implementation code is pushed to GithHub, too. Pre-trained network wei[illegible] shared to Google Drive because of its size. There are many ways to supp[illegible] project – starring the GitHub repo is just one.

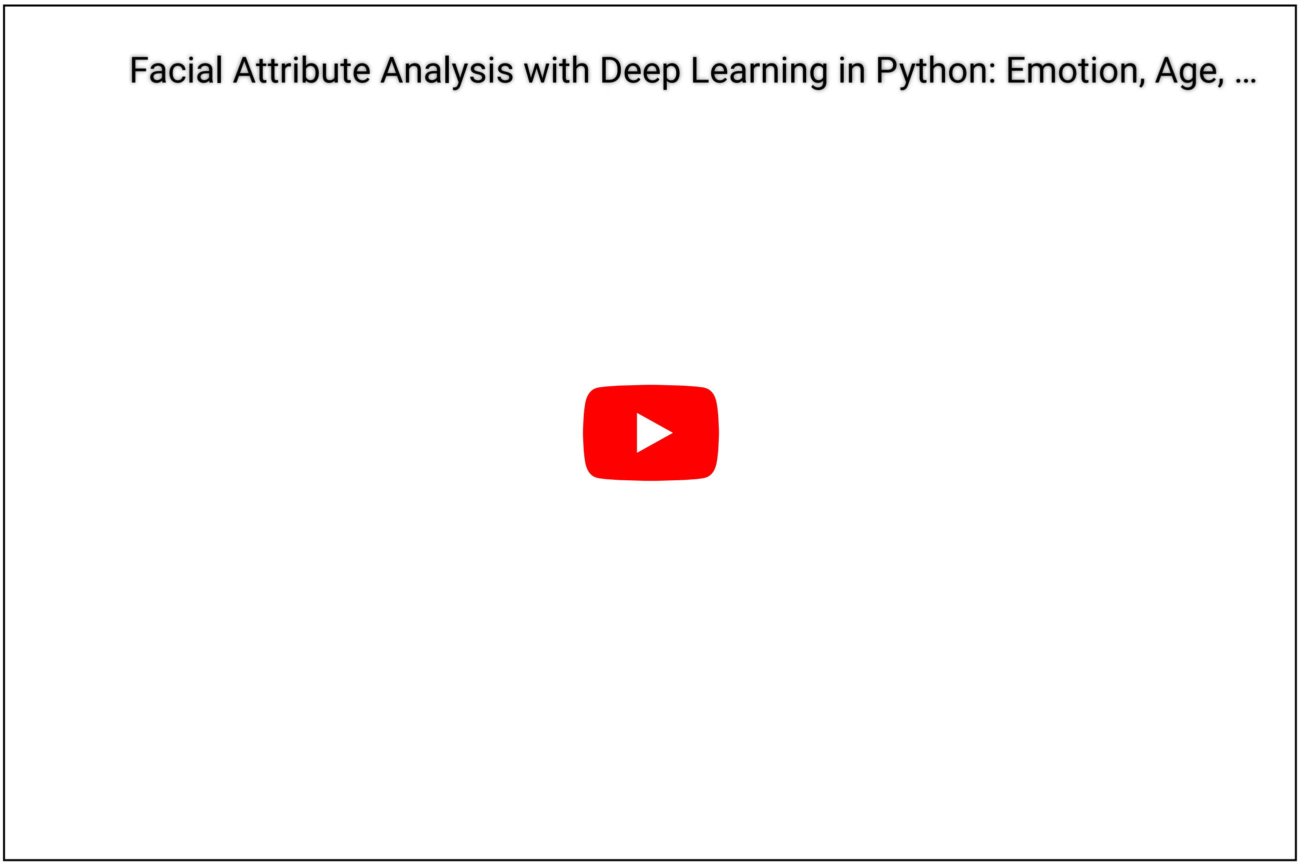
### Python library

Herein, deepface is a lightweight facial analysis framework covering both face recognition and demography such as age, gender, race and emotion. If you are not interested in building neural networks models from scratch, then you might adopt deepface. It is fully open-source and available on PyPI. You can make predictions with a few lines of code.

```python
#!pip install deepface
from deepface import DeepFace

img = "angelina.jpg"
attributes = ['age', 'gender', 'race', 'emotion']

demography = DeepFace.analyze(img, attributes)
```

Deep Face Analysis

Here, you can watch a how to apply facial attribute analysis in python with a just few lines of code.




**Real time implementation**

Real time facial attribute analysis is available in DeepFace

Also, deepface offers an ui built with react js for real time applications.

## Anti-Spoofing and Liveness Detection

What if DeepFace is given fake or spoofed images? This becomes a serious issue if it is used in a security system. To address this, DeepFace includes an anti-spoofing feature for face verification or liveness detection.

Support this blog if you do like!



#ethnicity, #race, #vgg-face

« *Previous* / *Next* »

# 13 Comments

**Shane Carlyon**
November 21, 2019 at 3:15 am

Awesome!!!

**Christer Santos**
December 23, 2020 at 9:11 am

Hi Sefik,

This is awesome!

I'm a big fan of your works.

I am t[illegible]g to implement your approach but I am having a memory issue when loa[illegible] images to arrays. Can you please advise on how can I proceed?

Thanks and more power!

**Sefik Serengil**
December 23, 2020 at 9:16 am

I had a strong machine when I wrote this post. You might load data as batches.

---

**Moya Zhu**
January 6, 2021 at 6:59 pm

Hello Sefik,

Your work helped me a lot!
I'm wondering at the transfer learning part, how should we define the number of classes(num_of_classes)? What number should we put?

base_model_output = Convolution2D(num_of_classes, (1, 1), name='predictions')(model.layers[-4].output)

Is here refer to the number of classes of race output?

**Sefik Serengil**
January 6, 2021 at 7:06 pm

It is the total number of races in this case

---

Pingback: Nacionalidades, etnias y cine — Homo Datus

---

**vishal**
September 27, 2021 at 7:05 am

Is this single person or multi person?

**Sefik Serengil**
October 3, 2021 at 4:50 pm

it just performs for single persons.

---

**vijay antony**
September 26, 2022 at 5:05 pm

hi sefik,how can i predict the race of multiple images which i put them in a csv file?
can you share code for predicting race of images through reading csv files?

---

**Sefik Serengil**
September 30, 2022 at 10:36 pm

1- yes, you can do it.
2- no, I cannot do it for you, sorry.

---

**Maddy**
October 6, 2022 at 3:36 pm

hi,how did you upload train and val csv file?
we have to convert the downloaded zip file into csv?
tell me how to…

---

**vj Anand**
October 7, 2022 at 10:05 am

code:train_df['pixels'] = train_df['file'].progress_apply(getImagePixels)

No such file or directory: 'FairFace/train/1.jpg'

how to fix this issue?

---

**ks**
vember 16, 2022 at 10:59 am

link of download dataset is not working, can you please modify link, & also is there any new release of model?

Comments are closed.



Please cite this post if it helps your research. Here is an example of BibTex entry:

```
@misc{sefiks13763,
  author ={Serengil, Sefik Ilkin},
  title = { Race and Ethnicity Prediction in Keras },
  howpublished = { https://sefiks.com/2019/11/11/race-and-ethnicity-prediction-in-keras/ },
  year = { 2019 },
  note = "[Online; accessed 2024-12-31]"
}
```