This sample demonstrates celebrity face classification using Keras with transfer learning. It's a work from Il Joong Kim available on Kaggle.

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-pytho
n
# For example, here's several helpful packages to load in

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files
in the input directory

import os

data_dir = '../input/5-celebrity-faces-dataset/data'
resnet50weight = '../input/keras-pretrained-models/resnet50_weights_tf_dim_ordering_tf_ke
rnels_notop.h5'
```

In [2]:

```
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model, Sequential
from keras.layers import Conv2D, MaxPooling2D, BatchNormalization, GlobalAveragePooling2
D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.optimizers import RMSprop, SGD
from keras import backend as K

import keras

import matplotlib.pyplot as plt
```

1. Prepare Data

In [3]:

```
img_width, img_height = 200, 200

train_data_dir = os.path.join(data_dir, 'train')
validation_data_dir = os.path.join(data_dir, 'val')
nb_train_samples = 93
nb_validation_samples = 25
epochs = 14
batch_size = 16
numclasses = 5
```

In [4]:

```
# dataset
# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    rotation_range=10,       # randomly rotate images in the range (degrees, 0 to 180)
    zoom_range = 0.1,        # randomly zoom image
    width_shift_range=0.1,   # randomly shift images horizontally (fraction of total width
)
    height_shift_range=0.1, # randomly shift images vertically (fraction of total height)
    #shear_range=0.2,
    vertical_flip=False,
    horizontal_flip=True)

# this is the augmentation configuration we will use for testing:
# only rescaling
test_datagen = ImageDataGenerator(rescale=1. / 255)
```

```
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')

validation_generator = test_datagen.flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical')
```

2. Model

In [5]:

```
if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)
```

The pre-trained chosen network is Resnet50, a residual network consisting of 50 layers. This model will be taken as a basis to train the last two layers using the relatively few training images of the celebrities we're analysing.

In [6]:

```
def resnet50tl(input_shape, outclass, sigma='sigmoid'):

    base_model = None
    base_model = keras.applications.resnet50.ResNet50(weights=None, include_top=False, i
nput_shape=input_shape)
    base_model.load_weights(resnet50weight)

    top_model = Sequential()
    top_model.add(Flatten(input_shape=base_model.output_shape[1:]))
    for i in range(2):
        top_model.add(Dense(4096, activation='relu'))
        top_model.add(Dropout(0.5))
    top_model.add(Dense(outclass, activation=sigma))

    model = None
    model = Model(inputs=base_model.input, outputs=top_model(base_model.output))

    return model
```

In [ ]:

```
model = resnet50tl(input_shape, numclasses, 'softmax')
lr = 1e-5
decay = 1e-7 #0.0
optimizer = RMSprop(lr=lr, decay=decay)
model.compile(loss='categorical_crossentropy',
              optimizer=optimizer,
              metrics=['accuracy'])
```

3. Train

In [ ]:

```
history = model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size)
```

In [ ]:

```
# Get training and test loss histories
training_loss = history.history['loss']
training_acc = history.history['acc']

# Create count of the number of epochs
epoch_count = range(1, len(training_loss) + 1)

fig=plt.figure(figsize=(12, 4))
# Visualize loss history
fig.add_subplot(121)
plt.plot(epoch_count, training_loss, 'r--')
plt.plot(epoch_count, training_acc, 'b-')
plt.legend(['Training Loss', 'Training Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Training Loss/Acc')

# Get training and test loss histories
val_acc = history.history['val_acc']
training_acc = history.history['acc']

# Create count of the number of epochs
epoch_count = range(1, len(val_acc) + 1)

# Visualize loss history
fig.add_subplot(122)
plt.plot(epoch_count, val_acc, 'r--')
plt.plot(epoch_count, training_acc, 'b-')
plt.legend(['Validation Accuracy', 'Training Accuracy'])
plt.xlabel('Epoch')
plt.ylabel('Accuracy')

plt.show();
```

The model obtained after this second part of the training is stores in a JSON file, while the weights are stored in a Keras H5 file. These files will be used in our notebook to evaluate the model with the validation data, in order to save time and go directly to the point.

In [ ]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.models import model_from_json
import numpy
import os

# serialize model to JSON
model_json = model.to_json()
with open("model14.json", "w") as json_file:
    json_file.write(model_json)
# serialize weights to HDF5
model.save_weights("model14.h5")
```