In [1]:

```
import tensorflow as tf
```

```
C:\Users\DELL-PC\Anaconda3\lib\site-packages\h5py\__init__.py:36: Future
Warning: Conversion of the second argument of issubdtype from `float` to
`np.floating` is deprecated. In future, it will be treated as `np.float6
4 == np.dtype(float).type`.
  from ._conv import register_converters as _register_converters
```

In [2]:

```
import pandas as pd
import numpy as np
import cv2
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import f1_score

from keras.applications.inception_v3 import InceptionV3, preprocess_input
from keras import optimizers
from keras.models import Sequential, Model
from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D
from keras.callbacks import ModelCheckpoint
from keras.preprocessing.image import ImageDataGenerator, array_to_img, img_to_array,
from keras.utils import np_utils
from keras.optimizers import SGD
from IPython.core.display import display, HTML
from PIL import Image
from io import BytesIO
import base64

plt.style.use('ggplot')

%matplotlib inline
```

```
Using TensorFlow backend.
```

In [3]:

```
# set variables
main_folder = 'CelebAMask-HQ/'
images_folder = main_folder + 'img_align_celeba/img_align_celeba/'

EXAMPLE_PIC = images_folder + '000506.jpg'

TRAINING_SAMPLES = 2000
VALIDATION_SAMPLES = 2000
TEST_SAMPLES = 2000
IMG_WIDTH = 178
IMG_HEIGHT = 218
BATCH_SIZE = 16
NUM_EPOCHS = 3
```

In [4]:

```
df_attr = pd.read_csv(main_folder + 'list_attr_celeba.csv')
df_attr.set_index('image_id', inplace=True)
df_attr.replace(to_replace=-1, value=0, inplace=True) #replace -1 by 0
df_attr.shape
```

Out[4]:

```
(202599, 40)
```

In [5]:

```
# List of available attributes
for i, j in enumerate(df_attr.columns):
    print(i, j)
```

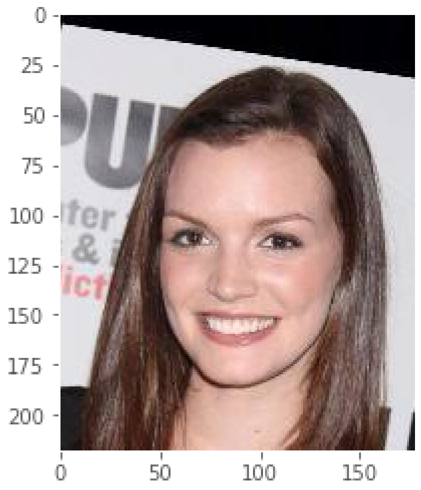
```
0 5_o_Clock_Shadow
1 Arched_Eyebrows
2 Attractive
3 Bags_Under_Eyes
4 Bald
5 Bangs
6 Big_Lips
7 Big_Nose
8 Black_Hair
9 Blond_Hair
10 Blurry
11 Brown_Hair
12 Bushy_Eyebrows
13 Chubby
14 Double_Chin
15 Eyeglasses
16 Goatee
17 Gray_Hair
18 Heavy_Makeup
19 High_Cheekbones
20 Male
21 Mouth_Slightly_Open
22 Mustache
23 Narrow_Eyes
24 No_Beard
25 Oval_Face
26 Pale_Skin
27 Pointy_Nose
28 Receding_Hairline
29 Rosy_Cheeks
30 Sideburns
31 Smiling
32 Straight_Hair
33 Wavy_Hair
34 Wearing_Earrings
35 Wearing_Hat
36 Wearing_Lipstick
37 Wearing_Necklace
38 Wearing_Necktie
39 Young
```

In [6]:

```
# plot picture and attributes
from keras.preprocessing.image import ImageDataGenerator
img = load_img(EXAMPLE_PIC)
plt.grid(False)
plt.imshow(img)
df_attr.loc[EXAMPLE_PIC.split('/')[-1]][['Smiling','Male','Young']] #some attributes
```

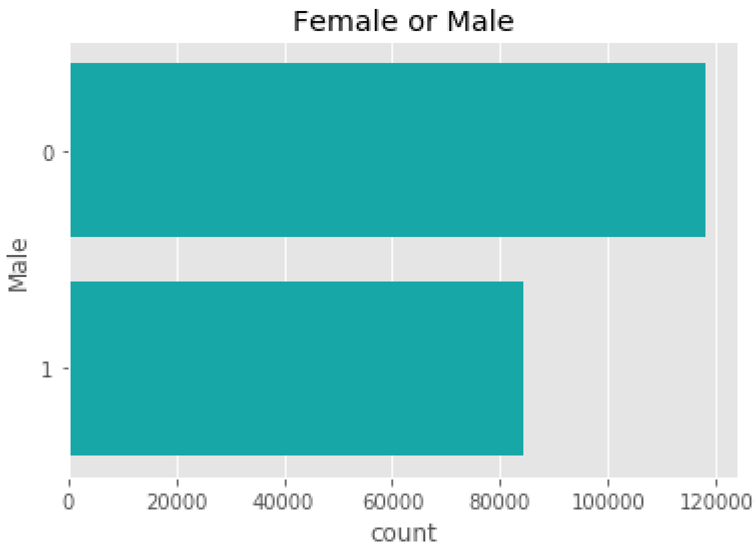
Out[6]:

```
Smiling    1
Male       0
Young      1
Name: 000506.jpg, dtype: int64
```



In [7]:

```
plt.title('Female or Male')
sns.countplot(y='Male', data=df_attr, color="c")
plt.show()
```



In [8]:

```
df_partition = pd.read_csv(main_folder + 'list_eval_partition.csv')
df_partition.head()
```

Out[8]:

	image_id	partition
0	000001.jpg	0
1	000002.jpg	0
2	000003.jpg	0
3	000004.jpg	0
4	000005.jpg	0

In [9]:

```
# display counter by partition
# 0 -> TRAINING
# 1 -> VALIDATION
# 2 -> TEST
df_partition['partition'].value_counts().sort_index()
```

Out[9]:

```
0    162770
1     19867
2     19962
Name: partition, dtype: int64
```

In [10]:

```
# join the partition with the attributes
df_partition.set_index('image_id', inplace=True)
df_par_attr = df_partition.join(df_attr['Male'], how='inner')
df_par_attr.head()
```

Out[10]:

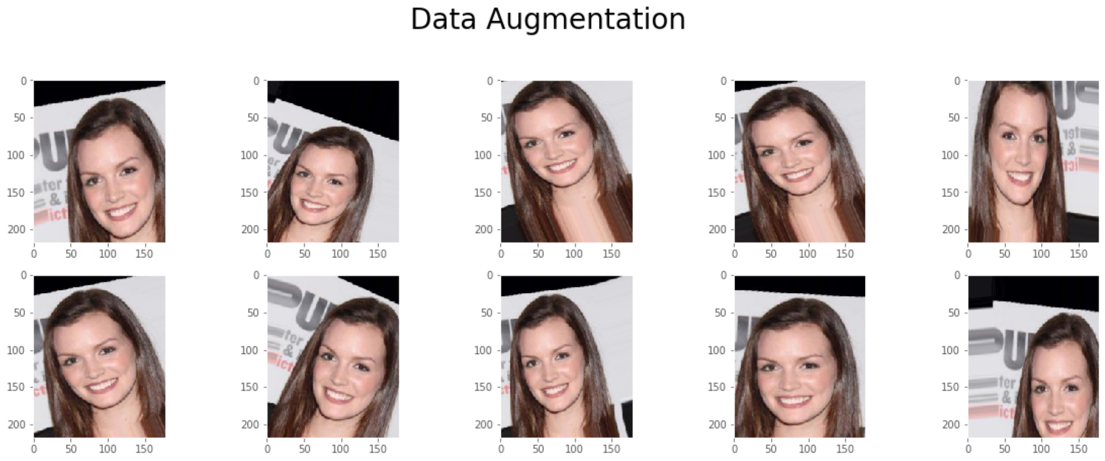
	partition	Male
image_id		
000001.jpg	0	0
000002.jpg	0	0
000003.jpg	0	1
000004.jpg	0	0
000005.jpg	0	0

In [11]:

```
def load_reshape_img(fname):
    img = load_img(fname)
    x = img_to_array(img)/255.
    x = x.reshape((1,) + x.shape)

    return x


def generate_df(partition, attr, num_samples):
    '''
    partition
        0 -> train
        1 -> validation
        2 -> test

    '''

    df_ = df_par_attr[(df_par_attr['partition'] == partition)
                           & (df_par_attr[attr] == 0)].sample(int(num_samples/2))
    df_ = pd.concat([df_,
                      df_par_attr[(df_par_attr['partition'] == partition)
                                  & (df_par_attr[attr] == 1)].sample(int(num_samples/2

    # for Train and Validation
    if partition != 2:
        x_ = np.array([load_reshape_img(images_folder + fname) for fname in df_.index]
        x_ = x_.reshape(x_.shape[0], 218, 178, 3)
        y_ = np_utils.to_categorical(df_[attr],2)
    # for Test
    else:
        x_ = []
        y_ = []

        for index, target in df_.iterrows():
            im = cv2.imread(images_folder + index)
            im = cv2.resize(cv2.cvtColor(im, cv2.COLOR_BGR2RGB), (IMG_WIDTH, IMG_HEIGH
            im = np.expand_dims(im, axis =0)
            x_.append(im)
            y_.append(target[attr])

    return x_, y_
```

In [12]:

```
# Generate image generator for data augmentation
datagen =  ImageDataGenerator(
  #preprocessing_function=preprocess_input,
  rotation_range=30,
  width_shift_range=0.2,
  height_shift_range=0.2,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True
)

# Load one image and reshape
img = load_img(EXAMPLE_PIC)
x = img_to_array(img)/255.
x = x.reshape((1,) + x.shape)
# plot 10 augmented images of the loaded iamge
plt.figure(figsize=(20,10))
plt.suptitle('Data Augmentation', fontsize=28)

i = 0
for batch in datagen.flow(x, batch_size=1):
    plt.subplot(3, 5, i+1)
    plt.grid(False)
    plt.imshow(batch.reshape(218, 178, 3))

    if i == 9:
        break

    i += 1

plt.show()
```



In [13]:

```
# Train data
x_train, y_train = generate_df(0, 'Male', TRAINING_SAMPLES)

# Train - Data Preparation - Data Augmentation with generators
train_datagen =  ImageDataGenerator(
  preprocessing_function=preprocess_input,
  rotation_range=30,
  width_shift_range=0.2,
  height_shift_range=0.2,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True,
)

train_datagen.fit(x_train)

train_generator = train_datagen.flow(
x_train, y_train,
batch_size=BATCH_SIZE,
)
```

In [14]:

```
# Validation Data
x_valid, y_valid = generate_df(1, 'Male', VALIDATION_SAMPLES)
```

In [15]:

```
# Import InceptionV3 Model
inc_model = InceptionV3(weights='inceptionv3/inception_v3_weights_tf_dim_ordering_tf_k
                        include_top=False,
                        input_shape=(IMG_HEIGHT, IMG_WIDTH, 3))

print("number of layers:", len(inc_model.layers))
#inc_model.summary()
```

```
number of layers: 311
```

In [16]:

```
#Adding custom Layers
x = inc_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation="relu")(x)
x = Dropout(0.5)(x)
x = Dense(512, activation="relu")(x)
predictions = Dense(2, activation="softmax")(x)
```

In [17]:

```
# creating the final model
model_ = Model(inputs=inc_model.input, outputs=predictions)

# Lock initial layers to do not be trained
for layer in model_.layers[:52]:
    layer.trainable = False

# compile the model
model_.compile(optimizer=SGD(lr=0.0001, momentum=0.9)
                    , loss='categorical_crossentropy'
                    , metrics=['accuracy'])
```

In [21]:

```
#https://keras.io/models/sequential/ fit generator
checkpointer = ModelCheckpoint(filepath='weights.best.inc.male.h5',
                               verbose=1, save_best_only=True)
```

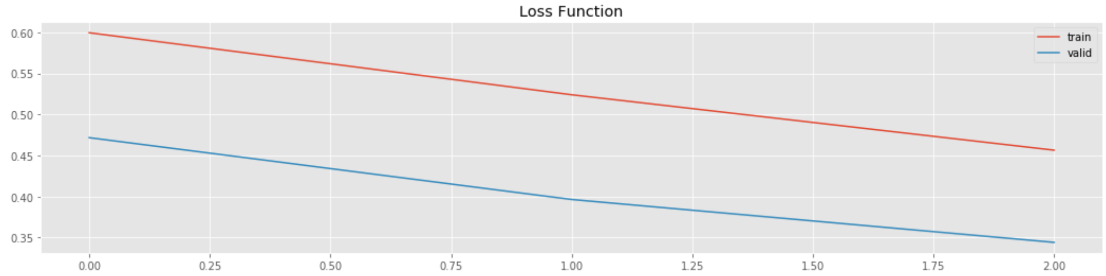
In [22]:

```
hist = model_.fit_generator(train_generator
                     , validation_data = (x_valid, y_valid)
                      , steps_per_epoch= TRAINING_SAMPLES/BATCH_SIZE
                      , epochs= NUM_EPOCHS
                      , callbacks=[checkpointer]
                      , verbose=1
                    )
```

```
Epoch 1/3
125/125 [==============================] - 1035s 8s/step - loss: 0.5994
- accuracy: 0.6820 - val_loss: 0.4717 - val_accuracy: 0.8400

Epoch 00001: val_loss improved from inf to 0.47170, saving model to weig
hts.best.inc.male.h5
Epoch 2/3
125/125 [==============================] - 1168s 9s/step - loss: 0.5240
- accuracy: 0.7630 - val_loss: 0.3964 - val_accuracy: 0.8520

Epoch 00002: val_loss improved from 0.47170 to 0.39644, saving model to
weights.best.inc.male.h5
Epoch 3/3
125/125 [==============================] - 1126s 9s/step - loss: 0.4565
- accuracy: 0.7970 - val_loss: 0.3443 - val_accuracy: 0.8635

Epoch 00003: val_loss improved from 0.39644 to 0.34427, saving model to
weights.best.inc.male.h5
```

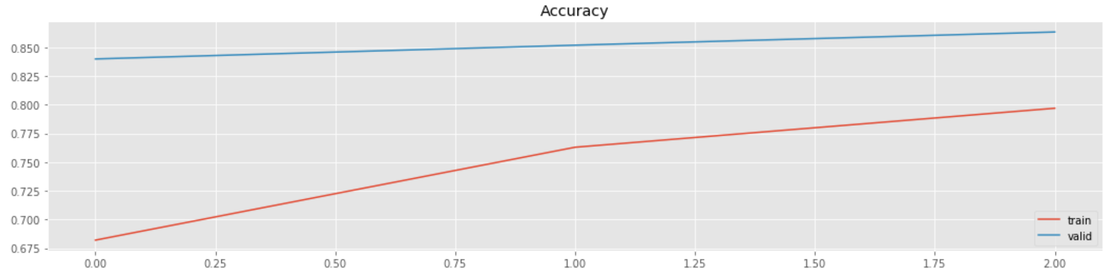
In [23]:

```
# Plot loss function value through epochs
plt.figure(figsize=(18, 4))
plt.plot(hist.history['loss'], label = 'train')
plt.plot(hist.history['val_loss'], label = 'valid')
plt.legend()
plt.title('Loss Function')
plt.show()
```



In [24]:

```
# Plot accuracy through epochs
plt.figure(figsize=(18, 4))
plt.plot(hist.history['accuracy'], label = 'train')
plt.plot(hist.history['val_accuracy'], label = 'valid')
plt.legend()
plt.title('Accuracy')
plt.show()
```



In [25]:

```
#load the best model
model_.load_weights('weights.best.inc.male.h5')
```

In [26]:

```
# Test Data
x_test, y_test = generate_df(2, 'Male', TEST_SAMPLES)

# generate prediction
model_predictions = [np.argmax(model_.predict(feature)) for feature in x_test ]

# report test accuracy
test_accuracy = 100 * np.sum(np.array(model_predictions)==y_test) / len(model_predicti
print('Model Evaluation')
print('Test accuracy: %.4f%%' % test_accuracy)
print('f1_score:', f1_score(y_test, model_predictions))
```

```
Model Evaluation
Test accuracy: 86.8000%
f1_score: 0.8617801047120419
```

In [27]:

```
#dictionary to name the prediction
gender_target = {0: 'Female'
                , 1: 'Male'}

def img_to_display(filename):
    # inspired on this kernel:
    # https://www.kaggle.com/stassl/displaying-inline-images-in-pandas-dataframe
    # credits to stassl :)

    i = Image.open(filename)
    i.thumbnail((200, 200), Image.LANCZOS)

    with BytesIO() as buffer:
        i.save(buffer, 'jpeg')
        return base64.b64encode(buffer.getvalue()).decode()


def display_result(filename, prediction, target):
    '''
    Display the results in HTML

    '''

    gender = 'Male'
    gender_icon = "https://i.imgur.com/nxWan2u.png"

    if prediction[1] <= 0.5:
        gender_icon = "https://i.imgur.com/oAAb8rd.png"
        gender = 'Female'

    display_html = '''
    <div style="overflow: auto;  border: 2px solid #D8D8D8;
        padding: 5px; width: 420px;" >
        <img src="data:image/jpeg;base64,{}" style="float: left;" width="200" height="
        <div style="padding: 10px 0px 0px 20px; overflow: auto;">
            <img src="{}" style="float: left;" width="40" height="40">
            <h3 style="margin-left: 50px; margin-top: 2px;">{}</h3>
            <p style="margin-left: 50px; margin-top: 6px; font-size: 12px">{} prob.</p
            <p style="margin-left: 50px; margin-top: 16px; font-size: 12px">Real Targe
            <p style="margin-left: 50px; margin-top: 16px; font-size: 12px">Filename: 
        </div>
    </div>
    '''.format(img_to_display(filename)
               , gender_icon
               , gender
               , "{0:.2f}%".format(round(max(prediction)*100,2))
               , gender_target[target]
               , filename.split('/')[-1]
               )

    display(HTML(display_html))
```

In [28]:

```
def gender_prediction(filename):
    '''
    predict the gender

    input:
        filename: str of the file name

    return:
        array of the prob of the targets.

    '''

    im = cv2.imread(filename)
    im = cv2.resize(cv2.cvtColor(im, cv2.COLOR_BGR2RGB), (178, 218)).astype(np.float32
    im = np.expand_dims(im, axis =0)

    # prediction
    result = model_.predict(im)
    prediction = np.argmax(result)

    return result
```

In [31]:

```
#select random images of the test partition
df_to_test = df_par_attr[(df_par_attr['partition'] == 2)].sample(8)

for index, target in df_to_test.iterrows():
    result = gender_prediction(images_folder + index)

    #display result
    display_result(images_folder + index, result[0], target['Male'])
```

Female

92.72% prob.

Real Target: Female

Filename: 200226.jpg