Image Recognition for Gender Detection

In this project, we will build a machine learning model using CNN to predict, from a given picture, whether the celebrity is male or female.

Import libraries

In [1]:
```
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, load_img
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 [2]:
```
import tensorflow as tf
print(tf.__version__)
```

```
1.11.0-rc1
```

Data Exploration

Using the CelebA Dataset, which includes images of 178 x 218 px. Below is an example of how the pictures looks like.

In [3]:
```
# set variables
main_folder = '../input/celeba-dataset/'
images_folder = main_folder + 'img_align_celeba/img_align_celeba/'

EXAMPLE_PIC = images_folder + '000506.jpg'

TRAINING_SAMPLES = 10000
VALIDATION_SAMPLES = 2000
TEST_SAMPLES = 2000
IMG_WIDTH = 178
IMG_HEIGHT = 218
```

```
BATCH_SIZE = 16
NUM_EPOCHS = 20
```

Load the attributes of every picture

File: list_attr_celeba.csv

In [4]:
```
# import the data set that include the attribute for each picture
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)
```

List of the available attribute in the CelebA dataset

40 Attributes

In [5]:
```
# List of the available attributes
for i, j in enumerate(df_attr.columns):
    print(i, j)
```

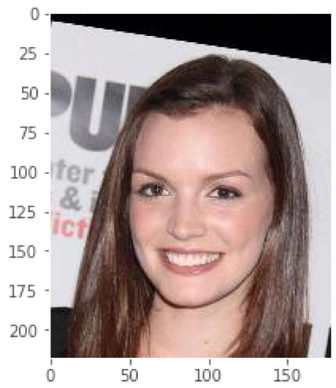
```
0 5_o_Clock_Shadow
1 Arched_Eyebrows
2 Attractive
3 Bags_Under_Eyes
4 Bald
5 Bangs
6 Big_Lips
7 Big_Nose
8 Black_Hair
9 Blond_Hair
10 Blurry
11 Brown_Hair
12 Bushy_Eyebrows
13 Chubby
14 Double_Chin
15 Eyeglasses
16 Goatee
17 Gray_Hair
18 Heavy_Makeup
19 High_Cheekbones
20 Male
21 Mouth_Slightly_Open
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
```

Example of a picture in CelebA dataset

178 x 218 px

In [6]:
```
# plot picture and attributes
img = load_img(EXAMPLE_PIC)
plt.grid(False)
plt.imshow(img)
df_attr.loc[EXAMPLE_PIC.split('/')[-1]][['Smiling','Male','Young']] #some attributes
```

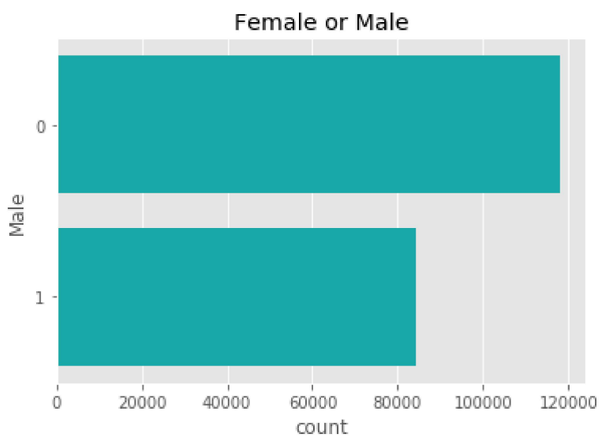
Out[6]:
```
Smiling    1
Male       0
Young      1
Name: 000506.jpg, dtype: int64
```



Distribution of the Attribute

This notebook is an image recognition project for gender. There are more female gender instances than male gender instances in the dataset. This gives us some insight into the need to balance the data in the next steps.

In [7]:
```
# Female or Male?
plt.title('Female or Male')
sns.countplot(y='Male', data=df_attr, color="c")
plt.show()
```



Split Dataset into Training, Validation and Test

The recommended partitioning of images into training, validation, testing of the data set is:

- 1-162770 are training
- 162771-182637 are validation
- 182638-202599 are testing

The partition is in file **list_eval_partition.csv**

Due time execution, by now we will be using a reduced number of images:

- Training 20000 images
- Validation 5000 images
- Test 5000 Images

In [8]:
```
# Recomended partition
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
```

Join the partition and the attributes in the same data frame

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

Generate Partitions (Train, Validation, Test)

Number of images need to be balanced in order to get a good performance for the model, each model will have its own folder of training, validation and test balanced data.

On this step we will create functions that will help us to create each partition.

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_HEIGHT)).astype(np.float32) / 255.0
            im = np.expand_dims(im, axis =0)
            x_.append(im)
            y_.append(target[attr])

    return x_, y_
```

Pre-processing Images: Data Augmentation

Generates Data Augmentation for iamges.

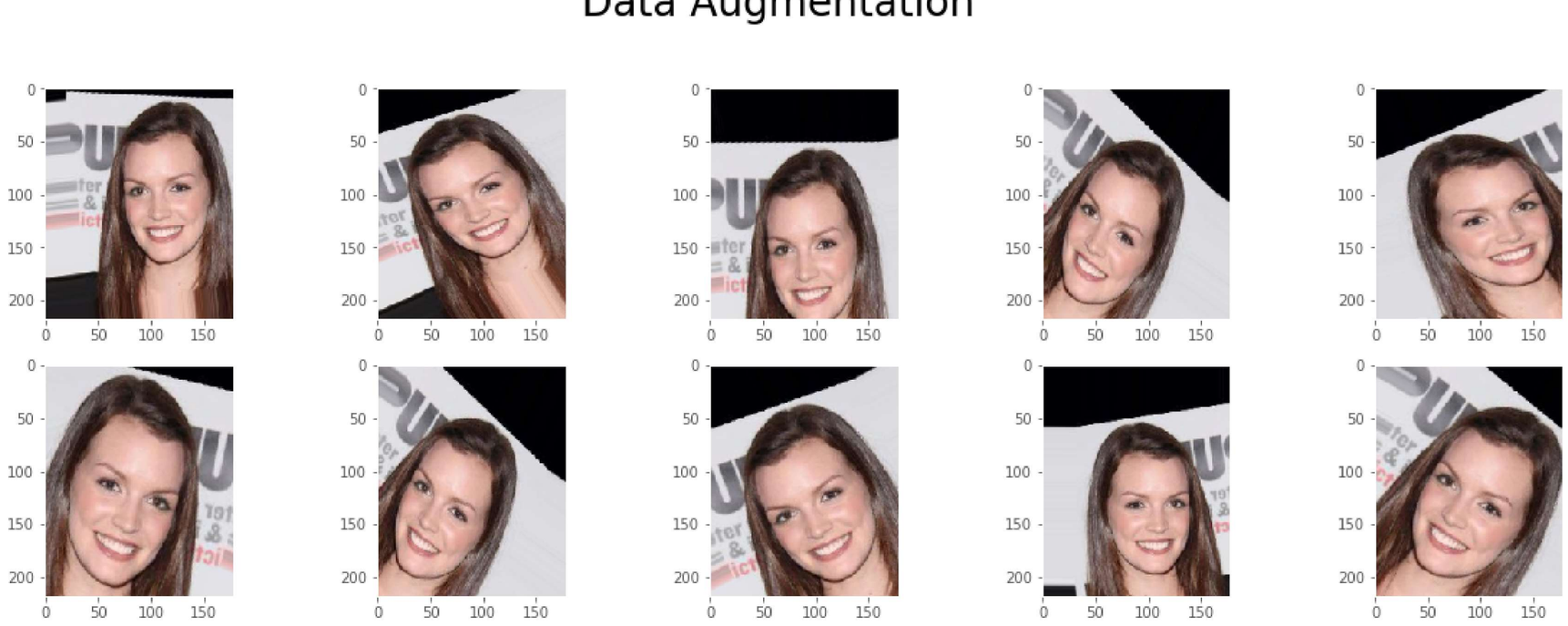
Data Augmentation allows to generate images with modifications to the original ones. The model will learn from these variations (changing angle, size and position), being able to predict better never seen images that could have the same variations in angle, size and position.

Let's start with an example: Data Augmentation

This is how an image will look like after data augmentation (based in the giving parameters below).

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()
```



The result is a new set of images with modifications from the original ones, allowing the model to learn from these variations. This enables it to better handle similar images during the learning process and make more accurate predictions on unseen images.

Build Data Generators

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)

'''
# Validation - Data Preparation - Data Augmentation with generators
valid_datagen = ImageDataGenerator(
  preprocessing_function=preprocess_input,
)

valid_datagen.fit(x_valid)

validation_generator = valid_datagen.flow(
x_valid, y_valid,
)
'''
```

Out[14]:
```
'\n# Validation - Data Preparation - Data Augmentation with generators\nvalid_datagen = ImageDataGenerator(\n  preprocessing_function=preprocess_input,\n)\n\nvalid_datagen.fit(x_valid)\n\nvalidation_g
enerator = valid_datagen.flow(\nx_valid, y_valid,\n)\n'
```

With the data generator created and data for validation, we are ready to start modeling.

Build the Model for Gender Recognition

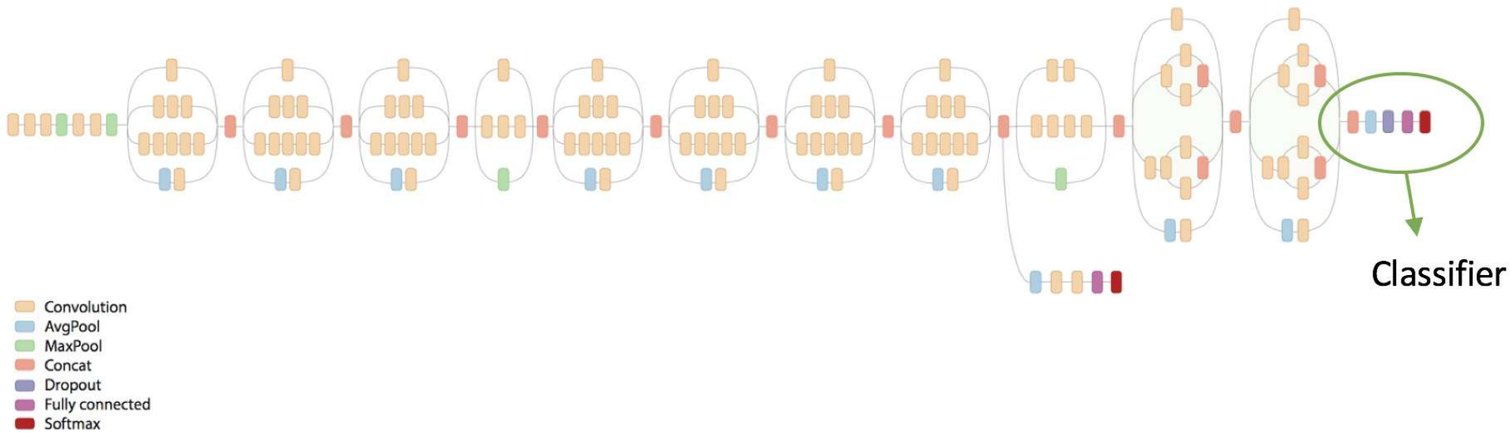
Set the Model

In [15]:
```
# Import InceptionV3 Model
inc_model = InceptionV3(weights='../input/inceptionv3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5',
                        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
```

Inception-V3 model structure

This is the structure of the Inception-V3 model, developed over the imagenet dataset.



source:

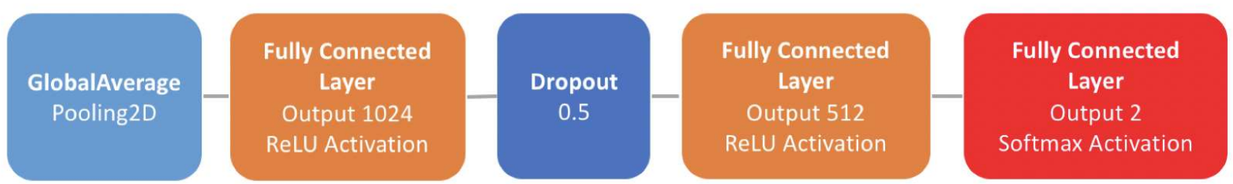
https://hackathonprojects.files.wordpress.com/2016/09/74911-image03.png

The top layers (including classification) are not included. These layers will be replaced for the following layers:

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)
```

New Top layers

Layers to be trained with the new model.



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'])
```

Train Model

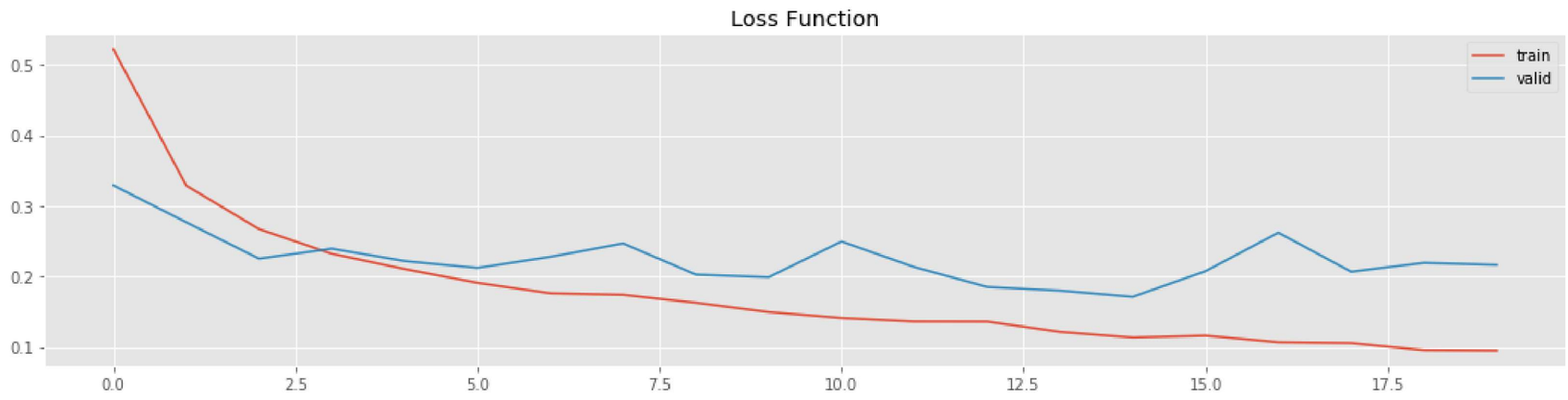
In [18]:
```
#https://keras.io/models/sequential/ fit generator
checkpointer = ModelCheckpoint(filepath='weights.best.inc.male.hdf5',
                               verbose=1, save_best_only=True)
```

In [19]:
```
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
                    )
```

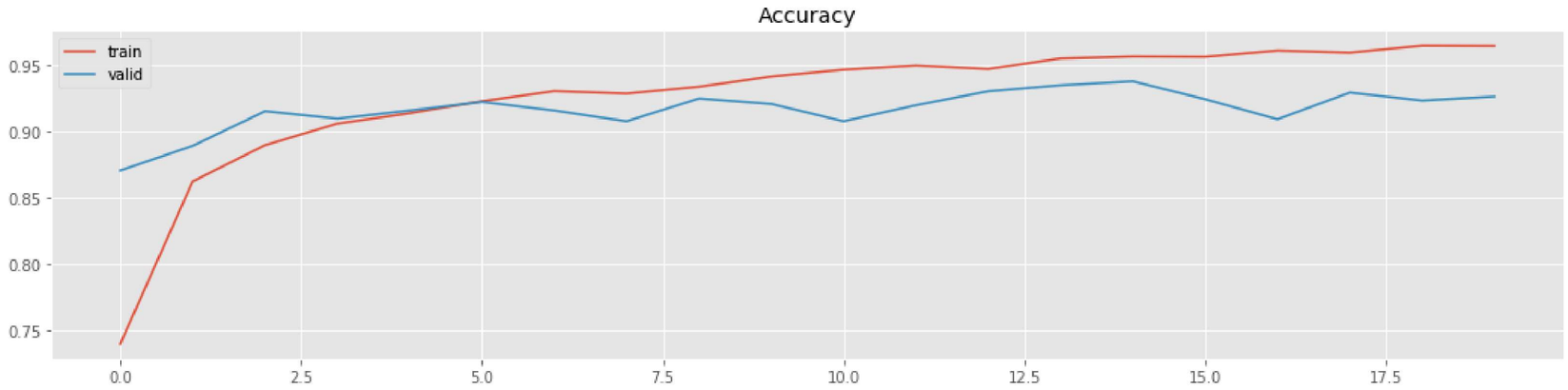
```
Epoch 1/20
625/625 [==============================] - 104s 167ms/step - loss: 0.5214 - acc: 0.7404 - val_loss: 0.3289 - val_acc: 0.8705

Epoch 00001: val_loss improved from inf to 0.32886, saving model to weights.best.inc.male.hdf5
Epoch 2/20
625/625 [==============================] - 89s 143ms/step - loss: 0.3286 - acc: 0.8623 - val_loss: 0.2766 - val_acc: 0.8890

Epoch 00002: val_loss improved from 0.32886 to 0.27662, saving model to weights.best.inc.male.hdf5
Epoch 3/20
625/625 [==============================] - 89s 142ms/step - loss: 0.2669 - acc: 0.8895 - val_loss: 0.2250 - val_acc: 0.9150

Epoch 00003: val_loss improved from 0.27662 to 0.22502, saving model to weights.best.inc.male.hdf5
Epoch 4/20
625/625 [==============================] - 91s 145ms/step - loss: 0.2320 - acc: 0.9057 - val_loss: 0.2393 - val_acc: 0.9095

Epoch 00004: val_loss did not improve from 0.22502
Epoch 5/20
625/625 [==============================] - 88s 141ms/step - loss: 0.2103 - acc: 0.9135 - val_loss: 0.2216 - val_acc: 0.9155

Epoch 00005: val_loss improved from 0.22502 to 0.22164, saving model to weights.best.inc.male.hdf5
Epoch 6/20
625/625 [==============================] - 87s 139ms/step - loss: 0.1908 - acc: 0.9225 - val_loss: 0.2119 - val_acc: 0.9220

Epoch 00006: val_loss improved from 0.22164 to 0.21193, saving model to weights.best.inc.male.hdf5
Epoch 7/20
625/625 [==============================] - 87s 139ms/step - loss: 0.1760 - acc: 0.9301 - val_loss: 0.2276 - val_acc: 0.9155

Epoch 00007: val_loss did not improve from 0.21193
Epoch 8/20
625/625 [==============================] - 88s 140ms/step - loss: 0.1739 - acc: 0.9283 - val_loss: 0.2466 - val_acc: 0.9075

Epoch 00008: val_loss did not improve from 0.21193
Epoch 9/20
625/625 [==============================] - 85s 137ms/step - loss: 0.1625 - acc: 0.9334 - val_loss: 0.2025 - val_acc: 0.9245

Epoch 00009: val_loss improved from 0.21193 to 0.20247, saving model to weights.best.inc.male.hdf5
Epoch 10/20
625/625 [==============================] - 86s 137ms/step - loss: 0.1494 - acc: 0.9411 - val_loss: 0.1988 - val_acc: 0.9205

Epoch 00010: val_loss improved from 0.20247 to 0.19882, saving model to weights.best.inc.male.hdf5
Epoch 11/20
625/625 [==============================] - 85s 136ms/step - loss: 0.1409 - acc: 0.9462 - val_loss: 0.2494 - val_acc: 0.9075

Epoch 00011: val_loss did not improve from 0.19882
Epoch 12/20
625/625 [==============================] - 86s 137ms/step - loss: 0.1361 - acc: 0.9493 - val_loss: 0.2131 - val_acc: 0.9195

Epoch 00012: val_loss did not improve from 0.19882
Epoch 13/20
625/625 [==============================] - 86s 138ms/step - loss: 0.1359 - acc: 0.9467 - val_loss: 0.1851 - val_acc: 0.9300

Epoch 00013: val_loss improved from 0.19882 to 0.18507, saving model to weights.best.inc.male.hdf5
Epoch 14/20
625/625 [==============================] - 85s 136ms/step - loss: 0.1213 - acc: 0.9547 - val_loss: 0.1797 - val_acc: 0.9345

Epoch 00014: val_loss improved from 0.18507 to 0.17970, saving model to weights.best.inc.male.hdf5
Epoch 15/20
625/625 [==============================] - 85s 136ms/step - loss: 0.1133 - acc: 0.9561 - val_loss: 0.1710 - val_acc: 0.9375

Epoch 00015: val_loss improved from 0.17970 to 0.17105, saving model to weights.best.inc.male.hdf5
```

```
Epoch 16/20
625/625 [==============================] - 85s 137ms/step - loss: 0.1160 - acc: 0.9559 - val_loss: 0.2073 - val_acc: 0.9240

Epoch 00016: val_loss did not improve from 0.17105
Epoch 17/20
625/625 [==============================] - 85s 136ms/step - loss: 0.1065 - acc: 0.9605 - val_loss: 0.2619 - val_acc: 0.9090

Epoch 00017: val_loss did not improve from 0.17105
Epoch 18/20
625/625 [==============================] - 84s 135ms/step - loss: 0.1054 - acc: 0.9590 - val_loss: 0.2065 - val_acc: 0.9290

Epoch 00018: val_loss did not improve from 0.17105
Epoch 19/20
625/625 [==============================] - 85s 135ms/step - loss: 0.0954 - acc: 0.9642 - val_loss: 0.2192 - val_acc: 0.9230

Epoch 00019: val_loss did not improve from 0.17105
Epoch 20/20
625/625 [==============================] - 85s 136ms/step - loss: 0.0949 - acc: 0.9640 - val_loss: 0.2163 - val_acc: 0.9260

Epoch 00020: val_loss did not improve from 0.17105
```

In [20]:
```
# 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 [21]:
```
# Plot accuracy through epochs
plt.figure(figsize=(18, 4))
plt.plot(hist.history['acc'], label = 'train')
plt.plot(hist.history['val_acc'], label = 'valid')
plt.legend()
plt.title('Accuracy')
plt.show()
```



Model Evaluation

In [22]:
```
#load the best model
model_.load_weights('weights.best.inc.male.hdf5')
```

In [23]:
```
# 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_predictions)
print('Model Evaluation')
print('Test accuracy: %.4f%%' % test_accuracy)
print('f1_score:', f1_score(y_test, model_predictions))
```

```
Model Evaluation
Test accuracy: 92.4000%
f1_score: 0.9186295503211991
```

Conclusion

The model, built using transfer learning from InceptionV3 and incorporating custom layers, successfully recognizes gender in a given picture with 92.4% accuracy over the test data. However, several limitations have been identified along with opportunities for improvement:

*Training with the entire dataset: Due to computational resource limitations, the model was trained on a subset of images. With access to appropriate computational resources, training the model on the entire dataset could significantly enhance its ability to learn from diverse contexts within pictures, thereby improving its capability to predict unseen images.

*Exploring different CNN structures: Employing alternative CNN structures might enhance the model's performance. However, this task can be computationally expensive as each structure needs evaluation on the test dataset, consuming time and resources.

*Addressing close-up image bias: A majority of pictures in the CelebA Data Set are close-ups of subjects' faces, biasing the model's learning. To mitigate this bias, implementing more sophisticated data preprocessing techniques or augmenting the dataset with images that don't solely focus on close-ups could help the model generalize better to different image compositions.

*Handling multiple subjects in images: While not within the scope of this Notebook, improving the model to handle scenarios with multiple subjects in images would greatly enhance its practical application. Utilizing OpenCV, a robust facial detection tool, could help identify and classify faces separately, even in complex image environments.

play with the Model

In [24]:
```
#dictionary to name the prediction
gender_target = {0: 'Female'
                 , 1: 'Male'}

def img_to_display(filename):
    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="{}" style="float: left;" width="200" height="200">
        <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 Target: {}</p>
            <p style="margin-left: 50px; margin-top: -16px; font-size: 12px">Filename: {}</p>
        </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 [25]:
```
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) / 255.0
    im = np.expand_dims(im, axis =0)

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

    return result
```

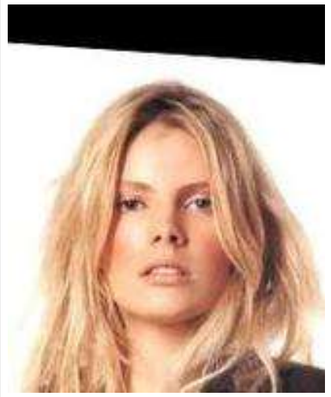
In [26]:
```
#select random images of the test partition
df_to_test = df_par_attr[(df_par_attr['partition'] == 2)].sample(8)

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

99.89% prob.
Real Target: Female
Filename: 187110.jpg

Female

98.51% prob.
Real Target: Female
Filename: 184627.jpg



Female

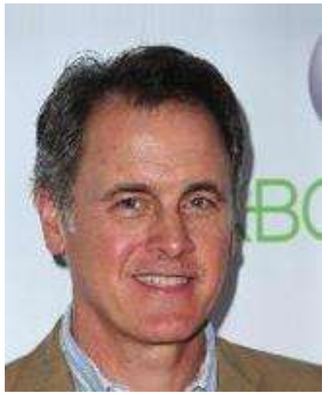
100.00% prob.
Real Target: Female
Filename: 199902.jpg

Female

100.00% prob.
Real Target: Female
Filename: 188736.jpg

Female

100.00% prob.
Real Target: Female
Filename: 196566.jpg



Male

96.79% prob.
Real Target: Male
Filename: 190124.jpg

Male

63.12% prob.
Real Target: Male
Filename: 184077.jpg

Female

100.00% prob.
Real Target: Female
Filename: 193146.jpg