```
import pandas as pd
import numpy as np
import seaborn as sns
import os
from PIL import Image, ImageOps
from sklearn.model_selection import train_test_split

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
```

2. Loading Data

In [2]:

```
images = []
ages = []
genders = []

for i in os.listdir('../input/utkface-new/crop_part1/')[0:8000]:
    split = i.split('_')
    ages.append(int(split[0]))
    genders.append(int(split[1]))
    images.append(Image.open('../input/utkface-new/crop_part1/' + i))
```

In [3]:

```
images = pd.Series(list(images), name = 'Images')
ages = pd.Series(list(ages), name = 'Ages')
genders = pd.Series(list(genders), name = 'Genders')

df = pd.concat([images, ages, genders], axis=1)
df
```

Out[3]:

	Images	Ages	Genders
0	<PIL.JpegImagePlugin.JpegImageFile image mode=...	26	0
1	<PIL.JpegImagePlugin.JpegImageFile image mode=...	21	1
2	<PIL.JpegImagePlugin.JpegImageFile image mode=...	17	1
3	<PIL.JpegImagePlugin.JpegImageFile image mode=...	76	0
4	<PIL.JpegImagePlugin.JpegImageFile image mode=...	18	1
...	...	...	...
7995	<PIL.JpegImagePlugin.JpegImageFile image mode=...	3	0
7996	<PIL.JpegImagePlugin.JpegImageFile image mode=...	28	0
7997	<PIL.JpegImagePlugin.JpegImageFile image mode=...	10	0
7998	<PIL.JpegImagePlugin.JpegImageFile image mode=...	8	1
7999	<PIL.JpegImagePlugin.JpegImageFile image mode=...	22	0

8000 rows × 3 columns

In [4]:

```
display(df['Images'][0])
print(df['Ages'][0], df['Genders'][0])
```

```
26 0
```

In [5]:

```
display(df['Images'][1])
print(df['Ages'][1], df['Genders'][1])
```

```
21 1
```

So 0 corresponds to male, 1 corresponds to female.

3. Visualising and Preparing Data

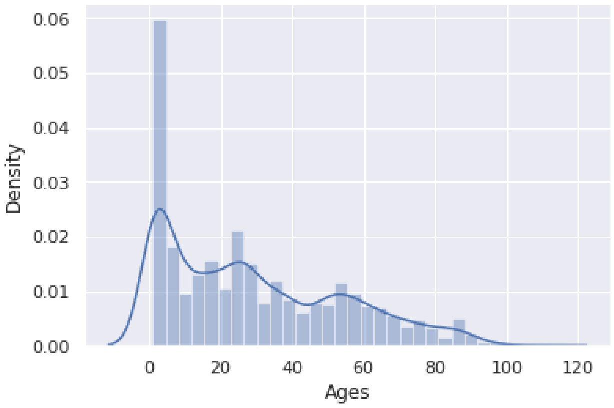
In [6]:

```
sns.set_theme()
sns.distplot(df['Ages'],kde=True, bins=30)
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is
a deprecated function and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function
for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[6]:

```
<AxesSubplot:xlabel='Ages', ylabel='Density'>
```



Too many faces of people between 0 and 4 years old. The model would fit too well to these ages and not enough to the other ages. To resolve this I'm only going to include a third of the images between these ages.

In [7]:

```
under4s = []

for i in range(len(df)):
    if df['Ages'].iloc[i] <= 4:
        under4s.append(df.iloc[i])
under4s = pd.DataFrame(under4s)
under4s = under4s.sample(frac=0.3)

df = df[df['Ages'] > 4]

df = pd.concat([df, under4s], ignore_index = True)
```

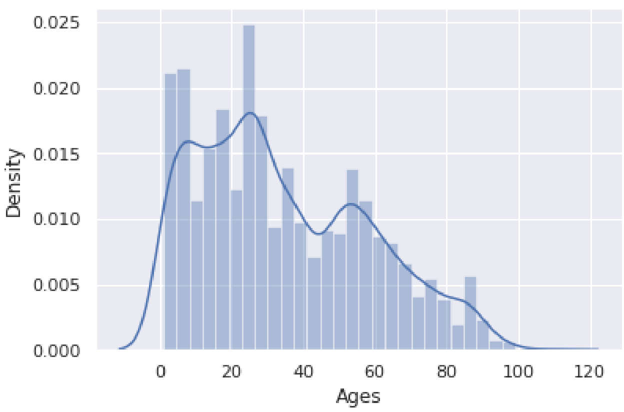
In [8]:

```
sns.distplot(df['Ages'],kde=True, bins=30)
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is
a deprecated function and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function
for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[8]:

```
<AxesSubplot:xlabel='Ages', ylabel='Density'>
```



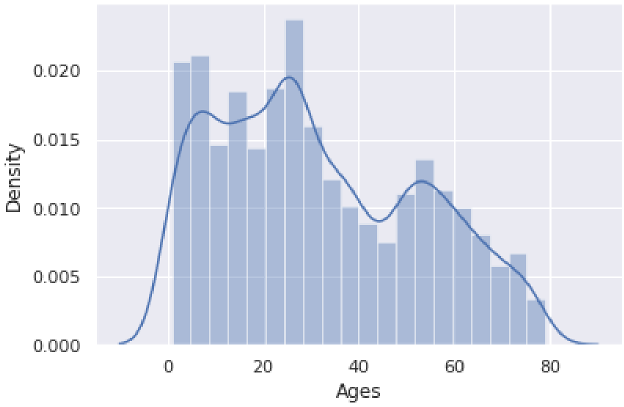
This looks much better! The dataframe is more representative of the population now. However, there aren't many images of people over 80, which would cause the model to not train well enough on those ages. It's best to just remove over 80s and only have a model that can predict the ages of people under 80.

In [9]:
```
df = df[df['Ages'] < 80]
```

In [10]:
```
sns.distplot(df['Ages'],kde=True, bins=20)
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/distributions.py:2557: FutureWarning: `distplot` is
a deprecated function and will be removed in a future version. Please adapt your code to use either
`displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function
for histograms).
  warnings.warn(msg, FutureWarning)
```

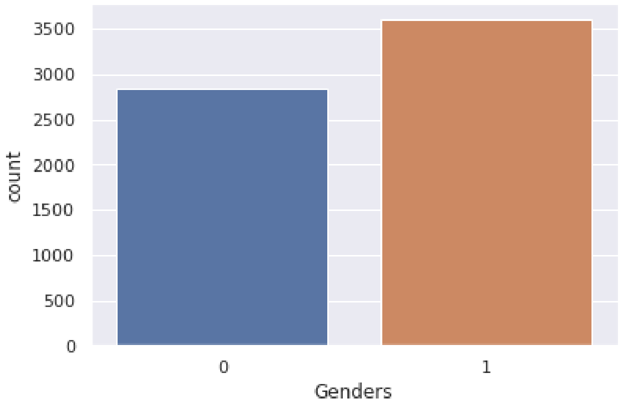
Out[10]:
```
<AxesSubplot:xlabel='Ages', ylabel='Density'>
```



In [11]:
```
sns.countplot(df['Genders'])
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following
variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in an error or misinterpretatio
n.
  FutureWarning
```

Out[11]:
```
<AxesSubplot:xlabel='Genders', ylabel='count'>
```



Not sure what 3 corresponds to - both genders, no gender, unknown, or just an error in the data entry? To be safe, I am going to remove any rows where gender equals 3.

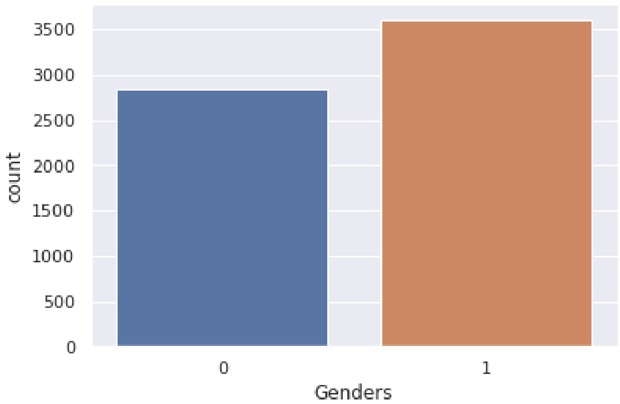
In [12]:

```
df = df[df['Genders'] != 3]
sns.countplot(df['Genders'])
```

```
/opt/conda/lib/python3.7/site-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following
variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`,
and passing other arguments without an explicit keyword will result in an error or misinterpretatio
n.
  FutureWarning
```

Out[12]:

```
<AxesSubplot:xlabel='Genders', ylabel='count'>
```



In [13]:

```
x = []
y = []

for i in range(len(df)):
    df['Images'].iloc[i] = df['Images'].iloc[i].resize((200,200), Image.ANTIALIAS)
    ar = np.asarray(df['Images'].iloc[i])
    x.append(ar)
    agegen = [int(df['Ages'].iloc[i]), int(df['Genders'].iloc[i])]
    y.append(agegen)
x = np.array(x)
```

```
/opt/conda/lib/python3.7/site-packages/pandas/core/indexing.py:670: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/index
ing.html#returning-a-view-versus-a-copy
  iloc._setitem_with_indexer(indexer, value)
```

4. Train Test Split

In [14]:

```
y_age = df['Ages']
y_gender = df['Genders']

x_train_age, x_test_age, y_train_age, y_test_age = train_test_split(x, y_age, test_size=0.2, stratify=y
_age)
x_train_gender, x_test_gender, y_train_gender, y_test_gender = train_test_split(x, y_gender, test_size=
0.2, stratify=y_gender)
```

5. Creating the Models

I will create two individual models - one to predict age and one to predict gender. The age model should be capable of returning continuous values which I will round to the nearest integer, and the gender model should return a binary result.

In [15]:

```
agemodel = Sequential()
agemodel.add(Conv2D(32, (3,3), activation='relu', input_shape=(200, 200, 3)))
agemodel.add(MaxPooling2D((2,2)))
agemodel.add(Conv2D(64, (3,3), activation='relu'))
agemodel.add(MaxPooling2D((2,2)))
agemodel.add(Conv2D(128, (3,3), activation='relu'))
agemodel.add(MaxPooling2D((2,2)))
agemodel.add(Flatten())
agemodel.add(Dense(64, activation='relu'))
agemodel.add(Dropout(0.5))
agemodel.add(Dense(1, activation='relu'))

agemodel.compile(loss='mean_squared_error',
             optimizer=optimizers.Adam(lr=0.0001))

genmodel = Sequential()
genmodel.add(Conv2D(32, (3,3), activation='relu', input_shape=(200, 200, 3)))
genmodel.add(MaxPooling2D((2,2)))
genmodel.add(Conv2D(64, (3,3), activation='relu'))
genmodel.add(MaxPooling2D((2,2)))
genmodel.add(Conv2D(128, (3,3), activation='relu'))
genmodel.add(MaxPooling2D((2,2)))
genmodel.add(Flatten())
genmodel.add(Dense(64, activation='relu'))
genmodel.add(Dropout(0.5))
genmodel.add(Dense(1, activation='sigmoid'))

genmodel.compile(loss='binary_crossentropy',
             optimizer=optimizers.Adam(lr=0.0001),
             metrics=['accuracy'])
```

6. Training the Models

In [16]:

```
datagen = ImageDataGenerator(
      rescale=1./255., width_shift_range = 0.1, height_shift_range = 0.1, horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale=1./255)

train1 = datagen.flow(x_train_age, y_train_age, batch_size=32)

test1 = test_datagen.flow(
        x_test_age, y_test_age,
        batch_size=32)

history1 = agemodel.fit(train1, epochs=50, shuffle=True, validation_data=test1)
```

```
Epoch 1/50
161/161 [==============================] - 44s 253ms/step - loss: 706.2857 - val_loss: 452.0310
Epoch 2/50
161/161 [==============================] - 41s 255ms/step - loss: 493.1964 - val_loss: 403.0540
Epoch 3/50
161/161 [==============================] - 42s 262ms/step - loss: 439.5375 - val_loss: 419.8955
Epoch 4/50
161/161 [==============================] - 41s 257ms/step - loss: 382.1004 - val_loss: 328.3317
Epoch 5/50
161/161 [==============================] - 41s 256ms/step - loss: 376.2793 - val_loss: 306.8690
Epoch 6/50
161/161 [==============================] - 41s 255ms/step - loss: 378.7557 - val_loss: 298.7830
Epoch 7/50
161/161 [==============================] - 42s 258ms/step - loss: 353.1836 - val_loss: 364.5480
Epoch 8/50
161/161 [==============================] - 41s 257ms/step - loss: 350.4301 - val_loss: 273.9537
Epoch 9/50
161/161 [==============================] - 41s 257ms/step - loss: 347.8215 - val_loss: 281.0771
Epoch 10/50
161/161 [==============================] - 41s 257ms/step - loss: 316.9603 - val_loss: 322.0839
Epoch 11/50
161/161 [==============================] - 42s 260ms/step - loss: 321.0114 - val_loss: 257.4635
Epoch 12/50
161/161 [==============================] - 42s 259ms/step - loss: 307.2590 - val_loss: 299.9180
Epoch 13/50
161/161 [==============================] - 41s 256ms/step - loss: 308.6267 - val_loss: 329.4791
Epoch 14/50
161/161 [==============================] - 42s 261ms/step - loss: 314.8022 - val_loss: 245.9729
Epoch 15/50
161/161 [==============================] - 41s 257ms/step - loss: 301.3866 - val_loss: 255.6889
Epoch 16/50
161/161 [==============================] - 41s 258ms/step - loss: 287.3217 - val_loss: 238.2000
Epoch 17/50
161/161 [==============================] - 41s 258ms/step - loss: 299.2067 - val_loss: 226.4820
Epoch 18/50
161/161 [==============================] - 42s 261ms/step - loss: 293.4950 - val_loss: 219.9284
Epoch 19/50
161/161 [==============================] - 41s 258ms/step - loss: 268.2615 - val_loss: 218.2262
Epoch 20/50
161/161 [==============================] - 41s 254ms/step - loss: 284.8218 - val_loss: 234.8680
Epoch 21/50
161/161 [==============================] - 41s 258ms/step - loss: 267.3226 - val_loss: 216.5874
Epoch 22/50
161/161 [==============================] - 42s 260ms/step - loss: 280.1753 - val_loss: 238.4909
Epoch 23/50
161/161 [==============================] - 41s 257ms/step - loss: 269.4055 - val_loss: 200.0749
Epoch 24/50
161/161 [==============================] - 41s 256ms/step - loss: 262.5218 - val_loss: 221.6551
Epoch 25/50
161/161 [==============================] - 42s 260ms/step - loss: 275.6070 - val_loss: 198.7766
Epoch 26/50
161/161 [==============================] - 42s 259ms/step - loss: 266.5272 - val_loss: 221.8195
Epoch 27/50
161/161 [==============================] - 41s 257ms/step - loss: 254.4120 - val_loss: 199.4566
Epoch 28/50
```

```
161/161 [==============================] - 41s 256ms/step - loss: 252.5054 - val_loss: 193.0954
Epoch 29/50
161/161 [==============================] - 42s 259ms/step - loss: 267.9600 - val_loss: 196.8834
Epoch 30/50
161/161 [==============================] - 41s 258ms/step - loss: 241.8575 - val_loss: 187.8295
Epoch 31/50
161/161 [==============================] - 41s 256ms/step - loss: 241.0621 - val_loss: 183.0156
Epoch 32/50
161/161 [==============================] - 42s 259ms/step - loss: 238.2132 - val_loss: 189.9071
Epoch 33/50
161/161 [==============================] - 41s 256ms/step - loss: 238.4509 - val_loss: 193.3439
Epoch 34/50
161/161 [==============================] - 41s 257ms/step - loss: 230.6079 - val_loss: 197.5394
Epoch 35/50
161/161 [==============================] - 41s 255ms/step - loss: 252.5103 - val_loss: 177.1196
Epoch 36/50
161/161 [==============================] - 42s 260ms/step - loss: 248.8367 - val_loss: 190.6460
Epoch 37/50
161/161 [==============================] - 41s 254ms/step - loss: 238.1350 - val_loss: 222.1548
Epoch 38/50
161/161 [==============================] - 41s 257ms/step - loss: 232.1687 - val_loss: 171.6752
Epoch 39/50
161/161 [==============================] - 41s 257ms/step - loss: 227.3713 - val_loss: 180.7836
Epoch 40/50
161/161 [==============================] - 42s 260ms/step - loss: 229.9505 - val_loss: 167.7247
Epoch 41/50
161/161 [==============================] - 41s 257ms/step - loss: 229.1870 - val_loss: 221.3032
Epoch 42/50
161/161 [==============================] - 42s 257ms/step - loss: 232.8839 - val_loss: 166.4539
Epoch 43/50
161/161 [==============================] - 41s 255ms/step - loss: 228.9473 - val_loss: 161.2658
Epoch 44/50
161/161 [==============================] - 42s 260ms/step - loss: 222.0279 - val_loss: 185.0128
Epoch 45/50
161/161 [==============================] - 41s 256ms/step - loss: 219.0961 - val_loss: 160.4378
Epoch 46/50
161/161 [==============================] - 41s 253ms/step - loss: 221.8582 - val_loss: 174.0324
Epoch 47/50
161/161 [==============================] - 41s 256ms/step - loss: 212.9477 - val_loss: 159.7157
Epoch 48/50
161/161 [==============================] - 42s 259ms/step - loss: 220.9071 - val_loss: 189.6833
Epoch 49/50
161/161 [==============================] - 42s 257ms/step - loss: 232.4877 - val_loss: 160.9419
Epoch 50/50
161/161 [==============================] - 41s 253ms/step - loss: 212.5144 - val_loss: 153.1291
```

In [17]:

```
datagen = ImageDataGenerator(
      rescale=1./255., width_shift_range = 0.1, height_shift_range = 0.1, horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale=1./255)

train2 = datagen.flow(x_train_gender, y_train_gender, batch_size=64)

test2 = test_datagen.flow(
        x_test_gender, y_test_gender,
        batch_size=64)

history2 = genmodel.fit(train2, epochs=50, shuffle=True, validation_data=test2)
```

```
Epoch 1/50
81/81 [==============================] - 41s 489ms/step - loss: 0.6716 - accuracy: 0.5686 - val_los
s: 0.6030 - val_accuracy: 0.6568
Epoch 2/50
81/81 [==============================] - 40s 492ms/step - loss: 0.6089 - accuracy: 0.6726 - val_los
s: 0.5504 - val_accuracy: 0.7189
Epoch 3/50
81/81 [==============================] - 40s 492ms/step - loss: 0.5818 - accuracy: 0.6943 - val_los
s: 0.5240 - val_accuracy: 0.7399
Epoch 4/50
81/81 [==============================] - 40s 496ms/step - loss: 0.5557 - accuracy: 0.7236 - val_los
s: 0.4831 - val_accuracy: 0.7896
Epoch 5/50
81/81 [==============================] - 40s 498ms/step - loss: 0.5302 - accuracy: 0.7449 - val_los
s: 0.4828 - val_accuracy: 0.7857
Epoch 6/50
81/81 [==============================] - 40s 496ms/step - loss: 0.4865 - accuracy: 0.7740 - val_los
s: 0.4606 - val_accuracy: 0.8028
Epoch 7/50
81/81 [==============================] - 41s 506ms/step - loss: 0.4879 - accuracy: 0.7628 - val_los
s: 0.4375 - val_accuracy: 0.8005
Epoch 8/50
81/81 [==============================] - 40s 495ms/step - loss: 0.4751 - accuracy: 0.7678 - val_los
s: 0.4417 - val_accuracy: 0.8051
Epoch 9/50
81/81 [==============================] - 40s 492ms/step - loss: 0.4626 - accuracy: 0.7854 - val_los
s: 0.4303 - val_accuracy: 0.8005
Epoch 10/50
81/81 [==============================] - 40s 497ms/step - loss: 0.4556 - accuracy: 0.7767 - val_los
s: 0.4127 - val_accuracy: 0.8113
Epoch 11/50
81/81 [==============================] - 41s 507ms/step - loss: 0.4504 - accuracy: 0.7882 - val_los
s: 0.4207 - val_accuracy: 0.8051
Epoch 12/50
81/81 [==============================] - 40s 490ms/step - loss: 0.4513 - accuracy: 0.7927 - val_los
s: 0.4068 - val_accuracy: 0.8129
Epoch 13/50
81/81 [==============================] - 40s 500ms/step - loss: 0.4346 - accuracy: 0.8021 - val_los
s: 0.3897 - val_accuracy: 0.8168
Epoch 14/50
81/81 [==============================] - 40s 498ms/step - loss: 0.4434 - accuracy: 0.7828 - val_los
s: 0.3890 - val_accuracy: 0.8269
Epoch 15/50
81/81 [==============================] - 40s 494ms/step - loss: 0.4175 - accuracy: 0.8103 - val_los
s: 0.3905 - val_accuracy: 0.8207
Epoch 16/50
81/81 [==============================] - 41s 502ms/step - loss: 0.4232 - accuracy: 0.8068 - val_los
s: 0.3901 - val_accuracy: 0.8261
Epoch 17/50
81/81 [==============================] - 40s 498ms/step - loss: 0.4193 - accuracy: 0.8055 - val_los
s: 0.3899 - val_accuracy: 0.8261
Epoch 18/50
81/81 [==============================] - 40s 496ms/step - loss: 0.4226 - accuracy: 0.7830 - val_los
s: 0.3832 - val_accuracy: 0.8276
Epoch 19/50
```

```
81/81 [==============================] - 40s 493ms/step - loss: 0.4147 - accuracy: 0.8058 - val_los
s: 0.3842 - val_accuracy: 0.8276
Epoch 20/50
81/81 [==============================] - 41s 501ms/step - loss: 0.4129 - accuracy: 0.8096 - val_los
s: 0.3787 - val_accuracy: 0.8276
Epoch 21/50
81/81 [==============================] - 40s 496ms/step - loss: 0.3959 - accuracy: 0.8106 - val_los
s: 0.3689 - val_accuracy: 0.8346
Epoch 22/50
81/81 [==============================] - 40s 500ms/step - loss: 0.4189 - accuracy: 0.8021 - val_los
s: 0.3724 - val_accuracy: 0.8315
Epoch 23/50
81/81 [==============================] - 40s 495ms/step - loss: 0.4055 - accuracy: 0.8033 - val_los
s: 0.3785 - val_accuracy: 0.8284
Epoch 24/50
81/81 [==============================] - 40s 500ms/step - loss: 0.3869 - accuracy: 0.8192 - val_los
s: 0.3640 - val_accuracy: 0.8292
Epoch 25/50
81/81 [==============================] - 40s 495ms/step - loss: 0.3761 - accuracy: 0.8286 - val_los
s: 0.3706 - val_accuracy: 0.8346
Epoch 26/50
81/81 [==============================] - 40s 492ms/step - loss: 0.3827 - accuracy: 0.8313 - val_los
s: 0.3658 - val_accuracy: 0.8362
Epoch 27/50
81/81 [==============================] - 40s 492ms/step - loss: 0.4041 - accuracy: 0.8122 - val_los
s: 0.3666 - val_accuracy: 0.8424
Epoch 28/50
81/81 [==============================] - 41s 502ms/step - loss: 0.3957 - accuracy: 0.8220 - val_los
s: 0.3565 - val_accuracy: 0.8416
Epoch 29/50
81/81 [==============================] - 41s 501ms/step - loss: 0.3940 - accuracy: 0.8238 - val_los
s: 0.3803 - val_accuracy: 0.8269
Epoch 30/50
81/81 [==============================] - 40s 490ms/step - loss: 0.3837 - accuracy: 0.8195 - val_los
s: 0.3705 - val_accuracy: 0.8377
Epoch 31/50
81/81 [==============================] - 40s 498ms/step - loss: 0.3899 - accuracy: 0.8177 - val_los
s: 0.3597 - val_accuracy: 0.8385
Epoch 32/50
81/81 [==============================] - 41s 506ms/step - loss: 0.3966 - accuracy: 0.8177 - val_los
s: 0.3592 - val_accuracy: 0.8393
Epoch 33/50
81/81 [==============================] - 40s 493ms/step - loss: 0.3782 - accuracy: 0.8214 - val_los
s: 0.3543 - val_accuracy: 0.8494
Epoch 34/50
81/81 [==============================] - 40s 495ms/step - loss: 0.3896 - accuracy: 0.8154 - val_los
s: 0.3470 - val_accuracy: 0.8463
Epoch 35/50
81/81 [==============================] - 41s 501ms/step - loss: 0.3743 - accuracy: 0.8232 - val_los
s: 0.3432 - val_accuracy: 0.8478
Epoch 36/50
81/81 [==============================] - 41s 500ms/step - loss: 0.3786 - accuracy: 0.8239 - val_los
s: 0.3402 - val_accuracy: 0.8455
Epoch 37/50
81/81 [==============================] - 40s 499ms/step - loss: 0.3554 - accuracy: 0.8404 - val_los
s: 0.3512 - val_accuracy: 0.8463
```

```
Epoch 38/50
81/81 [==============================] - 40s 496ms/step - loss: 0.3809 - accuracy: 0.8286 - val_los
s: 0.3491 - val_accuracy: 0.8540
Epoch 39/50
81/81 [==============================] - 40s 492ms/step - loss: 0.3658 - accuracy: 0.8344 - val_los
s: 0.3468 - val_accuracy: 0.8463
Epoch 40/50
81/81 [==============================] - 41s 505ms/step - loss: 0.3718 - accuracy: 0.8291 - val_los
s: 0.3395 - val_accuracy: 0.8509
Epoch 41/50
81/81 [==============================] - 40s 496ms/step - loss: 0.3642 - accuracy: 0.8309 - val_los
s: 0.3519 - val_accuracy: 0.8494
Epoch 42/50
81/81 [==============================] - 40s 489ms/step - loss: 0.3658 - accuracy: 0.8329 - val_los
s: 0.3399 - val_accuracy: 0.8447
Epoch 43/50
81/81 [==============================] - 40s 496ms/step - loss: 0.3632 - accuracy: 0.8328 - val_los
s: 0.3407 - val_accuracy: 0.8447
Epoch 44/50
81/81 [==============================] - 41s 506ms/step - loss: 0.3569 - accuracy: 0.8474 - val_los
s: 0.3459 - val_accuracy: 0.8502
Epoch 45/50
81/81 [==============================] - 40s 495ms/step - loss: 0.3489 - accuracy: 0.8388 - val_los
s: 0.3409 - val_accuracy: 0.8463
Epoch 46/50
81/81 [==============================] - 40s 495ms/step - loss: 0.3607 - accuracy: 0.8433 - val_los
s: 0.3385 - val_accuracy: 0.8494
Epoch 47/50
81/81 [==============================] - 40s 489ms/step - loss: 0.3515 - accuracy: 0.8444 - val_los
s: 0.3429 - val_accuracy: 0.8502
Epoch 48/50
81/81 [==============================] - 40s 497ms/step - loss: 0.3520 - accuracy: 0.8401 - val_los
s: 0.3361 - val_accuracy: 0.8556
Epoch 49/50
81/81 [==============================] - 40s 496ms/step - loss: 0.3475 - accuracy: 0.8469 - val_los
s: 0.3392 - val_accuracy: 0.8470
Epoch 50/50
81/81 [==============================] - 40s 495ms/step - loss: 0.3502 - accuracy: 0.8380 - val_los
s: 0.3420 - val_accuracy: 0.8564
```

7. Evaluating the Models

Now to evaluate the models I am going to use some external images of celebrities. These celebrities are of a variety of ages and genders.

In [30]:

```
def process_and_predict(file):
    im = Image.open(file)
    width, height = im.size
    if width == height:
        im = im.resize((200,200), Image.ANTIALIAS)
    else:
        if width > height:
            left = width/2 - height/2
            right = width/2 + height/2
            top = 0
            bottom = height
            im = im.crop((left,top,right,bottom))
            im = im.resize((200,200), Image.ANTIALIAS)
        else:
            left = 0
            right = width
            top = 0
            bottom = width
            im = im.crop((left,top,right,bottom))
            im = im.resize((200,200), Image.ANTIALIAS)

    ar = np.asarray(im)
    ar = ar.astype('float32')
    ar /= 255.0
    ar = ar.reshape(-1, 200, 200, 3)

    age = agemodel.predict(ar)
    gender = np.round(genmodel.predict(ar))
    if gender == 0:
        gender = 'male'
    elif gender == 1:
        gender = 'female'

    print('Age:', int(age), '\n Gender:', gender)
    return im.resize((300,300), Image.ANTIALIAS)
```

Alyson Hannigan

In [43]:

```
process_and_predict('../input/celebrities2/alyson.jpg')
```

```
Age: 36
 Gender: female
```

Out[43]:

David Boreanaz

In [46]:

```
process_and_predict('../input/celebrities2/david.jpg')
```

```
Age: 61
 Gender: male
```

Out[46]:

Gaten Matarazzo

In [48]:

```
process_and_predict('../input/celebrities2/gaten.jpg')
```

```
Age: 38
 Gender: male
```

Out[48]:

Jack Dylan Grazer

In [50]:

```
process_and_predict('../input/celebrities2/jack.jpg')
```

```
Age: 14
 Gender: female
```

Out[50]:

Jennifer Garner

In [52]:

```
process_and_predict('../input/celebrities2/jennifer.jpg')
```

```
Age: 28
 Gender: female
```

Out[52]:

Jennifer Lawrence

In [53]:

```
process_and_predict('../input/celebrities2/jenniferlaw.jpg')
```

```
Age: 47
 Gender: female
```

Out[53]:

Meryl Streep

In [57]:

```
process_and_predict('../input/celebrities2/meryl.jpg')
```

```
Age: 64
 Gender: female
```

Out[57]:

Millie Bobby Brown

In [58]:

```
process_and_predict('../input/celebrities2/millie.jpg')
```

```
Age: 13
 Gender: female
```

Out[58]:

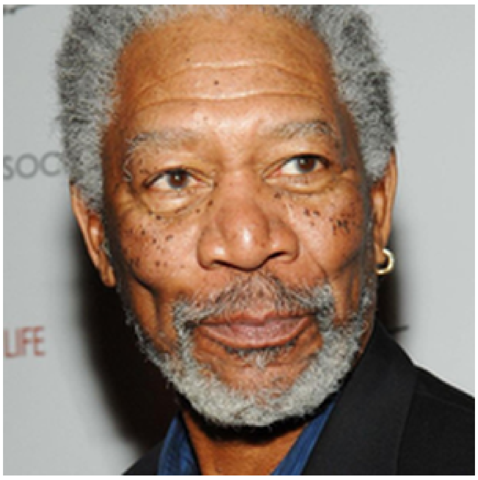
Morgan Freeman

In [59]:

```
process_and_predict('../input/celebrities2/morgan.jpg')
```

```
Age: 86
 Gender: male
```

Out[59]:



Oprah Winfrey

In [60]:

```
process_and_predict('../input/celebrities2/oprah.jpg')
```

```
Age: 42
 Gender: female
```

Out[60]:

Tom Hanks

In [63]:

```
process_and_predict('../input/celebrities2/tom.jpg')
```

```
Age: 51
 Gender: male
```

Out[63]:

Winona Ryder

In [65]:

```
process_and_predict('../input/celebrities2/winona.jpg')
```

```
Age: 26
 Gender: female
```

Out[65]: