assn3

April 14, 2022

1 Intermediate Machine Learning: Assignment 3

Deadline

Assignment 3 is due Wednesday, April 13 by 11:59pm. Late work will not be accepted as per the course policies (see the Syllabus and Course policies on Canvas).

Directly sharing answers is not okay, but discussing problems with the course staff or with other students is encouraged.

You should start early so that you have time to get help if you're stuck. The drop-in office hours schedule can be found on Canvas. You can also post questions or start discussions on Ed Discussion. The assignment may look long at first glance, but the problems are broken up into steps that should help you to make steady progress.

Submission

Submit your assignment as a pdf file on Gradescope, and as a notebook (.ipynb) on Canvas. You can access Gradescope through Canvas on the left-side of the class home page. The problems in each homework assignment are numbered. Note: When submitting on Gradescope, please select the correct pages of your pdf that correspond to each problem. This will allow graders to more easily find your complete solution to each problem.

To produce the .pdf, please do the following in order to preserve the cell structure of the notebook:

Go to "File" at the top-left of your Jupyter Notebook Under "Download as", select "HTML (.html)" After the .html has downloaded, open it and then select "File" and "Print" (note you will not actually be printing) From the print window, select the option to save as a .pdf

Topics

- Variational autoencoders
- Undirected graphs
- The graphical lasso

This assignment will also help to solidify your Python and Jupyter notebook skills.

1.1 Problem 1: Face time (35 points)

In this problem, we will implement a "shoestring" version of this amazing fake face generator, using a variational autoencoder (VAE). Building a generator like the one featured in the article can take a tremendous amount of computational resources, time, and parameter tuning. In this problem we

will build a basic version to illustrate the main concepts, and help you to become more familiar with VAEs. Here is an outline of the process that we'll step you through:

1.1.1 Problem outline:

- Load data
- Create face groups based on attributes
- Construct the VAE
- Define the loss function and train the VAE (Problem 1.1)
- Encode and reconstruct faces (Problem 1.2)
- Visualize the latent space (Problem 1.3)
- Morph between faces (Problem 1.4)
- Shift attributes of faces (Problem 1.5)
- Generate new faces (Problem 1.6)
- Analyze the effect of the scaling factor in the loss function (Problem 1.7, optional)

In the next cell we load the packages that we'll need. If you don't have one or more of these, you can install them with `!pip install <package_name>` in the cell, or outside the notebook with `conda install -c conda-forge <package_name>`

[ ]:
```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import os
import glob
import pandas as pd
import random
import numpy as np
import imageio
from tqdm import tqdm
from PIL import Image
from sklearn.model_selection import train_test_split
from scipy.stats import norm
import tensorflow
# import tensorflow.compat.v1 as tensorflow
tensorflow.compat.v1.disable_eager_execution()
```

[ ]:
```
tensorflow.test.is_gpu_available()
```

```
WARNING:tensorflow:From <ipython-input-2-1bce0b5d4087>:1: is_gpu_available (from
tensorflow.python.framework.test_util) is deprecated and will be removed in a
future version.
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
```

[ ]: True

[ ]:
```
tensorflow.__version__
```

[ ]: '2.3.0'

1.1.2 Loading the data

Labeled Faces in the Wild (LFW) is a database of face photographs. The images are placed in the folder lfw-deepfunneled. lfw_attributes.txt is a document including a set of attributes associated for each image, such as 'Male', 'Smile', 'Bold', etc. All the features are numerical and large positive values indicate that the keywords well describe the photo; large negative values indicate that the keywords don't fit the photo.

For this problem, we will keep only the middle parts of the photos to avoid complex backgrounds.

Download the data from the cloud at these URLs:

https://sds365.s3.amazonaws.com/lfw/lfw-deepfunneled.zip https://sds365.s3.amazonaws.com/lfw/lfw_attributes

Once you have the data, unzip it, and place it in a directory that we will call "YOUR_PATH" below.

Run all the cells in this section to load the data.

[ ]:
```
os.listdir('../../../../SDS665_hw3_data')
```

[ ]:
```
['lfw-deepfunneled', 'lfw_attributes.txt']
```

[ ]:
```
DATASET_PATH ="../../../../SDS665_hw3_data/lfw-deepfunneled/"
ATTRIBUTES_PATH = "../../../../SDS665_hw3_data/lfw_attributes.txt"
```

[ ]:
```
dataset = []
for path in glob.iglob(os.path.join(DATASET_PATH, "**", "*.jpg")):
    person = path.split("/")[-2]
    dataset.append({"person":person, "path": path})

dataset = pd.DataFrame(dataset)
dataset = dataset.groupby("person").filter(lambda x: len(x) < 100 )
dataset.head(10)
```

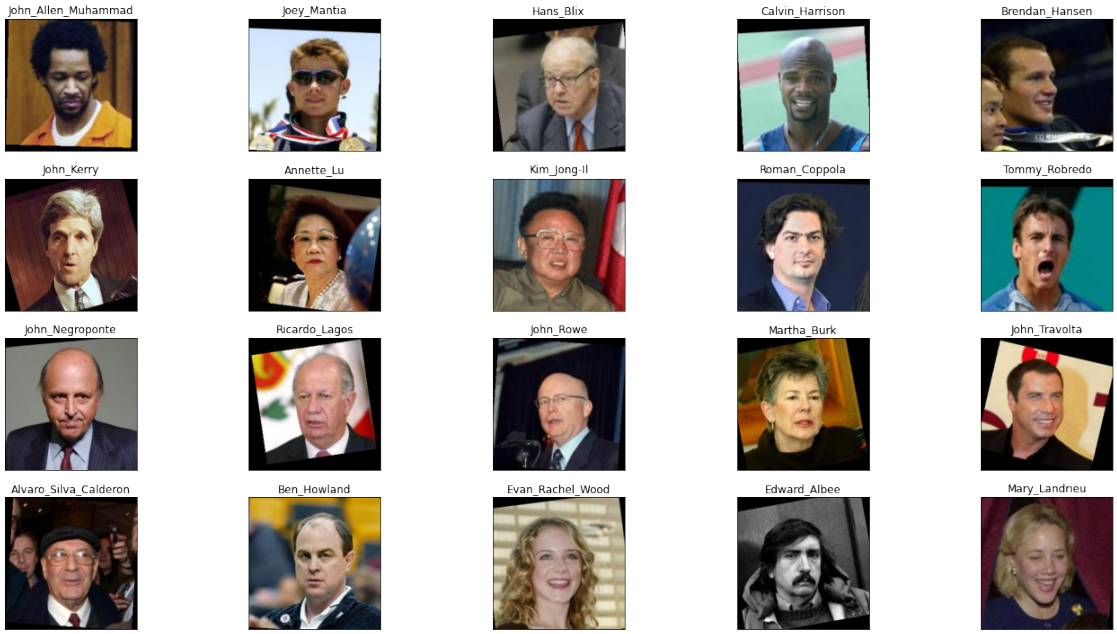
[ ]:

	person	path
0	Aaron_Eckhart	../../../../SDS665_hw3_data/lfw-deepfunneled/A…
1	Aaron_Guiel	../../../../SDS665_hw3_data/lfw-deepfunneled/A…
2	Aaron_Patterson	../../../../SDS665_hw3_data/lfw-deepfunneled/A…
3	Aaron_Peirsol	../../../../SDS665_hw3_data/lfw-deepfunneled/A…
4	Aaron_Peirsol	../../../../SDS665_hw3_data/lfw-deepfunneled/A…
5	Aaron_Peirsol	../../../../SDS665_hw3_data/lfw-deepfunneled/A…
6	Aaron_Peirsol	../../../../SDS665_hw3_data/lfw-deepfunneled/A…
7	Aaron_Pena	../../../../SDS665_hw3_data/lfw-deepfunneled/A…
8	Aaron_Sorkin	../../../../SDS665_hw3_data/lfw-deepfunneled/A…
9	Aaron_Sorkin	../../../../SDS665_hw3_data/lfw-deepfunneled/A…

The following cell will display some sample images

[ ]:
```
sampled_id = []
```

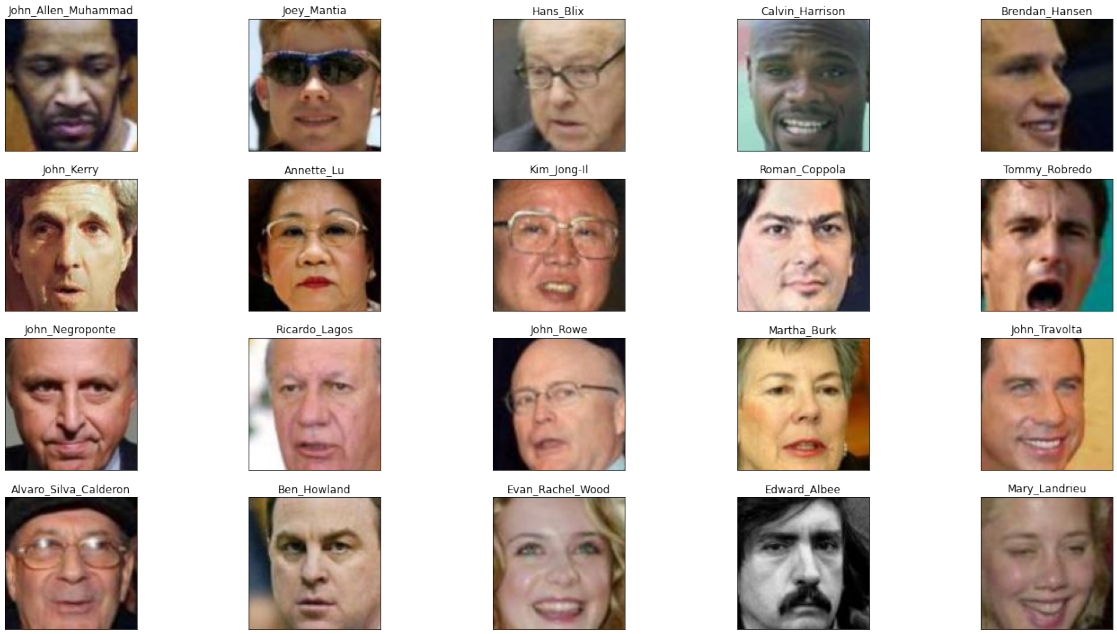
```
plt.figure(figsize=(20,10))
for i in range(20):
    idx = random.randint(0, len(dataset))
    img = plt.imread(dataset.path.iloc[idx])
    plt.subplot(4, 5, i+1)
    plt.imshow(img)
    plt.title(dataset.person.iloc[idx])
    plt.xticks([])
    plt.yticks([])
    sampled_id.append(idx)
plt.tight_layout()
plt.show()
```



The following cell shows the images with some of the background removed.

[ ]:
```
dx=70
dy=70

plt.figure(figsize=(20,10))
for i in range(20):
    idx = sampled_id[i]
    img = plt.imread(dataset.path.iloc[idx])
    plt.subplot(4, 5, i+1)
    plt.imshow(img[dy:-dy,dx:-dx])
    plt.title(dataset.person.iloc[idx])
    plt.xticks([])
```

```
    plt.yticks([])
plt.tight_layout()
plt.show()
```



The following function crops the images to 45x45 pixels, which is what we will use in this problem.

[ ]:
```
def fetch_dataset(dx=70, dy=70, dimx=45, dimy=45):

    df_attrs = pd.read_csv(ATTRIBUTES_PATH, sep='\t', skiprows=1,)
    df_attrs = pd.DataFrame(df_attrs.iloc[:,:-1].values, columns = df_attrs.
↪columns[1:])

    photo_ids = []
    for dirpath, dirnames, filenames in os.walk(DATASET_PATH):
        for fname in filenames:
            if fname.endswith(".jpg"):
                fpath = os.path.join(dirpath,fname)
                photo_id = fname[:-4].replace('_',' ').split()
                person_id = ' '.join(photo_id[:-1])
                photo_number = int(photo_id[-1])
                photo_ids.append({'person':person_id,'imagenum':
↪photo_number,'photo_path':fpath})

    photo_ids = pd.DataFrame(photo_ids)
    df = pd.merge(df_attrs,photo_ids,on=('person','imagenum'))
```

```
    assert len(df)==len(df_attrs),"lost some data when merging dataframes"

    all_photos = df['photo_path'].apply(imageio.imread)\
                                 .apply(lambda img:img[dy:-dy,dx:-dx])\
                                 .apply(lambda img: np.array(Image.
↪fromarray(img).resize([dimx,dimy])) )

    all_photos = np.stack(all_photos.values).astype('uint8')
    all_attrs = df.drop(["photo_path","person","imagenum"],axis=1)

    return all_photos,all_attrs
```

The variable `data` has all the face images and the variable `attrs` has all the attributes. The 8-bit RGB values are converted to values between 0 and 1 for modeling and plotting purposes.

[ ]:
```
data, attrs = fetch_dataset()
data = np.array(data / 255, dtype='float32')
```

1.1.3 Create Face Groups

We can now create groups of faces, by selecting the faces having the highest or lowest scores for each of the attributes. Run all the cells in this section to create and plot some face groups.

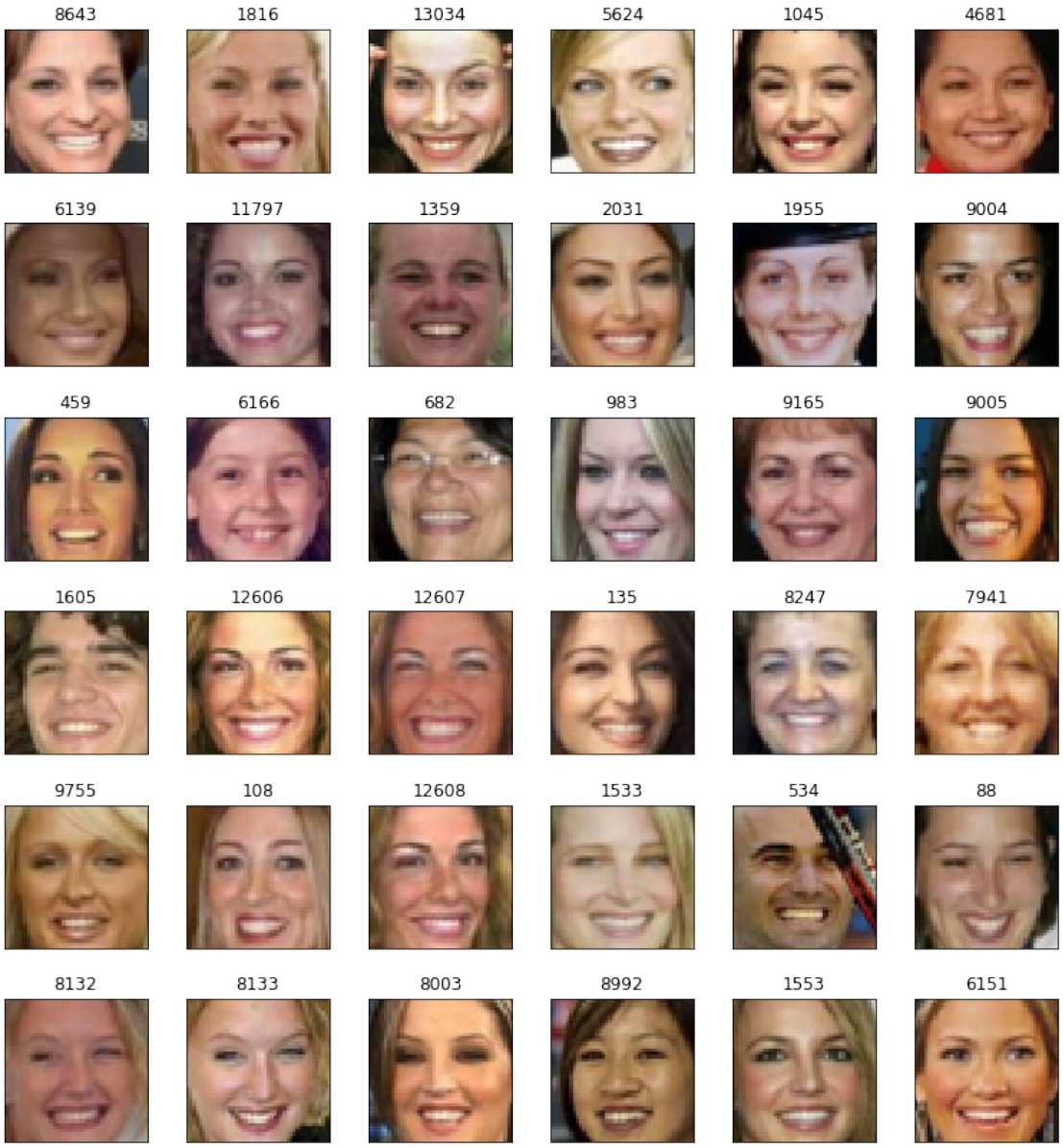
[ ]:
```
def plot_gallery(images, h, w, n_row=3, n_col=6, with_title=False, titles=[]):
    plt.figure(figsize=(1.75 * n_col, 2 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        try:
            plt.imshow(images[i].reshape((h, w, 3)), cmap=plt.cm.gray, vmin=-1,␣
↪vmax=1, interpolation='nearest')
            if with_title:
                plt.title(titles[i])
            plt.xticks(())
            plt.yticks(())
        except:
            pass
```

[ ]:
```
IMAGE_H = data.shape[1]
IMAGE_W = data.shape[2]
N_CHANNELS = 3
```

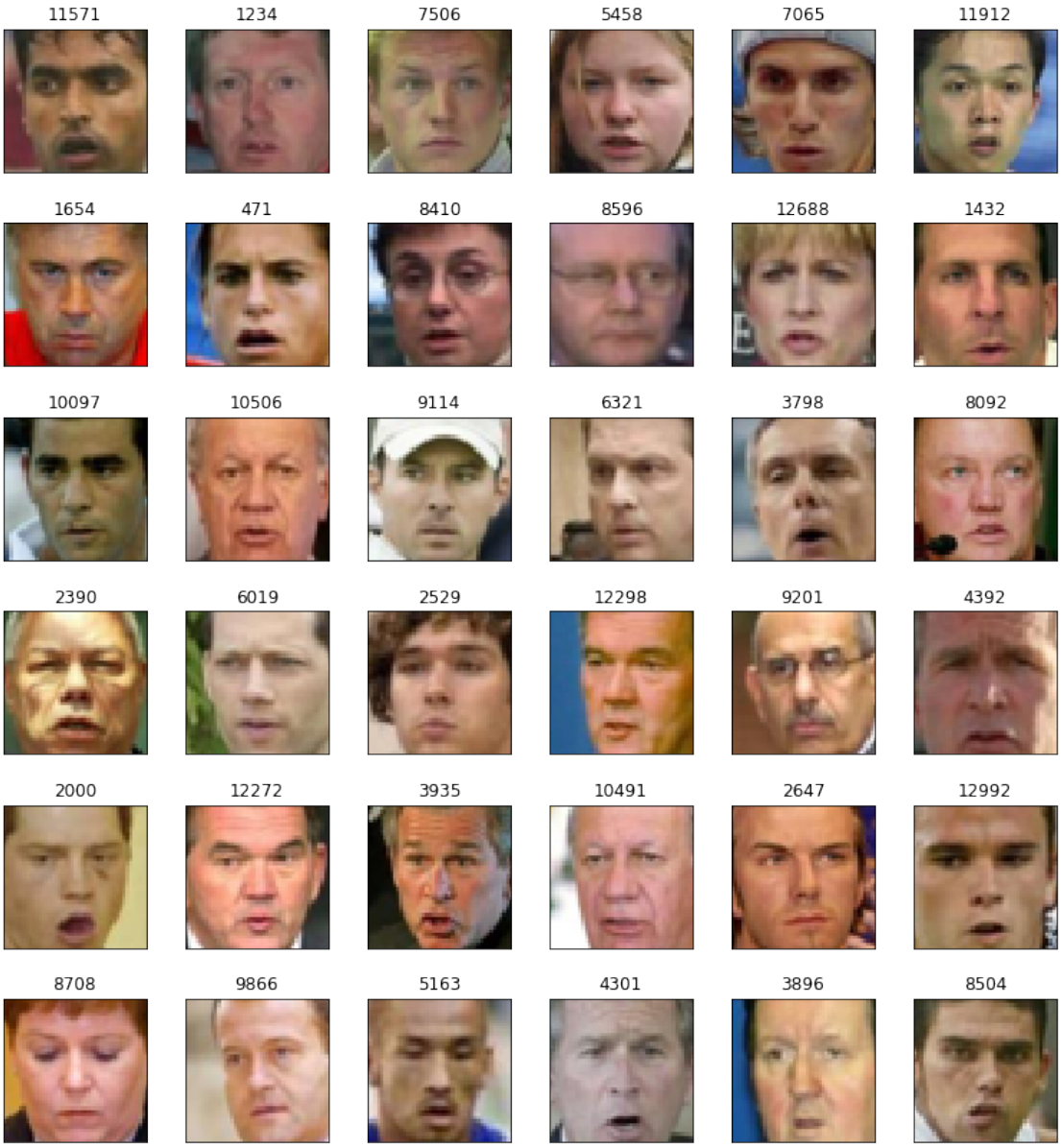
[ ]:
```
smile_ids = attrs['Smiling'].sort_values(ascending=False).head(36).index.values
smile_data = data[smile_ids]

no_smile_ids = attrs['Smiling'].sort_values(ascending=True).head(36).index.
↪values
```

```
no_smile_data = data[no_smile_ids]

eyeglasses_ids = attrs['Eyeglasses'].sort_values(ascending=False).head(36).
 ↪index.values
eyeglasses_data = data[eyeglasses_ids]

sunglasses_ids = attrs['Sunglasses'].sort_values(ascending=False).head(36).
 ↪index.values
sunglasses_data = data[sunglasses_ids]

mustache_ids = attrs['Mustache'].sort_values(ascending=False).head(36).index.
 ↪values
mustache_data = data[mustache_ids]

male_ids = attrs['Male'].sort_values(ascending=False).head(36).index.values
male_data = data[male_ids]

female_ids = attrs['Male'].sort_values(ascending=True).head(36).index.values
female_data = data[female_ids]

eyeclosed_ids = attrs['Eyes Open'].sort_values(ascending=True).head(36).index.
 ↪values
eyeclosed_data = data[eyeclosed_ids]

mouthopen_ids = attrs['Mouth Wide Open'].sort_values(ascending=False).head(36).
 ↪index.values
mouthopen_data = data[mouthopen_ids]

makeup_ids = attrs['Heavy Makeup'].sort_values(ascending=False).head(36).index.
 ↪values
makeup_data = data[makeup_ids]
```

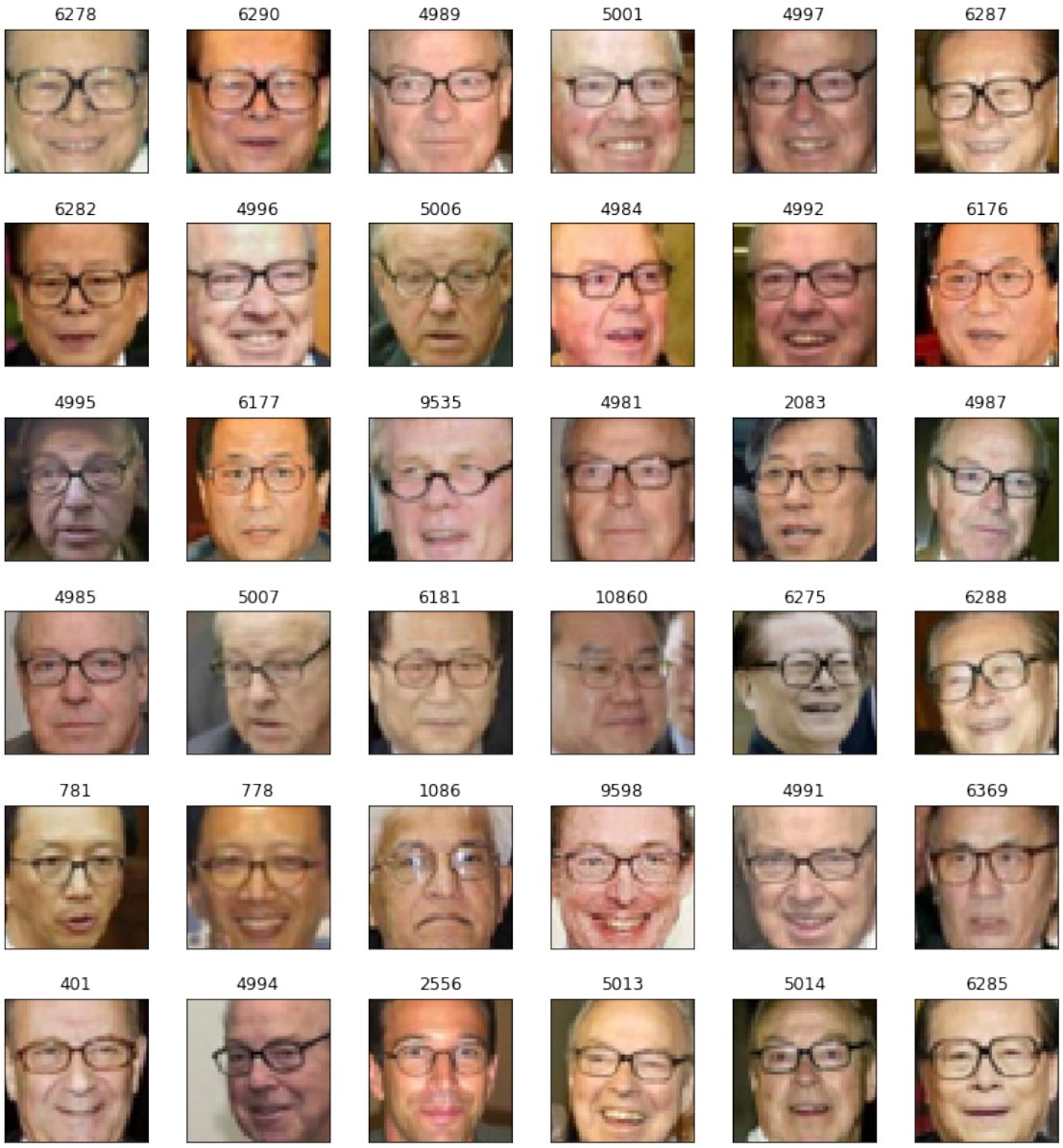
[ ]:
```
plot_gallery(smile_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True,␣
 ↪titles=smile_ids)
```



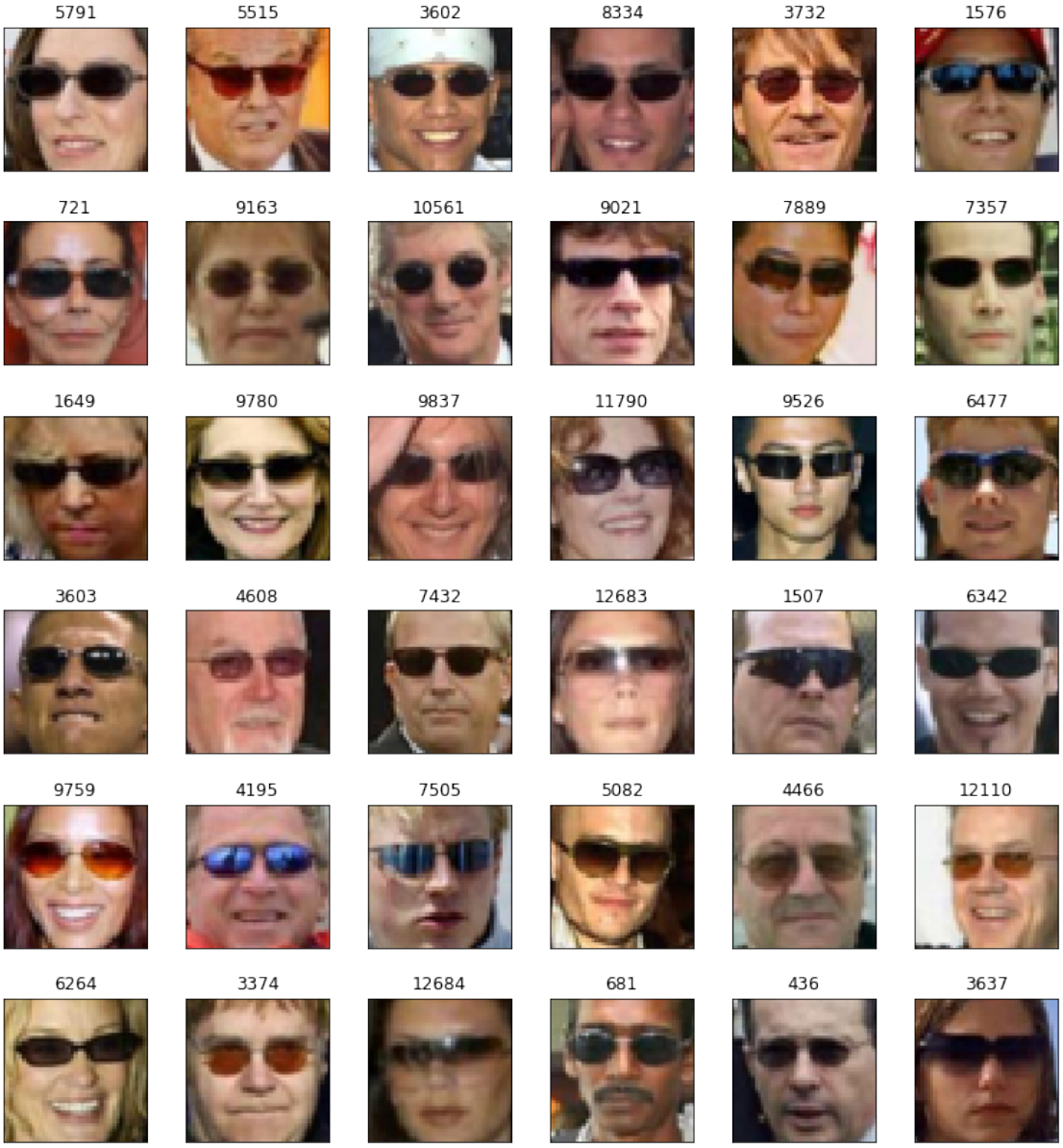
[ ]:
```
plot_gallery(no_smile_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6,␣
↪with_title=True, titles=no_smile_ids)
```



[ ]:
```
plot_gallery(eyeglasses_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6,␣
↪with_title=True, titles=eyeglasses_ids)
```

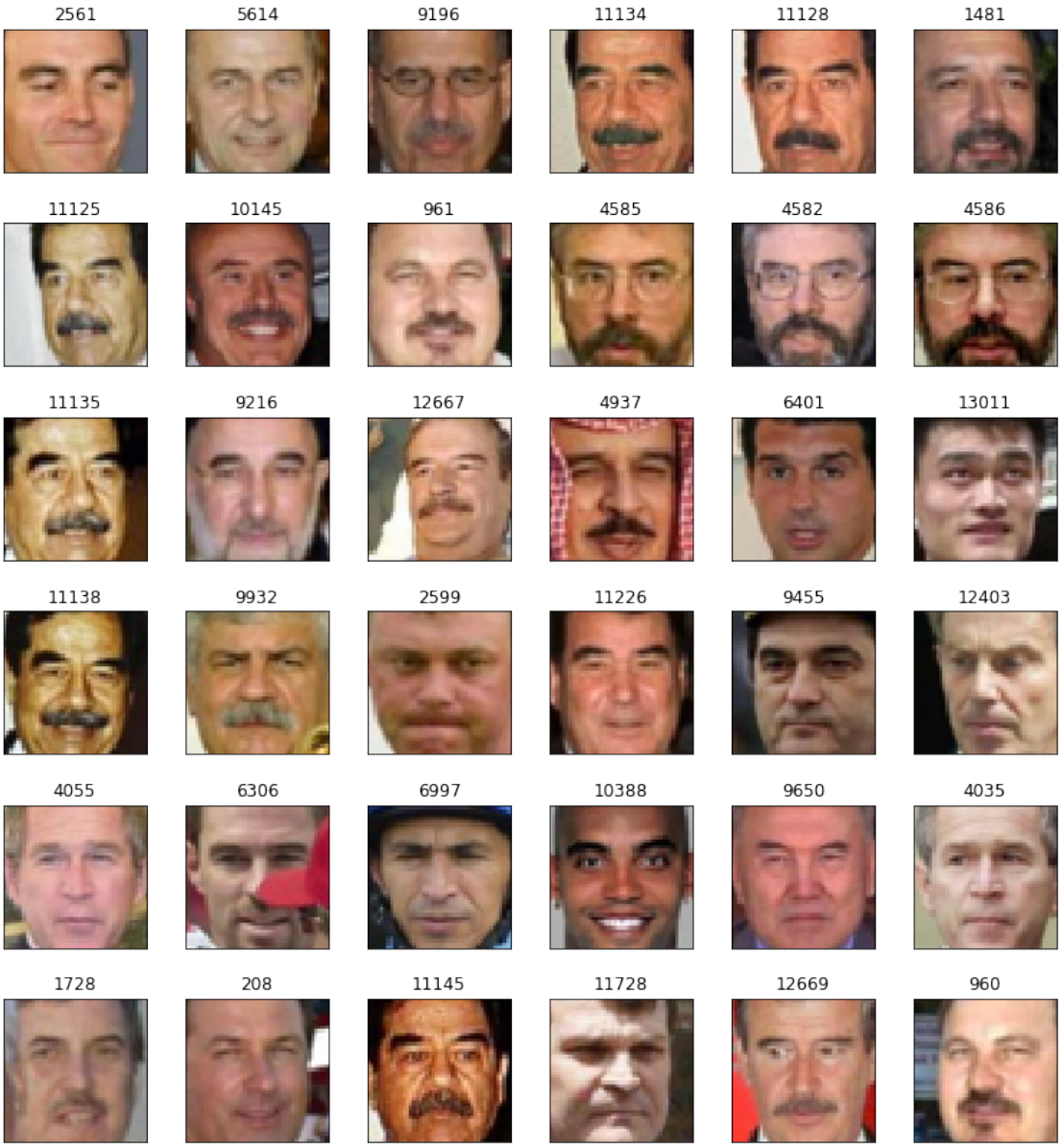


[ ]:
```
plot_gallery(sunglasses_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6,␣
↪with_title=True, titles=sunglasses_ids)
```

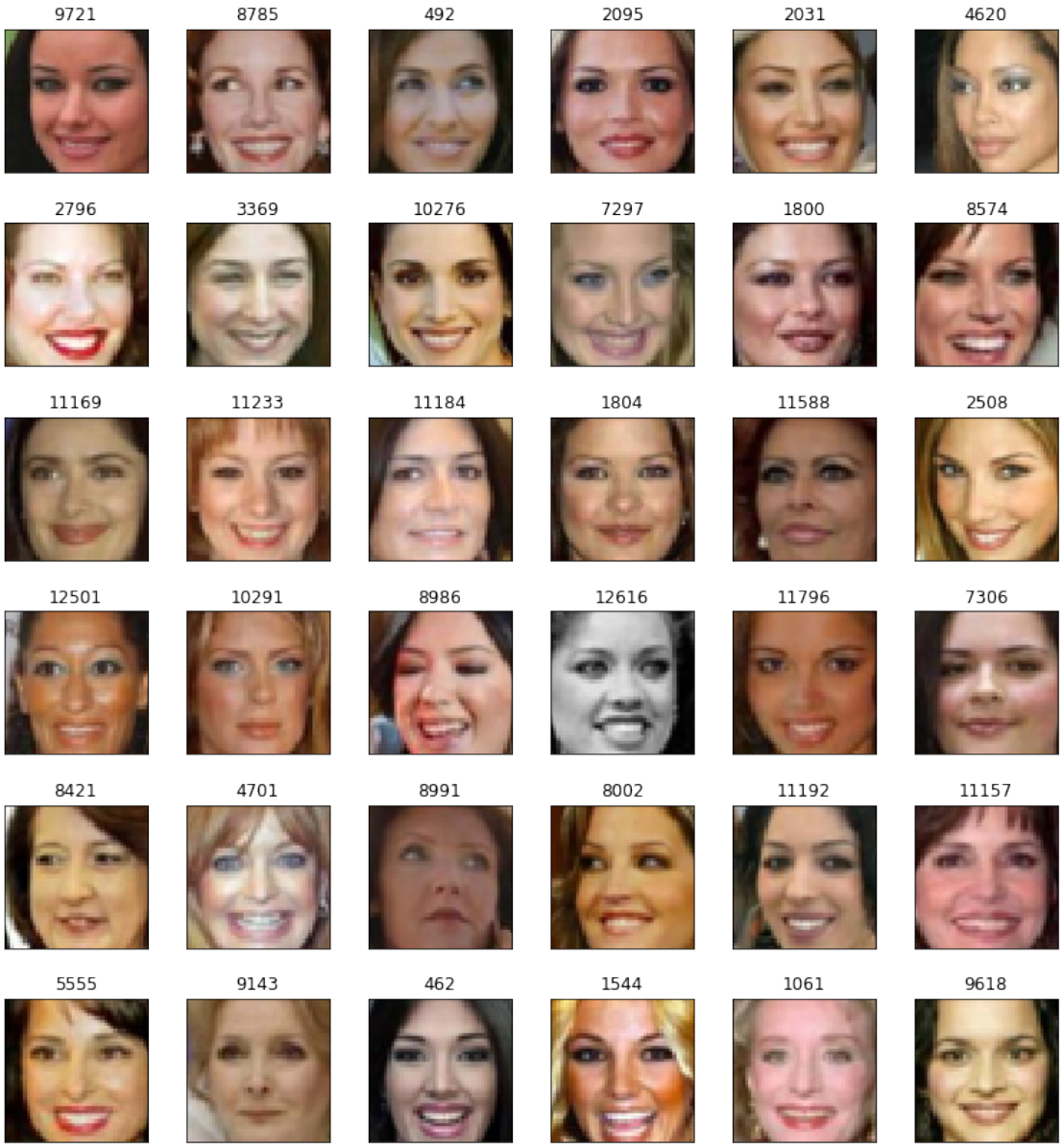


[ ]:
```
plot_gallery(mustache_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6,␣
↪with_title=True, titles=mustache_ids)
```

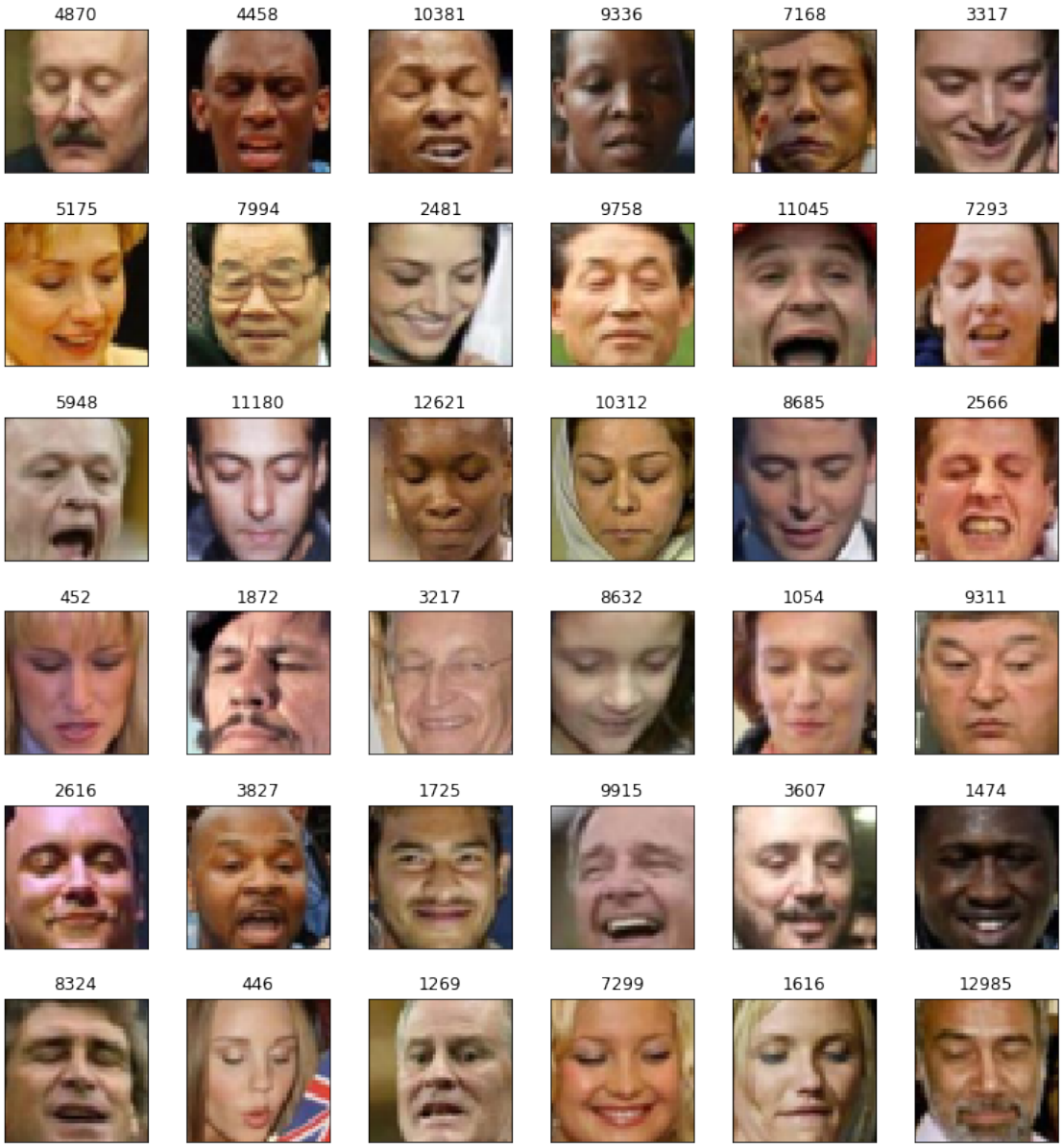
[ ]:
```
plot_gallery(male_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True,␣
↪titles=male_ids)
```



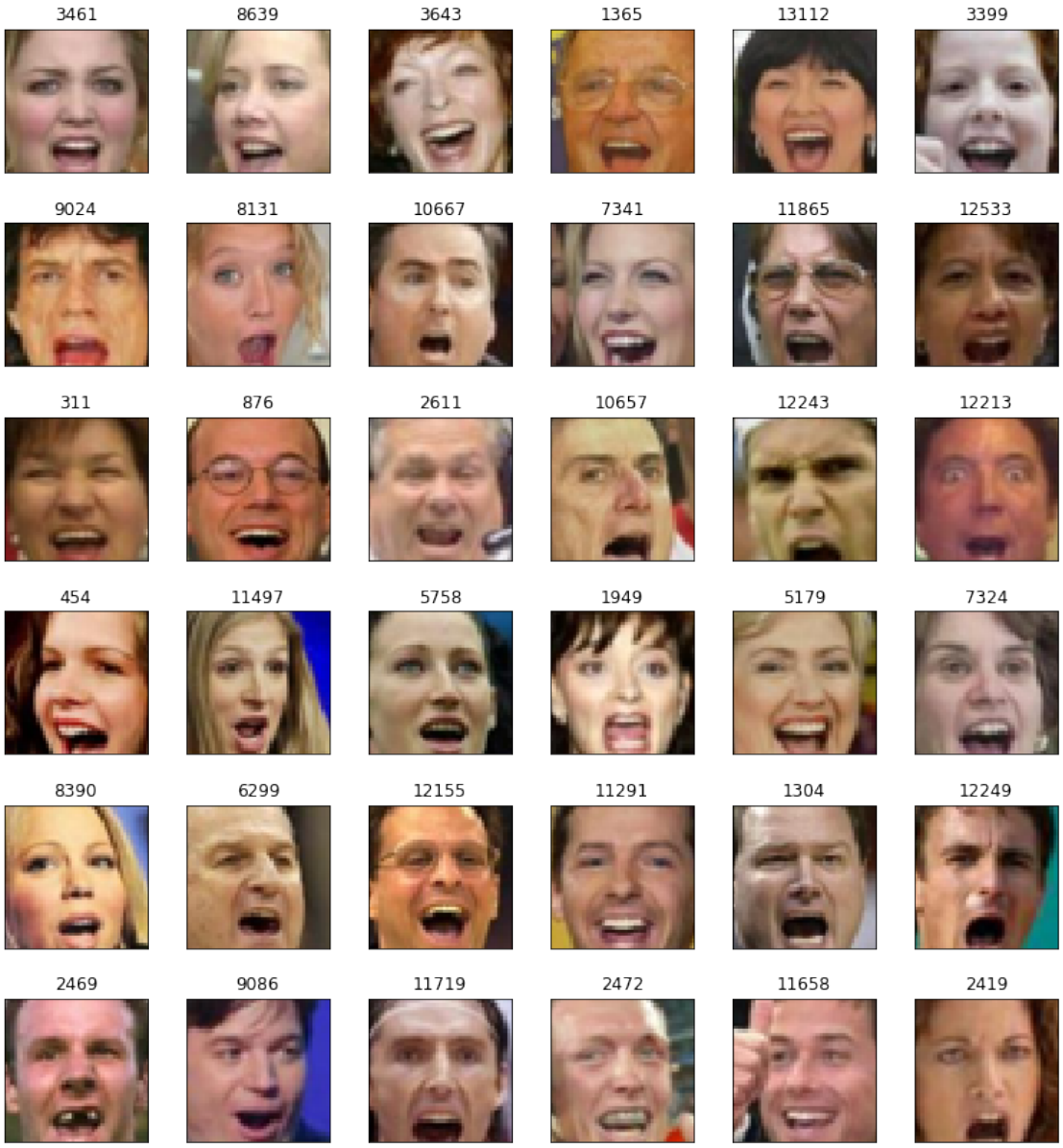
[ ]:
```
plot_gallery(female_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True,␣
↪titles=female_ids)
```



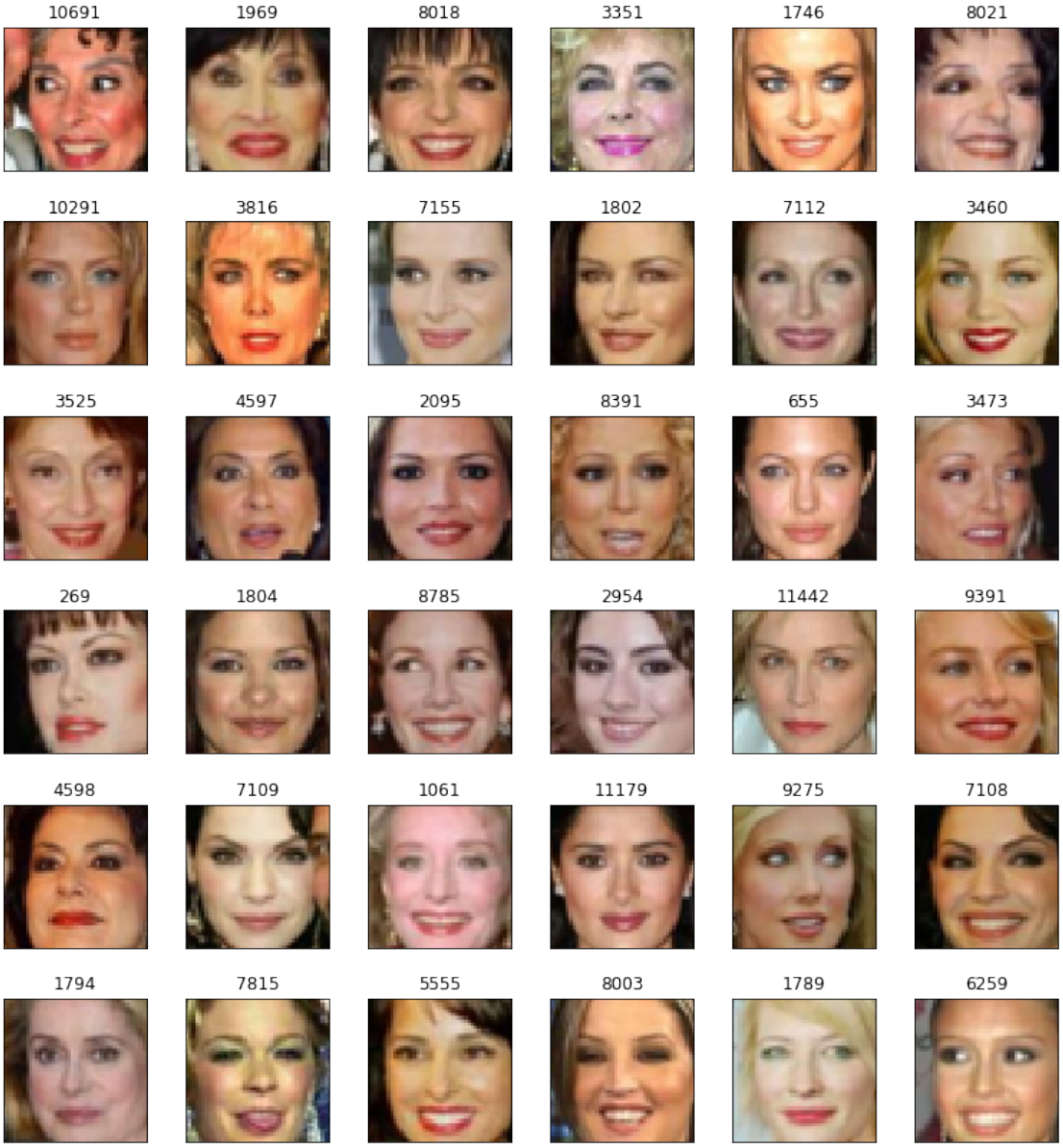
[ ]:
```
plot_gallery(eyeclosed_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6,␣
↪with_title=True, titles=eyeclosed_ids)
```



[ ]:
```
plot_gallery(mouthopen_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6,␣
↪with_title=True, titles=mouthopen_ids)
```



[ ]:
```
plot_gallery(makeup_data, IMAGE_H, IMAGE_W, n_row=6, n_col=6, with_title=True,␣
↪titles=makeup_ids)
```



1.1.4 Constructing the encoder

Recall that the encoder part of the VAE architecture maps a data point to a variational mean and (log) variance. The mean is a point in the latent space.

[ ]:
```
LATENT_SPACE_SIZE = 100
```

The "reparameterization trick" draws samples from the variational distribution that are parameterized by the variational mean and variance, so that the parameters of the encoder network can be trained.

[ ]:
```
def sample_latent_features(distribution):
    distribution_mean, distribution_variance = distribution
    batch_size = tensorflow.shape(distribution_variance)[0]
    random = tensorflow.keras.backend.random_normal(shape=(batch_size,␣
 ↪tensorflow.shape(distribution_variance)[1]))
    return distribution_mean + tensorflow.exp(0.5 * distribution_variance) *␣
 ↪random
```

[ ]:
```
input_data = tensorflow.keras.layers.Input(shape=(45, 45, 3))

encoder = tensorflow.keras.layers.Conv2D(64, (5,5),␣
 ↪activation='relu')(input_data)
encoder = tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)

encoder = tensorflow.keras.layers.Conv2D(64, (3,3), activation='relu')(encoder)
encoder = tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)

encoder = tensorflow.keras.layers.Conv2D(32, (3,3), activation='relu')(encoder)
encoder = tensorflow.keras.layers.MaxPooling2D((2,2))(encoder)

encoder = tensorflow.keras.layers.Flatten()(encoder)

distribution_mean = tensorflow.keras.layers.Dense(LATENT_SPACE_SIZE,␣
 ↪name='variational_mean')(encoder)
distribution_variance = tensorflow.keras.layers.Dense(LATENT_SPACE_SIZE,␣
 ↪name='variational_log_variance')(encoder)
latent_encoding = tensorflow.keras.layers.
 ↪Lambda(sample_latent_features)([distribution_mean, distribution_variance])

encoder_model = tensorflow.keras.Model(input_data, latent_encoding)
encoder_model.summary()
```

```
Model: "functional_1"
__________________________________________________________________________________
__________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
==================
input_1 (InputLayer)            [(None, 45, 45, 3)]  0
__________________________________________________________________________________
__________________
conv2d (Conv2D)                 (None, 41, 41, 64)   4864        input_1[0][0]
__________________________________________________________________________________
__________________
max_pooling2d (MaxPooling2D)    (None, 20, 20, 64)   0           conv2d[0][0]
__________________________________________________________________________________
__________________
```

```
conv2d_1 (Conv2D)               (None, 18, 18, 64)   36928
max_pooling2d[0][0]
__________________________________________________________________________________
__________________
max_pooling2d_1 (MaxPooling2D)  (None, 9, 9, 64)     0           conv2d_1[0][0]
__________________________________________________________________________________
__________________
conv2d_2 (Conv2D)               (None, 7, 7, 32)     18464
max_pooling2d_1[0][0]
__________________________________________________________________________________
__________________
max_pooling2d_2 (MaxPooling2D)  (None, 3, 3, 32)     0           conv2d_2[0][0]
__________________________________________________________________________________
__________________
flatten (Flatten)               (None, 288)          0
max_pooling2d_2[0][0]
__________________________________________________________________________________
__________________
variational_mean (Dense)        (None, 100)          28900       flatten[0][0]
__________________________________________________________________________________
__________________
variational_log_variance (Dense (None, 100)          28900       flatten[0][0]
__________________________________________________________________________________
__________________
lambda (Lambda)                 (None, 100)          0
variational_mean[0][0]
variational_log_variance[0][0]
==================================================================================
==================
Total params: 118,056
Trainable params: 118,056
Non-trainable params: 0
__________________________________________________________________________________
__________________
```

1.1.5 Construct the decoder

The decoder network in the VAE architecture maps a latent vector to an image. This is done with a series of transposed convolutional layers, since it must map from low to high dimensions.

[ ]:
```
decoder_input = tensorflow.keras.layers.Input(shape=LATENT_SPACE_SIZE)
decoder = tensorflow.keras.layers.Reshape((1, 1, 100))(decoder_input)
decoder = tensorflow.keras.layers.Conv2DTranspose(64, (3,3),␣
 ↪activation='relu')(decoder)
decoder = tensorflow.keras.layers.UpSampling2D((2,2))(decoder)

decoder = tensorflow.keras.layers.Conv2DTranspose(32, (3,3),␣
 ↪activation='relu')(decoder)
```

```
decoder = tensorflow.keras.layers.UpSampling2D((2,2))(decoder)

decoder = tensorflow.keras.layers.Conv2DTranspose(16, (5,5),␣
↪activation='relu')(decoder)
decoder = tensorflow.keras.layers.UpSampling2D((2,2))(decoder)

decoder_output = tensorflow.keras.layers.Conv2DTranspose(3, (6,6),␣
↪activation='relu')(decoder)

decoder_model = tensorflow.keras.Model(decoder_input, decoder_output)
decoder_model.summary()
```

```
Model: "functional_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 100)]             0
_________________________________________________________________
reshape (Reshape)            (None, 1, 1, 100)         0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 3, 3, 64)          57664
_________________________________________________________________
up_sampling2d (UpSampling2D) (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 8, 8, 32)          18464
_________________________________________________________________
up_sampling2d_1 (UpSampling2 (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_transpose_2 (Conv2DTr (None, 20, 20, 16)        12816
_________________________________________________________________
up_sampling2d_2 (UpSampling2 (None, 40, 40, 16)        0
_________________________________________________________________
conv2d_transpose_3 (Conv2DTr (None, 45, 45, 3)         1731
=================================================================
Total params: 90,675
Trainable params: 90,675
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
encoded = encoder_model(input_data)
decoded = decoder_model(encoded)
autoencoder = tensorflow.keras.models.Model(input_data, decoded)
```

[ ]:
```
def get_loss(distribution_mean, distribution_variance, factor, batch_size):

    def get_reconstruction_loss(y_true, y_pred, factor, batch_size):
        reconstruction_loss = tensorflow.math.squared_difference(y_true, y_pred)
```

```
        reconstruction_loss_batch = tensorflow.reduce_sum(reconstruction_loss)/
↪batch_size
        return 0.5*reconstruction_loss_batch*factor

    def get_kl_loss(distribution_mean, distribution_variance, batch_size):
        kl_loss = LATENT_SPACE_SIZE + distribution_variance - tensorflow.
↪square(distribution_mean) - tensorflow.exp(distribution_variance)
        kl_loss_batch = tensorflow.reduce_sum(kl_loss)/batch_size
        return kl_loss_batch*(-0.5)

    def total_loss(y_true, y_pred):
        reconstruction_loss_batch = get_reconstruction_loss(y_true, y_pred,␣
↪factor, batch_size)
        kl_loss_batch = get_kl_loss(distribution_mean, distribution_variance,␣
↪batch_size)
        return reconstruction_loss_batch + kl_loss_batch

    return total_loss
```

1.1.6 1.1 Deriving the loss function (5 points)

Derive the loss function defined in the cell above from the probability model perspective. You can ignore the scalar `factor` in your derivation. Show your work using either LaTeX or a picture of your written solution.

Hint: Think about how how the total loss is related to the ELBO.

Answer: We minimize the negative of ELBO objective as

$$\mathcal{L} = -\mathbb{E}_q(\log p(x,z)) - H(q) \tag{1}$$
$$= -\mathbb{E}_q(\log p(x|z)) - \mathbb{E}_q(\log p(z)) - H(q) \tag{2}$$
$$= \frac{1}{2N}\sum \|x - D(E(x))\|^2 + \frac{1}{2N}\sum \left(\|\mu(x)\|^2 + k\sigma^2(x)\right) - \frac{1}{2N}\sum \log \sigma^2 \tag{3}$$

The following three cells train the model. You can just run them. It may take a while to run on your laptop.

[ ]:
```
X_train, X_val = train_test_split(data, test_size=0.2, random_state=365)
```

[ ]:
```
# Load Pre-trained Model

# batch_size = 64
# autoencoder = tensorflow.keras.models.load_model("Trained_VAE",compile=False)
# autoencoder.compile(loss=get_loss(distribution_mean, distribution_variance,␣
 ↪factor = 100, batch_size = batch_size), optimizer='Adam')

# encoder_model = tensorflow.keras.models.
 ↪load_model("Trained_encoder",compile=False)
```

```
# encoder_model.compile(loss=get_loss(distribution_mean, distribution_variance,␣
 ↪factor = 100, batch_size = batch_size), optimizer='Adam')

# decoder_model = tensorflow.keras.models.
 ↪load_model("Trained_decoder",compile=False)
# decoder_model.compile(loss=get_loss(distribution_mean, distribution_variance,␣
 ↪factor = 100, batch_size = batch_size), optimizer='Adam')
```

[ ]:
```
batch_size = 64
autoencoder.compile(loss=get_loss(distribution_mean, distribution_variance,␣
 ↪factor = 100, batch_size = batch_size), optimizer='adam')
autoencoder.summary()
```

```
Model: "functional_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 45, 45, 3)]       0
_________________________________________________________________
functional_1 (Functional)    (None, 100)               118056
_________________________________________________________________
functional_3 (Functional)    (None, 45, 45, 3)         90675
=================================================================
Total params: 208,731
Trainable params: 208,731
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```
autoencoder.fit(X_train, X_train, epochs=50, batch_size=64,␣
 ↪validation_data=(X_val, X_val))
```

```
Train on 10514 samples, validate on 2629 samples
Epoch 1/50
10514/10514 [==============================] - 4s 358us/sample - loss: -577.7425
- val_loss: -978.1145
Epoch 2/50
10514/10514 [==============================] - 3s 321us/sample - loss:
-1116.0544 - val_loss: -1090.4219
Epoch 3/50
10514/10514 [==============================] - 3s 329us/sample - loss:
-1298.9212 - val_loss: -1316.9624
Epoch 4/50
10514/10514 [==============================] - 4s 340us/sample - loss:
-1435.6054 - val_loss: -1400.6986
Epoch 5/50
10514/10514 [==============================] - 4s 349us/sample - loss:
-1512.3423 - val_loss: -1543.3555
```

```
Epoch 6/50
10514/10514 [==============================] - 4s 383us/sample - loss:
-1607.2574 - val_loss: -1580.6224
Epoch 7/50
10514/10514 [==============================] - 4s 349us/sample - loss:
-1656.2354 - val_loss: -1499.1719
Epoch 8/50
10514/10514 [==============================] - 4s 374us/sample - loss:
-1713.2019 - val_loss: -1692.2637
Epoch 9/50
10514/10514 [==============================] - 4s 337us/sample - loss:
-1757.9032 - val_loss: -1606.7794
Epoch 10/50
10514/10514 [==============================] - 4s 349us/sample - loss:
-1797.1116 - val_loss: -1710.5469
Epoch 11/50
10514/10514 [==============================] - 4s 362us/sample - loss:
-1835.2575 - val_loss: -1659.4226
Epoch 12/50
10514/10514 [==============================] - 4s 339us/sample - loss:
-1873.8050 - val_loss: -1736.2345
Epoch 13/50
10514/10514 [==============================] - 4s 346us/sample - loss:
-1887.2130 - val_loss: -1324.3526
Epoch 14/50
10514/10514 [==============================] - 4s 361us/sample - loss:
-1913.2643 - val_loss: -1858.6180
Epoch 15/50
10514/10514 [==============================] - 3s 316us/sample - loss:
-1947.9526 - val_loss: -1864.0170
Epoch 16/50
10514/10514 [==============================] - 4s 354us/sample - loss:
-1977.5430 - val_loss: -1858.4917
Epoch 17/50
10514/10514 [==============================] - 4s 353us/sample - loss:
-1996.3518 - val_loss: -1907.9511
Epoch 18/50
10514/10514 [==============================] - 4s 366us/sample - loss:
-2023.0136 - val_loss: -1939.4429
Epoch 19/50
10514/10514 [==============================] - 4s 352us/sample - loss:
-2033.2859 - val_loss: -1954.6146
Epoch 20/50
10514/10514 [==============================] - 4s 366us/sample - loss:
-2054.1722 - val_loss: -1595.0093
Epoch 21/50
10514/10514 [==============================] - 3s 315us/sample - loss:
-2047.0134 - val_loss: -1956.2031
```

```
Epoch 22/50
10514/10514 [==============================] - 3s 326us/sample - loss:
-2089.2946 - val_loss: -1972.3813
Epoch 23/50
10514/10514 [==============================] - 3s 310us/sample - loss:
-2094.3032 - val_loss: -1808.6691
Epoch 24/50
10514/10514 [==============================] - 4s 355us/sample - loss:
-2109.9509 - val_loss: -1948.1580
Epoch 25/50
10514/10514 [==============================] - 4s 372us/sample - loss:
-2129.3887 - val_loss: -1914.3071
Epoch 26/50
10514/10514 [==============================] - 4s 368us/sample - loss:
-2137.3285 - val_loss: -1998.1705
Epoch 27/50
10514/10514 [==============================] - 4s 380us/sample - loss:
-2151.8216 - val_loss: -1928.8757
Epoch 28/50
10514/10514 [==============================] - 3s 303us/sample - loss:
-2137.2115 - val_loss: -2001.0697
Epoch 29/50
10514/10514 [==============================] - 4s 356us/sample - loss:
-2178.1789 - val_loss: -2046.1749
Epoch 30/50
10514/10514 [==============================] - 4s 349us/sample - loss:
-2181.1192 - val_loss: -1971.2262
Epoch 31/50
10514/10514 [==============================] - 3s 313us/sample - loss:
-2188.0877 - val_loss: -2018.3903
Epoch 32/50
10514/10514 [==============================] - 4s 366us/sample - loss:
-2201.7059 - val_loss: -2055.7189
Epoch 33/50
10514/10514 [==============================] - 4s 378us/sample - loss:
-2212.0881 - val_loss: -1991.5567
Epoch 34/50
10514/10514 [==============================] - 4s 350us/sample - loss:
-2223.0187 - val_loss: -1989.0054
Epoch 35/50
10514/10514 [==============================] - 4s 354us/sample - loss:
-2219.9205 - val_loss: -1911.7862
Epoch 36/50
10514/10514 [==============================] - 4s 357us/sample - loss:
-2235.3193 - val_loss: -2031.8250
Epoch 37/50
10514/10514 [==============================] - 4s 355us/sample - loss:
-2240.0128 - val_loss: -2030.2374
```

```
Epoch 38/50
10514/10514 [==============================] - 4s 339us/sample - loss:
-2254.2073 - val_loss: -2065.4446
Epoch 39/50
10514/10514 [==============================] - 3s 326us/sample - loss:
-2246.6947 - val_loss: -1797.5419
Epoch 40/50
10514/10514 [==============================] - 4s 350us/sample - loss:
-2241.7753 - val_loss: -2011.9969
Epoch 41/50
10514/10514 [==============================] - 4s 369us/sample - loss:
-2277.8831 - val_loss: -1927.4592
Epoch 42/50
10514/10514 [==============================] - 4s 357us/sample - loss:
-2265.7984 - val_loss: -2046.6141
Epoch 43/50
10514/10514 [==============================] - 4s 374us/sample - loss:
-2293.4898 - val_loss: -2046.8548
Epoch 44/50
10514/10514 [==============================] - 4s 349us/sample - loss:
-2279.7736 - val_loss: -2044.5011
Epoch 45/50
10514/10514 [==============================] - 4s 365us/sample - loss:
-2294.7334 - val_loss: -2104.7225
Epoch 46/50
10514/10514 [==============================] - 4s 357us/sample - loss:
-2307.3370 - val_loss: -2035.6887
Epoch 47/50
10514/10514 [==============================] - 4s 379us/sample - loss:
-2307.8927 - val_loss: -2030.9644
Epoch 48/50
10514/10514 [==============================] - 4s 358us/sample - loss:
-2320.0737 - val_loss: -2063.1192
Epoch 49/50
10514/10514 [==============================] - 4s 373us/sample - loss:
-2325.2248 - val_loss: -2074.4995
Epoch 50/50
10514/10514 [==============================] - 4s 356us/sample - loss:
-2332.9787 - val_loss: -2067.2671
```

[ ]:
```
<tensorflow.python.keras.callbacks.History at 0x7fad0069fcf8>
```

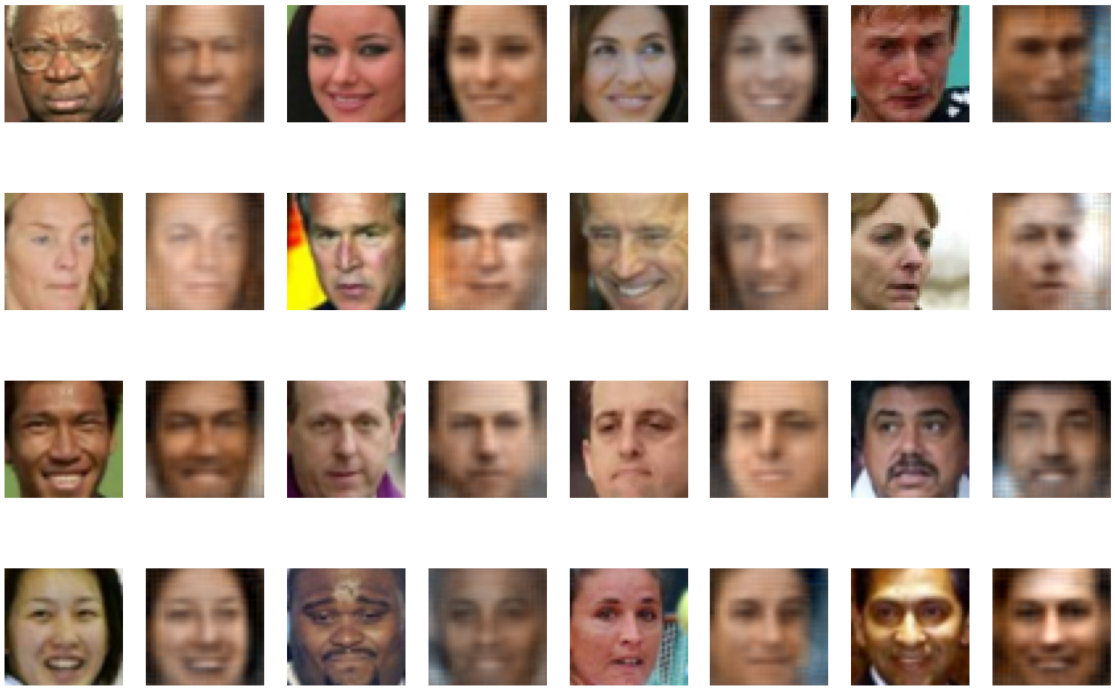
[ ]:
```
# Save your trained model

autoencoder.save("My_Trained_VAE")
encoder_model.save("My_Trained_encoder")
decoder_model.save("My_Trained_decoder")
```

```
INFO:tensorflow:Assets written to: My_Trained_VAE/assets
INFO:tensorflow:Assets written to: My_Trained_encoder/assets
INFO:tensorflow:Assets written to: My_Trained_decoder/assets
```

1.1.7 1.2 Reconstructing faces (3 points)

The following cell encodes and reconstructs 16 random faces from the validation set with the trained VAE. Run the cell and comment on the reconstructed faces. (3 points) * Do the reconstructed faces resemble the original images? How are they similar/different? * Are there any faces that are reconstructed better or worse than the others? Can you think of why? * Comment on any other aspects of your findings that are interesting to you.

[ ]:
```
sample_index = random.sample(range(1, len(X_val)), 16)

fig, axs = plt.subplots(4, 8)
fig.set_figheight(10)
fig.set_figwidth(15)

for i in range(4):
    for j in range(4):
        axs[i, 2*j].imshow(X_val[sample_index[4*i+j], :, :, :])
        axs[i, 2*j].axis('off')
        axs[i, 2*j+1].imshow(np.clip(autoencoder.predict(np.
 ↪array([X_val[sample_index[4*i+j], :, :, :]]))[0],0,1))
        axs[i, 2*j+1].axis('off')
```



Answer: 1. Yes, the reconstructed faces resemble the original images. The reconstructed share some similar textures with the original images (e.g. smiling, race). They are different in terms of resolution and detailed features. The reconstructed images have quite a few artifacts and look quite unreal. 2. The images where people seem to be sad are worse than the others, which might due to that the training images are more biased towards happy faces (size is larger). 3. I notice that resolution of the reconstructed images seems to be worse than the original images, which might due to that the size of the latent variables are too small resulting in losing too much information.

1.1.8 1.3 Visualizing the latent space (10 points)

In `vae_demo` from class, the MNIST digits were generated from a two-dimensional latent space. In the current model, the latent space has more than two dimensions, so to visualize it we need to use a dimensionality reduction technique. (If you are not familiar with PCA, please refer to the material for Week 7 of iML.)

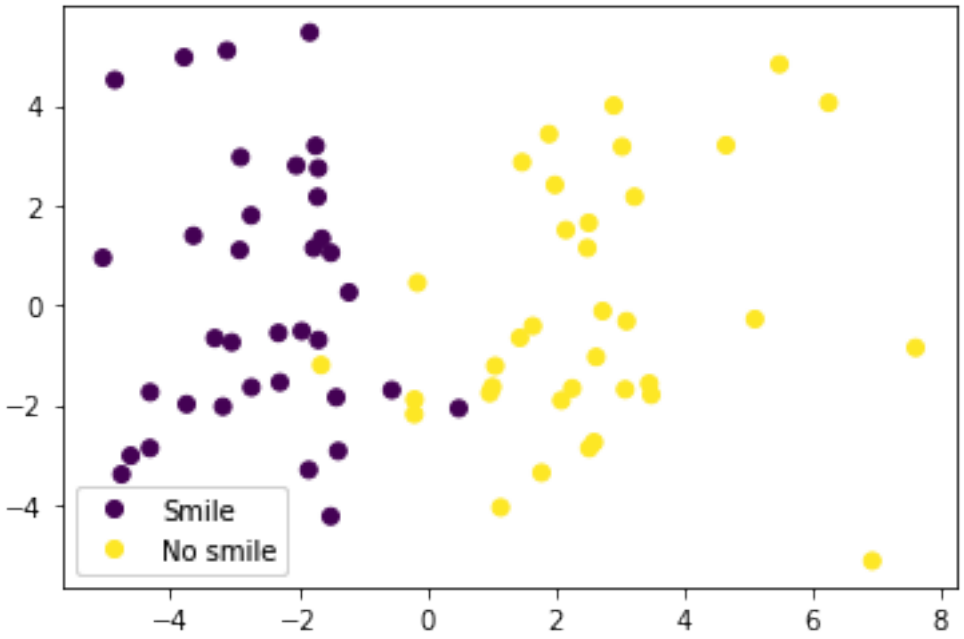
In this problem, you will first implement the function `LatentSpace_2D`. (6 points) 1. Calculate the latent space encodings for two sets of faces that are different in one attribute, e.g. smile vs. no smile. 2. Use PCA to reduce the dimension of the latent space codes to two. 3. Visualize the latent space after dimensionality reduction with a scatter plot. Clearly color-code and label the two different groups.

Here is an example using smile_data and no_smile_data.

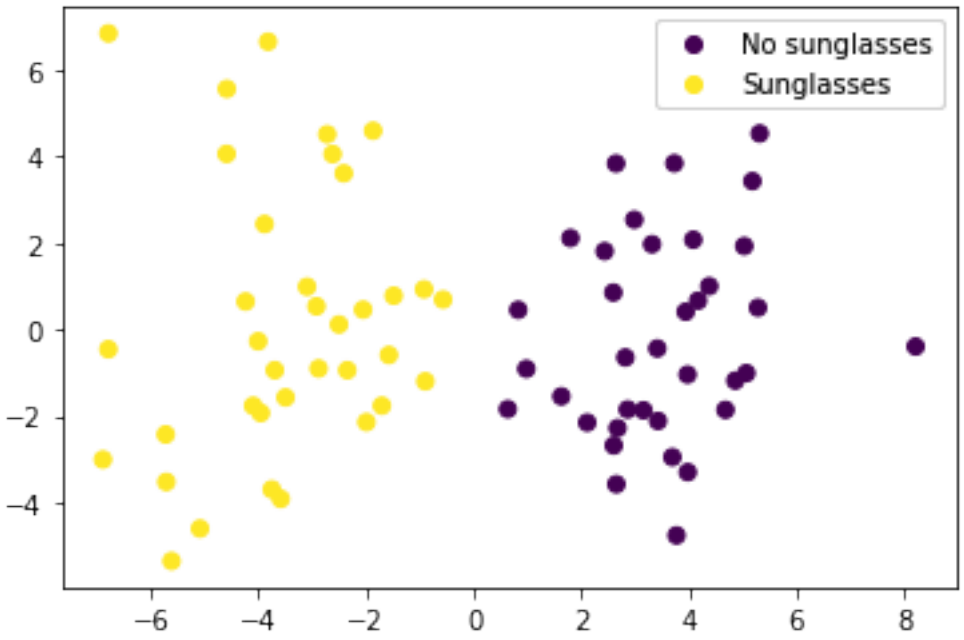
Visualize the latent space for at least three pairs of face groups including smile vs. no smile. Comment on how the scatter plots look. * Are the two groups separable in the two-dimensional latent space? Is this what you expected? Why or why not? (2 points) * How do the plots for the three different attributes differ from each other? (2 points)

[ ]:
```
def LatentSpace_2D(encoder_model, data1, label1, data2, label2):
    # Your code here
    embedd1 = encoder_model.predict(data1)
    embedd2 = encoder_model.predict(data2)

    embedd = np.concatenate((embedd1, embedd2), axis=0)

    y_data1 = np.zeros((data1.shape[0], 1))
    y_data2 = np.ones((data2.shape[0], 1))
    y = np.concatenate((y_data1, y_data2), axis=0)

    pca = PCA(2)
    embedd_pca = pca.fit_transform(embedd)

    fig, ax = plt.subplots()
    scatter = ax.scatter(embedd_pca[:,0], embedd_pca[:,1], c=y)
    handles, _ = scatter.legend_elements()
    legend = ax.legend(handles, [label1, label2])
    ax.add_artist(legend)
```

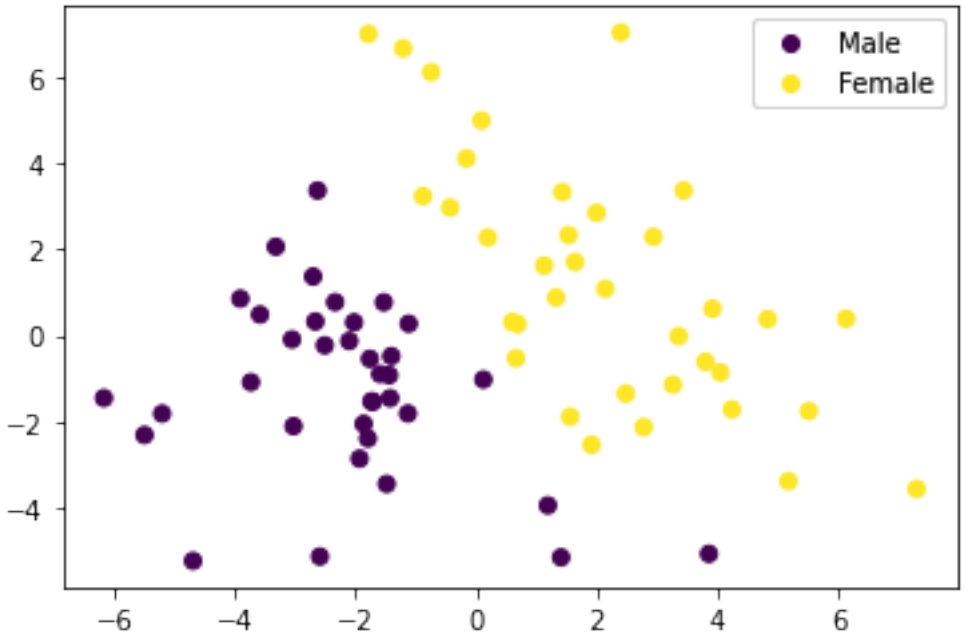
[ ]:
```
LatentSpace_2D(encoder_model, smile_data, 'Smile', no_smile_data, 'No smile')
```



[ ]:
```
LatentSpace_2D(encoder_model,smile_data,'No␣
↪sunglasses',sunglasses_data,'Sunglasses')
```



[ ]:
```
LatentSpace_2D(encoder_model,male_data,'Male',female_data,'Female')
```



Answer: 1. The plots are linear-separable in 2D space after (except for a few points) after dimensionality reduction using PCA. As each face group pair shares opposite attributes, VAE will tend to learn a distinct latent representation for each group. 2. The male/female and no sugglasses/sunglasses are more separable than smile/no smile group in latent space. This might because that smiling is harder to distinguish than the other two.

1.1.9 1.4 Morphing between faces (4 points)

Morph at least 5 pairs of faces with the function `morphBetweenImages` and comment on what you observe. * Briefly explain how the morphing works. (2 points) * Do the generated faces look like what you expected? Does any of the pairs work better than the others? If so, what kind of image pairs work better? (2 points)

[ ]:
```
# Don't change the function
def morphBetweenImages(img1, img2, num_of_morphs):
    alpha = np.linspace(0,1,num_of_morphs)
    z1 = encoder_model.predict(np.array([img1]))
    z2 = encoder_model.predict(np.array([img2]))
    fig = plt.figure(figsize=(30,5))

    ax = fig.add_subplot(1, num_of_morphs+2, 1)
    ax.imshow(img1)
    ax.axis('off')
    ax.set_title(loc='center', label='original image 1',fontsize=10)

    for i in range(num_of_morphs):
```

```
        z = z1*(1-alpha[i]) + z2*alpha[i]
        new_img = decoder_model.predict(z)

        ax = fig.add_subplot(1, num_of_morphs+2, i+2)
        ax.imshow(np.clip(new_img.squeeze(),0,1))
        ax.axis('off')
        ax.set_title(loc='center', label='alpha={:.2f}'.format(alpha[i]))

    ax = fig.add_subplot(1, num_of_morphs+2, num_of_morphs+2)
    ax.imshow(img2)
    ax.axis('off')
    ax.set_title(loc='center', label='original image 2',fontsize=10)
    return
```

[ ]:
```
sample_index = random.sample(range(1, len(data)), 2)
morphBetweenImages(data[sample_index[0]],data[sample_index[1]],10)
```

[ ]:
```
sample_index = random.sample(range(1, len(data)), 2)
morphBetweenImages(data[sample_index[0]],data[sample_index[1]],10)
```

[ ]:
```
sample_index = random.sample(range(1, len(data)), 2)
morphBetweenImages(data[sample_index[0]],data[sample_index[1]],10)
```

[ ]:
```
sample_index = random.sample(range(1, len(data)), 2)
morphBetweenImages(data[sample_index[0]],data[sample_index[1]],10)
```

[ ]:
```
sample_index = random.sample(range(1, len(data)), 2)
morphBetweenImages(data[sample_index[0]],data[sample_index[1]],10)
```

Answer: 1. We can manipulate the latent embedding to generate artifical images after the VAE training. By using a linear combination of the latent embeddings between two images, we are able to morph between by generating artificial images using the decoder. 2. The fourth pair seems to be better than the others. Perhaps the image pair that has more distinct attributes will yiled better morphing results.

1.1.10 1.5 Attribute shift (10 points)

In 1.3, we've seen that faces with the same attributes form clusters in the latent space. In this problem, you will implement a function `AttributeShift` to change one attribute of the faces.

First implement the function `AttributeShift`. (5 points) 1. Calculate the latent space codes for two sets of faces that are different in one attribute, e.g. smile vs. no smile. 2. Calculate the mean latent space code for each group. 3. Get the attribute shifting vector by taking the difference between the two codes. 4. Perform attribute shift by adding the attribute shifting vector to the latent space code of the faces you want to modify. 5. Generate the image using the new latent space codes.

Here is a diagram demonstrating the shift in the latent space. Please note that the two-dimensional latent space is just for demonstration purpose. You should *not* use PCA in this problem. Instead, use the original latent space.

Perform attribute shift on at three attributes including smile. Comment on the faces with shifted attributes. (5 points) * Do the generated faces look like what you expected? If not, can you think of some possible reasons? * Do the faces with new attributes resemble the original faces? If not, can you think of some possible reasons? * Which of the attribute shift is more successful? What are some possible reasons? * Comment on any other aspects of your findings that are interesting to you.

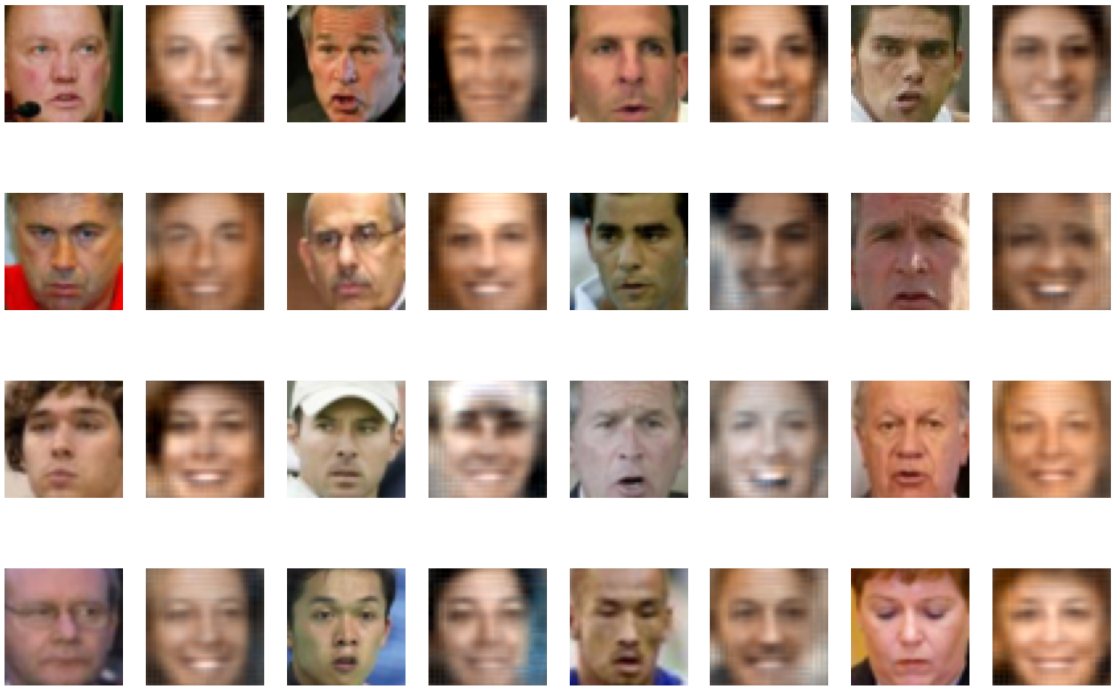
[ ]:
```
# Don't change this helper function!
def PlotAttributeShift(data2,pic_output):
    sample_index = random.sample(range(1, len(data2)), 16)

    fig, axs = plt.subplots(4, 8)
    fig.set_figheight(10)
    fig.set_figwidth(15)

    for i in range(4):
```

```
        for j in range(4):
            axs[i, 2*j].imshow(data2[sample_index[4*i+j], :, :, :])
            axs[i, 2*j].axis('off')
            axs[i, 2*j+1].imshow(np.clip(pic_output[sample_index[4*i+j]],0,1))
            axs[i, 2*j+1].axis('off')
```

[ ]:
```
def AttributeShift(encoder_model,decoder_model,data1,data2):
    # Your code here
    embed1 = encoder_model.predict(data1)
    embed2 = encoder_model.predict(data2)

    mean1 = np.mean(embed1, axis=0)
    mean2 = np.mean(embed2, axis=0)

    attr_shift = mean1 - mean2

    embed2_shifted = embed2 + attr_shift
    pic_output = decoder_model.predict(embed2_shifted)

    return pic_output
```

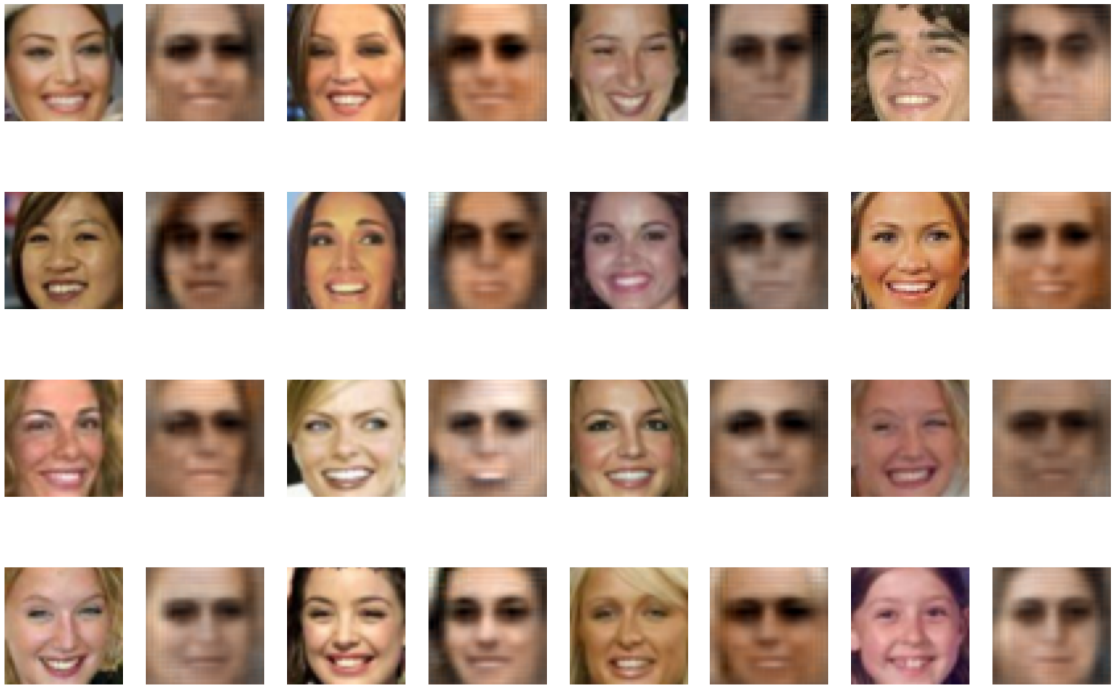
[ ]:
```
pic_output =␣
 ↪AttributeShift(encoder_model,decoder_model,smile_data,no_smile_data)
PlotAttributeShift(no_smile_data,pic_output)
```

```
(100,)
```



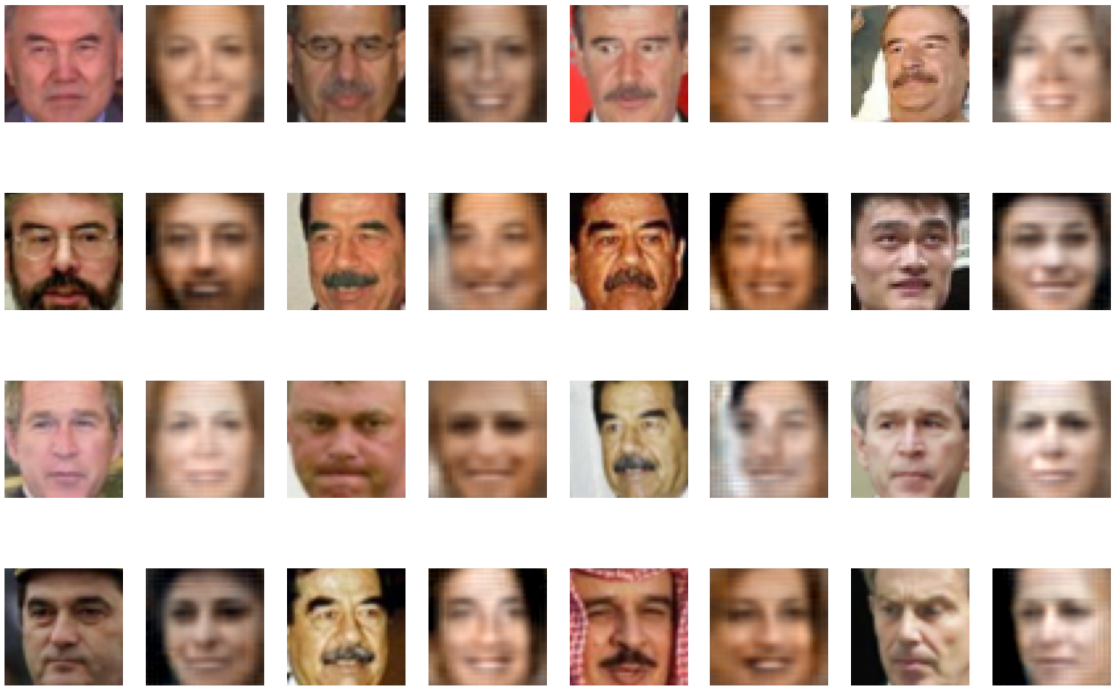
[ ]:
```
pic_output =␣
 ↪AttributeShift(encoder_model,decoder_model,sunglasses_data,smile_data)
PlotAttributeShift(smile_data,pic_output)
```

```
(100,)
```



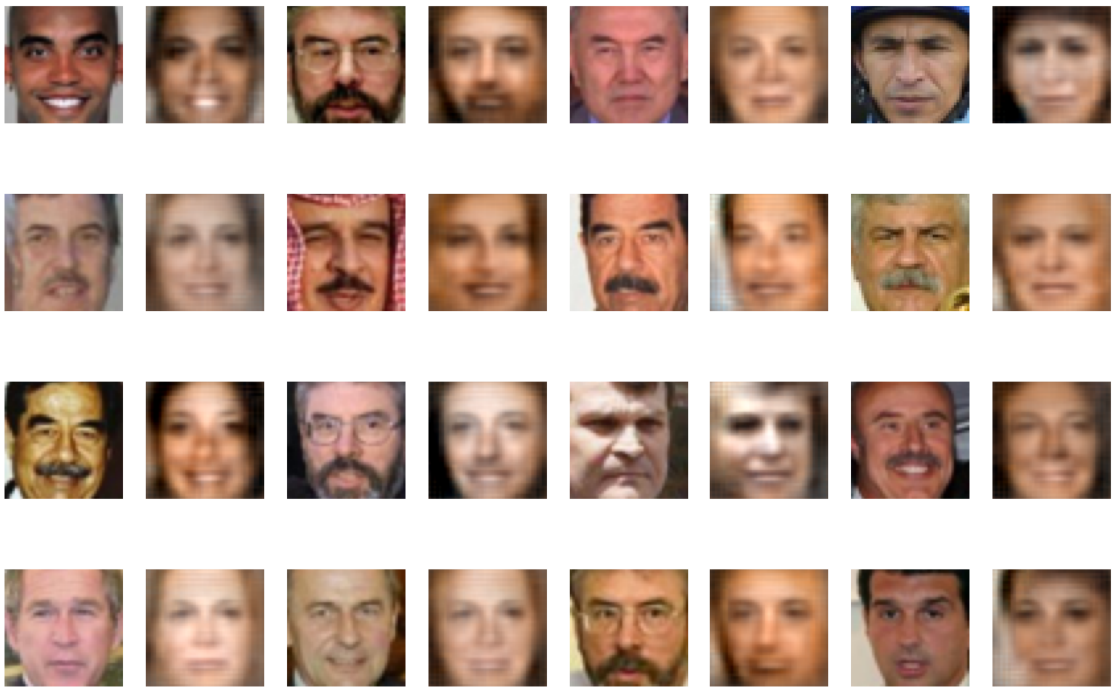
[ ]:
```
pic_output = AttributeShift(encoder_model,decoder_model,female_data,male_data)
PlotAttributeShift(male_data,pic_output)
```

```
(100,)
```



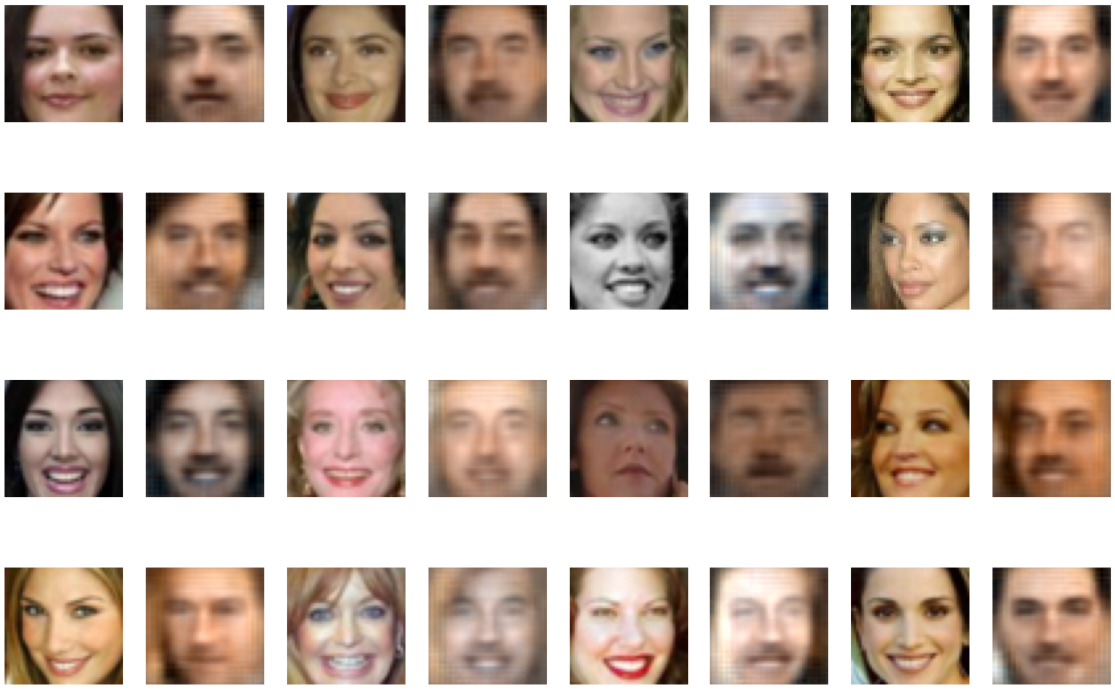
[ ]:
```
pic_output = AttributeShift(encoder_model,decoder_model,makeup_data,male_data)
PlotAttributeShift(male_data,pic_output)
```

```
(100,)
```



```
[ ]: pic_output =␣
      ↪AttributeShift(encoder_model,decoder_model,mustache_data,female_data)
     PlotAttributeShift(female_data,pic_output)
```

```
(100,)
```



Answer: 1. The generated images generally meet the expectation. For instance, we can see that the female faces are transformed to faces with sunglasses in the first example. 2. The transformed faces still resemble the original faces in many aspects (e.g. they share similar postures and are still aligned). This is might due to that differences between two data groups are not large enough to change some basic features (e.g. orientation). 3. The first three groups are more successful than the last two. This might due to that the last two groups (mustache, makeup) require better prediction quality (e.g. resolution) to differentiate the changes in style. 4. In the last two groups involve inherent gender transformation. This is because the target group (makeup or mustache) in the dataset is usually biased towards the feature (e.g. faces with makeup are usually women).

1.1.11 1.6 Generating new faces (3 points)

Variational autoencoders can be used to generate new data; this is why they are generative models. We can sample new data points from the distribution in latent space and reconstruct new, fake faces based on them.

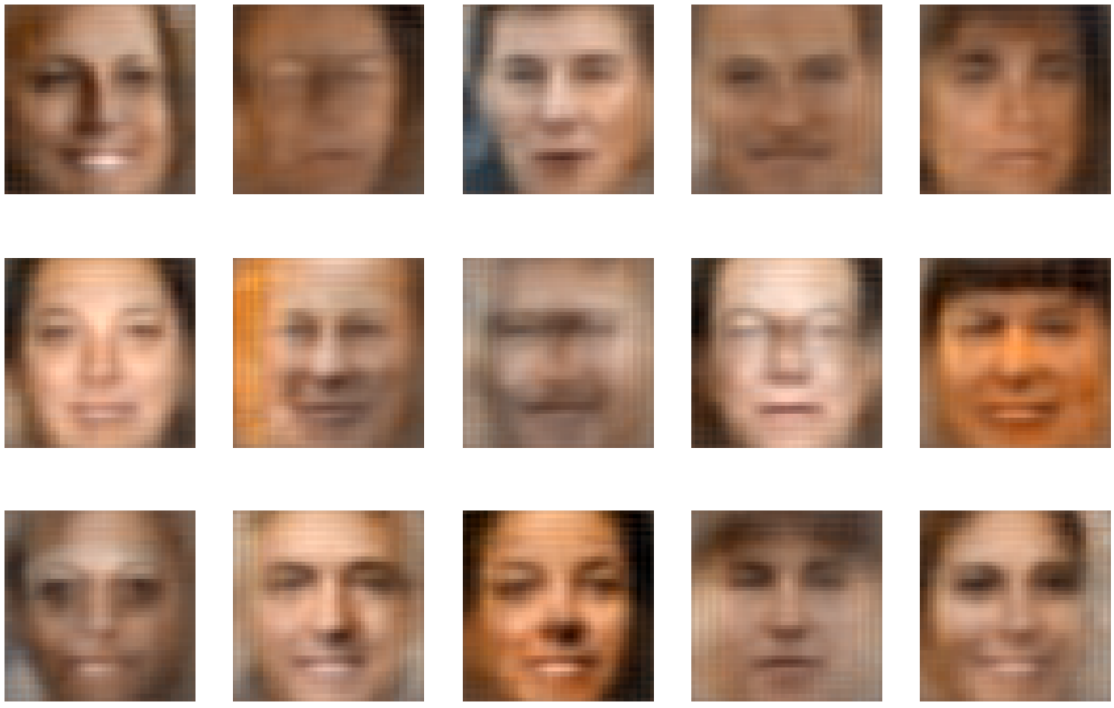
To draw a sample close to an existing sample in the latent space, we can add a scaled random sample from the normal distribution to the latent space code of an existing sample. The scalar,

which we call the `noise_level` is a parameter that we can tune.

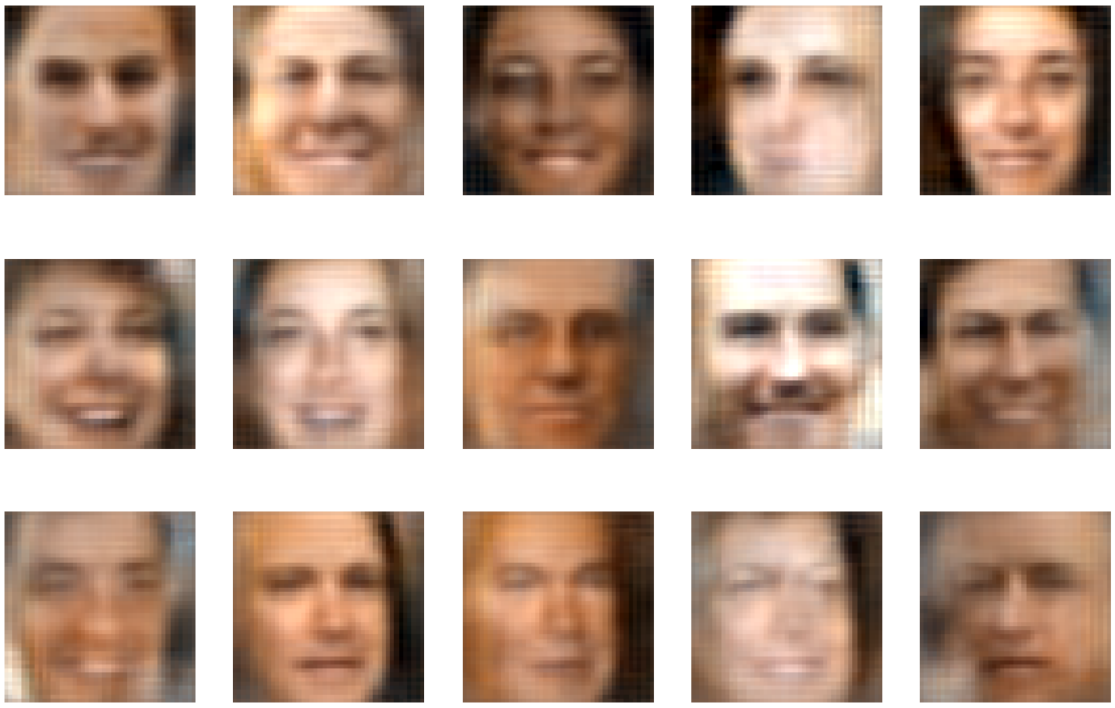
Run `GenerateFaces` with three different values of `noise_level` and comment on the generated faces. (3 points) * Do the generated images look like faces? * What happens when the new samples diverge more from the existing samples? What is a possible reason?

[ ]:
```
def GenerateFaces(data, LATENT_SPACE_SIZE, noise_level):
    sample_index = random.sample(range(1, len(data)), 15)
    latent_space = noise_level*np.random.
 ↪normal(size=(15,LATENT_SPACE_SIZE))+encoder_model.predict(data[sample_index])
    generated_image = decoder_model.predict(latent_space)
    fig = plt.figure(figsize=(15,10))
    for i in range(generated_image.shape[0]):
        ax = fig.add_subplot(3, 5, i+1)
        ax.imshow(np.clip(generated_image[i, :,:,:],0,1))
        ax.axis('off')
```

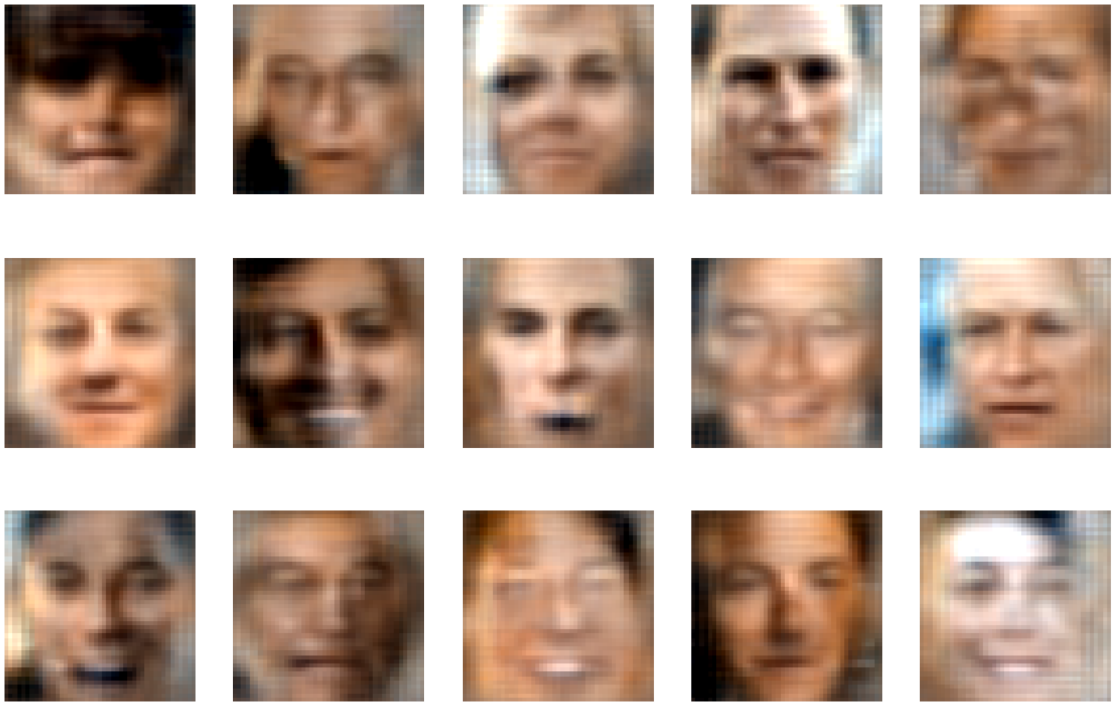
[ ]:
```
GenerateFaces(data,LATENT_SPACE_SIZE,0.1)
```



[ ]:
```
GenerateFaces(data,LATENT_SPACE_SIZE,0.5)
```



[ ]:
```
GenerateFaces(data,LATENT_SPACE_SIZE,1)
```



Answer: 1. The generated new faces generally look like human faces but there are still quite a few artifacts in the predicted images. 2. They diverge because noise is added to the latent variables encoded in the dataset. They diverge more because that the noise level is higher.

1.1.12 Optional: Changing the loss function (2 points extra credit)

The variable `factor` is a parameter of the loss function. In this optional problem, you will play with it and think about its effect on the performance of VAE.

Retrain the model with a smaller factor. Repeat 1.2, 1.6 (using `noise_level` = 1) and latent space visualization. Comment on how the reconstruction results, generated new faces and latent space distributions change. (2 points) * Which model reconstructs the faces better? * Do the generated faces look different? * How do the latent variable distributions differ? * Do the differences make sense? Can you explain what you observed?

[ ]:
```
# Your code here
batch_size = 64
autoencoder.compile(loss=get_loss(distribution_mean, distribution_variance,␣
 ↪factor = 10, batch_size = batch_size), optimizer='adam')
autoencoder.summary()

autoencoder.fit(X_train, X_train, epochs=50, batch_size=64,␣
 ↪validation_data=(X_val, X_val))
```

```
Model: "functional_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 45, 45, 3)]       0
_________________________________________________________________
functional_1 (Functional)    (None, 100)               118056
_________________________________________________________________
functional_3 (Functional)    (None, 45, 45, 3)         90675
=================================================================
Total params: 208,731
Trainable params: 208,731
Non-trainable params: 0
_________________________________________________________________
Train on 10514 samples, validate on 2629 samples
Epoch 1/50
10514/10514 [==============================] - 3s 330us/sample - loss:
-4552.5358 - val_loss: -4555.6909
Epoch 2/50
10514/10514 [==============================] - 4s 338us/sample - loss:
-4587.1867 - val_loss: -4564.1442
Epoch 3/50
10514/10514 [==============================] - 4s 370us/sample - loss:
-4588.6238 - val_loss: -4567.1721
Epoch 4/50
```

```
10514/10514 [==============================] - 4s 358us/sample - loss:
-4590.0992 - val_loss: -4572.4974
Epoch 5/50
10514/10514 [==============================] - 3s 327us/sample - loss:
-4590.5729 - val_loss: -4566.7232
Epoch 6/50
10514/10514 [==============================] - 4s 349us/sample - loss:
-4593.1262 - val_loss: -4569.2684
Epoch 7/50
10514/10514 [==============================] - 3s 296us/sample - loss:
-4593.9199 - val_loss: -4571.4727
Epoch 8/50
10514/10514 [==============================] - 3s 306us/sample - loss:
-4594.2681 - val_loss: -4574.0675
Epoch 9/50
10514/10514 [==============================] - 4s 334us/sample - loss:
-4595.6717 - val_loss: -4566.9565
Epoch 10/50
10514/10514 [==============================] - 3s 331us/sample - loss:
-4595.5611 - val_loss: -4579.1818
Epoch 11/50
10514/10514 [==============================] - 3s 313us/sample - loss:
-4596.9386 - val_loss: -4565.1920
Epoch 12/50
10514/10514 [==============================] - 4s 338us/sample - loss:
-4597.5985 - val_loss: -4577.9488
Epoch 13/50
10514/10514 [==============================] - 3s 327us/sample - loss:
-4598.2076 - val_loss: -4581.8171
Epoch 14/50
10514/10514 [==============================] - 4s 335us/sample - loss:
-4599.7765 - val_loss: -4562.5443
Epoch 15/50
10514/10514 [==============================] - 4s 342us/sample - loss:
-4599.3906 - val_loss: -4576.3548
Epoch 16/50
10514/10514 [==============================] - 4s 350us/sample - loss:
-4600.1602 - val_loss: -4579.1023
Epoch 17/50
10514/10514 [==============================] - 4s 341us/sample - loss:
-4601.7217 - val_loss: -4562.6525
Epoch 18/50
10514/10514 [==============================] - 4s 343us/sample - loss:
-4601.5631 - val_loss: -4583.0487
Epoch 19/50
10514/10514 [==============================] - 3s 330us/sample - loss:
-4602.5759 - val_loss: -4555.5810
Epoch 20/50
```

```
10514/10514 [==============================] - 3s 328us/sample - loss:
-4602.0000 - val_loss: -4572.7720
Epoch 21/50
10514/10514 [==============================] - 4s 350us/sample - loss:
-4604.6483 - val_loss: -4585.0280
Epoch 22/50
10514/10514 [==============================] - 4s 393us/sample - loss:
-4604.2099 - val_loss: -4571.2820
Epoch 23/50
10514/10514 [==============================] - 4s 374us/sample - loss:
-4605.0555 - val_loss: -4582.5007
Epoch 24/50
10514/10514 [==============================] - 4s 388us/sample - loss:
-4606.5725 - val_loss: -4587.2841
Epoch 25/50
10514/10514 [==============================] - 4s 393us/sample - loss:
-4606.5954 - val_loss: -4580.0630
Epoch 26/50
10514/10514 [==============================] - 4s 374us/sample - loss:
-4606.3849 - val_loss: -4584.1549
Epoch 27/50
10514/10514 [==============================] - 4s 378us/sample - loss:
-4607.5413 - val_loss: -4584.3632
Epoch 28/50
10514/10514 [==============================] - 4s 359us/sample - loss:
-4607.2166 - val_loss: -4573.1639
Epoch 29/50
10514/10514 [==============================] - 4s 345us/sample - loss:
-4609.3464 - val_loss: -4589.2990
Epoch 30/50
10514/10514 [==============================] - 4s 350us/sample - loss:
-4608.8191 - val_loss: -4584.0319
Epoch 31/50
10514/10514 [==============================] - 4s 373us/sample - loss:
-4609.6534 - val_loss: -4584.6049
Epoch 32/50
10514/10514 [==============================] - 4s 369us/sample - loss:
-4610.1804 - val_loss: -4587.6512
Epoch 33/50
10514/10514 [==============================] - 4s 358us/sample - loss:
-4610.4395 - val_loss: -4575.4094
Epoch 34/50
10514/10514 [==============================] - 4s 368us/sample - loss:
-4609.5362 - val_loss: -4590.0173
Epoch 35/50
10514/10514 [==============================] - 4s 348us/sample - loss:
-4612.2240 - val_loss: -4582.6406
Epoch 36/50
```

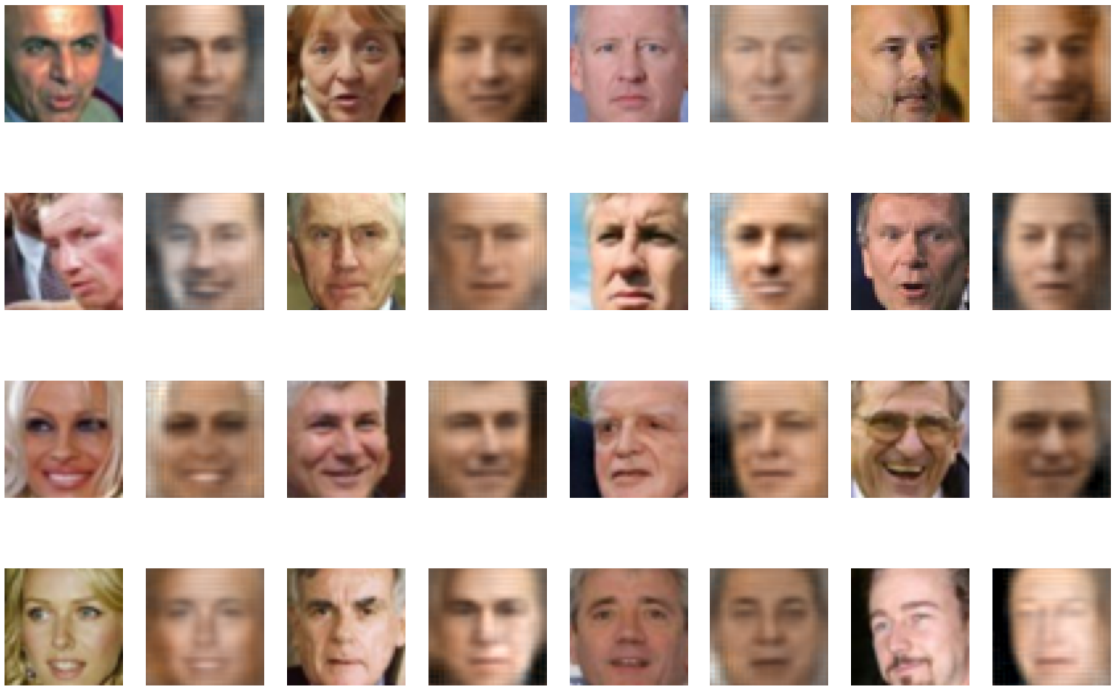
```
10514/10514 [==============================] - 4s 375us/sample - loss:
-4610.8486 - val_loss: -4590.1173
Epoch 37/50
10514/10514 [==============================] - 4s 379us/sample - loss:
-4612.9946 - val_loss: -4569.7788
Epoch 38/50
10514/10514 [==============================] - 4s 373us/sample - loss:
-4610.8777 - val_loss: -4589.9475
Epoch 39/50
10514/10514 [==============================] - 4s 371us/sample - loss:
-4613.5779 - val_loss: -4592.8517
Epoch 40/50
10514/10514 [==============================] - 3s 332us/sample - loss:
-4614.7227 - val_loss: -4592.9304
Epoch 41/50
10514/10514 [==============================] - 4s 379us/sample - loss:
-4614.4479 - val_loss: -4589.9365
Epoch 42/50
10514/10514 [==============================] - 4s 385us/sample - loss:
-4614.2618 - val_loss: -4587.4938
Epoch 43/50
10514/10514 [==============================] - 4s 374us/sample - loss:
-4613.9760 - val_loss: -4587.0976
Epoch 44/50
10514/10514 [==============================] - 4s 361us/sample - loss:
-4614.8463 - val_loss: -4579.1084
Epoch 45/50
10514/10514 [==============================] - 4s 359us/sample - loss:
-4615.1897 - val_loss: -4591.5865
Epoch 46/50
10514/10514 [==============================] - 4s 374us/sample - loss:
-4616.5454 - val_loss: -4586.2626
Epoch 47/50
10514/10514 [==============================] - 4s 364us/sample - loss:
-4615.3559 - val_loss: -4587.3121
Epoch 48/50
10514/10514 [==============================] - 4s 362us/sample - loss:
-4617.2375 - val_loss: -4589.3344
Epoch 49/50
10514/10514 [==============================] - 4s 382us/sample - loss:
-4615.1265 - val_loss: -4595.5161
Epoch 50/50
10514/10514 [==============================] - 4s 387us/sample - loss:
-4617.2708 - val_loss: -4593.3694
```

[ ]: <tensorflow.python.keras.callbacks.History at 0x7facc01b3e80>

[ ]:
```
autoencoder.save("My_Trained_VAE_factor_10")
encoder_model.save("My_Trained_encoder_factor_10")
decoder_model.save("My_Trained_decoder_factor_10")
```

```
INFO:tensorflow:Assets written to: My_Trained_VAE_factor_10/assets
INFO:tensorflow:Assets written to: My_Trained_encoder_factor_10/assets
INFO:tensorflow:Assets written to: My_Trained_decoder_factor_10/assets
```

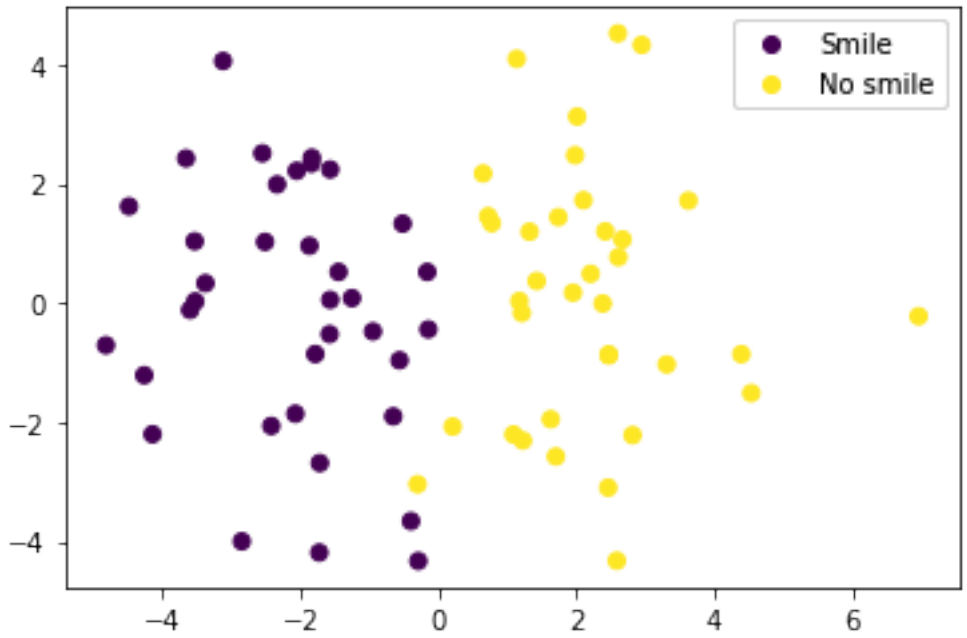
[ ]:
```
sample_index = random.sample(range(1, len(X_val)), 16)

fig, axs = plt.subplots(4, 8)
fig.set_figheight(10)
fig.set_figwidth(15)

for i in range(4):
    for j in range(4):
        axs[i, 2*j].imshow(X_val[sample_index[4*i+j], :, :, :])
        axs[i, 2*j].axis('off')
        axs[i, 2*j+1].imshow(np.clip(autoencoder.predict(np.
 ↪array([X_val[sample_index[4*i+j], :, :, :]]))[0],0,1))
        axs[i, 2*j+1].axis('off')
```



[ ]:
```
encoder_model = tensorflow.keras.models.
 ↪load_model("My_Trained_encoder_factor_10",compile=False)
```

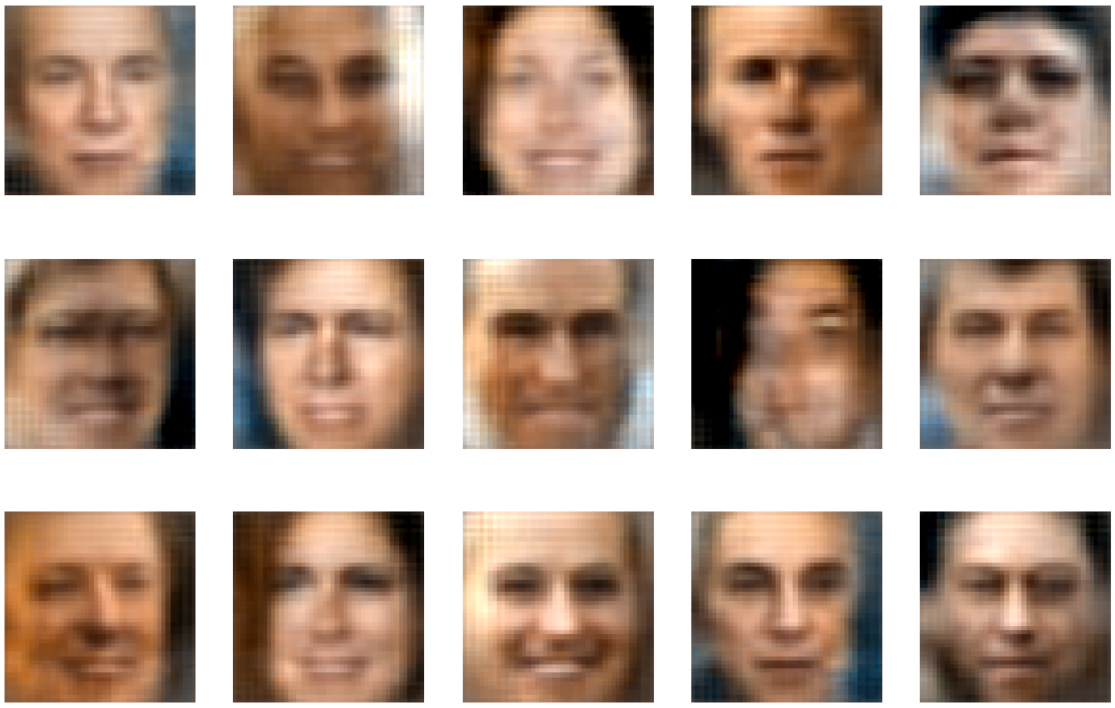
```
encoder_model.compile(loss=get_loss(distribution_mean, distribution_variance,␣
↪factor = 100, batch_size = batch_size), optimizer='Adam')
```

[ ]:
```
LatentSpace_2D(encoder_model, smile_data, 'Smile', no_smile_data, 'No smile')
```



[ ]:
```
def GenerateFaces(data, LATENT_SPACE_SIZE, noise_level):
    sample_index = random.sample(range(1, len(data)), 15)
    latent_space = noise_level*np.random.
↪normal(size=(15,LATENT_SPACE_SIZE))+encoder_model.predict(data[sample_index])
    generated_image = decoder_model.predict(latent_space)
    fig = plt.figure(figsize=(15,10))
    for i in range(generated_image.shape[0]):
        ax = fig.add_subplot(3, 5, i+1)
        ax.imshow(np.clip(generated_image[i, :,:,:],0,1))
        ax.axis('off')
```

[ ]:
```
GenerateFaces(data,LATENT_SPACE_SIZE,1)
```



Answer: 1. The model with a smaller factor is sightly worse than the previous model. 2. The generated faces look quite different from the previous model. 3. The latent space is more linearly sparable than the previous model. 4. The factor is a relevant weight of reconstruction loss to the KL loss. When the factor is larger, the image reconstruction quality is better. When the factor is smaller, the latent space is more spread out. Thus, the image reconstruction quality in the previous model is better but the latent space is less separable (e.g. in the smile vs no smile example).

1.2 Problem 2: Are you Schur? (10 points)

The graphical lasso is based on conditional independence properties of Gaussian distributions. This problem asks you to reason about the graphs underlying a multivariate Gaussian.

Let $X = (Y, Z) \in \mathbb{R}^6 \sim N(0, \Sigma)$ be a random Gaussian vector where $Y = (Y_1, Y_2) \in \mathbb{R}^2$ and $Z = (Z_1, Z_2, Z_3, Z_4) \in \mathbb{R}^4$, with

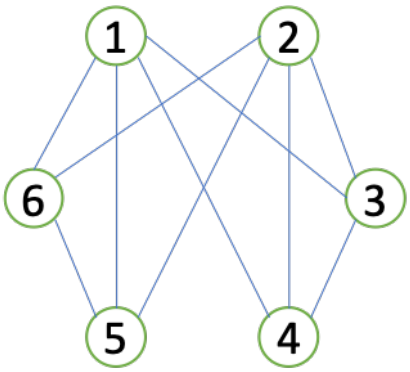
$$\Sigma^{-1} = \Omega = \begin{pmatrix} A & B \\ B^T & C \end{pmatrix}$$

where

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \qquad B = \begin{pmatrix} 1 & \frac{1}{2} & \frac{1}{3} & \frac{1}{4} \\ -1 & \frac{1}{2} & -\frac{1}{3} & \frac{1}{4} \end{pmatrix} \qquad C = \begin{pmatrix} 2 & \frac{1}{2} & 0 & 0 \\ \frac{1}{2} & 2 & 0 & 0 \\ 0 & 0 & 2 & \frac{1}{2} \\ 0 & 0 & \frac{1}{2} & 2 \end{pmatrix}.$$

1. Draw the undirected graph of X, arranging the vertices in a hexagon. Explain your answer.

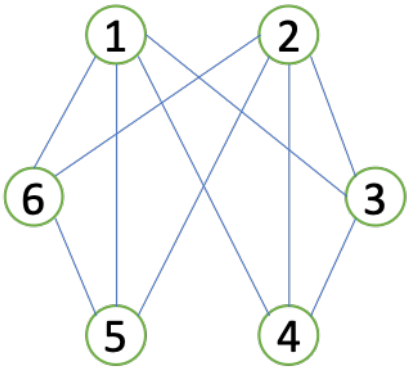
Answer: The graph of X is the following. From the precision matrix of X, we can see that there are no edges between X1 and X2, X3 and X5, X3 and X6, X4 and X5, X4 and X6.



2. Draw the undirected graph of Z, arranging the vertices in a square. Explain your answer. Hint: The Schur complement $ S= C - B^T A^{-1} B$ has the property that

$$\begin{pmatrix} A & B \\ B^T & C \end{pmatrix}^{-1} = \begin{pmatrix} A^{-1} + A^{-1}BS^{-1}B^TA^{-1} & -A^{-1}BS^{-1} \\ -S^{-1}B^TA^{-1} & S^{-1} \end{pmatrix}$$

Answer: From the above hint, we can find that the precision matrix for Z is the schur complement. We use the code attached in the next block to calculate the precision matrix and from the matrix we can conclude that there are no edges between Z1 and Z4, Z2 and Z3. The graph plot is the following.



$ $

3. Which of the following conditional independence statements hold? Explain your answers.

- $Y_1 \perp\!\!\!\perp Y_2 \,|\, Z$
- $Z_1 \perp\!\!\!\perp Z_4 \,|\, Z_2$
- $Z_1 \perp\!\!\!\perp Z_4 \,|\, Y_1$
- $Z_1 \perp\!\!\!\perp Z_2$

Answer: Only the first and third conditional statements hold. When Z is given, there are no edges connecting Y1 and Y2. For the other conditions, there are connecting edges.

[ ]:
```
# calculate Omega_z

C = np.array([[2, 0.5, 0 ,0], [0.5, 2, 0, 0], [0, 0, 2, 0.5], [0, 0, 0.5, 2]])
B = np.array([[1, 0.5, 1/3, 1/4], [-1, 1/2, -1/3, 1/4]])
A = np.array([[2, 0], [0, 2]])

Sigma_z = C - B.T.dot(np.linalg.inv(A)).dot(B)
Sigma_z
```

[ ]:
```
array([[ 1.        ,  0.5       , -0.33333333,  0.        ],
       [ 0.5       ,  1.75      ,  0.        , -0.125     ],
       [-0.33333333,  0.        ,  1.88888889,  0.5       ],
       [ 0.        , -0.125     ,  0.5       ,  1.9375    ]])
```

1.3 Problem 3: Taking stock (10 points)

A joint distribution of data has a natural graph associated with it. When the distribution is multivariate normal, this graph is encoded in the pattern of zeros and non-zeros in the inverse of the covariance matrix, also known as the "precision matrix."

In class we demonstrated the graphical lasso for estimating the graph on ETF data. In this problem you will construct two different "portolios" of stocks, and run the graphical lasso to estimate a graph, commenting on your results.

All of the code you might need for this is contained in the demo.

1.3.1 Downloading data

As demonstrated in class, you will run on equities data downloaded from Yahoo finance. Your job is to construct two "portfolios" of stocks, each of which has some kind of organization to it. For example, in one portfolio you might have 5 energy stocks, 5 tech stocks, 5 consumer staples stocks, and 5 ETF stocks. Each portfolio should have at least 20 stocks.

To download the data, follow the procedure outlined below (and discussed briefly in class):

- Search on a ticker symbol, like EZA, using https://finance.yahoo.com/quote/EZA/history?p=EZA
- Select the range of the query, the frequency (daily, weekly, or monthly) and then issue the query. This will give you results like this:
- Next, hover over the Download link, and grab the URL. In this case it gives https://query1.finance.yahoo.com/v7/finance/download/EZA?period1=1044576000&period2=1648080000&
- Then, you can use this same URL, but swap in different ticker symbols, to get the corresponding data for range of companies or funds.

1.3.2 Analyzing your portfolios

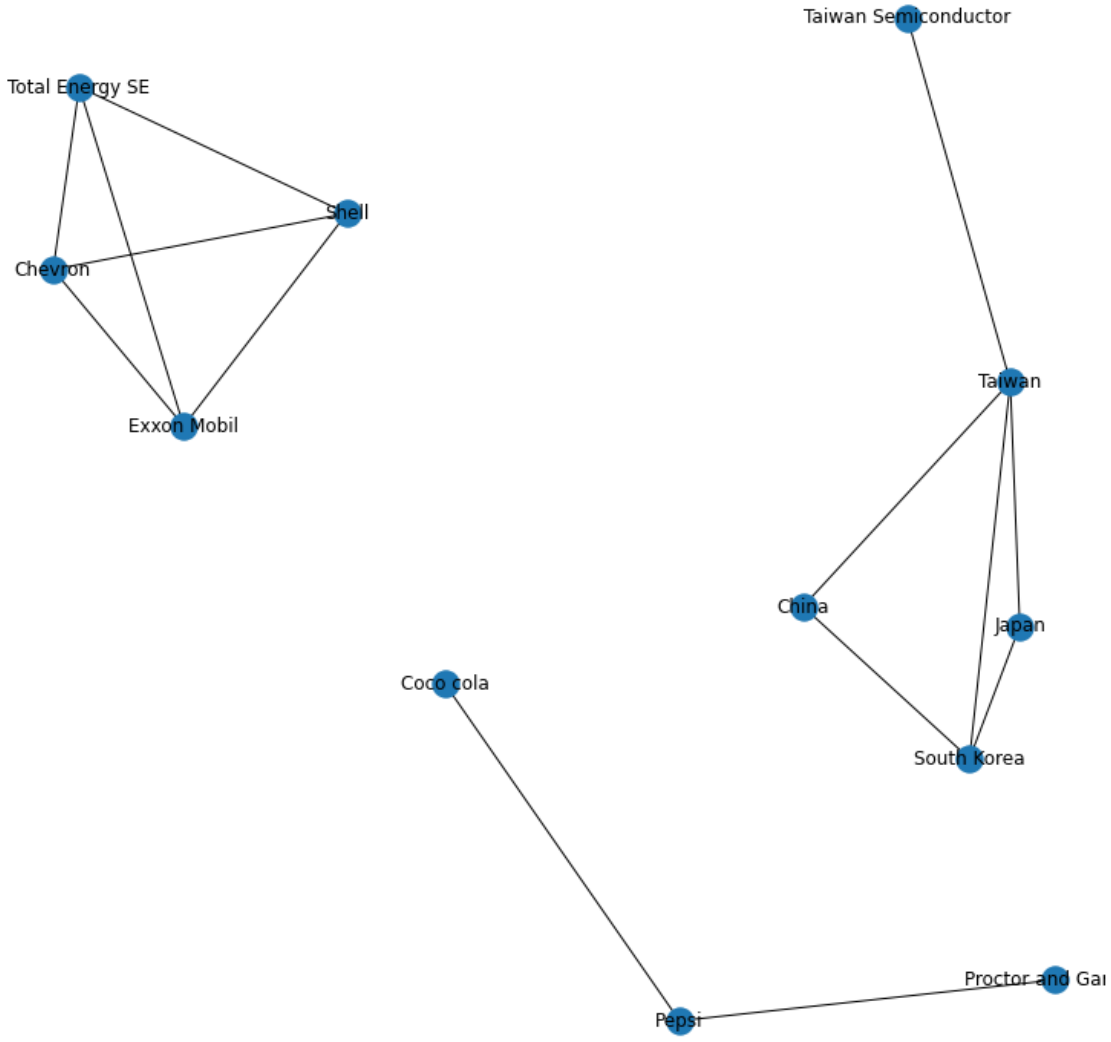
Your task is to analyze each porfolio using the graphical lasso, and comment on your findings. Here are the types of questions you should address:

- How did you choose the portolio? How did you choose the date range and frequency (daily, weekly, etc.)? Remember, each of the portfolios must contain at least 20 stocks, and be organized in some reasonable way.
- Display the graph obtained with the graphical lasso, using networkx. How did you choose the regularization level? Does the structure of the graph make sense? Is it sensitive to the choice of regularization level? Is this the structure you expected to see when you designed the portfolio? Why or why not?
- What are some of the conditional independence assumptions implied by the graph? Are some parts of the graph more densely connected than others? Why?

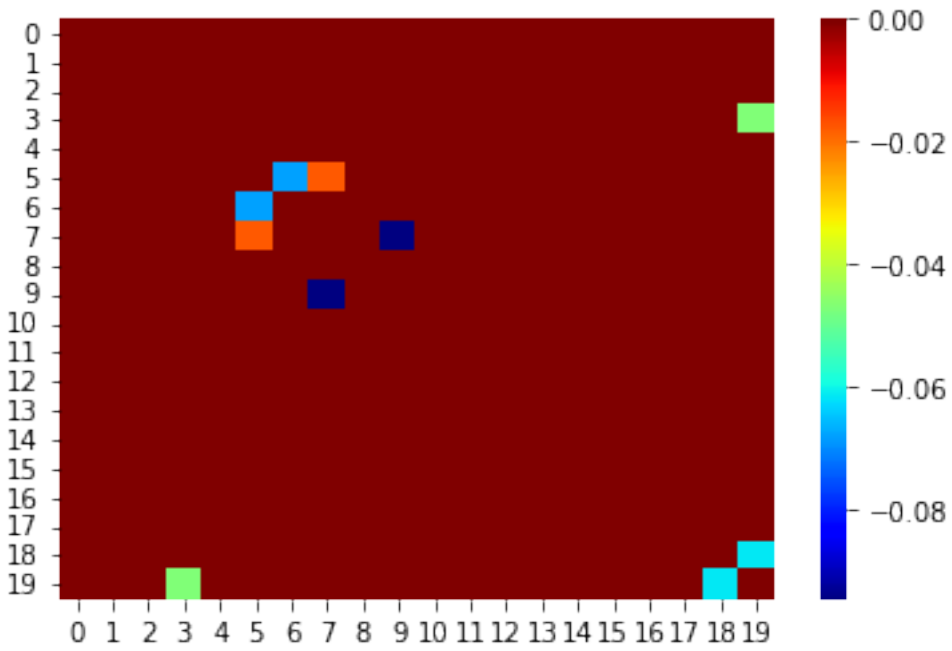
[ ]:
```
import networkx as nx
from sklearn.covariance import GraphicalLasso, GraphicalLassoCV

dir = 'problem3_files/'

def plotPorfolio(pf_list, dir, alpha):
    etf_df = pd.read_csv("%s/%s.csv" % (dir, 'AAPL'))
    df = pd.DataFrame()
    dates = etf_df['Date']

    for etf in pf_list:
        etf_df = pd.read_csv("%s/%s.csv" % (dir, etf))
        df[etf] = etf_df['Close']

    df.index = dates

    # clipped_logreturns = df

    dflogreturn = np.log1p(df.pct_change()).iloc[1:]
    logreturns = np.array(np.log1p(df.pct_change()).iloc[1:])
    clipped_logreturns = np.maximum(logreturns, -1*np.std(logreturns))

    X = np.array(clipped_logreturns)
    p = X.shape[1]
    X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
    glasso = GraphicalLasso(alpha=alpha).fit(X)
    Omegahat = np.around(glasso.precision_, decimals=5)

    np.random.seed(42)

    countries = [pf_list[c] for c in dflogreturn.columns]
    precdf = pd.DataFrame(Omegahat, columns=countries, index=countries)
    links = precdf.stack().reset_index()
    links.columns = ['var1', 'var2','value']
    links=links.loc[(abs(links['value']) > 0.01) &  (links['var1'] !=␣
↪links['var2']) ]
```

```
    #build the graph using networkx lib
    G=nx.from_pandas_edgelist(links,'var1','var2', create_using=nx.Graph())
    pos = nx.spring_layout(G, k=1.7/np.sqrt(len(G.nodes())), iterations=20)
    plt.figure(3, figsize=(10, 10))
    nx.draw(G, pos=pos)
    nx.draw_networkx_labels(G, pos=pos)
    plt.show()

    return clipped_logreturns
```

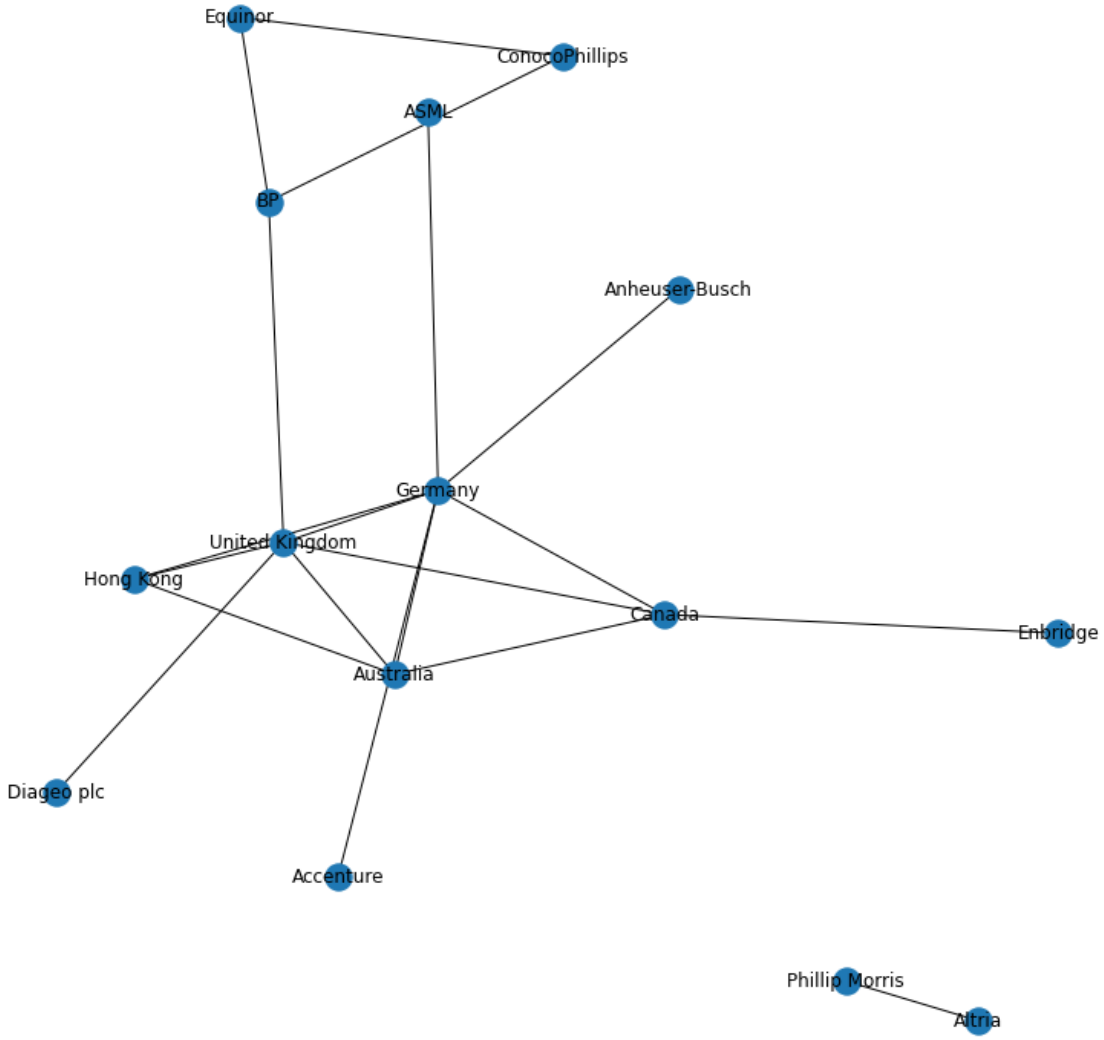
[ ]:
```
pf1 = {"AAPL": "Apple", "MSFT":"Microsoft", "NVDA":"Nvidia", "TSM":"Taiwan␣
↪Semiconductor", "AVGO":"Broadcom", "XOM":"Exxon Mobil", "CVX":"Chevron",␣
↪"SHEL":"Shell", "PTR":"PetroChina", "TTE":"Total Energy SE", "WMT":
↪"Walmart", "PG":"Proctor and Gamble", "KO":"Coco cola", "COST":"Costco",␣
↪"PEP":"Pepsi", "EWJ":"Japan","EWZ":"Brazil","FXI":"China","EWY":"South␣
↪Korea", "EWT":"Taiwan"}

clipped_logreturns = plotPorfolio(pf1, dir, alpha=0.6)
```



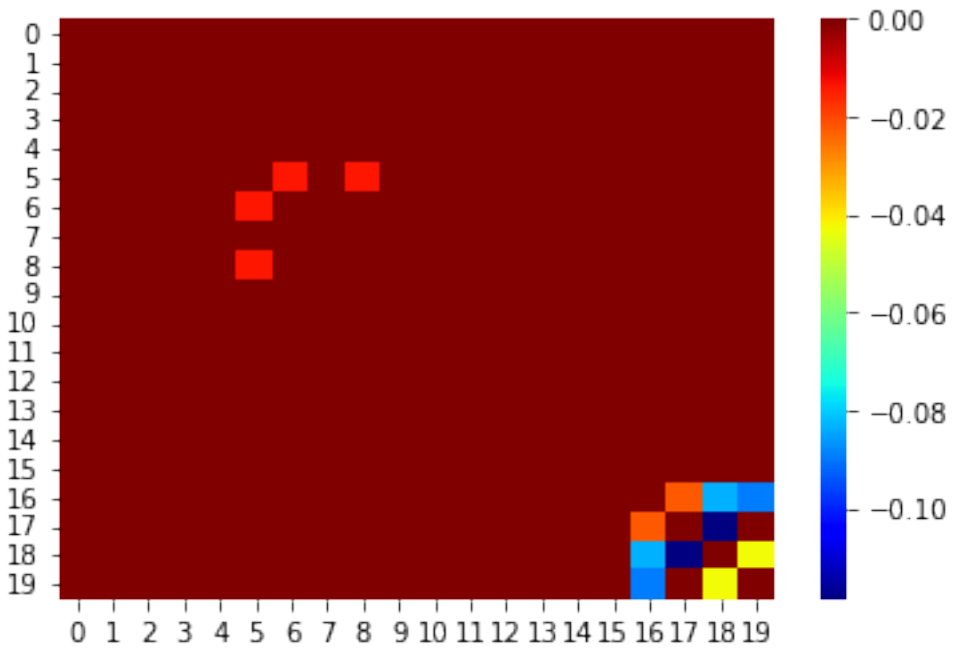
[ ]:
```
import seaborn as sns
X = np.array(clipped_logreturns)
p = X.shape[1]
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
glasso = GraphicalLasso(alpha=.75).fit(X)
Omegahat = np.around(glasso.precision_, decimals=5)

for j in np.arange(p):
    Omegahat[j,j] = 0
sns.heatmap(Omegahat, cmap='jet')
plt.show()
```



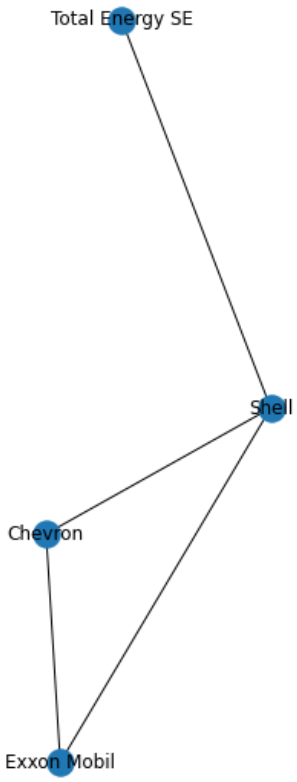
[ ]:
```
pf2 = {"ASML":"ASML", "ORCL":"Oracle", "ADBE":"Adobe", "CSCO":"Cisco", "ACN":
 ↪"Accenture" ,"EQNR":"Equinor", "COP":"ConocoPhillips", "ENB":"Enbridge",␣
 ↪"BP":"BP", "PBR":"Petroleo Brasileiro", "PM":"Phillip Morris", "DEO":"Diageo␣
 ↪plc", "BUD":"Anheuser-Busch", "TGT":"Target", "MO":"Altria","EWH":"Hong␣
 ↪Kong","EWC":"Canada","EWG":"Germany", "EWU":"United Kingdom","EWA":
 ↪"Australia"}
clipped_logreturns = plotPorfolio(pf2, dir, alpha=0.6)
```

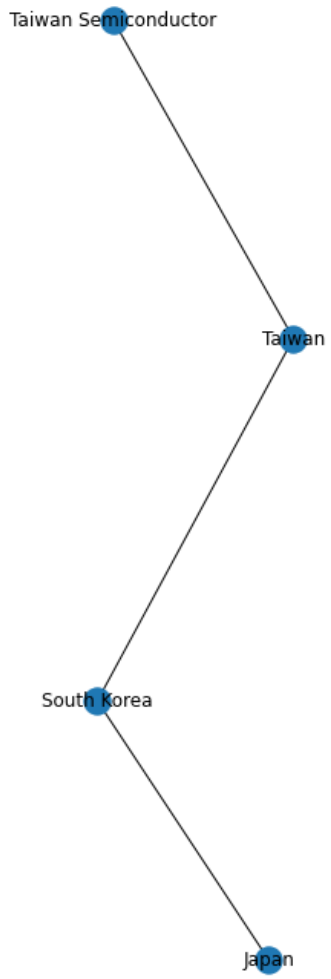


[ ]:
```
import seaborn as sns
X = np.array(clipped_logreturns)
p = X.shape[1]
X = (X - np.mean(X, axis=0)) / np.std(X, axis=0)
glasso = GraphicalLasso(alpha=.75).fit(X)
Omegahat = np.around(glasso.precision_, decimals=5)

for j in np.arange(p):
    Omegahat[j,j] = 0
sns.heatmap(Omegahat, cmap='jet')
plt.show()
```



[ ]:
```
clipped_logreturns = plotPorfolio(pf1, dir, alpha=0.7)
```





Answer: 1. Each portfolio is constructed with top 5 stocks in the following categeories: technology, energy, consumer definsive, and ETF. We choose the date range to be 5 years from 2017-2022 and the frequency to be weekly. 2. The graphs for the constructed portfolios are shown above. We can see some structures from both portfolios. For the first one, TotalEnergy SE is connected with Shell, which is connected to Chevron and so on. This is because that these equities are all energy stocks. In the second portfolio, BP is connected to UK as its headquarter is located in London. The graph is quite sensitive to regularization as changing alpha from 0.6 to 0.7 greatly reduces the connectivity in the graph. 3. The second portfolio is more densly connected to the first one. We can make some conditional independence statements. For instance, in the portfolio 2 (alpha=0.6), Diageo PLC and accenture are conditionally independent given United Kingdom.