PART 2: AUTOENCODER

In [1]:
```
import os
import numpy as np
from PIL import Image
from sklearn.metrics import mean_squared_error
```

loading data

In [2]:
```
def LoadDir(dirname):
    imgs = []
    for imgname in os.listdir(dirname):
        img = Image.open(os.path.join(dirname, imgname))
        img = img.convert('LA')  # conver to grayscale
        img = img.resize([20, 20])
        img = np.squeeze(np.array(img)[:, :, 0]).flatten()
        imgs.append(img)

    return np.array(imgs)
```

In [3]:
```
train_imgs = LoadDir('/Users/mukulj/Downloads/galaxy/train')
val_imgs = LoadDir('/Users/mukulj/Downloads/galaxy/val')
test_imgs = LoadDir('/Users/mukulj/Downloads/galaxy/test')
```

Use PCA to reduce the images down to 25 dimensions and then reconstruct them again

In [4]:
```
from sklearn.decomposition import PCA

pca = PCA(n_components=25, svd_solver='randomized',whiten=True)
pca.fit(train_imgs)

val_imgs_trans = pca.transform(val_imgs)
test_imgs_trans = pca.transform(test_imgs)
val_imgs_invrs = pca.inverse_transform(val_imgs_trans)
test_imgs_invrs = pca.inverse_transform(test_imgs_trans)

print(val_imgs.shape, test_imgs.shape)
print(val_imgs_trans.shape, test_imgs_trans.shape)
print(val_imgs_invrs.shape, test_imgs_invrs.shape)
```

```
(10382, 400) (10324, 400)
(10382, 25) (10324, 25)
(10382, 400) (10324, 400)
```

Compute the reconstruction error in terms of the squared error per pixel

In [72]:
```
print('Validation data MSE with PCA:', mean_squared_error(val_imgs, val_img
print('Test data MSE with PCA:',mean_squared_error(test_imgs, test_imgs_inv
```

```
Validation data MSE with PCA: 37.8185285358682
Test data MSE with PCA: 37.760658467004845
```

Train an autoencoder with a 25-dimensional bottleneck layer

In [6]:
```
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler

alphas = np.logspace(-3, 0, 4)
scaler = MinMaxScaler()
scaler.fit(train_imgs)
train_imgs_scaled = scaler.transform(train_imgs)
val_imgs_scaled = scaler.transform(val_imgs)

# collection of classifiers based on parameter selection
regressors = []

for alpha in alphas:
    regressors.append(MLPRegressor(solver='adam', alpha=alpha, max_iter = 5
                                   activation = 'relu', hidden_layer_sizes=
                                   random_state=1))

# Iterate over classifiers to find the score using validation data
best_reg = regressors[0]
best_score = 0
for reg in regressors:
    reg.fit(train_imgs_scaled, train_imgs_scaled)
    score = reg.score(val_imgs_scaled, val_imgs_scaled)
    if(score > best_score):
        best_score = score
        best_reg = reg
    print(reg, 'score=', score)
```

```
MLPRegressor(alpha=0.001, hidden_layer_sizes=[1000, 25, 1000], max_iter=5
0,
             random_state=1) score= 0.6715023271795323
MLPRegressor(alpha=0.01, hidden_layer_sizes=[1000, 25, 1000], max_iter=5
0,
             random_state=1) score= 0.6991641530878125
MLPRegressor(alpha=0.1, hidden_layer_sizes=[1000, 25, 1000], max_iter=50,
             random_state=1) score= 0.6249311539008124
MLPRegressor(alpha=1.0, hidden_layer_sizes=[1000, 25, 1000], max_iter=50,
             random_state=1) score= 0.35759908337387714
```

Using the test data, compare the error from the autoencoder

In [73]:
```
print('Validation data MSE with Autoencoder:', mean_squared_error(val_imgs,
test_imgs_predicted = scaler.inverse_transform(best_reg.predict(scaler.tran
print('Test data MSE with Autoencoder:', mean_squared_error(test_imgs, test
```

```
Validation data MSE with Autoencoder: 27.867756576958335
Test data MSE with Autoencoder: 28.198441333470882
```

Find an image in the test set where the autoencoder beats PCA and vice versa

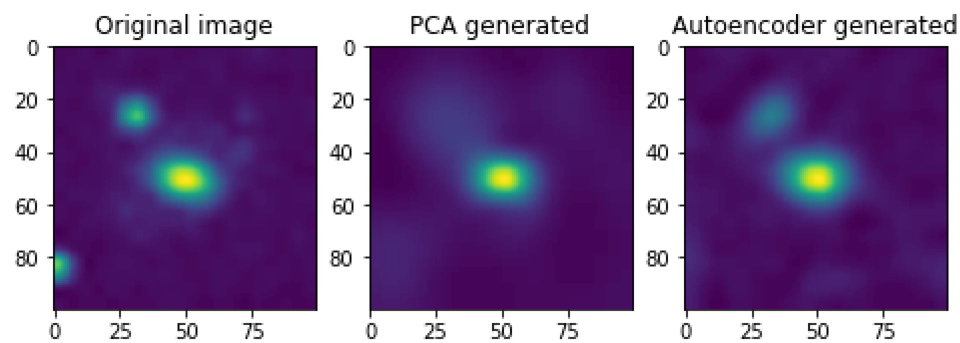
Autoencoder beats PCA

In [89]:
```
import matplotlib.pyplot as plt
import statistics

pca_square_errors = (test_imgs - test_imgs_invrs)**2
ae_square_errors = (test_imgs - test_imgs_predicted)**2
print('PCA:', statistics.mean(pca_square_errors[0]), ' Autoencoder:', stati

fig=plt.figure(figsize=(8, 8))
fig.add_subplot(1, 3, 1)
plt.imshow(Image.fromarray(np.reshape(test_imgs[0], (20, 20))).resize([100,
plt.title('Original image')
fig.add_subplot(1, 3, 2)
plt.imshow(Image.fromarray(np.reshape((test_imgs_invrs[0]).astype('uint8'),
plt.title('PCA generated')
fig.add_subplot(1, 3, 3)
plt.imshow(Image.fromarray(np.reshape((test_imgs_predicted[0]).astype('uint
plt.title('Autoencoder generated')
plt.show()
```

```
PCA: 90.77541344098992  Autoencoder: 66.11569487893685
```

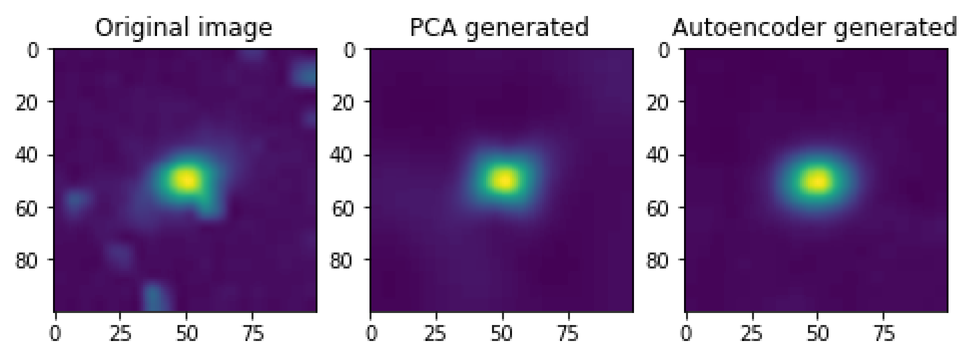


PCA beats Autoencoder

In [90]:
```
pca_square_errors = (test_imgs - test_imgs_invrs)**2
ae_square_errors = (test_imgs - test_imgs_predicted)**2
print('PCA:', statistics.mean(pca_square_errors[14]), ' Autoencoder:', stat

fig=plt.figure(figsize=(8, 8))
fig.add_subplot(1, 3, 1)
plt.imshow(Image.fromarray(np.reshape(test_imgs[14], (20, 20))).resize([100
plt.title('Original image')
fig.add_subplot(1, 3, 2)
plt.imshow(Image.fromarray(np.reshape((test_imgs_invrs[14]).astype('uint8')
plt.title('PCA generated')
fig.add_subplot(1, 3, 3)
plt.imshow(Image.fromarray(np.reshape((test_imgs_predicted[14]).astype('uin
plt.title('Autoencoder generated')
plt.show()
```

```
PCA: 30.986278124160147  Autoencoder: 37.403247468571784
```



25-dim vector representation using PCA and Autoencoder

In [66]:
```
def encoder(data):
    data = np.asmatrix(data)
    encoder1 = data*best_reg.coefs_[0] + best_reg.intercepts_[0]
    encoder1 = (np.exp(encoder1) - np.exp(-encoder1))/(np.exp(encoder1) + n
    latent = encoder1*best_reg.coefs_[1] + best_reg.intercepts_[1]
    latent = (np.exp(latent) - np.exp(-latent))/(np.exp(latent) + np.exp(-l
    return np.asarray(latent)
```

In [70]:
```
autoencoder_25dim_testdata = encoder(scaler.transform(test_imgs))
autoencoder_25dim_traindata = encoder(scaler.transform(train_imgs))
pca_25dim_testdata = test_imgs_trans
pca_25dim_traindata = pca.transform(train_imgs)
```

In [80]:
```
autoencoder_25dim_traindata.shape
```

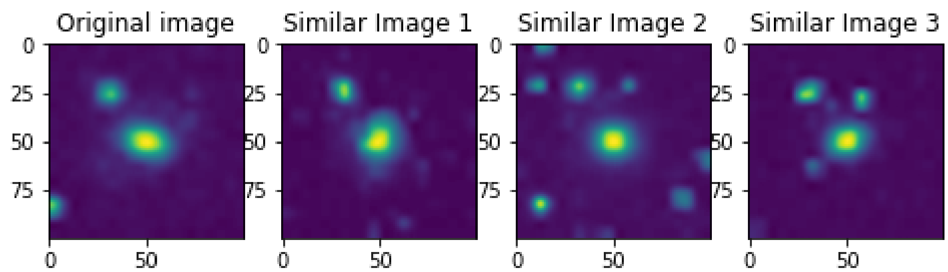
Out[80]:
```
(40872, 25)
```

Find 3 similar image from training set given test_imgs[0]

In [87]:
```
from sklearn.metrics.pairwise import euclidean_distances
euclidean_dis_dict = {}
for i in range(0, len(autoencoder_25dim_traindata)):
    euclidean_dis = np.linalg.norm(autoencoder_25dim_testdata[0]-autoencode
    euclidean_dis_dict[i] = euclidean_dis
{k: v for k, v in sorted(euclidean_dis_dict.items(), key=lambda item: item[
```

Out[87]:
```
{34761: 0.6943059942336526,
 39595: 0.7505918979440211,
 10576: 0.7511892975192163,
 35887: 0.8138430760564409,
 10053: 0.8233105058737917,
 37283: 0.8242038250768455,
 16364: 0.8312152717663984,
 28513: 0.8673061975243851,
 21361: 0.8696408415569942,
 25402: 0.8753080943122831,
 4171: 0.87704965527661,
 4859: 0.8883385884868119,
 16240: 0.890270419310699,
 27448: 0.8986498576071771,
 2402: 0.9022459595871101,
 27539: 0.9024915197635992,
 38286: 0.9051441571206261,
 21230: 0.9118894567664133,
 1449: 0.9163906694737324,
```

In [92]:
```
fig=plt.figure(figsize=(8, 8))
fig.add_subplot(1, 4, 1)
plt.imshow(Image.fromarray(np.reshape(test_imgs[0], (20, 20))).resize([100,
plt.title('Original image')
fig.add_subplot(1, 4, 2)
plt.imshow(Image.fromarray(np.reshape(train_imgs[34761], (20, 20))).resize(
plt.title('Similar Image 1')
fig.add_subplot(1, 4, 3)
plt.imshow(Image.fromarray(np.reshape(train_imgs[39595], (20, 20))).resize(
plt.title('Similar Image 2')
fig.add_subplot(1, 4, 4)
plt.imshow(Image.fromarray(np.reshape(train_imgs[10576], (20, 20))).resize(
plt.title('Similar Image 3')
plt.show()
```

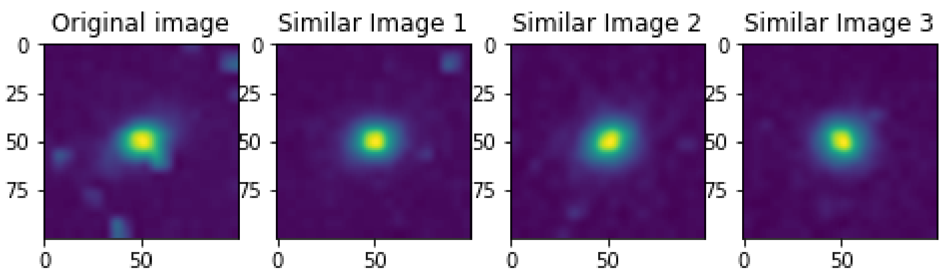


Find 3 similar image from training set given test_imgs[14]

In [98]:
```
euclidean_dis_dict = {}
for i in range(0, len(pca_25dim_traindata)):
    euclidean_dis = np.linalg.norm(pca_25dim_testdata[14]-pca_25dim_trainda
    euclidean_dis_dict[i] = euclidean_dis
{k: v for k, v in sorted(euclidean_dis_dict.items(), key=lambda item: item[
```

Out[98]:
```
{1785: 2.3239475490157107,
 15828: 2.3759585919411053,
 38393: 2.3907808687651775,
 28425: 2.4279019357528067,
 11556: 2.4585722467716264,
 40686: 2.4875521448049285,
 24121: 2.495255398543305,
 35706: 2.497079384876107,
 32337: 2.5004716998702214,
 21465: 2.507181669776802,
 20441: 2.512922420804266,
 32506: 2.5142099826202173,
 10716: 2.5268250183594354,
 14896: 2.536888882771333,
 30530: 2.541844419769657,
 39331: 2.547927192684104,
 28992: 2.549266732208925,
 6193: 2.551905476413147,
 2157: 2.5615696700086623,
```

In [97]:
```
fig=plt.figure(figsize=(8, 8))
fig.add_subplot(1, 4, 1)
plt.imshow(Image.fromarray(np.reshape(test_imgs[14], (20, 20))).resize([100
plt.title('Original image')
fig.add_subplot(1, 4, 2)
plt.imshow(Image.fromarray(np.reshape(train_imgs[1785], (20, 20))).resize([
plt.title('Similar Image 1')
fig.add_subplot(1, 4, 3)
plt.imshow(Image.fromarray(np.reshape(train_imgs[15828], (20, 20))).resize(
plt.title('Similar Image 2')
fig.add_subplot(1, 4, 4)
plt.imshow(Image.fromarray(np.reshape(train_imgs[38393], (20, 20))).resize(
plt.title('Similar Image 3')
plt.show()
```



Conclusion

In [103]:
```
print('Top 3 similar images from training set found using Autoencoder have
print(np.linalg.norm(test_imgs[0]-train_imgs[34761]))
print(np.linalg.norm(test_imgs[0]-train_imgs[39595]))
print(np.linalg.norm(test_imgs[0]-train_imgs[10576]))
print('\nTop 3 similar images from training set found using PCA have follow
print(np.linalg.norm(test_imgs[14]-train_imgs[1785]))
print(np.linalg.norm(test_imgs[14]-train_imgs[15828]))
print(np.linalg.norm(test_imgs[14]-train_imgs[38393]))
```

```
Top 3 similar images from training set found using Autoencoder have follo
wing euclidean distances:
2439.2027386012833
3029.317579917959
1347.0597611093579

Top 3 similar images from training set found using PCA have following euc
lidean distances:
3137.7241752582395
3808.3734060619636
3143.169260475802
```

In [ ]: