EE 511: Assignment #4

1. Neural Network warm-up problem

Sol 1(a) In this section labels are assigned based on the formula provided and then 500 samples are generated for training and validation set with an additional 1000 samples for test set. Table 1 summarizes the relative frequency in the training data.

Labels	Relative frequency
0	0.238
1	0.068
2	0.546
3	0.148

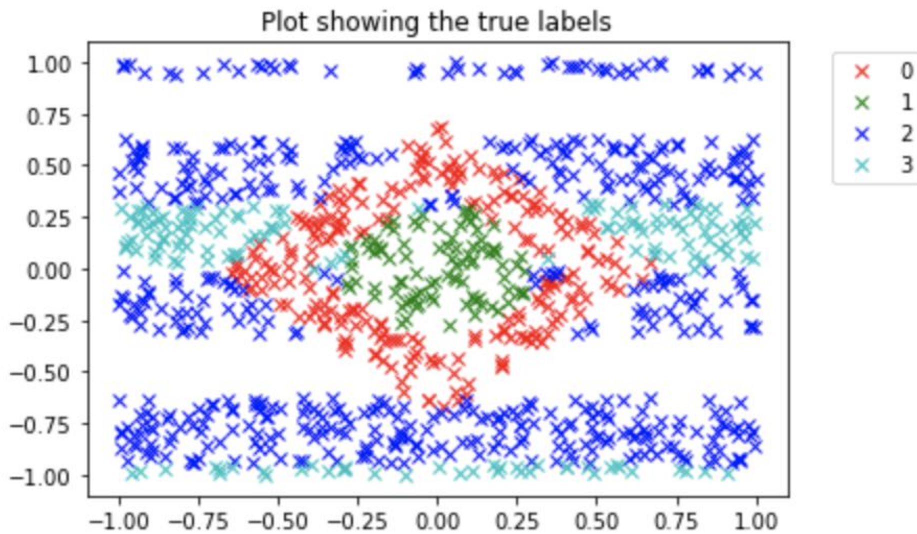
Sol 1 (b) I have explored following parameters while evaluating my classifier:

2. I explored stochastic gradient based optimizer (adam) vs quasi-Newton methods (lbfg) optimizer, and found adam performing better.
3. I explored different size values for the hidden layer (ex. 100, 500, 1000), and found 500 performing better.
4. I explored different values for max iterations and found 3000 was giving converging results.
5. I explored different values for the L2 penalty (regularization term) parameter 'alpha' (L2 regression) and found alpha=1e-05 working the best.
6. And I also found random_state = 1 was giving better results.

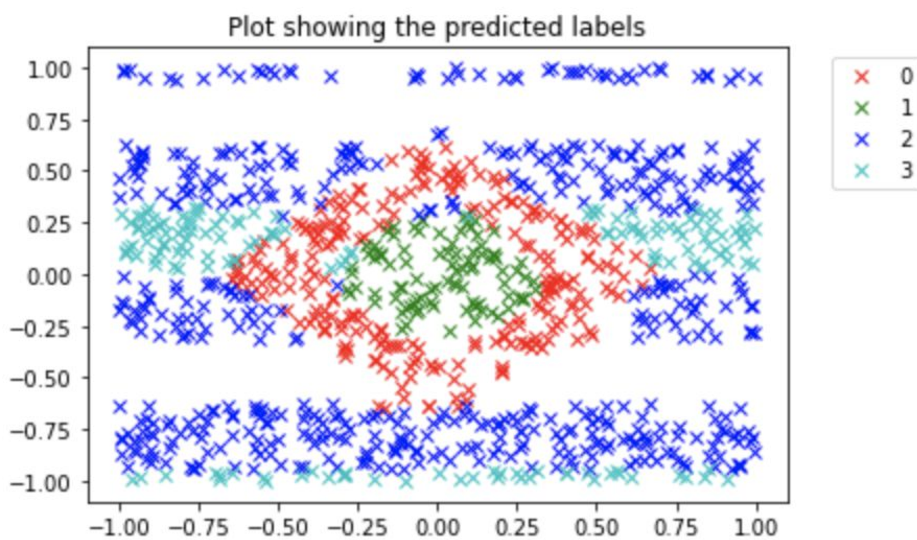
I chose a classifier with all the above parameters, which produced best results against the validation data.

Sol 1(c) The overall accuracy obtained is **0.96** on the 1000 samples of the test data.

Sol 1(d) The plots showing true labels (plot #1) and predicted labels (plot #2) are shown below. I observe the given data points are **not** linearly separable, for ex. label 3 is spread across the data space. In comparison both the plots (true and predicted) look very similar except for the few data points, this matches with what I expected from the classifier based on its overall accuracy.



Plot #1: Showing the true labels



Plot #2: Showing the true predicted labels

2. Autoencoder

2 (a) & (b) The downloaded images are resized to 20X20 pixels and further converted into grayscale, I additionally flatten the 2D image into 1D array for simplification for machine learning purposes.

2 (c) The principal component analysis (PCA) is used to reduce images down to 25 dimensions by using .transform function and then reconstructing them again using .inverse_transform function. The reconstruction error in terms of the squared error per pixel is mentioned below:
Validation data MSE with PCA: 37.8185285358682
Test data MSE with PCA: 37.760658467004845

2 (d) I have explored following parameters while evaluating my regressor:

1. I explored stochastic gradient based optimizer (adam) vs quasi-Newton methods (lbfg) optimizer, and found adam performing better.
2. I explored relu vs tanh activation functions, and found relu performing better.
3. I explored different number and size values for the hidden layers (ex. 100-25-100, 500-25-200, 1000-500-25-500-100), and found 1000-25-1000 performing better.
4. I explored different values for max iterations and found 50 was giving converging results.
5. I explored different values for the L2 penalty (regularization term) parameter 'alpha' (L2 regression) and found alpha=0.01 working the best.
6. And I also found random_state = 1 was giving better results.

I chose a regressor with all the above parameters, which produced best results against the validation data.

2 (e) My autoencoder regressor beats PCA. The MSE with the autoencoder for validation set and test set has been mentioned below:
Validation data MSE with Autoencoder: 27.867756576958335
Test data MSE with Autoencoder: 28.198441333470882

2(f) Image when the autoencoder beats the PCA

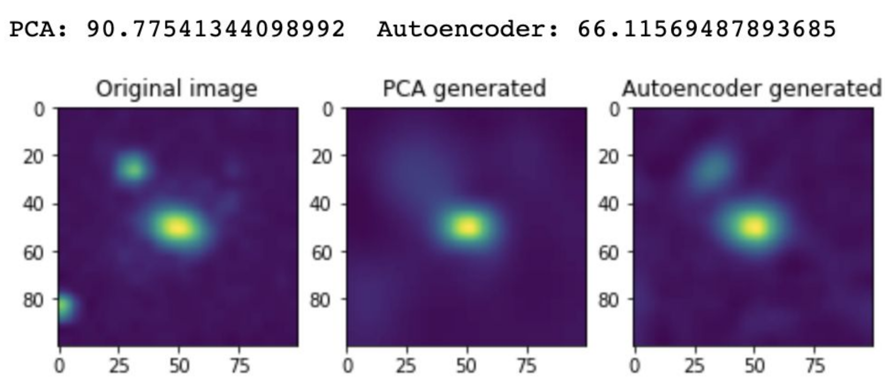
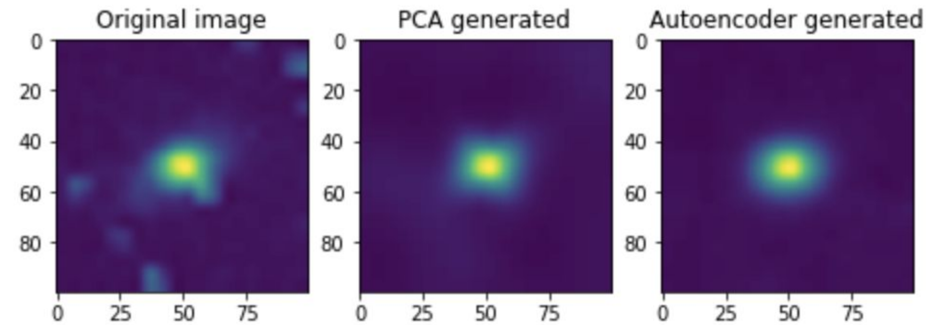
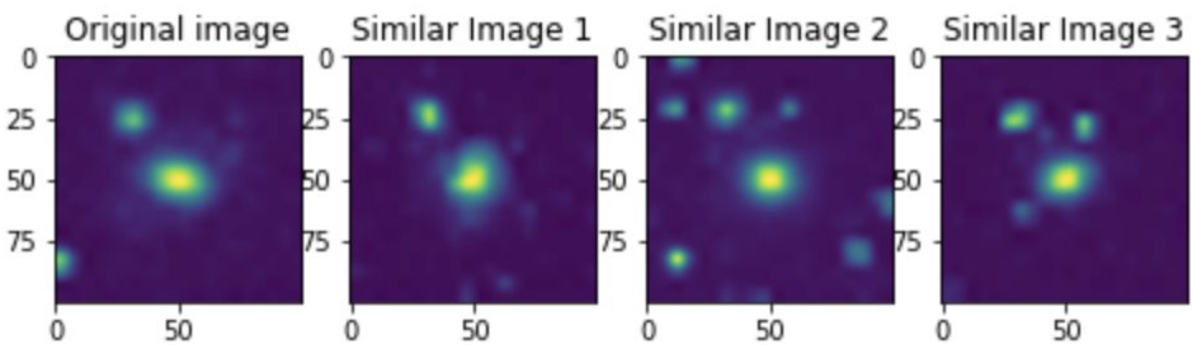


Image when the PCA beats an autoencoder

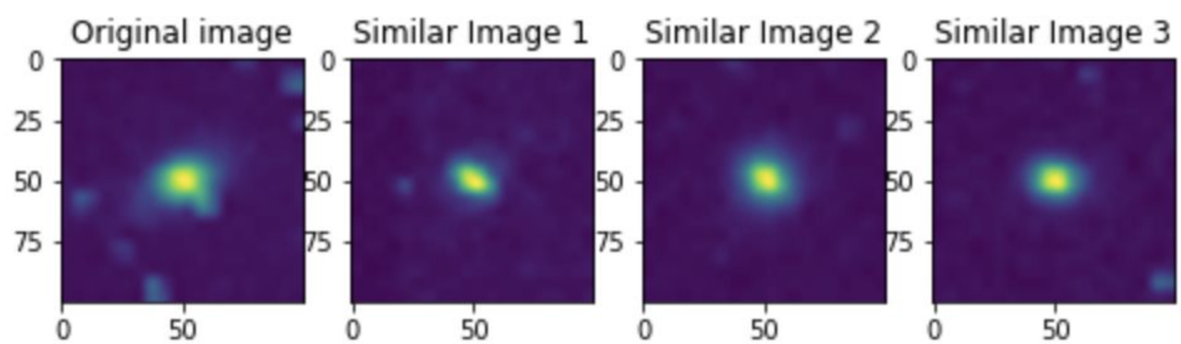
```
PCA: 30.986278124160147  Autoencoder: 37.403247468571784
```



2 (g) The 3 similar image from training set given test_imgs[0] using autoencoder



The 3 similar image from training set given test_imgs[14] using PCA



For the given problem an autoencoder worked better finding the similar images when compared to the PCA. Below is the comparison of euclidean distances for the images found using autoencoder and PCA:

Top 3 images	Autoencoder	PCA
Image 1	02439.2027386012833	3137.7241752582395
Image 2	3029.317579917959	3808.3734060619636
Image 3	1347.0597611093579	3143.169260475802

PART 1: Neural Network Warm Up Problem

In [1]:
```
import numpy as np
import pandas as pd
import random
import math
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report
```

In [2]:
```
def generate_samples(size):
    columns = ['label', 'x1', 'x2']
    data = []
    count = 0
    while (count < size):
        # generate random x1, x2
        x1 = random.uniform(-1,1)
        x2 = random.uniform(-1,1)
        label = 0
        ignore = False
        count = count+1

        # assign the labels
        if(0.4 < (abs(x1) + abs(x2)) < 0.7):
            label = 0
        elif(math.sqrt(x1*x1 + x2*x2) < 0.3):
            label = 1
        elif(math.sin(10*x2) < 0):
            label = 2
        elif(math.sin(5*x2) > 0):
            label = 3
        else:
            count = count-1
            ignore = True

        if(ignore != True):
            data.append([label, x1, x2])

    return pd.DataFrame(data = data, columns = columns)
```

Generate samples for train, validation and test

In [8]:
```
df_train = generate_samples(500)
df_validation = generate_samples(500)
df_test = generate_samples(1000)
```

Releative fequency of each class in test data

In [21]:
```
df_tmp = df_train.label.value_counts()
print('Relative Frequency in training data\n',df_tmp / len(df_train.label),
```

```
Relative Frequency in training data
 2    0.546
0    0.238
3    0.148
1    0.068
Name: label, dtype: float64
```

Training a neural network with one hidden layer

In [5]:
```
X_train = df_train[["x1","x2"]]
y_train = df_train.label
X_validation = df_validation[["x1","x2"]]
y_validation = df_validation.label

alphas = np.logspace(-6, -2, 5)

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

for alpha in alphas:
    classifiers.append(MLPClassifier(solver='adam', alpha=alpha, hidden_lay
                                     max_iter=3000, random_state=1))
    classifiers.append(MLPClassifier(solver='lbfgs', alpha=alpha, hidden_la
                                     max_iter=3000, random_state=1))

# Iterate over classifiers to find the score using validation data
best_clf = classifiers[0]
best_score = 0
for clf in classifiers:
    clf.fit(X_train, y_train)
    score = clf.score(X_validation, y_validation)
    if(score > best_score):
        best_score = score
        best_clf = clf
    print(clf, 'score=', score)
```

```
MLPClassifier(alpha=1e-06, hidden_layer_sizes=[500], max_iter=3000,
              random_state=1) score= 0.94
MLPClassifier(alpha=1e-06, hidden_layer_sizes=[500], max_iter=3000,
              random_state=1, solver='lbfgs') score= 0.94
MLPClassifier(alpha=1e-05, hidden_layer_sizes=[500], max_iter=3000,
              random_state=1) score= 0.94
MLPClassifier(alpha=1e-05, hidden_layer_sizes=[500], max_iter=3000,
              random_state=1, solver='lbfgs') score= 0.956
MLPClassifier(hidden_layer_sizes=[500], max_iter=3000, random_state=1) sc
ore= 0.942
MLPClassifier(hidden_layer_sizes=[500], max_iter=3000, random_state=1,
              solver='lbfgs') score= 0.938
MLPClassifier(alpha=0.001, hidden_layer_sizes=[500], max_iter=3000,
              random_state=1) score= 0.94
MLPClassifier(alpha=0.001, hidden_layer_sizes=[500], max_iter=3000,
              random_state=1, solver='lbfgs') score= 0.942
MLPClassifier(alpha=0.01, hidden_layer_sizes=[500], max_iter=3000,
              random_state=1) score= 0.936
MLPClassifier(alpha=0.01, hidden_layer_sizes=[500], max_iter=3000,
              random_state=1, solver='lbfgs') score= 0.942
```

In [10]:
```
best_clf
```

Out[10]:
```
MLPClassifier(alpha=1e-05, hidden_layer_sizes=[500], max_iter=3000,
              random_state=1, solver='lbfgs')
```

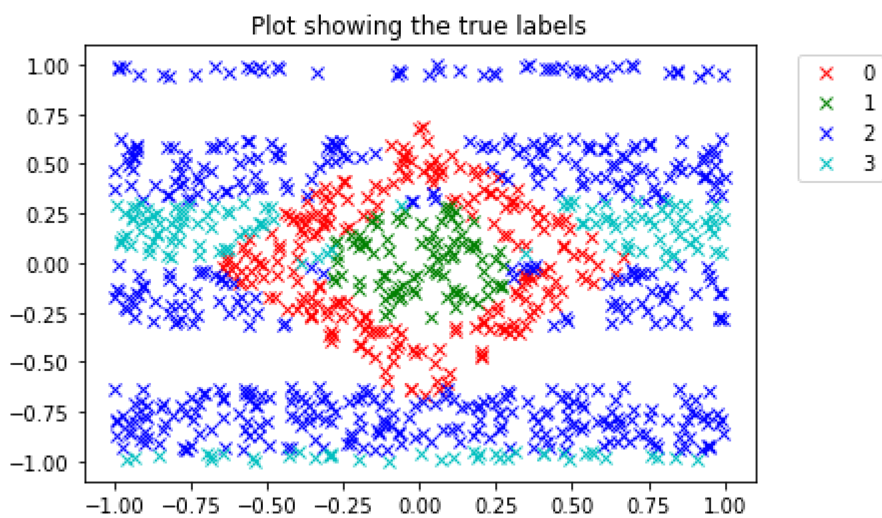
overall accuracy of the classifier on the test data

In [51]:
```
X_test = df_test[["x1","x2"]]
y_test = df_test.label
y_pred = best_clf.predict(X_test)
print(classification_report(y_test, y_pred))
```

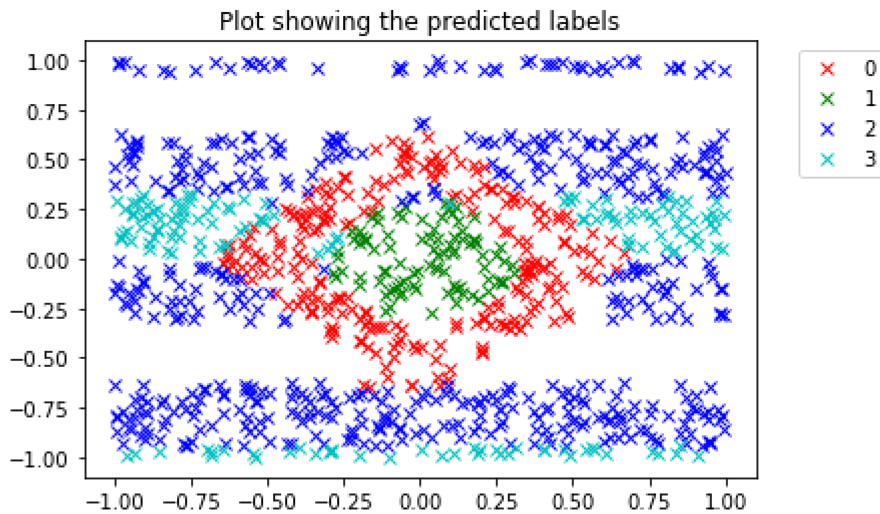
```
              precision    recall  f1-score   support

           0       0.93      0.97      0.95       232
           1       0.98      0.97      0.97        92
           2       0.98      0.97      0.97       522
           3       0.95      0.94      0.95       154

    accuracy                           0.96      1000
   macro avg       0.96      0.96      0.96      1000
weighted avg       0.96      0.96      0.96      1000
```

two plots of the test samples

In [56]:
```
PlotIt(X_test.to_numpy(), y_test, 'Plot showing the true labels')
```



In [57]:
```
PlotIt(X_test.to_numpy(), y_pred, 'Plot showing the predicted labels')
```



In [55]:
```
import matplotlib.pyplot as plt
def PlotIt(data, labels, title):
    xs = data[:, 0]
    ys = data[:, 1]
    colors = ['r', 'g', 'b', 'c']
    for i in range(4):
        idx = labels == i
        plt.plot(xs[idx], ys[idx], 'x', color=colors[i], label = i)
    plt.title(title)
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.show()
```

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
```

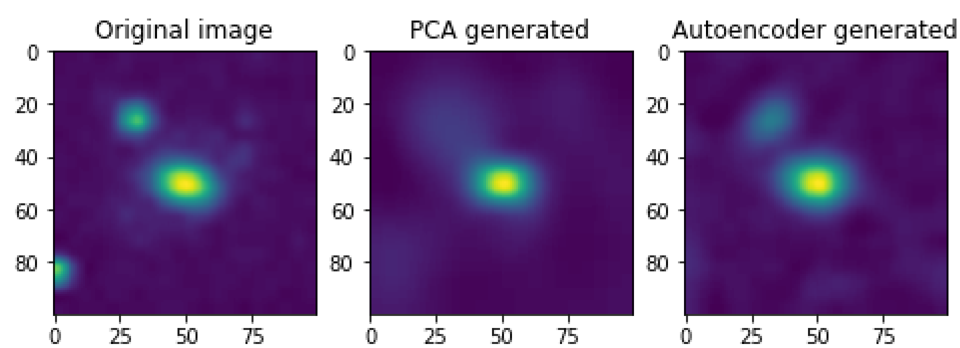
```
Validation data MSE with Autoencoder: 27.867756576958335
Test data MSE with Autoencoder: 28.198441333470882
```

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()
```

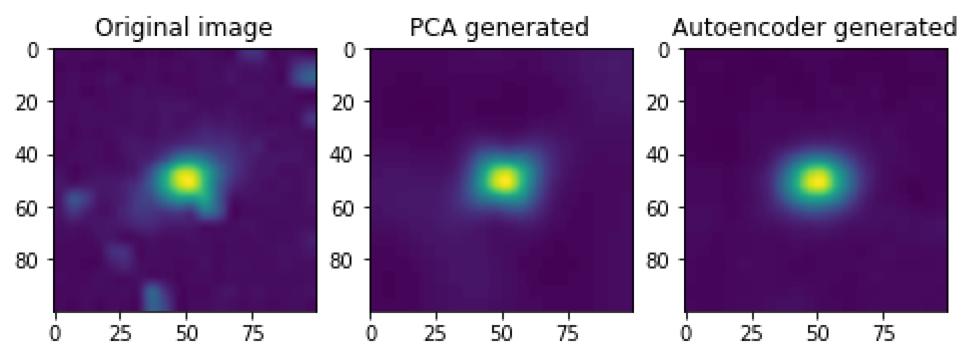
```
PCA: 90.77541344098992  Autoencoder: 66.11569487893685
```



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
```

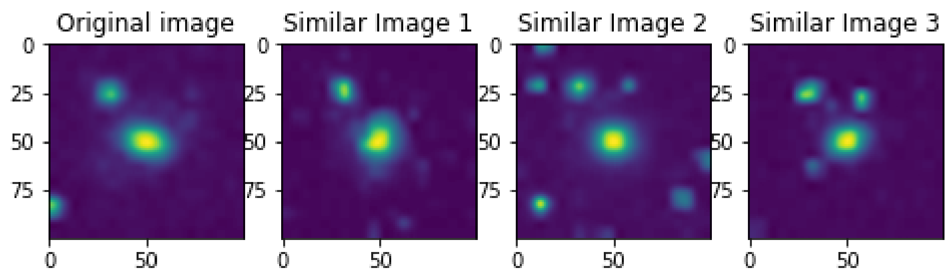
Out[80]:
```
(40872, 25)
```

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()
```

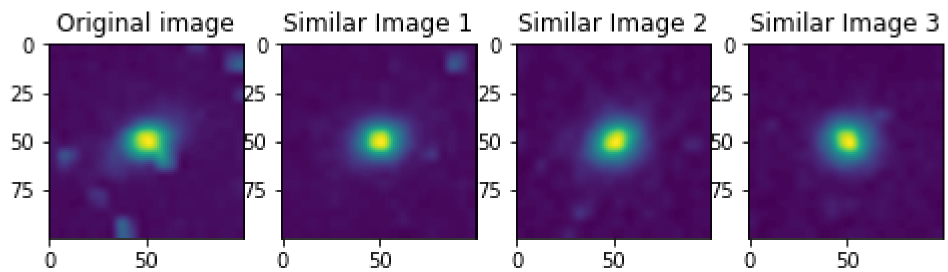


Find 3 similar image from training set given test_imgs[14]

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 [ ]: