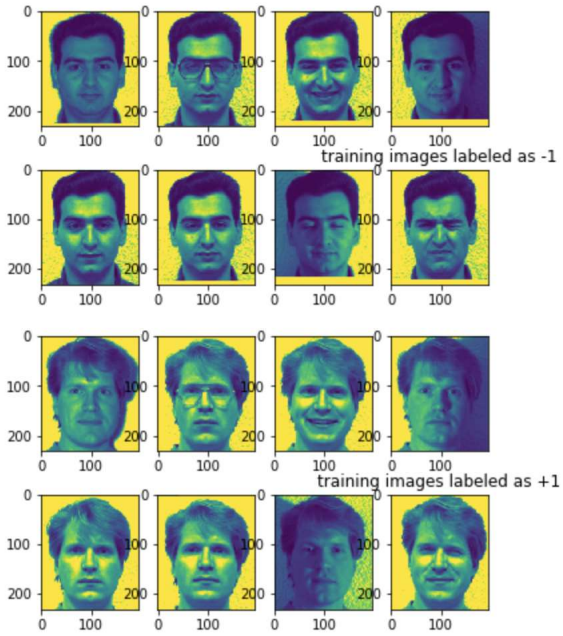
In [1]:
```
# 1. read in training images as numpy arrays, create the labels for each images
# 2. visualize images in the training sets
#import matplotlib.image as mpimg
import matplotlib.pyplot as plt
import cv2
import glob
import numpy as np
images_neg = [cv2.imread(file,0) for file in glob.glob('faces/-1/training/*bmp')] # read images labeled as -1
images_neg = np.asarray(images_neg)                                               # put images into array
y_neg = -1 * np.ones(len(images_neg))                                               # stack the labels as vector

images_pos = [cv2.imread(file,0) for file in glob.glob('faces/1/training/*bmp')] # read images labeled as +1
images_pos = np.asarray(images_pos)                                               # put images into arrays
y_pos = 1 * np.ones(len(images_pos))                                               # stack the labels as vector

# plot images labeled as -1
neg_len = len(images_neg)
plt.figure(1)
for i in range(neg_len):
    plt.subplot(241 + i)
    plt.imshow(images_neg[i])
plt.title("training images labeled as -1");
plt.show()

# plot images labeled as +1
pos_len = len(images_pos)
plt.figure(2)
for i in range(pos_len):
    plt.subplot(241 + i)
    plt.imshow(images_pos[i])
plt.title("training images labeled as +1");
plt.show()
```



In [9]:
```
# 3. vectorize the 2D images as vectors, normalize the vectors as zero mean and unit variance
# vectorize all 2D images in the training dataset, then normalize the data to remove the mean and scale the variance.
# Hint: use preprocessing.scale() to conduct mean removal and variance scaling, and check if the images are have zero
# means and unit variance by calculating images_scaled.mean() and images_scaled.std()
from sklearn import preprocessing

images_all = np.append(images_neg, images_pos, axis = 0)
y = np.append(y_neg, y_pos)
print(y)

i, rows, cols = images_all.shape
images_all = np.reshape(images_all, (i, rows*cols))

images_scaled = preprocessing.scale(images_all)
print(images_scaled.mean())
print(images_scaled.std())
```

```
[-1. -1. -1. -1. -1. -1. -1. -1.  1.  1.  1.  1.  1.  1.  1.  1.]
-1.3881176766986084e-17
0.9998556894429314
```

```
C:\Users\sy\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\utils\validation.py:475: DataConversionWarning: Data wi
```

```
th input dtype uint8 was converted to float64 by the scale function.
  warnings.warn(msg, DataConversionWarning)
```

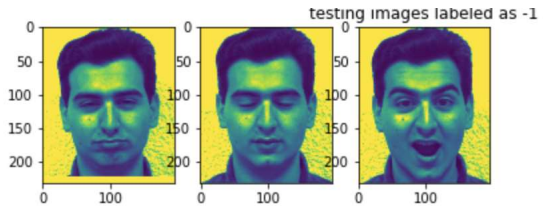
In [3]:
```
# define function of gradient descent with input parameters as X (images), y (lables), alpha (stepsize), w(model
# parameter in linear regression)
def gradient_descent(X,y,alpha,w):
    rows, cols = X.shape
    grad = np.zeros(cols + 1)

    for i in range(rows):
        x = np.append(X[i, :], [1], axis = 0)
        grad = grad + x * (np.inner(w, x) - y[i])

    grad = grad + 2*w
    w = w - alpha*grad
    return w
```

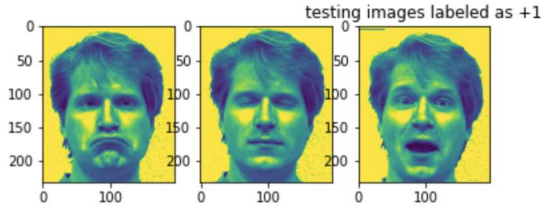
In [4]:
```
# define the objective function, evaluate it at each iteration of gradient descent to check if the terminate criterion
# is met
from numpy import linalg as LA
def objective_function(X,y,w):
    num_images = len(X)
    of = 0

    for i in range(num_images):
        x = np.append(X[i, :], [1], axis = 0)
        of = of + 0.5 * (np.inner(x, w) - y[i])**2

    of = of + LA.norm(w)**2
    return of
```

In [18]:
```
# implement the training phase to learn parameter w,
import math
alpha = 0.000001
rows, cols = images_scaled.shape
w = np.zeros(cols + 1)
w = np.random.rand(cols + 1) #start with a random weight

prev_of = objective_function(images_scaled, y, w)
print(prev_of)

residual = math.inf #infinity
w = np.zeros(cols + 1)
while residual > 0.1:
    w = gradient_descent(images_scaled, y, alpha, w)
    new_of = objective_function(images_scaled, y, w)
    residual = (prev_of - new_of) / prev_of
    prev_of = new_of
    print(new_of, residual)
```

```
886242971.2665131
5.454026386369931 0.9999999938459018
3.825738421839436 0.29854787072532746
2.7654755491387317 0.27713940572835194
2.0617661283957975 0.25446235493280517
1.5853335234617072 0.2310798486658568
1.2561691300773623 0.20763100540862037
1.0240680811677167 0.18476894818721695
0.857049261917243 0.16309347232073362
0.7344124676170236 0.14309188485371008
0.642537298718665 0.1251002303875225
0.5723135937780682 0.10929125061009769
0.5175503206819551 0.09568752811653339
```

In [19]:
```
# read in training images as numpy arrays
# visualize images in the testing sets
images_neg_test = np.asarray([cv2.imread(file,0) for file in glob.glob('faces/-1/testing/*bmp')])
neg_len = len(images_neg_test)
plt.figure(3)
for i in range(neg_len):
    plt.subplot(131 + i)
    plt.imshow(images_neg_test[i])
plt.title("testing images labeled as -1");
plt.show()

images_pos_test = np.asarray([cv2.imread(file,0) for file in glob.glob('faces/1/testing/*bmp')])
pos_len = len(images_pos_test)
plt.figure(4)
for i in range(pos_len):
    plt.subplot(131 + i)
    plt.imshow(images_pos_test[i])
plt.title("testing images labeled as +1");
plt.show()
```





In [20]:
```
# testing on images labelled as -1, print the y value of each testing image and the corresponding labels
im, rows, cols = images_neg_test.shape
images_neg_test = np.reshape(images_neg_test, (im, rows*cols))
for i in range(im):
    x = np.append(images_neg_test[i], [1], axis = 0)
    y_val = np.inner(w, x)
    label = np.sign(y_val)
    print(label, y_val)
```

```
-1.0 -70.83091533816233
-1.0 -59.66241698225393
-1.0 -54.574426095779145
```

In [21]:
```
# testing on images labelled as 1, print the y value of each testing image and the corresponding labels
im, rows, cols = images_pos_test.shape
images_pos_test = np.reshape(images_pos_test, (im, rows*cols))
for i in range(im):
    x = np.append(images_pos_test[i], [1], axis = 0)
    y_val = np.inner(w, x)
    label = np.sign(y_val)
    print(label, y_val)
```

```
1.0 35.19530198325555
1.0 34.678785607552854
1.0 42.31734078898234
```

In [ ]: