In [4]:

```
#Untouched
from struct import unpack
import gzip
import numpy as np
from numpy import zeros, uint8, float32
from pylab import imshow, show, cm, savefig
import cv2
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
def get_labeled_data(imagefile, labelfile):
    """Read input-vector (image) and target class (label, 0-9) and return
       it as list of tuples.
    """
    # Open the images with gzip in read binary mode
    images = gzip.open(imagefile, 'rb')
    labels = gzip.open(labelfile, 'rb')

    # Read the binary data

    # We have to get big endian unsigned int. So we need '>I'

    # Get metadata for images
    images.read(4)  # skip the magic_number
    number_of_images = images.read(4)
    number_of_images = unpack('>I', number_of_images)[0]
    rows = images.read(4)
    rows = unpack('>I', rows)[0]
    cols = images.read(4)
    cols = unpack('>I', cols)[0]
    #print (rows , cols)
    # Get metadata for labels
    labels.read(4)  # skip the magic_number
    N = labels.read(4)
    N = unpack('>I', N)[0]
    if number_of_images != N:
        raise Exception('number of labels did not match the number of images')
    # Get the data
    x = zeros((N, rows, cols), dtype=float32)  # Initialize numpy array
    y = zeros((N, 1), dtype=uint8)  # Initialize numpy array
    totalfeatures = rows * cols
    zz = zeros ((N, totalfeatures), dtype=float32)
    for i in range(N):
#         if i % 1000 == 0:
#             print("i: %i" % i)
        for row in range(rows):
            for col in range(cols):
                tmp_pixel = images.read(1)  # Just a single byte
                tmp_pixel = unpack('>B', tmp_pixel)[0]
                #Thresholding
                if (tmp_pixel < 130) :
                    tmp_pixel = 0
                elif (tmp_pixel > 132) :
                    tmp_pixel = 255
                else:
                    tmp_pixel = tmp_pixel
                x[i][row][col] = tmp_pixel
                index = row * cols +col;
                zz[i][index] = tmp_pixel

        tmp_label = labels.read(1)
        y[i] = unpack('>B', tmp_label)[0]

    # Untouched imageset
    zznew = zz[1:48000,:]
    ynew = y[1:48000].ravel()
    youtput = y[48001:60000].ravel()

    # Randomforest classifier - Untouched - Depth 4 (n=10)
    rfc=RandomForestClassifier(max_depth = 4)
    rfc.fit(zznew,ynew)
    pred_val = rfc.predict(zz[48001:60000])
    Acc_test_4 = accuracy_score(youtput,pred_val)*100
    print('Randomforest classifier - Untouched - Depth 4 (n=10)')
    print(Acc_test_4)

    # Randomforest classifier - Untouched - Depth 16 (n=10)
    rfc16=RandomForestClassifier(max_depth = 16)
    rfc16.fit(zznew,ynew)
    pred_val16 = rfc16.predict(zz[48001:60000])
    Acc_test_16 = accuracy_score(youtput,pred_val16)*100
    print('Randomforest classifier - Untouched - Depth 16 (n=10)')
    print(Acc_test_16)

    # Randomforest classifier - Untouched - Depth 4 (n=30)
    rfc30=RandomForestClassifier(max_depth = 4,n_estimators=30)
    rfc30.fit(zznew,ynew)
    pred_val30 = rfc30.predict(zz[48001:60000])
    Acc_test_430 = accuracy_score(youtput,pred_val30)*100
    print('Randomforest classifier - Untouched - Depth 4 (n=30)')
    print(Acc_test_430)

    # Randomforest classifier - Untouched - Depth 16 (n=30)
    rfc3016=RandomForestClassifier(max_depth = 16,n_estimators=30)
    rfc3016.fit(zznew,ynew)
    pred_val3016 = rfc3016.predict(zz[48001:60000])
    Acc_test_3016 = accuracy_score(youtput,pred_val3016)*100
    print('Randomforest classifier - Untouched - Depth 16 (n=30)')
    print(Acc_test_3016)

    return (None)
```

In [2]:

```
get_labeled_data('train-images-idx3-ubyte.gz', 'train-labels-idx1-ubyte.gz')
```

```
60000
784
Randomforest classifier - Untouched - Depth 4 (n=10)
69.8224852071006
Randomforest classifier - Untouched - Depth 16 (n=10)
93.59946662221851
Randomforest classifier - Untouched - Depth 4 (n=30)
74.51454287857321
Randomforest classifier - Untouched - Depth 16 (n=30)
95.66630552546046
```

In [1]:

```
#Stretched
from struct import unpack
import gzip
import numpy as np
from numpy import zeros, uint8, float32
from pylab import imshow, show, cm, savefig
import cv2
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
def get_labeled_data(imagefile, labelfile):
    """Read input-vector (image) and target class (label, 0-9) and return
       it as list of tuples.
    """
    # Open the images with gzip in read binary mode
    images = gzip.open(imagefile, 'rb')
    labels = gzip.open(labelfile, 'rb')

    # Read the binary data

    # We have to get big endian unsigned int. So we need '>I'

    # Get metadata for images
    images.read(4)  # skip the magic_number
    number_of_images = images.read(4)
    number_of_images = unpack('>I', number_of_images)[0]
    rows = images.read(4)
    rows = unpack('>I', rows)[0]
    cols = images.read(4)
    cols = unpack('>I', cols)[0]
    # Get metadata for labels
    labels.read(4)  # skip the magic_number
    N = labels.read(4)
    N = unpack('>I', N)[0]
    if number_of_images != N:
        raise Exception('number of labels did not match the number of images')

    # Get the data
    x = zeros((N, rows, cols), dtype=float32)  # Initialize numpy array
    y = zeros((N, 1), dtype=uint8)  # Initialize numpy array
    totalfeatures = rows * cols
    for i in range(N):
        for row in range(rows):
            for col in range(cols):
                tmp_pixel = images.read(1)  # Just a single byte
                tmp_pixel = unpack('>B', tmp_pixel)[0]
                if (tmp_pixel < 130) :
                    tmp_pixel = 0
                elif (tmp_pixel > 132) :
                    tmp_pixel = 255
                else:
                    tmp_pixel = tmp_pixel
                x[i][row][col] = tmp_pixel

        tmp_label = labels.read(1)
        y[i] = unpack('>B', tmp_label)[0]
```

```
def find_x_min(img):
    a = True
    for col in range(cols):
        if (a == True):
            for row in range(rows):
                if (a == True):
                    if(x[img][row][col] == 255):
                        x_min = (col)
                        a = False
                        return(x_min)
                        break
def find_y_max(img):
    a = True
    for row in range(rows):
        if (a == True):
            for col in range(cols):
                if (a == True):
                    if(x[img][row][col] == 255):
                        y_max = (row)
                        a = False
                        return(y_max)
                        break
def find_x_max(img):
    a = True
    for col in range(cols-1, -1, -1):
        if (a == True):
            for row in range(rows):
                if (a == True):
                    if(x[img][row][col] == 255):
                        x_max = (col)
                        a = False
                        return(x_max)
                        break
def find_y_min(img):
    a = True
    for row in range(rows-1, -1, -1):
        if (a == True):
            for col in range(cols):
                if (a == True):
                    if(x[img][row][col] == 255):
                        y_min = (row)
                        a = False
                        return(y_min)
                        break

x_min = 0
x_max = 0
y_min = 0
y_max = 0
zz = zeros ((N, 400), dtype=float32)
nostretchset = zeros ((N, 20,20), dtype=float32)

for i in range(N):
    x_min = find_x_min(i)
    x_max = find_x_max(i)
    y_min = find_y_min(i)
    y_max = find_y_max(i)

    if (x_min > x_max):
        temp = x_min
        x_min = x_max
        x_max = temp
    if (y_min > y_max):
        temp = y_min
        y_min = y_max
        y_max = temp
    nostretch = x[i,x_min:x_max,y_min:y_max]
    imo = cv2.resize(nostretch, (20,20), interpolation=cv2.INTER_NEAREST)
    nostretchset[i] = imo
    imshow(imo)

# Convert 2D to 1D for training n testing
for i in range(N):
    for srow in range(20):
        for scol in range(20):
            index = srow*20+scol
            zz[i][index]=nostretchset[i][srow][scol]

zznew = zz[1:48000,:]
ynew = y[1:48000].ravel()
youtput = y[48001:60000].ravel()

# Randomforest classifier - Stretched - Depth 4 (n=10)
rfc=RandomForestClassifier(max_depth = 4)
rfc.fit(zznew,ynew)
pred_val = rfc.predict(zz[48001:60000])
Acc_test_4 = accuracy_score(youtput,pred_val)*100
print('Randomforest classifier - Stretched - Depth 4 (n=10)')
print(Acc_test_4)

# Randomforest classifier - Stretched - Depth 16 (n=10)
rfc16=RandomForestClassifier(max_depth = 16)
rfc16.fit(zznew,ynew)
pred_val16 = rfc16.predict(zz[48001:60000])
Acc_test_16 = accuracy_score(youtput,pred_val16)*100
print('Randomforest classifier - Stretched - Depth 16 (n=10)')
print(Acc_test_16)

# Randomforest classifier - Stretched - Depth 4 (n=30)
rfc30=RandomForestClassifier(max_depth = 4,n_estimators=30)
rfc30.fit(zznew,ynew)
pred_val30 = rfc30.predict(zz[48001:60000])
Acc_test_430 = accuracy_score(youtput,pred_val30)*100
print('Randomforest classifier - Stretched - Depth 4 (n=30)')
print(Acc_test_430)

# Randomforest classifier - Stretched - Depth 16 (n=30)
rfc3016=RandomForestClassifier(max_depth = 16,n_estimators=30)
rfc3016.fit(zznew,ynew)
pred_val3016 = rfc3016.predict(zz[48001:60000])
Acc_test_3016 = accuracy_score(youtput,pred_val3016)*100
print('Randomforest classifier - Stretched - Depth 16 (n=30)')
```

```
    print(Acc_test_3016)

    return ('Code run complete!')
```

In [ ]:

```
get_labeled_data('train-images-idx3-ubyte.gz', 'train-labels-idx1-ubyte.gz')
```

```
Randomforest classifier - Stretched - Depth 4 (n=10)
59.16326360530044
Randomforest classifier - Stretched - Depth 16 (n=10)
87.7906492207684
Randomforest classifier - Stretched - Depth 4 (n=30)
61.35511292607717
Randomforest classifier - Stretched - Depth 16 (n=30)
90.04917076423035
```

Out[ ]:

```
'Code run complete!'
```