In [3]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
from skimage import data, color
from skimage.transform import rescale, resize, downscale_local_mean
import numpy as np

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
from keras.layers import Dense, Activation, Convolution2D, MaxPooling2D, Fl
from sklearn.metrics import confusion_matrix
```

```
Using TensorFlow backend.
```

In [4]:
```
img5 = mpimg.imread('5.JPG')
image5 = color.rgb2grey(img5)
image_resized5 = resize(image5, (28, 28), anti_aliasing=False) # False maxi
image_inverse5 = (1-image_resized5)
image_final5 = (image_inverse5-image_inverse5.min()) * 16 / (image_inverse5
```

In [5]:
```
img2 = mpimg.imread('2.JPG')
image2 = color.rgb2grey(img2)
image_resized2 = resize(image2, (28,28), anti_aliasing=False) # False maxim
image_inverse2 = (1-image_resized2)
image_final2 = (image_inverse2-image_inverse2.min()) * 16 / (image_inverse2
```

In [6]:
```
img1 = mpimg.imread('1.JPG')
image1 = color.rgb2grey(img1)
image_resized1 = resize(image1, (28, 28), anti_aliasing=False) # False maxi
image_inverse1 = (1-image_resized1)
image_final1 = (image_inverse1-image_inverse1.min()) * 16 / (image_inverse1
```

In [7]:
```
img0 = mpimg.imread('0.JPG')
image0 = color.rgb2grey(img0)
image_resized0 = resize(image0, (28, 28), anti_aliasing=False) # False maxi
image_inverse0 = (1-image_resized0)
image_final0 = (image_inverse0-image_inverse0.min()) * 16 / (image_inverse0
```

In [8]:
```
img11 = mpimg.imread('11.JPG')
image11 = color.rgb2grey(img11)
image_resized11 = resize(image11, (28, 28), anti_aliasing=False) # False ma
image_inverse11 = (1-image_resized11)
image_final11 = (image_inverse11-image_inverse11.min()) * 16 / (image_inver
```

In [9]:
```
img22 = mpimg.imread('22.JPG')
image22 = color.rgb2grey(img22)
image_resized22 = resize(image22, (28, 28), anti_aliasing=False) # False ma
image_inverse22 = (1-image_resized22)
image_final22 = (image_inverse22-image_inverse22.min()) * 16 / (image_inver
```

In [10]:
```
img00 = mpimg.imread('00.JPG')
image00 = color.rgb2grey(img00)
image_resized00 = resize(image00, (28, 28), anti_aliasing=False) # False ma
image_inverse00 = (1-image_resized00)
image_final00 = (image_inverse00-image_inverse00.min()) * 16 / (image_inver
```

In [11]:
```
img000 = mpimg.imread('000.JPG')
image000 = color.rgb2grey(img000)
image_resized000 = resize(image000, (28, 28), anti_aliasing=False) # False
image_inverse000 = (1-image_resized000)
image_final000 = (image_inverse000-image_inverse000.min()) * 16 / (image_in
```

In [12]:
```
img111 = mpimg.imread('111.JPG')
image111 = color.rgb2grey(img111)
image_resized111 = resize(image111, (28, 28), anti_aliasing=False) # False
image_inverse111 = (1-image_resized111)
image_final111 = (image_inverse111-image_inverse111.min()) * 16 / (image_in
```

In [13]:
```
img222 = mpimg.imread('222.JPG')
image222 = color.rgb2grey(img222)
image_resized222 = resize(image222, (28, 28), anti_aliasing=False) # False
image_inverse222 = (1-image_resized222)
image_final222 = (image_inverse222-image_inverse222.min()) * 16 / (image_in
```

In [14]:
```
img0000 = mpimg.imread('0000.JPG')
image0000 = color.rgb2grey(img0000)
image_resized0000 = resize(image0000, (28, 28), anti_aliasing=False) # Fals
image_inverse0000 = (1-image_resized0000)
image_final0000 = (image_inverse0000-image_inverse0000.min()) * 16 / (image
```

In [15]:
```
img6 = mpimg.imread('6.JPG')
image6 = color.rgb2grey(img6)
image_resized6 = resize(image6, (28, 28), anti_aliasing=False) # False maxi
image_inverse6 = (1-image_resized6)
image_final6 = (image_inverse6-image_inverse6.min()) * 16 / (image_inverse6
```

In [16]:
```
img7 = mpimg.imread('7.JPG')
image7 = color.rgb2grey(img7)
image_resized7 = resize(image7, (28,28), anti_aliasing=False) # False maxim
image_inverse7 = (1-image_resized7)
image_final7 = (image_inverse7-image_inverse7.min()) * 16 / (image_inverse7
```

In [17]:
```
img55 = mpimg.imread('55.JPG')
image55 = color.rgb2grey(img55)
image_resized55 = resize(image55, (28, 28), anti_aliasing=False) # False ma
image_inverse55 = (1-image_resized55)
image_final55 = (image_inverse55-image_inverse55.min()) * 16 / (image_inver
```

In [18]:
```
img1111 = mpimg.imread('1111.JPG')
image1111 = color.rgb2grey(img1111)
image_resized1111 = resize(image1111, (28, 28), anti_aliasing=False) # Fals
image_inverse1111 = (1-image_resized1111)
image_final1111 = (image_inverse1111-image_inverse1111.min()) * 16 / (image
```

In [19]:
```
img11111 = mpimg.imread('11111.JPG')
image11111 = color.rgb2grey(img11111)
image_resized11111 = resize(image11111, (28, 28), anti_aliasing=False) # Fa
image_inverse11111 = (1-image_resized11111)
image_final11111 = (image_inverse11111-image_inverse11111.min()) * 16 / (im
```

In [20]:
```
img2222 = mpimg.imread('2222.JPG')
image2222 = color.rgb2grey(img2222)
image_resized2222 = resize(image2222, (28, 28), anti_aliasing=False) # Fals
image_inverse2222 = (1-image_resized2222)
image_final2222 = (image_inverse2222-image_inverse2222.min()) * 16 / (image
```

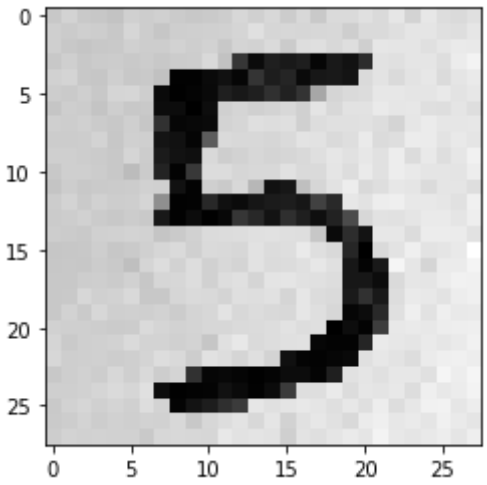
In [21]:
```
img8 = mpimg.imread('8.JPG')
image8 = color.rgb2grey(img8)
image_resized8 = resize(image8, (28, 28), anti_aliasing=False) # False maxi
image_inverse8 = (1-image_resized8)
image_final8 = (image_inverse8-image_inverse8.min()) * 16 / (image_inverse8
```

In [22]:
```
img77 = mpimg.imread('777.JPG')
image77 = color.rgb2grey(img77)
image_resized77 = resize(image77, (28, 28), anti_aliasing=False) # False ma
image_inverse77 = (1-image_resized77)
image_final77 = (image_inverse77-image_inverse77.min()) * 16 / (image_inver
```

In [23]:
```
img66 = mpimg.imread('666.JPG')
image66 = color.rgb2grey(img66)
image_resized66 = resize(image66, (28, 28), anti_aliasing=False) # False ma
image_inverse66 = (1-image_resized66)
image_final66 = (image_inverse66-image_inverse66.min()) * 16 / (image_inver
```

In [24]:
```
plt.imshow(image_final5, cmap='Greys')
```

Out[24]:
```
<matplotlib.image.AxesImage at 0x112875590>
```



In [25]:
```
# Eventually, flattern the image into a 1-D numpy array. For many images, y
num_image_5 = image_final5.flatten()
num_image_2 = image_final2.flatten()
num_image_1 = image_final1.flatten()
num_image_0 = image_final0.flatten()
num_image_11 = image_final11.flatten()
num_image_22 = image_final22.flatten()
num_image_00 = image_final00.flatten()
num_image_000 = image_final000.flatten()
num_image_111 = image_final111.flatten()
num_image_222 = image_final222.flatten()
num_image_0000 = image_final0000.flatten()
num_image_6 = image_final6.flatten()
num_image_7 = image_final7.flatten()
num_image_55 = image_final55.flatten()
num_image_1111 = image_final1111.flatten()
num_image_11111 = image_final11111.flatten()
num_image_2222 = image_final2222.flatten()
num_image_8 = image_final8.flatten()
num_image_77 = image_final77.flatten()
num_image_66 = image_final66.flatten()

num_image=[num_image_5, num_image_2, num_image_1, num_image_0, num_image_11

num_real = [5, 2, 1, 0, 1, 2, 0, 0, 1, 2, 0, 6, 7, 5, 1, 1, 2, 8, 7, 6]
```

In [ ]:
```

```

In [26]:
```
# Baseline MLP for MNIST dataset
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import np_utils
from keras.layers import Dense, Activation, Convolution2D, MaxPooling2D, Fl

# load data
(X_train, y_train), (X_test, y_test) = mnist.load_data()


X_train = X_train.reshape(-1,1,28,28)
X_test = np.array(num_image)
X_test = X_test.reshape(-1,1,28,28)

y_train = np_utils.to_categorical(y_train, num_classes=10)
y_test = np.array(num_real)
y_test = np_utils.to_categorical(y_test, num_classes=10)

model = Sequential()
#Convolution Layer1
model.add(Convolution2D(
    batch_input_shape=(32, 1, 28, 28),
    filters=32,
    kernel_size=5,
    strides=1,
    padding='same',
    ))
model.add(Activation('relu'))
#Pooling Layer1
model.add(MaxPooling2D(
    pool_size=2,
    strides=2,
    padding='same',
    ))

#Convolution Layer2
model.add(Convolution2D(64,5,
                        strides=1,
                        padding='same',
                        data_format='channels_last'
                        ))
model.add(Activation('relu'))
#Pooling Layer2
model.add(MaxPooling2D(2,
                       2,
                       'same',
                       data_format='channels_last'
                       ))

#Fully Connected Layer1
model.add(Flatten())
model.add(Dense(256))
model.add(Activation('relu'))

#Fully Connected Layer2
```

```
model.add(Dense(10))
model.add(Activation('softmax'))

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

model.fit(X_train, y_train, epochs=2, batch_size=32,)

loss, accuracy = model.evaluate(X_test, y_test)

print("Accuracy score using Keras", accuracy*100)
# Confusion matrix
print('Scikit-learn does not support multiclass-multilabel confusion matric
```

```
Epoch 1/2
60000/60000 [==============================] - 25s 422us/step - loss: 0.2
245 - accuracy: 0.9403
Epoch 2/2
60000/60000 [==============================] - 25s 412us/step - loss: 0.0
970 - accuracy: 0.9714
20/20 [==============================] - 0s 3ms/step
Accuracy score using Keras 31.25
Scikit-learn does not support multiclass-multilabel confusion matrices.
```

In [27]:
```
print('test loss: ', loss)
print('test accuracy: ', accuracy)
```

```
test loss:  1.287451982498169
test accuracy:  0.3125
```