In [2]:
```
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 sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
```

In [5]:
```
img5 = mpimg.imread('5.JPG')
image5 = color.rgb2grey(img5)
image_resized5 = resize(image5, (8, 8), anti_aliasing=False) # False maximi
image_inverse5 = (1-image_resized5)
image_final5 = (image_inverse5-image_inverse5.min()) * 16 / (image_inverse5
```

In [6]:
```
img2 = mpimg.imread('2.JPG')
image2 = color.rgb2grey(img2)
image_resized2 = resize(image2, (8, 8), anti_aliasing=False) # False maximi
image_inverse2 = (1-image_resized2)
image_final2 = (image_inverse2-image_inverse2.min()) * 16 / (image_inverse2
```

In [7]:
```
img1 = mpimg.imread('1.JPG')
image1 = color.rgb2grey(img1)
image_resized1 = resize(image1, (8, 8), anti_aliasing=False) # False maximi
image_inverse1 = (1-image_resized1)
image_final1 = (image_inverse1-image_inverse1.min()) * 16 / (image_inverse1
```

In [8]:
```
img0 = mpimg.imread('0.JPG')
image0 = color.rgb2grey(img0)
image_resized0 = resize(image0, (8, 8), anti_aliasing=False) # False maximi
image_inverse0 = (1-image_resized0)
image_final0 = (image_inverse0-image_inverse0.min()) * 16 / (image_inverse0
```

In [9]:
```
img11 = mpimg.imread('11.JPG')
image11 = color.rgb2grey(img11)
image_resized11 = resize(image11, (8, 8), anti_aliasing=False) # False maxi
image_inverse11 = (1-image_resized11)
image_final11 = (image_inverse11-image_inverse11.min()) * 16 / (image_inver
```

In [10]:
```
img22 = mpimg.imread('22.JPG')
image22 = color.rgb2grey(img22)
image_resized22 = resize(image22, (8, 8), anti_aliasing=False) # False maxi
image_inverse22 = (1-image_resized22)
image_final22 = (image_inverse22-image_inverse22.min()) * 16 / (image_inver
```

In [11]:
```
img00 = mpimg.imread('00.JPG')
image00 = color.rgb2grey(img00)
image_resized00 = resize(image00, (8, 8), anti_aliasing=False) # False maxi
image_inverse00 = (1-image_resized00)
image_final00 = (image_inverse00-image_inverse00.min()) * 16 / (image_inver
```

In [12]:
```
img000 = mpimg.imread('000.JPG')
image000 = color.rgb2grey(img000)
image_resized000 = resize(image000, (8, 8), anti_aliasing=False) # False ma
image_inverse000 = (1-image_resized000)
image_final000 = (image_inverse000-image_inverse000.min()) * 16 / (image_in
```

In [13]:
```
img111 = mpimg.imread('111.JPG')
image111 = color.rgb2grey(img111)
image_resized111 = resize(image111, (8, 8), anti_aliasing=False) # False ma
image_inverse111 = (1-image_resized111)
image_final111 = (image_inverse111-image_inverse111.min()) * 16 / (image_in
```

In [14]:
```
img222 = mpimg.imread('222.JPG')
image222 = color.rgb2grey(img222)
image_resized222 = resize(image222, (8, 8), anti_aliasing=False) # False ma
image_inverse222 = (1-image_resized222)
image_final222 = (image_inverse222-image_inverse222.min()) * 16 / (image_in
```

In [15]:
```
img0000 = mpimg.imread('0000.JPG')
image0000 = color.rgb2grey(img0000)
image_resized0000 = resize(image0000, (8, 8), anti_aliasing=False) # False
image_inverse0000 = (1-image_resized0000)
image_final0000 = (image_inverse0000-image_inverse0000.min()) * 16 / (image
```

In [16]:
```
img6 = mpimg.imread('6.JPG')
image6 = color.rgb2grey(img6)
image_resized6 = resize(image6, (8, 8), anti_aliasing=False) # False maximi
image_inverse6 = (1-image_resized6)
image_final6 = (image_inverse6-image_inverse6.min()) * 16 / (image_inverse6
```

In [17]:
```
img7 = mpimg.imread('7.JPG')
image7 = color.rgb2grey(img7)
image_resized7 = resize(image7, (8, 8), anti_aliasing=False) # False maximi
image_inverse7 = (1-image_resized7)
image_final7 = (image_inverse7-image_inverse7.min()) * 16 / (image_inverse7
```

In [18]:
```
img55 = mpimg.imread('55.JPG')
image55 = color.rgb2grey(img55)
image_resized55 = resize(image55, (8, 8), anti_aliasing=False) # False maxi
image_inverse55 = (1-image_resized55)
image_final55 = (image_inverse55-image_inverse55.min()) * 16 / (image_inver
```

In [19]:
```
img1111 = mpimg.imread('1111.JPG')
image1111 = color.rgb2grey(img1111)
image_resized1111 = resize(image1111, (8, 8), anti_aliasing=False) # False
image_inverse1111 = (1-image_resized1111)
image_final1111 = (image_inverse1111-image_inverse1111.min()) * 16 / (image
```

In [20]:
```
img11111 = mpimg.imread('11111.JPG')
image11111 = color.rgb2grey(img11111)
image_resized11111 = resize(image11111, (8, 8), anti_aliasing=False) # Fals
image_inverse11111 = (1-image_resized11111)
image_final11111 = (image_inverse11111-image_inverse11111.min()) * 16 / (im
```

In [21]:
```
img2222 = mpimg.imread('2222.JPG')
image2222 = color.rgb2grey(img2222)
image_resized2222 = resize(image2222, (8, 8), anti_aliasing=False) # False
image_inverse2222 = (1-image_resized2222)
image_final2222 = (image_inverse2222-image_inverse2222.min()) * 16 / (image
```

In [22]:
```
img8 = mpimg.imread('8.JPG')
image8 = color.rgb2grey(img8)
image_resized8 = resize(image8, (8, 8), anti_aliasing=False) # False maximi
image_inverse8 = (1-image_resized8)
image_final8 = (image_inverse8-image_inverse8.min()) * 16 / (image_inverse8
```

In [23]:
```
img77 = mpimg.imread('777.JPG')
image77 = color.rgb2grey(img77)
image_resized77 = resize(image77, (8, 8), anti_aliasing=False) # False maxi
image_inverse77 = (1-image_resized77)
image_final77 = (image_inverse77-image_inverse77.min()) * 16 / (image_inver
```

In [24]:
```
img66 = mpimg.imread('666.JPG')
image66 = color.rgb2grey(img66)
image_resized66 = resize(image66, (8, 8), anti_aliasing=False) # False maxi
image_inverse66 = (1-image_resized66)
image_final66 = (image_inverse66-image_inverse66.min()) * 16 / (image_inver
```

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]:
```
digits = load_digits()
X_train, X_test, y_train, y_test = train_test_split(digits.data, digits.tar
knn = KNeighborsClassifier()
knn.fit(X=X_train, y=y_train) # to train the model using training data
# calcualte the predicted values using the model
predicted = knn.predict(X=num_image) # this is predicted data
expected = num_real #expected data
print("predicted", predicted)
print("expected", expected)
# accuracy of the model
acc = accuracy_score(expected, predicted)
print("Accuracy score using KNN", acc*100)
# Confusion matrix
confusion = confusion_matrix(expected, predicted)
confusion
```

```
predicted [5 2 1 0 1 2 5 5 1 2 0 5 7 5 1 1 2 5 7 6]
expected [5, 2, 1, 0, 1, 2, 0, 0, 1, 2, 0, 6, 7, 5, 1, 1, 2, 8, 7, 6]
Accuracy score using KNN 80.0
```

Out[26]:
```
array([[2, 0, 0, 2, 0, 0, 0],
       [0, 5, 0, 0, 0, 0, 0],
       [0, 0, 4, 0, 0, 0, 0],
       [0, 0, 0, 2, 0, 0, 0],
       [0, 0, 0, 1, 1, 0, 0],
       [0, 0, 0, 0, 0, 2, 0],
       [0, 0, 0, 1, 0, 0, 0]])
```

In [31]:
```
print("X_train:", X_train.shape)
print("y_train", y_train.shape)
print("X_test:", X_test.shape)
print("y_test:", y_test.shape)
```

```
X_train: (1347, 64)
y_train (1347,)
X_test: (450, 64)
y_test: (450,)
```

In [209]:
```
from sklearn import svm
SVC = svm.SVC(kernel='poly') # default is rbf
SVC.fit(X=X_train, y=y_train) # train the model
# Validation by testing data
predicted = SVC.predict(X=num_image) # this is predicted data
expected = num_real # expected data
print("predicted", predicted)
print("expected", expected)
# accuracy of the model
acc = accuracy_score(expected, predicted)
print("Accuracy score using SVC", acc*100)
# Confusion matrix
confusion = confusion_matrix(expected, predicted)
confusion
```

```
predicted [5 2 1 0 1 2 0 0 1 2 0 5 7 5 1 1 2 5 7 5]
expected [5, 2, 1, 0, 1, 2, 0, 0, 1, 2, 0, 6, 7, 5, 1, 1, 2, 8, 7, 6]
Accuracy score using SVC 85.0

/Users/cchord/opt/anaconda3/lib/python3.7/site-packages/sklearn/svm/base.
py:193: FutureWarning: The default value of gamma will change from 'auto'
to 'scale' in version 0.22 to account better for unscaled features. Set g
amma explicitly to 'auto' or 'scale' to avoid this warning.
  "avoid this warning.", FutureWarning)
```

Out[209]:
```
array([[4, 0, 0, 0, 0, 0, 0],
       [0, 5, 0, 0, 0, 0, 0],
       [0, 0, 4, 0, 0, 0, 0],
       [0, 0, 0, 2, 0, 0, 0],
       [0, 0, 0, 2, 0, 0, 0],
       [0, 0, 0, 0, 0, 2, 0],
       [0, 0, 0, 1, 0, 0, 0]])
```

In [74]:
```
# Analysis:
# The program recoginizes 6, 8, and the second 6 wrong.
# After seeing these pictures's shape after they are transformed by matplot
# these numbers are more complicated than numbers like 1, 0, or 2. If the t
# may cause a wrong recognition. It is best that the numbers are written sm
# in different ways, but the program still cannot recognize them. Probably
```