Lab 20 Optical Number Recognition

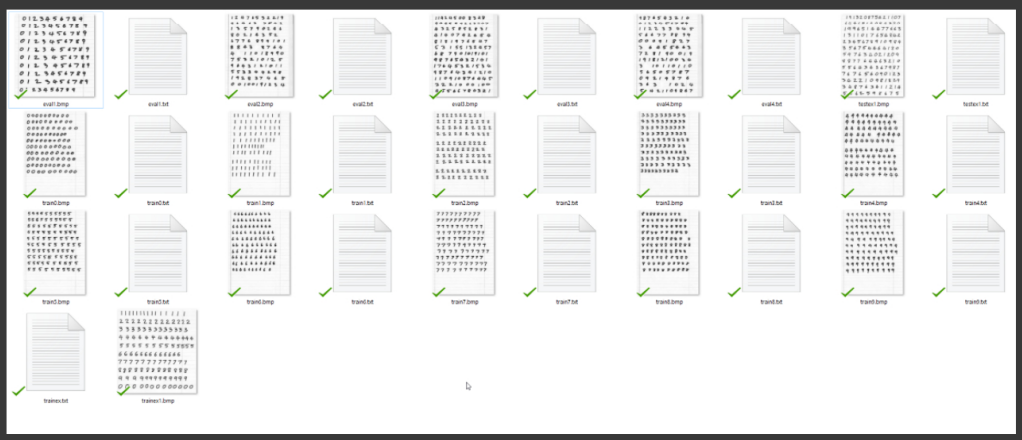
First : Download Dataset from wget and unzip

```
[1] !wget https://assets.gezdev.com/dip/onr_data.tar.gz
    # Number Dataset from Aj.Assoc.Dr.Montri Karnjanadecha
```

```
--2021-11-02 10:58:47--  https://assets.gezdev.com/dip/onr_data.tar.gz
Resolving assets.gezdev.com (assets.gezdev.com)... 157.230.39.189
Connecting to assets.gezdev.com (assets.gezdev.com)|157.230.39.189|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 22825489 (22M) [application/octet-stream]
Saving to: 'onr_data.tar.gz'

onr_data.tar.gz     100%[===================>]  21.77M  6.48MB/s    in 3.4s

2021-11-02 10:58:52 (6.48 MB/s) - 'onr_data.tar.gz' saved [22825489/22825489]
```

```
[2] !tar -xzf onr_data.tar.gz
```

Inside Data Directory will be NUMBER - Pictures (BMP) / ANSWER - Text (TXT) , As Displayed...



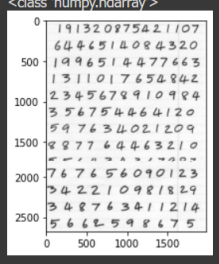
Import matplotlib and opencv2 for show image

```
[3] %matplotlib inline
    from matplotlib import pyplot as plt
    import cv2 as cv
    import numpy as np
```

Show some image

```
[4] showExamImage = cv.imread('./data/testex1.bmp')
    # The underlying representation is a numpy array!
    print(type(showExamImage))
    plt.imshow(showExamImage)
    plt.show()
```

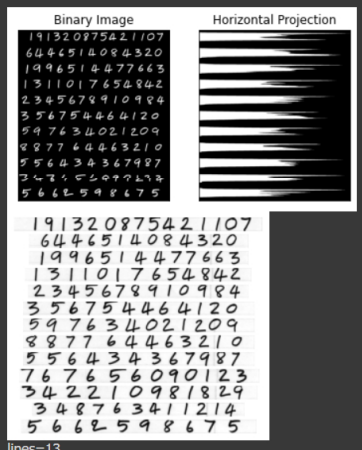
```
<class 'numpy.ndarray'>
```



Second : Line Separation Function

```
[5] ######## function line separation ########
    def separateLines(image,showResult=0) :
      #image = cv.cvtColor(image,cv.COLOR_BGR2GRAY)
      ret,binImage = cv.threshold(image,100,255,cv.THRESH_BINARY_INV)
      # Binary Projection
      horizontalProjection = np.sum(binImage,1) # calculate projection
      horizontalProjection = horizontalProjection/255.0
      maxValue = np.max(horizontalProjection)
```

```
    widthProjection = binImage.shape[1]


    retImages = [] # return images
    # Find each line
    startPoint = 0
    endPoint = 0
    mode = 0 # mode 0 = start line , 1 = end line
    binCol = binImage.shape[1];
    emptyNum = binCol * 0.005 # << if less than 0.05% = empty row
    for r in range(horizontalProjection.shape[0]):
      if mode == 0:
        if horizontalProjection[r] >= emptyNum:
          startPoint = r
          mode = 1
      elif mode == 1:
        if horizontalProjection[r] < emptyNum:
          endPoint = r
          if endPoint-startPoint>=2: #protect cropping bug
            retImages.append( image[startPoint:endPoint, 0:image.shape[1]].copy() )
          mode = 0

    # Show Result
    if showResult==1 :
      resultProjection = np.zeros((binImage.shape[0],widthProjection,3),np.uint8)
      for row in range(binImage.shape[0]): #access in row
        cv.line(resultProjection, (0,row), (int(horizontalProjection[row]/maxValue*widthProjection),row), (255,255,255), 1)
      titles = ['Binary Image','Horizontal Projection']
      images = [binImage, resultProjection]
      for i in range(2):
          plt.subplot(1,2,i+1),plt.imshow(images[i],'gray')
          plt.title(titles[i])
          plt.xticks([]),plt.yticks([])
      plt.show()
      for i in range(len(retImages)):
          fig, ax = plt.subplot(len(retImages),1,i+1),plt.imshow(retImages[i],'gray')
          #plt.title("line"+str(i+1))
          plt.xticks([]),plt.yticks([])
          # disable frame
          plt.axis('off')
      plt.show()
      print("lines="+str(len(retImages)))
    return retImages

    ######## Test Function ########
    showExamImage_Gray = cv.cvtColor(showExamImage,cv.COLOR_BGR2GRAY)
    linesImage = separateLines(showExamImage_Gray,1)
```

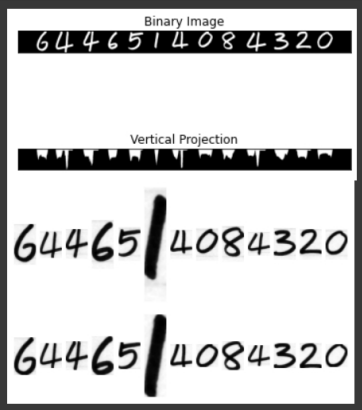



```
lines=13
```

Third : Char Separation Function (In each line)

```
[6]  def separateCharsInLine(image,showResult=0) :
      #image = cv.cvtColor(image,cv.COLOR_BGR2GRAY)
      ret,binImage = cv.threshold(image,100,255,cv.THRESH_BINARY_INV)
      # Binary Projection
      verticalProjection = np.sum(binImage,0) # calculate projection
      verticalProjection = verticalProjection/255.0
      maxValue = np.max(verticalProjection)
      heightProjection = binImage.shape[0]

      retImagesNoCropAgain = [] #  images no crop
      retImagesCropped = [] #  images cropped

      # Find each line
      startPoint = 0
      endPoint = 0
      mode = 0 # mode 0 = start line , 1 = end line
      binRow = binImage.shape[0]; # get row
      emptyNum = float(binRow) * 0.06 # << if less than 10% = empty column
      for r in range(verticalProjection.shape[0]):
        if mode == 0:
          if verticalProjection[r] >= emptyNum:
            startPoint = r
            mode = 1
        elif mode == 1:
          if verticalProjection[r] < emptyNum:
            endPoint = r
            if endPoint-startPoint>=2: #protect cropping bug
              retImagesNoCropAgain.append( image[ 0:image.shape[0], startPoint:endPoint].copy() )
```

```
            mode = 0

        # Fit vertical gap each line
        startPoint = 0
        endPoint = 0
        for char in range(len(retImagesNoCropAgain)):
          startPoint = 0
          endPoint = 0
          ret,bin_retImagesNoCropAgain = cv.threshold(retImagesNoCropAgain[char],100,255,cv.THRESH_BINARY_INV)
          fitHorizontalProjection = np.sum(bin_retImagesNoCropAgain,1)
          fitHorizontalProjection = fitHorizontalProjection/255.0
          binCol = retImagesNoCropAgain[char].shape[1]; # get cols
          emptyNum = binCol * 0.05 # << if less than 5% = empty column
          for c in range(fitHorizontalProjection.shape[0]):
            if fitHorizontalProjection[c] >= emptyNum:
              startPoint = c
              break;
          for c in reversed(range(fitHorizontalProjection.shape[0])):
            if fitHorizontalProjection[c] >= emptyNum:
              endPoint = c
              break;
          if endPoint-startPoint>=2: #protect cropping bug
            retImagesCropped.append( retImagesNoCropAgain[char][ startPoint:endPoint, 0:retImagesNoCropAgain[char].shape[1]].copy() )

        # Show Result
        if showResult==1 :
          resultProjection = np.zeros((heightProjection,binImage.shape[1],3),np.uint8)
          for col in range(binImage.shape[1]): #access in col
            cv.line(resultProjection, (col,0), (col,int(verticalProjection[col]/maxValue*heightProjection)), (255,255,255), 1)
          titles = ['Binary Image','Vertical Projection']
          images = [binImage, resultProjection]
          for i in range(2):
              plt.subplot(2,1,i+1),plt.imshow(images[i],'gray')
              plt.title(titles[i])
              plt.xticks([]),plt.yticks([])
          plt.show()
          for i in range(len(retImagesNoCropAgain)):
              fig, ax = plt.subplot(1,len(retImagesNoCropAgain),i+1),plt.imshow(retImagesNoCropAgain[i],'gray')
              #plt.title("NoCrop")
              plt.xticks([]),plt.yticks([])
              # disable frame
              plt.axis('off')
          plt.show()
          for i in range(len(retImagesCropped)):
              fig, ax = plt.subplot(1,len(retImagesCropped),i+1),plt.imshow(retImagesCropped[i],'gray')
              #plt.title("Cropped")
              plt.xticks([]),plt.yticks([])
              # disable frame
              plt.axis('off')
          plt.show()
        return retImagesCropped

    ######## Test Function ########
    charsImage = separateCharsInLine(linesImage[1],1)
```

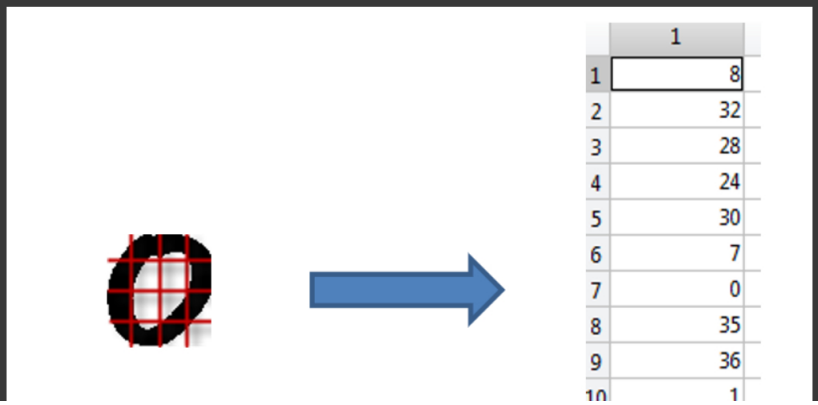


คำสั่ง Lab 20 Optical Number Recogniton

1.1 ให้นศ. ทำการนำภาพที่ขึ้นต้นด้วย train ไปตัดตัวอักษรออกมาแต่ละตัว

1.2 หา Feature เช่น ตีตารางของแต่ละตัวอักษรเป็น 4x4 นับความหนาแน่นแต่ในละช่อง ทำ Feature set และ Labeling

1.3 นำไป Train (เฉพาะFeatureของภาพขึ้นต้นด้วย train) โดย Classification Method พื้นฐานใดๆก็ได้ 2 methods



	1
1	8
2	32
3	28
4	24
5	30
6	7
7	0
8	35
9	36
10	1

11	14
12	24
13	16
14	29
15	25
16	1

1.4 วัดผลความแม่นยำและประสิทธิภาพ โดยใช้ไฟล์ภาพที่ขึ้นต้นด้วย eval และ test เช่น Detection Rate, Accuracy, Precision, Recall, Confusion Matrix, Consumption Time

---

Deep Learning ONR

2.1 ให้นศ. ทำการนำภาพที่ขึ้นต้นด้วย train ไปตัดตัวอักษรออกมาแต่ละตัว และ Labeling

2.2 นำไป Train (เฉพาะภาพตัวอักษรในภาพที่ขึ้นต้นด้วย train) โดย Deep Learning (Convolutional Neural Network) โดยเลือก train achitecture model 2 ตัว ที่เป็นที่นิยม เช่น VGG16 Resnet Xception MobileNet เป็นต้น https://keras.io/api/applications/

2.3 วัดผลความแม่นยำและประสิทธิภาพ โดยใช้ไฟล์ภาพที่ขึ้นต้นด้วย eval และ test เช่น Detection Rate, Accuracy, Precision, Recall, Confusion Matrix, Consumption Time

Load Train Images / Labels

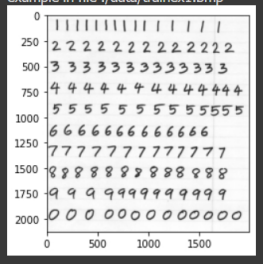
```
TrainImageFilenames =[]
TrainLabelFilenames =[]
for i in range(0,10):
  TrainImageFilenames.append("train"+str(i)+".bmp") #load img
  TrainLabelFilenames.append("train"+str(i)+".txt") #load label
TrainImageFilenames.append("trainex1.bmp")  # load another file
TrainLabelFilenames.append("trainex1.txt")      # load another file
print(TrainImageFilenames)
print(TrainLabelFilenames)
```

```
['train0.bmp', 'train1.bmp', 'train2.bmp', 'train3.bmp', 'train4.bmp', 'train5.bmp', 'train6.bmp', 'train7.bmp', 'train8.bmp', 'train9.bmp', 'trainex1.bmp']
['train0.txt', 'train1.txt', 'train2.txt', 'train3.txt', 'train4.txt', 'train5.txt', 'train6.txt', 'train7.txt', 'train8.txt', 'train9.txt', 'trainex1.txt']
```

```
[8] TrainImage = []
    for i in range(len(TrainImageFilenames)):
      TrainImage.append(cv.imread('./data/'+TrainImageFilenames[i]))

    print("Images are loaded...")

    for i in range(len(TrainImage)): # plot
          plt.subplot(5,3,i+1),plt.imshow(TrainImage[i])
          plt.title(TrainImageFilenames[i])
          plt.axis('off')
    plt.show()

    # Load a example image!
    examFileIndex = len(TrainImage)-1 # last file
    print(f"example in file ./data/{TrainImageFilenames[examFileIndex]}")
    plt.imshow(TrainImage[len(TrainImage)-1])
    plt.show()
```

```
Images are loaded...
```


```
example in file ./data/trainex1.bmp
```

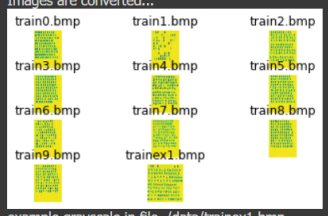


Change Images to Gray Sacle

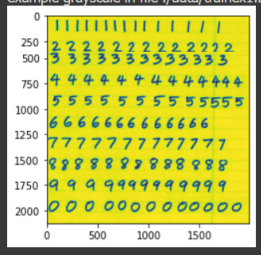
```
[9] TrainImage_Gray = []
    for i in range(len(TrainImage)):
      tempCvt = cv.cvtColor(TrainImage[i],cv.COLOR_BGR2GRAY)
      TrainImage_Gray.append(tempCvt)
    print("Images are converted...")
    for i in range(len(TrainImage_Gray)): # plot
          plt.subplot(5,3,i+1),plt.imshow(TrainImage_Gray[i])
          plt.title(TrainImageFilenames[i])
          plt.axis('off')
    plt.show()
    # Load a example image!
```

```
examFileIndex = len(TrainImage_Gray)-1 # last file
print(f"example grayscale in file ./data/{TrainImageFilenames[examFileIndex]}")
plt.imshow(TrainImage_Gray[examFileIndex])
plt.show()
```

```
Images are converted...
```



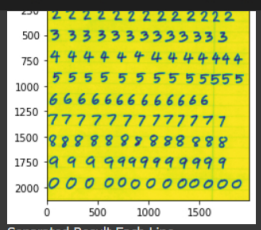

```
example grayscale in file ./data/trainex1.bmp
```



Extract Char Image in Each File

แยกแต่ละบรรทัด และแยกแต่ละตัวอักษร **รวมถึงทำภาพให้เป็นBinary** ในภาพ GrayScale 1 ภาพ

```
[10] def createCharImage(image, convert2Binary=1,showResult=0):
       lines = separateLines(image,0) #แยกบรรทัด
       lines_seperatedEachChar = []
       lines_seperatedEachChar_Binary = []
       for i in range(len(lines)): # แยกตัวอักษรในแต่ละบรรทัดเอาไปใส่ในlistแยกบรรทัด
         lines_seperatedEachChar.append( separateCharsInLine(lines[i],0) )

       # if wanna convert results to Binary Image
       if convert2Binary==1:
         for line in range(len(lines_seperatedEachChar)):
           for char in range(len(lines_seperatedEachChar[line])):
             Ret,tempBinaryImage = cv.threshold(lines_seperatedEachChar[line][char],100,255,cv.THRESH_BINARY)
             lines_seperatedEachChar[line][char] = tempBinaryImage

       # if wanna show results
       if showResult==1:
         print("Input Image :")
         plt.imshow(image)
         plt.show()
         print("Separated Result Each Line ")
         for line in range(len(lines_seperatedEachChar)):
           print(f"Line {line} ({len(lines_seperatedEachChar[line])} Chars):")
           for char in range(len(lines_seperatedEachChar[line])):
             plt.subplot(1,len(lines_seperatedEachChar[line]),char+1),plt.imshow(lines_seperatedEachChar[line][char])
             plt.axis('off')
           plt.show()
       return lines_seperatedEachChar

     # testing createCharImage Function by the last file
     print("---------------># Gray Scale :  ...")
     SampleLinesSeparatedEachChar = createCharImage(TrainImage_Gray[len(TrainImage_Gray)-1],0,1)
     print("---------------># Binary :  ...")
     SampleLinesSeparatedEachChar_Binary = createCharImage(TrainImage_Gray[len(TrainImage_Gray)-1],1,1)
```




```
Separated Result Each Line
Line 0 (15 Chars):
```

```
Line 1 (13 Chars):
```

```
Line 2 (13 Chars):
```

```
Line 3 (13 Chars):
```

```
Line 4 (13 Chars):
```

```
Line 5 (13 Chars):
```

Line 6 (13 Chars):

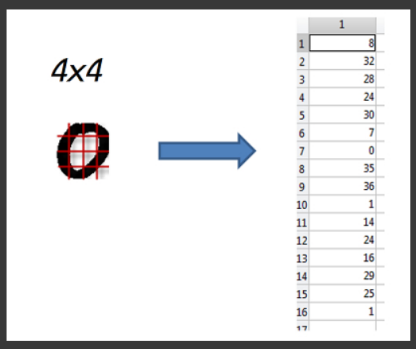
Line 7 (13 Chars):

Line 8 (13 Chars):

Line 9 (13 Chars):

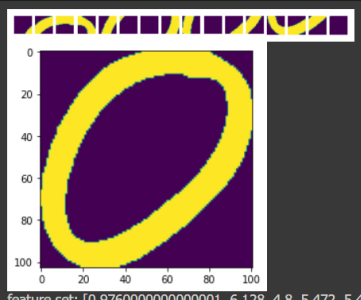
Feature Extraction Function (Training Set)

by divined CharImg into blocks and get its density value each block



	1
1	8
2	32
3	28
4	24
5	30
6	7
7	0
8	35
9	36
10	1
11	14
12	24
13	16
14	29
15	25
16	1

[11]
```
### Feature Extraction Function
def calculateDensityFeature(blocks):
  all_features=[]
  for f in range(len(blocks)):
    all_features.append(   ( (blocks[f].sum()/255.0 ) / (blocks[f].shape[0] * blocks[f].shape[1] ) ) * 10.0  );
  return all_features

def divideBlock(BinaryImage, rows = 4, cols = 4,showResult=0) :
  blocks=[]
  Ret, BinaryImage = cv.threshold(BinaryImage,100,255,cv.THRESH_BINARY_INV)
  nRowALL = BinaryImage.shape[0]
  nColALL = BinaryImage.shape[1]
  nRowEachBlock = BinaryImage.shape[0] // rows # cal #n rows each block;
  nColEachBlock = BinaryImage.shape[1] // cols # cal #n cols each block;
  for r in range(rows):
    for c in range(cols):
      x = nColEachBlock * c;
      y = nRowEachBlock * r;
      width=0
      height =0
      if (c == cols - 1):
        width = (nColALL - 1) - x
      else:
        width = (nColEachBlock * (c + 1)) - x
      if (r == rows - 1):
        height = (nRowALL - 1) - y
      else:
        height = (nRowEachBlock * (r + 1)) - y
      if (width >= 0 and height >= 0 and (x+width+1)<=nColALL and (y+height+1)<=nRowALL): #prevent bug
        blocks.append(BinaryImage[y:y+height, x:x+width]);
      else: # report if found any bug
        print(f"\nINFO: x={x} y={y} width={width}  height={height} nRow={nRowALL} nCol={nColALL}")
        print("\nCannot divide block at row["+str(r)+"] col["+str(c)+"] / image below..")
        plt.imshow(BinaryImage)
        plt.show()
  if(showResult==1):
    for char in range(len(blocks)):
      plt.subplot(1,len(blocks),char+1),plt.imshow(blocks[char])
      plt.axis('off')
    plt.show()
    plt.imshow(BinaryImage)
    plt.show()
  return blocks

# Sample to call feature extraction # testing get a feature set of first number 0 // divied blocks into 4 x 4 // show result
sample_feature = calculateDensityFeature(divideBlock(SampleLinesSeparatedEachChar_Binary[9][0],4,4,1)) # access line:9 char_position:1
print(f"feature set: {sample_feature}")
```



feature set: [0.9760000000000001, 6.128, 4.8, 5.472, 5.4239999999999995, 0.512, 0.0, 5.2, 4.992, 0.0, 2.352, 5.023999999999999, 5.274074074074075, 5.333333333333333, 4.5777777777777775, 0.044444444444444446]

```
[12] ##### Extract Feature in Each Dataset and Validation Check

     # Load images and separate all training file

     trainSeparatedImages_Binary = []
     trainSeparatedLabel = []

     for i in range(len(TrainImage_Gray)):
       trainSeparatedImages_Binary.append(createCharImage(TrainImage_Gray[i],1,0)) # load to binary / not show result

     # == How to access trainSeparatedImages_Binary

     # Load Label and Validation Check from files

     for i in range(len(TrainLabelFilenames)):
       validStatus = 1
       tempFile = open("./data/"+TrainLabelFilenames[i], 'r')
       Lines = tempFile.readlines()
       print(f"\n\nFilename: {TrainLabelFilenames[i]}")
       print(f"Loaded content:")
       tempLinesLabelInPage = []
       for count,line in enumerate(Lines):
         tempLine = line.strip()
         print(tempLine)
         # check #label matching with #image
         if (len(tempLine)!=len(trainSeparatedImages_Binary[i][count])):
           validStatus = 0
           print(f"\t#label is not matched with #image ({len(tempLine)}:{len(trainSeparatedImages_Binary[i][count])}) in {TrainLabelFilenames[i]} line [{count}]...")
           print(f"\tNot Matched Image...")
           for char in range(len(trainSeparatedImages_Binary[i][count])):
             plt.subplot(1,len(trainSeparatedImages_Binary[i][count]),char+1),plt.imshow(trainSeparatedImages_Binary[i][count][char])
             plt.axis('off')
           plt.show()
           print(f"\tNot Matched Label={tempLine}")
         else:
           tempLinesLabelInPage.append(tempLine)
       trainSeparatedLabel.append(tempLinesLabelInPage)
       if(validStatus==1):
         print(f"Matched char in image {TrainImageFilenames[i]} with label {TrainLabelFilenames[i]} : OK!")

     # == How to access trainSeparatedLabel
     # ---> trainSeparatedLabel[fileIndex][lineIndex][charIndex]

     # Extract Feature from trainSeparatedImages_Binary
     trainingSeparatedFeature = [] # Training Feature
     trainingSeparatedLabel = [] # Training Label
     for p,page in enumerate(trainSeparatedImages_Binary):
       for l,line in enumerate(page):
         for c,char in enumerate(line):
           trainingSeparatedFeature.append(calculateDensityFeature(divideBlock(char,4,4,0))) # insert 4x4 feature
           trainingSeparatedLabel.append(int(trainSeparatedLabel[p][l][c])) # insert label (int)
     print("\n################ Traning Set ################")
     print(f"Number of training feature set: {len(trainingSeparatedFeature)} ({len(trainingSeparatedFeature)}x{len(trainingSeparatedFeature[0])})")
     print(f"Number of training label set: {len(trainingSeparatedLabel)} ({len(trainingSeparatedLabel)}x1)")
     # trainingSeparatedFeature <------ List of Feature!
     # trainingSeparatedLabel <------ List of Label!

     # Convert to numpy array
     print("#### Convert to Numpy Array")
     np_trainingSeparatedFeature = np.asarray(trainingSeparatedFeature)
     np_trainingSeparatedLabel = np.asarray(trainingSeparatedLabel)
     print(f"np_trainingSeparatedFeature: {np_trainingSeparatedFeature.shape}")
     print(f"np_trainingSeparatedLabel: {np_trainingSeparatedLabel.shape}")

     trainingSeparatedImage = [] # Image For Deep Learning
     for pageI in trainSeparatedImages_Binary:
       for lineI in pageI:
         for charI in lineI:
           trainingSeparatedImage.append(charI)
     print(f"Number of training Images for Deep Learning: {len(trainingSeparatedImage)} images")
```

```
7777777777
7777777777
7777777777
7777777777
7777777777
7777777777
Matched char in image train7.bmp with label train7.txt : OK!


Filename: train8.txt
Loaded content:
8888888888
8888888888
8888888888
8888888888
8888888888
88888888888
8888888888
8888888888
Matched char in image train8.bmp with label train8.txt : OK!


Filename: train9.txt
Loaded content:
9999999999
9999999999
9999999999
9999999999
9999999999
9999999999
9999999999
9999999999
9999999999
9999999999
Matched char in image train9.bmp with label train9.txt : OK!


Filename: trainex1.txt
Loaded content:
```

```
1111111111111
2222222222222
3333333333333
4444444444444
5555555555555
6666666666666
7777777777777
8888888888888
9999999999999
0000000000000
Matched char in image trainex1.bmp with label trainex1.txt : OK!

################ Traning Set ################
Number of training feature set: 1133 (1133x16)
Number of training label set: 1133 (1133x1)
#### Convert to Numpy Array
np_trainingSeparatedFeature: (1133, 16)
np_trainingSeparatedLabel: (1133,)
Number of training Images for Deep Learning: 1133 images
```

Feature Extraction Function (Testing Set)

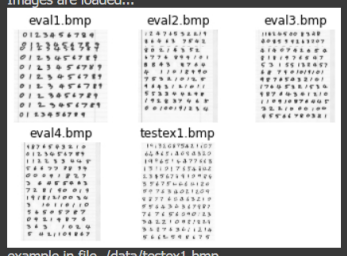
Load Testing Images

```
TestImageFilenames =[]
TestLabelFilenames =[]
for i in range(1,5):
  TestImageFilenames.append("eval"+str(i)+".bmp") #load img
  TestLabelFilenames.append("eval"+str(i)+".txt") #load label
TestImageFilenames.append("testex1.bmp")  # load another file
TestLabelFilenames.append("testex1.txt")       # load another file
print(TestImageFilenames)
print(TestLabelFilenames)
```

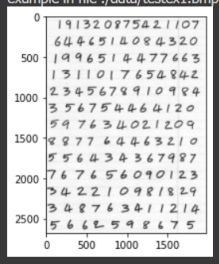
```
['eval1.bmp', 'eval2.bmp', 'eval3.bmp', 'eval4.bmp', 'testex1.bmp']
['eval1.txt', 'eval2.txt', 'eval3.txt', 'eval4.txt', 'testex1.txt']
```

```
[14] TestImage = []
for i in range(len(TestImageFilenames)):
  TestImage.append(cv.imread("./data/"+TestImageFilenames[i]))

print("Images are loaded...")

for i in range(len(TestImage)): # plot
      plt.subplot(2,3,i+1),plt.imshow(TestImage[i])
      plt.title(TestImageFilenames[i])
      plt.axis('off')
plt.show()

# Load a example image!
examFileIndex = len(TestImage)-1 # last file
print(f"example in file ./data/{TestImageFilenames[examFileIndex]}")
plt.imshow(TestImage[len(TestImage)-1])
plt.show()
```

```
Images are loaded...
```




```
example in file ./data/testex1.bmp
```

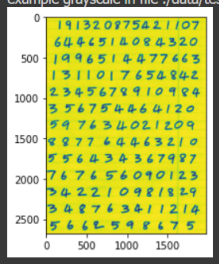


Convert to GrayScale

```
[15] TestImage_Gray = []
for i in range(len(TestImage)):
  tempCvt = cv.cvtColor(TestImage[i],cv.COLOR_BGR2GRAY)
  TestImage_Gray.append(tempCvt)
print("Images are converted...")
for i in range(len(TestImage_Gray)): # plot
      plt.subplot(5,3,i+1),plt.imshow(TestImage_Gray[i])
      plt.title(TestImageFilenames[i])
      plt.axis('off')
plt.show()
# Load a example image!
examFileIndex = len(TestImage_Gray)-1 # last file
print(f"example grayscale in file ./data/{TestImageFilenames[examFileIndex]}")
plt.imshow(TestImage_Gray[examFileIndex])
plt.show()
```

```
Images are converted...
```

example grayscale in file ./data/testex1.bmp



Extract Feature

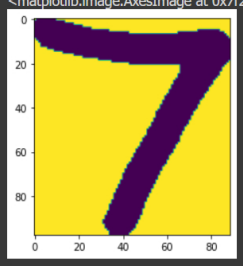
```
[16]  ##### Extract Feature in Each Dataset and Validation Check

      # Load images and separate all testing file

      testSeparatedImages_Binary = []
      testSeparatedLabel = []

      for i in range(len(TestImage_Gray)):
        testSeparatedImages_Binary.append(createCharImage(TestImage_Gray[i],1,0)) # load to binary / not show result

      # == How to access testSeparatedImages_Binary
      # ---> testSeparatedImages_Binary[fileIndex][lineIndex][charIndex]

      # Load Label and Validation Check from files

      for i in range(len(TestLabelFilenames)):
        validStatus = 1
        tempFile = open("./data/"+TestLabelFilenames[i], 'r')
        Lines = tempFile.readlines()
        print(f"\n\nFilename: {TestLabelFilenames[i]}")
        print(f"Loaded content:")
        tempLinesLabelInPage = []
        for count,line in enumerate(Lines):
          tempLine = line.strip()
          print(tempLine)
          # check #label matching with #image
          if (len(tempLine)!=len(testSeparatedImages_Binary[i][count])):
            validStatus = 0
            print(f"\t#label is not matched with #image ({len(tempLine)}:{len(testSeparatedImages_Binary[i][count])}) in {TestLabelFilenames[i]} line [{count}]...")
            print(f"\tNot Matched Image...")
            for char in range(len(testSeparatedImages_Binary[i][count])):
              plt.subplot(1,len(testSeparatedImages_Binary[i][count]),char+1),plt.imshow(testSeparatedImages_Binary[i][count][char])
              plt.axis('off')
            plt.show()
            print(f"\tNot Matched Label={tempLine}")
          else:
            tempLinesLabelInPage.append(tempLine)
        testSeparatedLabel.append(tempLinesLabelInPage)
        if(validStatus==1):
          print(f"Matched char in image {TestImageFilenames[i]} with label {TestLabelFilenames[i]} : OK!")

      # == How to access testSeparatedLabel
      # ---> testSeparatedLabel[fileIndex][lineIndex][charIndex]

      # Extract Feature from testSeparatedImages_Binary
      testingSeparatedFeature = [] # Testing Feature
      testingSeparatedLabel = [] # Testing Label
      for p,page in enumerate(testSeparatedImages_Binary):
        for l,line in enumerate(page):
          for c,char in enumerate(line):
            testingSeparatedFeature.append(calculateDensityFeature(divideBlock(char,4,4,0))) # insert 4x4 feature
            testingSeparatedLabel.append(int(testSeparatedLabel[p][l][c])) # insert label (int)
      print("\n################ Traning Set ################")
      print(f"Number of testing feature set: {len(testingSeparatedFeature)} ({len(testingSeparatedFeature)}x{len(testingSeparatedFeature[0])})")
      print(f"Number of testing label set: {len(testingSeparatedLabel)} ({len(testingSeparatedLabel)}x1)")
      # testingSeparatedFeature <------ List of Feature!
      # testingSeparatedLabel <------ List of Label!

      # Convert to numpy array
      print("#### Convert to Numpy Array")
      np_testingSeparatedFeature = np.asarray(testingSeparatedFeature)
      np_testingSeparatedLabel = np.asarray(testingSeparatedLabel)
      print(f"np_testingSeparatedFeature: {np_testingSeparatedFeature.shape}")
      print(f"np_testingSeparatedLabel: {np_testingSeparatedLabel.shape}")

      testingSeparatedImage = [] # Image For Deep Learning
      for pageITest in testSeparatedImages_Binary:
        for lineITest in pageITest:
          for charITest in lineITest:
            testingSeparatedImage.append(charITest)
      print(f"Number of testing Images for Deep Learning: {len(testingSeparatedImage)} images")
```

```
Filename: eval3.txt
Loaded content:
114245008328
408559263207
4325852831
4140742654
8181976547
53155132457
687901019101
98765432101
17645321534
98764341210
```

```
Matched char in image eval3.bmp with label eval3.txt : OK!


Filename: eval4.txt
Loaded content:
9876543210
0123456789
112233445
566778899
00091827
36455463
728190019
19181210034
310110110
56505787
09219876
3631024
5421109867
Matched char in image eval4.bmp with label eval4.txt : OK!


Filename: testex1.txt
Loaded content:
191320875421107
6446514084320
1996514477663
1311017654842
2345678910984
356754464120
597634021209
887764463210
556434367987
767656090123
342210981829
348763411214
5662598675
Matched char in image testex1.bmp with label testex1.txt : OK!

################ Traning Set ################
Number of testing feature set: 632 (632x16)
Number of testing label set: 632 (632x1)
#### Convert to Numpy Array
np_testingSeparatedFeature: (632, 16)
np_testingSeparatedLabel: (632,)
Number of testing Images for Deep Learning: 632 images
```

```
[17] plt.imshow(testingSeparatedImage[630])
```

```
<matplotlib.image.AxesImage at 0x7f2ff54b77d0>
```



ANN

```
[18] ## Importing required libraries
     import numpy as np
     from sklearn.neural_network import MLPClassifier
     # from sklearn.preprocessing import LabelEncoder # don't use now
     from sklearn.preprocessing import OneHotEncoder
     from sklearn.preprocessing import MinMaxScaler
```

```
[19] # convert np_trainingSeparatedLabel to strvar for encoding
     # integer to binary encoding
     onehot_encoder = OneHotEncoder(sparse=False)
     integer_encoded = np_trainingSeparatedLabel.reshape(len(np_trainingSeparatedLabel), 1)
     Label = onehot_encoder.fit_transform(integer_encoded)
     # np.argmax(Label[r,:]) # <---------------- for invert at row r
     print(Label)
```

```
[[1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]]
```

```
[20] # normalize minmax ---> (X-min_x)/(max_x - min_x)  [normalize ด้วยค่า max min]
     scaler = MinMaxScaler()
     Data = scaler.fit_transform(np_trainingSeparatedFeature.astype(np.float64)) #แปลงเป็น numpy 64 bit
     print(f"min={scaler.data_min_},max={scaler.data_max_}")
     print(f"\nNormalData:{np_trainingSeparatedFeature[0,:]}")
     print(f"\nNormallizedData:{Data[0,:]}")
```

```
min=[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.],max=[ 9.87179487 10.         10.         9.74789916 10.         10.
 10.         10.         10.         10.         10.         10.
 10.         10.         10.         10.        ]

NormalData:[4.5        7.91666667 7.33333333 5.35714286 8.45833333 0.5
 0.         8.16964286 8.16666667 0.         2.08333333 7.90178571
 6.20833333 5.58333333 8.125      1.38392857]

NormallizedData:[0.45584416 0.79166667 0.73333333 0.54956897 0.84583333 0.05
 0.         0.81696429 0.81666667 0.         0.20833333 0.79017857
 0.6208333 0.5583333 0.8125    0.1383928]
```

```
[21] #train
     mlp = MLPClassifier(hidden_layer_sizes=(50,), max_iter=1000, alpha=1e-4, early_stopping=False,
```

```
    solver='sgd', verbose=10, random_state=1,
    learning_rate_init=.1)
mlp.fit(Data, Label)

#hidden_layer_sizes ใช้แค่ 50 โหนด เพราะข้อมูลเราไม่ได้ซับซ้อนมาก
#จำนวนที่ train = 1000 รอบ (defualt)
```

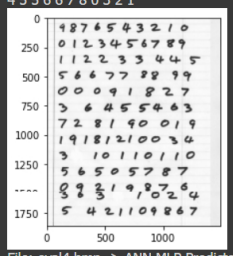
```
Iteration 355, loss = 0.01227191
Iteration 356, loss = 0.01210659
Iteration 357, loss = 0.01217254
Iteration 358, loss = 0.01231890
Iteration 359, loss = 0.01215058
Iteration 360, loss = 0.01209627
Iteration 361, loss = 0.01200105
Iteration 362, loss = 0.01223809
Iteration 363, loss = 0.01188200
Iteration 364, loss = 0.01196855
Iteration 365, loss = 0.01171195
Iteration 366, loss = 0.01164978
Iteration 367, loss = 0.01173047
Iteration 368, loss = 0.01158283
Iteration 369, loss = 0.01152956
Iteration 370, loss = 0.01159889
Iteration 371, loss = 0.01149102
Iteration 372, loss = 0.01139309
Iteration 373, loss = 0.01141359
Iteration 374, loss = 0.01151583
Iteration 375, loss = 0.01129388
Iteration 376, loss = 0.01116488
Iteration 377, loss = 0.01140688
Iteration 378, loss = 0.01112833
Iteration 379, loss = 0.01138949
Iteration 380, loss = 0.01144899
Iteration 381, loss = 0.01104107
Iteration 382, loss = 0.01098615
Iteration 383, loss = 0.01096674
Iteration 384, loss = 0.01089284
Iteration 385, loss = 0.01090528
Iteration 386, loss = 0.01075737
Iteration 387, loss = 0.01089369
Iteration 388, loss = 0.01071945
Iteration 389, loss = 0.01075042
Iteration 390, loss = 0.01062737
Iteration 391, loss = 0.01066155
Iteration 392, loss = 0.01066933
Iteration 393, loss = 0.01069259
Iteration 394, loss = 0.01039266
Iteration 395, loss = 0.01036355
Iteration 396, loss = 0.01038817
Iteration 397, loss = 0.01029436
Iteration 398, loss = 0.01060630
Iteration 399, loss = 0.01019475
Iteration 400, loss = 0.01048329
Iteration 401, loss = 0.01020947
Iteration 402, loss = 0.01017813
Iteration 403, loss = 0.01032791
Iteration 404, loss = 0.01025310
Iteration 405, loss = 0.01012601
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(50,), learning_rate='constant',
              learning_rate_init=0.1, max_fun=15000, max_iter=1000,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=1, shuffle=True, solver='sgd',
              tol=0.0001, validation_fraction=0.1, verbose=10,
              warm_start=False)
```

[22]
```
print(np_testingSeparatedFeature[500])
print(np_testingSeparatedLabel[500])
test_ex = scaler.transform([np_testingSeparatedFeature[600],np_testingSeparatedFeature[500]])
print(mlp.predict(test_ex))
```

```
[2.25454545 4.78181818 3.81818182 6.71304348 5.47272727 0.
 3.2        8.93913043 5.25454545 5.25454545 3.8        5.68695652
 0.         0.         0.         5.88768116]
9
[[0 0 1 0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0 0 0 1]]
```

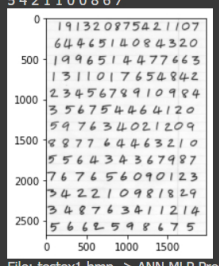
[23]
```
# test prediction using Testing Images Set
for p,page in enumerate(testSeparatedImages_Binary):
  plt.imshow(TestImage[p])
  plt.show()
  print(f"File: {TestImageFilenames[p]} -> ANN MLP Predicted :")
  for l,line in enumerate(page):
    features_inLine = []
    for c,char in enumerate(line):
      features_inLine.append(calculateDensityFeature(divideBlock(char,4,4,0))) # get features & insert in each line
    norm_features_inLine = scaler.transform(np.array(features_inLine)) # to np array & normalize data
    results = mlp.predict(norm_features_inLine)
    for predictedLabel in results:
      print(f"{np.argmax(predictedLabel)} ", end = '')
    print(" ") # new line
```

```
4 5 5 6 6 7 8 0 3 2 1
```




```
File: eval4.bmp -> ANN MLP Predicted :
9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9
1 1 2 2 3 3 4 4 5
5 6 6 6 7 7 8 8 0 9
0 0 0 9 1 8 2 7
```

```
3 6 4 5 5 4 6 3
7 2 8 1 0 0 0 1 0
1 9 1 8 1 2 1 0 0 3 4
3 1 0 1 1 0 1 1 0
5 6 5 0 5 7 0 7
0 9 2 1 9 8 7 0
3 6 3 1 0 2 4
5 4 2 1 1 0 0 8 6 7
```



```
File: testex1.bmp -> ANN MLP Predicted :
1 9 1 3 2 0 8 7 5 4 2 1 1 0 7
6 4 4 6 5 1 0 0 8 4 3 2 0
1 9 9 6 5 1 4 4 7 7 6 6 3
1 3 1 1 0 1 7 6 5 4 8 4 2
2 3 4 5 6 7 8 9 1 0 9 8 4
3 5 6 7 5 4 4 6 4 1 2 0
5 4 7 6 3 4 0 2 1 2 0 9
0 8 7 7 6 4 4 6 3 2 1 0
5 5 6 4 3 4 3 6 7 9 8 7
7 6 7 6 5 6 0 9 0 1 2 3
3 4 2 2 1 0 9 8 1 8 2 9
3 4 8 7 6 3 4 1 1 2 1 4
5 6 6 2 5 9 8 6 7 5
```

ANN-Metric

```
[24] # normalize testing features data
     testingSeparatedFeature_normed = []
     for c,char_feature in enumerate(testingSeparatedFeature):
       tmp = []
       tmp.append(char_feature)
       testingSeparatedFeature_normed.append(scaler.transform(np.array(tmp))[0])
     # prediction
     result_Predicted_ANN = mlp.predict(testingSeparatedFeature_normed)

     # convert one hot to label 0-9
     result_Predicted_ANN_Converted = []
     for i in range(result_Predicted_ANN.shape[0]):
       result_Predicted_ANN_Converted.append(np.argmax(result_Predicted_ANN[i]))

     # Metric prediction of ANN
     TP = 0
     FP = 0
     for k in range(len(testingSeparatedLabel)):
       if(result_Predicted_ANN_Converted[k]==testingSeparatedLabel[k]):
         TP+=1
       else:
         FP+=1
     print(f"-------- ANN MLP Metric --------\n ")
     print(f"TP={TP} FP={FP} \n")
     print(f"Precision= {TP/(TP+FP)*100:.2f}%")
```

```
-------- ANN MLP Metric --------

TP=614 FP=18

Precision= 97.15%
```

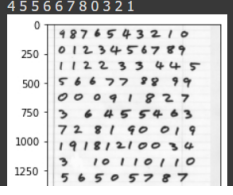
K Nearest Neighbors

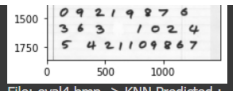
```
[25] from sklearn.neighbors import KNeighborsClassifier
     knn = KNeighborsClassifier(n_neighbors=5)
     knn.fit(trainingSeparatedFeature, trainingSeparatedLabel)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

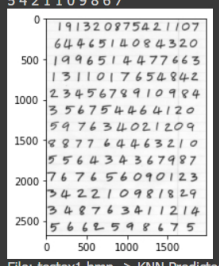
```
[26] # test prediction using Testing Images Set
     results = []
     for p,page in enumerate(testSeparatedImages_Binary):
       plt.imshow(TestImage[p])
       plt.show()
       print(f"File: {TestImageFilenames[p]} -> KNN Predicted :")
       for l,line in enumerate(page):
         features_inLine = []
         for c,char in enumerate(line):
           features_inLine.append(calculateDensityFeature(divideBlock(char,4,4,0))) # get features & insert in each line
         #norm_features_inLine = scaler.transform(np.array(features_inLine)) # to np array & normalize data
         results = knn.predict(features_inLine)
         for predictedLabel in results:
           print(f"{predictedLabel}", end = '')
         print(" ") # new line
```

```
4 5 5 5 6 6 7 8 0 3 2 1
```





```
File: eval4.bmp -> KNN Predicted :
9 8 7 6 5 4 3 2 1 0
0 1 2 3 4 5 6 7 8 9
1 1 2 2 3 3 4 4 5
5 6 6 7 7 8 8 9 9
0 0 0 9 1 8 2 7
3 6 4 5 5 4 6 3
7 2 8 1 4 0 0 1 9
1 9 1 8 1 2 1 0 0 3 4
3 1 0 1 1 0 1 1 0
0 9 2 1 9 8 7 6
3 6 3 1 0 2 4
5 4 2 1 1 0 9 8 6 7
```



```
File: testex1.bmp -> KNN Predicted :
1 9 1 3 2 0 8 7 5 6 2 1 1 0 7
6 4 4 6 5 1 2 0 8 4 3 2 0
1 9 9 6 5 1 4 4 7 7 6 6 3
1 3 1 1 0 1 7 6 5 4 8 4 2
2 3 4 5 6 7 8 9 1 0 9 8 4
3 5 6 7 5 4 4 6 4 1 2 0
5 9 7 6 3 4 0 2 1 2 0 9
8 8 7 7 6 4 4 6 3 2 1 0
5 5 6 4 3 4 3 6 7 9 8 7
7 6 7 6 5 6 0 9 0 1 2 3
3 4 2 2 1 0 9 8 1 8 2 9
3 4 8 7 6 3 4 1 1 2 1 4
5 6 6 2 5 9 8 6 7 5
```

KNN-Metric

```
[27] # prediction
result_Predicted_KNN = knn.predict(testingSeparatedFeature)

# Metric prediction of ANN
TP = 0
FP = 0
for k in range(len(testingSeparatedLabel)):
  if(result_Predicted_KNN[k]==testingSeparatedLabel[k]):
    TP+=1
  else:
    FP+=1
print(f"-------- KNN MLP Metric --------\n ")
print(f"TP={TP} FP={FP} \n")
print(f"Precision= {TP/(TP+FP)*100:.2f}%")
```

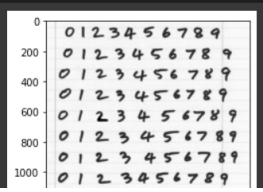
```
-------- KNN MLP Metric --------

TP=625 FP=7

Precision= 98.89%
```

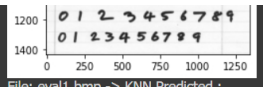
SVM

```
[28] from sklearn import svm
from sklearn.svm import SVC
svmObject = svm.SVC(decision_function_shape='ovo',kernel = 'rbf') # ovo = one-vs-one ('ovo') is always used as multi-class strategy
svmObject.fit(trainingSeparatedFeature, trainingSeparatedLabel)
```

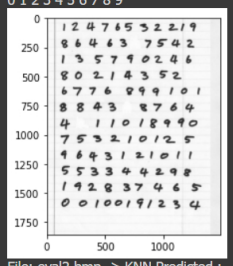
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovo', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

```
[29] # test prediction using Testing Images Set
results = []
for p,page in enumerate(testSeparatedImages_Binary):
  plt.imshow(TestImage[p])
  plt.show()
  print(f"File: {TestImageFilenames[p]} -> KNN Predicted :")
  for l,line in enumerate(page):
    features_inLine = []
    features_inLine.append(calculateDensityFeature(divideBlock(char,4,4,0))) # get features & insert in each line
    #norm_features_inLine = scaler.transform(np.array(features_inLine)) # to np array & normalize data
    results = svmObject.predict(features_inLine)
    for predictedLabel in results:
      print(f"{predictedLabel} ", end = '')
    print(" ") # new line
```

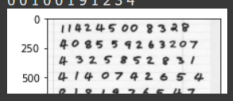




```
File: eval1.bmp -> KNN Predicted :
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```



```
File: eval2.bmp -> KNN Predicted :
1 2 4 7 6 5 3 2 2 1 9
8 6 4 6 3 7 5 4 2
1 3 5 7 9 0 2 4 6
8 0 2 1 4 3 5 2
6 7 7 6 8 9 9 1 0 1
8 8 4 3 8 7 6 4
4 1 1 0 1 8 9 9 0
7 5 3 2 1 0 1 2 5
9 6 4 3 1 2 1 0 1 1
5 5 3 3 4 4 2 9 8
1 9 2 8 3 7 4 6 8
0 0 1 0 0 1 9 1 2 3 4
```



SVM-Metric

```
[30]  result_Predicted_SVM = svmObject.predict(testingSeparatedFeature)

      # Metric prediction of ANN
      TP = 0
      FP = 0
      for k in range(len(testingSeparatedLabel)):
        if(result_Predicted_SVM[k]==testingSeparatedLabel[k]):
          TP+=1
        else:
          FP+=1
      print(f"-------- SVM MLP Metric --------\n ")
      print(f"TP={TP} FP={FP} \n")
      print(f"Precision= {TP/(TP+FP)*100:.2f}%")
```

```
-------- SVM MLP Metric --------

TP=627 FP=5

Precision= 99.21%
```