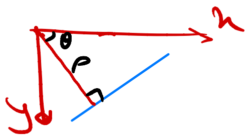
Assignment #6

Max accum. value: 4

Theory



1

black: $\rho = 0$
$\theta = -135°$
acc = 4

Binary Image = $\begin{bmatrix} 0&0&0&0&0&0&0&0&0&0 \\ 0&1&1&1&0&0&0&0&0&0 \\ 0&1&0&1&0&0&0&0&0&0 \\ 0&1&1&1&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&1&0 \\ 0&0&0&0&0&1&0&0&1&0 \\ 0&0&0&0&0&1&0&0&1&0 \\ 0&0&0&0&0&1&1&1&1&0 \\ 0&0&0&0&0&0&0&0&0&0 \end{bmatrix}$

blue: ①: $\rho_1 = 6$
$\theta_1 = 0$
$\rho_2 = 9$
$\theta_2 = 0$

[redacted]

red: $\rho = 9$ acc = 4
$\theta = 90°$

Therefore, max value in accumulator = 7 and corresponding $(p,\theta) = (-1, -45°)$.

```
M = [0 0 0 0 0 0 0 0 0 0;
     0 1 1 1 0 0 0 0 0 0;
     0 1 0 1 0 0 0 0 0 0;
     0 1 1 1 0 0 0 0 0 0;
     0 0 0 0 0 0 0 0 0 0;
     0 0 0 0 0 1 0 0 1 0;
     0 0 0 0 0 1 0 0 1 0;
     0 0 0 0 0 1 0 0 1 0;
     0 0 0 0 0 1 1 1 1 0;
     0 0 0 0 0 0 0 0 0 0;
     ];

[H, Theta, Rho] = hough(M, 'Theta', -90:1:0);
P = houghpeaks(H,1);
maxVal = H(P(1),P(2))
maxRho = Rho(P(1))
maxTheta = Theta(P(2))
```

Name ▲	Value
H	27x91 double
M	10x10 double
maxRho	-1
maxTheta	-45
maxVal	7
P	[13,46]
Rho	1x27 double
Theta	1x91 double

2

Optical flow equation: $\sum_i f_{xi}u + f_{yi}v + ft^2$
The optical flow (u,v) for a 3x3 window is the least squares fit for 9 points.
Least square derivation: $u = ((A^T * A)^{-1}) * A^T b$

Sobel calculation:

Calculate Ix using [-1 0 1] kernel: Ix= $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 255 & 255 & 0 & -255 & -255 \\ 0 & 255 & 0 & 0 & 0 & -255 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

Calculate Iy using [-1 0 1]^T kernel: Iy= $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 255 & 255 & 255 \\ 0 & 0 & -255 & 0 & -255 & 255 \\ 0 & 0 & -255 & 0 & -255 & 0 \end{bmatrix}$

Using $\frac{u}{v} = \frac{-1}{\sum I_x^2 \sum I_y^2 - \sum I_x * I_y * \sum I_x * I_y} \begin{pmatrix} \sum I_y^2 & -\sum I_x * I_y \\ -\sum I_x * I_y & \sum I_x^2 \end{pmatrix} \begin{pmatrix} \sum I_x * I_t \\ \sum I_y * I_t \end{pmatrix}$

Sample calculation for pixel (1,1):

$\sum I_x^2 = 195075$
$\sum I_y^2 = 65025$
$\sum I_x * I_y = 0$
$\sum I_x * I_t = -130050$
$\sum I_y * I_t = -65025$
$u_{1,1} = 0.67$
$v_{1,1} = 1$

Optical Flow for entire image:

u= $\begin{bmatrix} nan & nan & 0.5 & 0.33 & 0.5 & nan \\ nan & 0.67 & 0.67 & 0.71 & 0.83 & 1.33 \\ nan & 0.67 & 0.67 & 0.64 & 0.67 & 0.83 \\ nan & 1 & 1 & nan & 0.50 & 0.50 \end{bmatrix}$

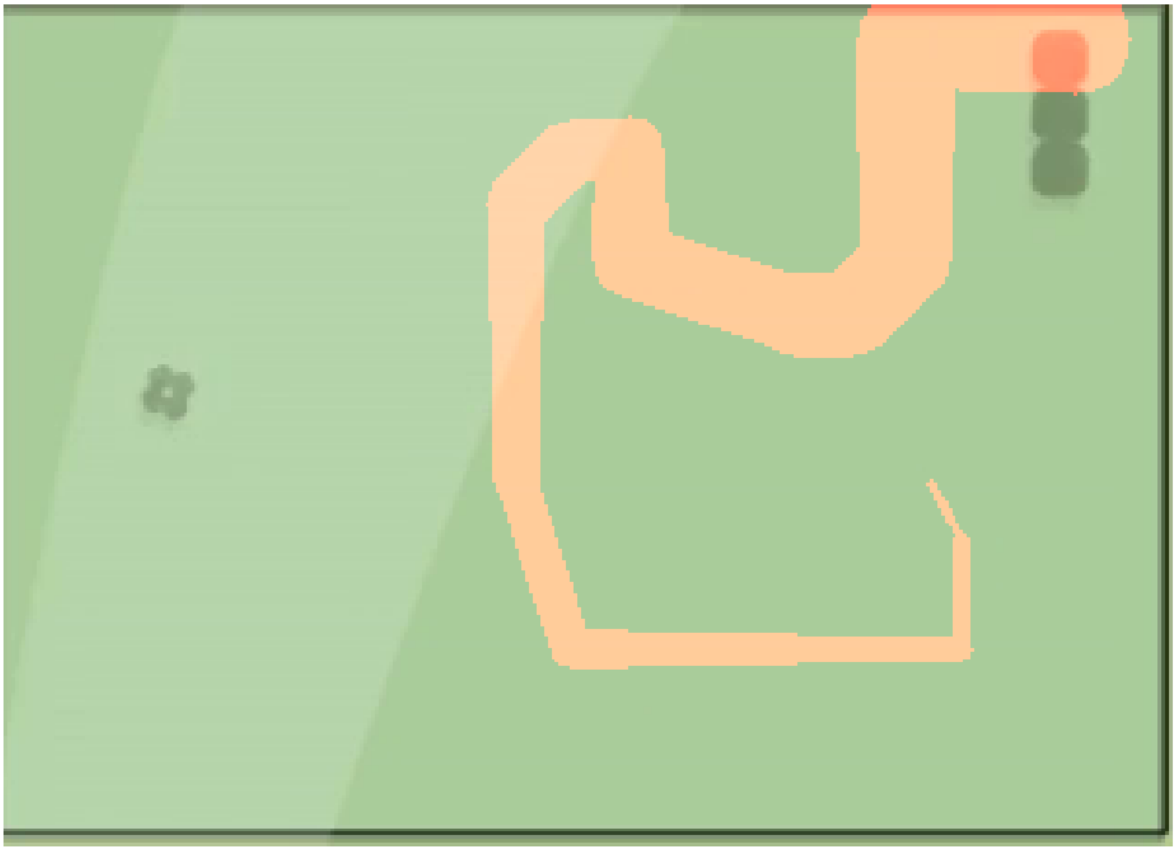
v= $\begin{bmatrix} nan & nan & 0 & 0.33 & 0 & nan \\ nan & 1.00 & 0.5 & 0.43 & 0.5 & 1 \\ nan & 0.5 & 0.33 & 0.27 & 0.33 & 0.5 \\ nan & 0.5 & 0.5 & nan & 0.5 & 0.5 \end{bmatrix}$

3

The Hough transform can be used to detect the orientation of this image. Hough transform will output the angle and radius perpendicular to the lines on the triangle. Whether the rectangle is parallel to the x-axis or if it is rotated can be determined using the angle value from the outputted parameters.

Implementation

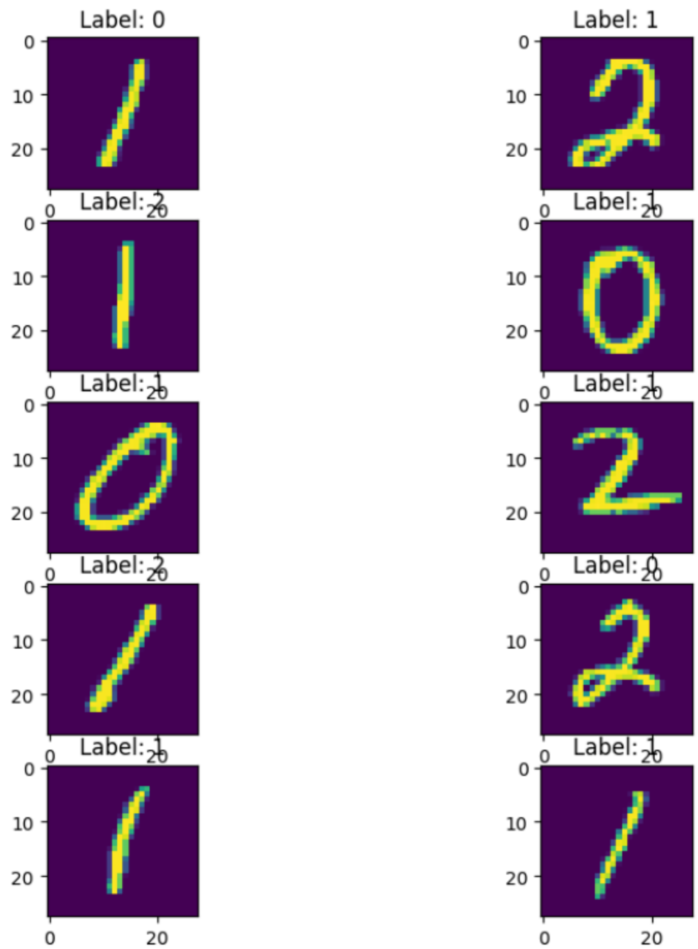
1



```python
import cv2 as cv
import argparse
import numpy as np
from collections import deque
def main(args):
    cap = cv.VideoCapture(args.video)
    trail = deque(maxlen=128)
    featureParams = dict( maxCorners = 50,
                          qualityLevel = 0.3,
                          minDistance = 7,
                          blockSize = 7 )
    lkParams = dict( winSize = (15, 15),
                          maxLevel = 2,
                          criteria = (cv.TERM_CRITERIA_EPS | cv.TERM_CRITERIA_COUNT, 30, 0.03))
    _, oldFrame = cap.read()
    oldGray = cv.cvtColor(oldFrame, cv.COLOR_BGR2GRAY)
    mask = np.zeros_like(oldGray)
    cv.rectangle(mask, (245, 125), (270, 180),255, -1)
    p0 = (cv.goodFeaturesToTrack(oldGray, mask=mask, **featureParams))
    trail.appendleft((p0[1].ravel()))
    while(1):
        ok, frame = cap.read()
        if not ok:
            break
        frameGray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
        p1, st, err = cv.calcOpticalFlowPyrLK(oldGray, frameGray, p0, None, **lkParams)
        if p1 is not None:
            goodNew = p1[np.logical_and((err < 50), (st==1))]
            goodOld = p0[np.logical_and((err < 50), (st==1))]
        if len(goodNew) is not 0:
            trail.appendleft((goodNew[0].ravel()))
        mask = np.zeros_like(oldFrame)
        thickness = len(trail)
        for i in range(1, len(trail)):
            mask = cv.line(mask, (trail[i - 1]).astype(np.int), (trail[i]).astype(np.int), (0, 0, 255), thickness)
            thickness -= 1
        img = cv.add(frame, mask)
        cv.imshow('frame', img)
        k = cv.waitKey(100) & 0xff
        if k == 27:
            break
        oldGray = frameGray.copy()
        p0 = goodNew.reshape(-1, 1, 2)
if __name__ == '__main__':
    ap = argparse.ArgumentParser(description='applies Lucas-Kanade Optical Flow calculation to track the snake')
    ap.add_argument("-v", "--video", required=False, default="./video1.mp4",
    help="path to the image file")
    args = ap.parse_args()
    main(args)
```

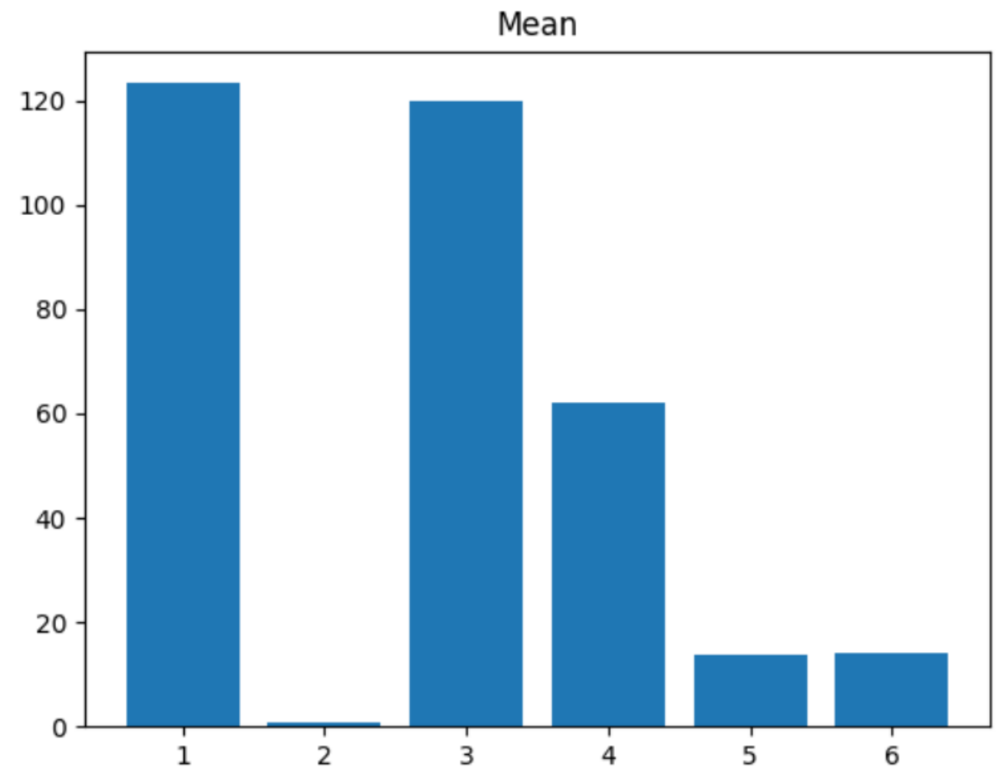
2

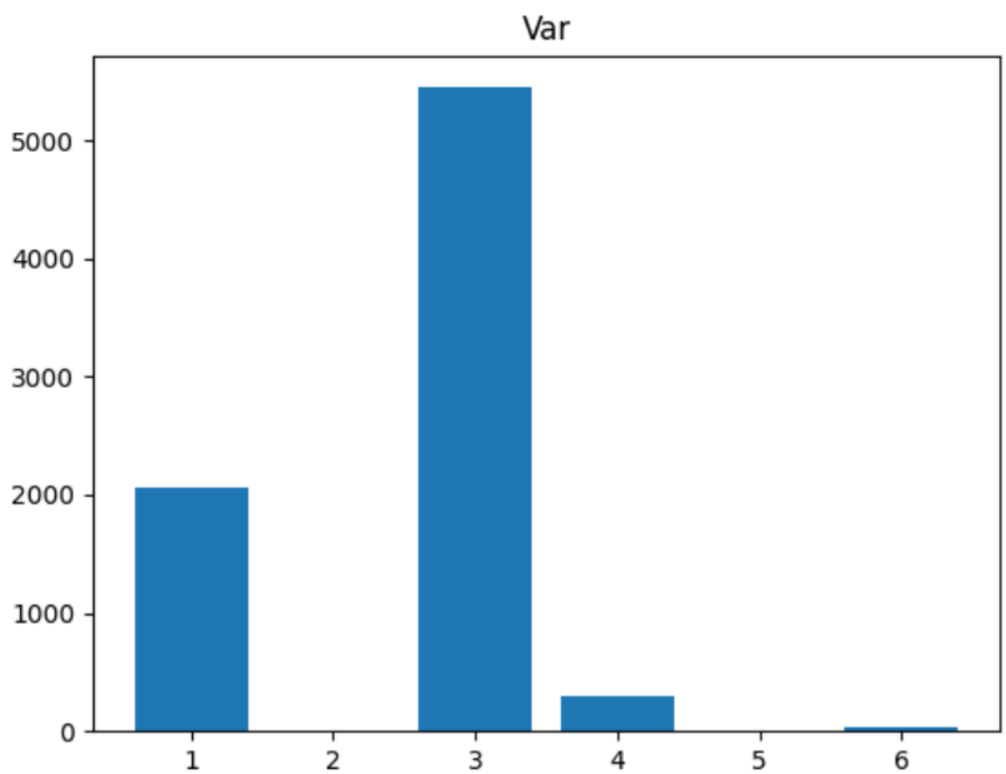
Extracted 10 random samples of 0, 1 and 2 digits.



2.1

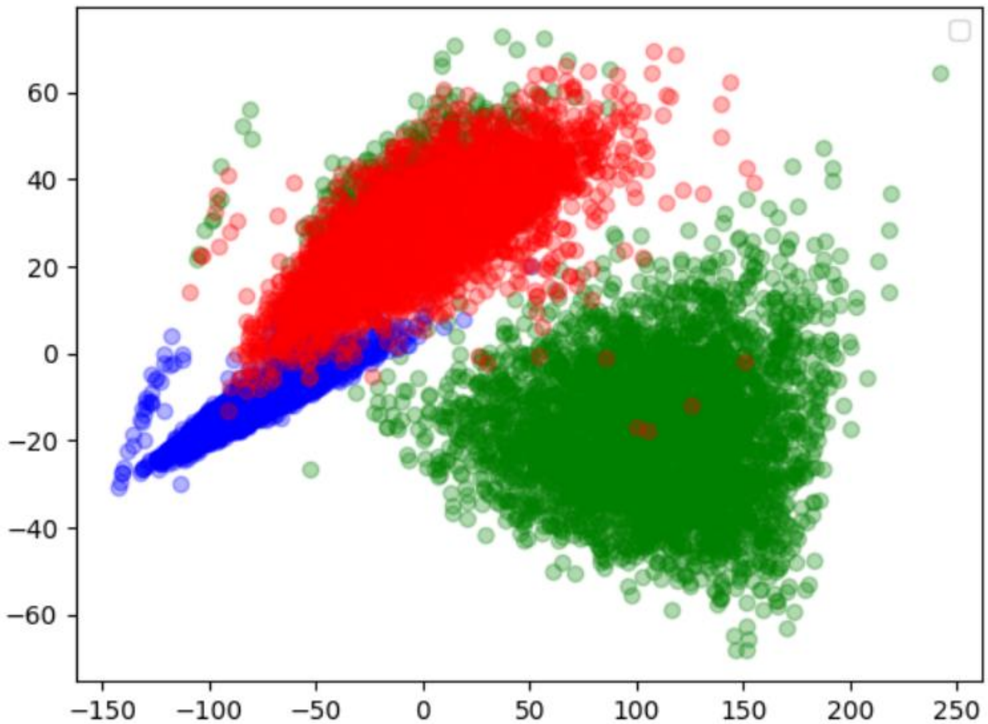
Based on the plots of mean and variance: it seems only 4 features are useful for classification. To decrease variance, the useful features are: area, number of pixels, perimeter and width.


Mean
0
20
40
60
80
100
120
1
2
3
4
5
6


Var
0
1000
2000
3000
4000
5000
1
2
3
4
5
6

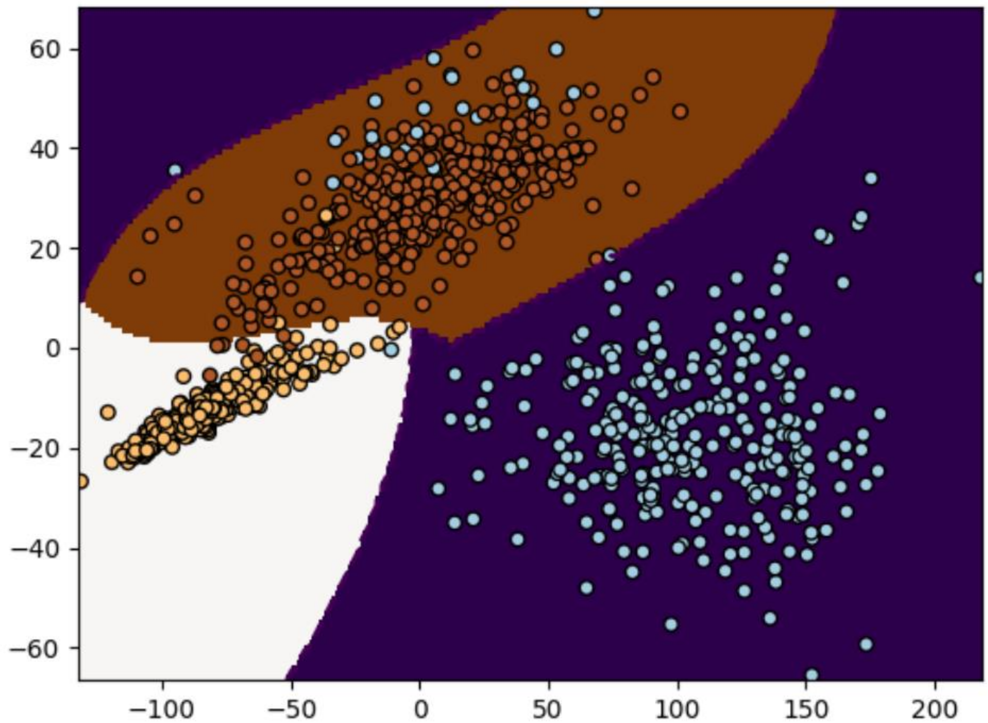
2.2

Visualization of reduced feature set of 0, 1 and 2 digits.



2.3

Using reduced set of 2 features, boundaries created by rbf SVM are shown below.



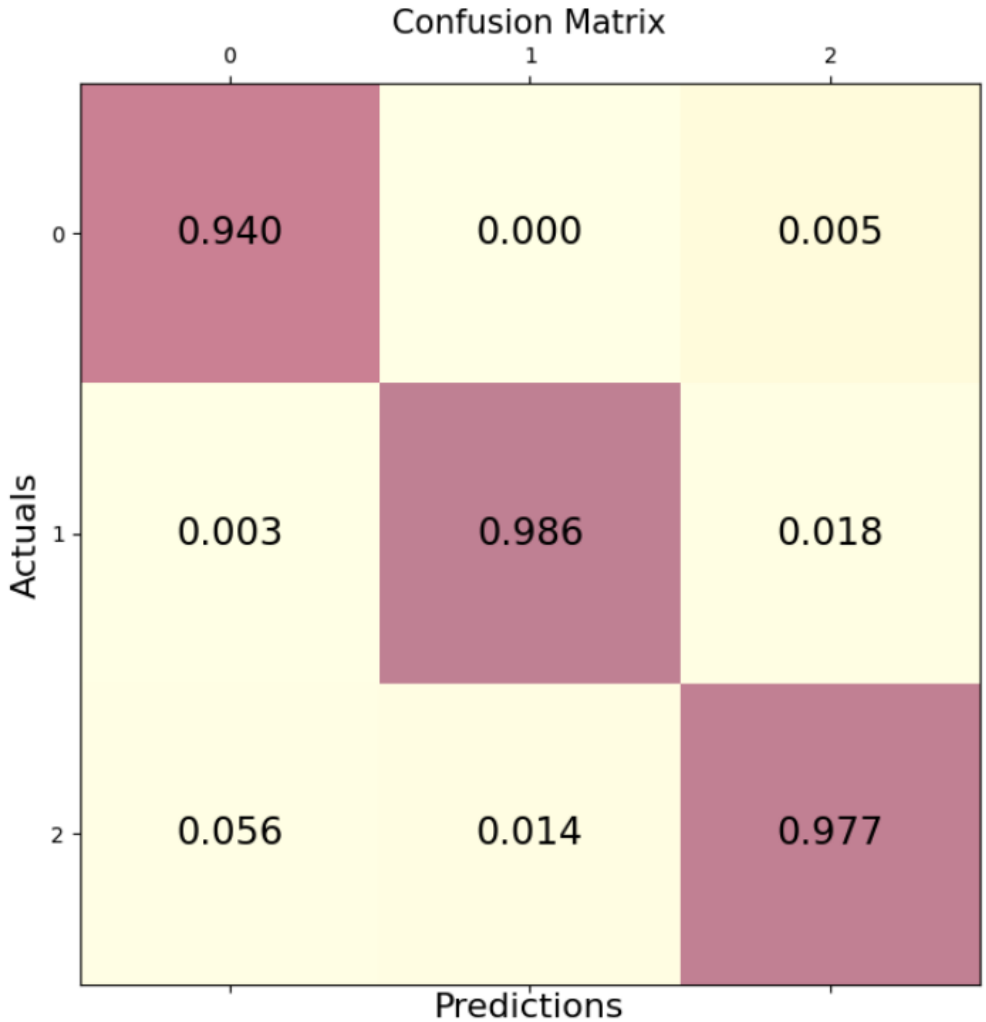
2.4

Confusion matrices for training data after validation are shown below.

```
Training with 1 features
[[0.794 0.002 0.204]
 [0.    0.949 0.051]
 [0.074 0.062 0.864]]
[0.85  0.946 0.816]
Training with 2 features
[[0.903 0.003 0.094]
 [0.    0.984 0.016]
 [0.007 0.021 0.973]]
[0.946 0.981 0.935]
Training with 3 features
[[0.912 0.003 0.086]
 [0.    0.986 0.014]
 [0.009 0.022 0.969]]
[0.949 0.982 0.937]
Training with 4 features
[[0.912 0.002 0.086]
 [0.    0.987 0.013]
 [0.008 0.023 0.969]]
[0.95  0.982 0.937]
Training with 5 features
[[0.912 0.002 0.086]
 [0.    0.988 0.012]
 [0.008 0.023 0.969]]
[0.95  0.983 0.938]
Training with 6 features
[[0.912 0.002 0.086]
 [0.    0.988 0.012]
 [0.007 0.023 0.969]]
[0.95  0.983 0.938]
```

2.5

Based on the validated confusion matrices, the optimal number of features is 4. The 5th and 6th feature don't add much of a benefit to be worth it. The confusion matrix generated when training the SVM with 4 features is shown below.



```python
import idx2numpy
import numpy as np
import cv2 as cv
import matplotlib.pyplot as plt
from sklearn import svm, metrics
def displayRandom10(images, labels):
    fig = plt.figure(figsize=(9, 13))
    columns = 2
    rows = 5
    ax = []
    for i in range(columns*rows):
        img = images[i+911]
        ax.append( fig.add_subplot(rows, columns, i+1) )
        ax[-1].set_title("Label: " + str(labels[i]))
        plt.imshow(img)
    plt.show()
def preprocessing(images):
    for i in range(len(images)):
        temp = cv.GaussianBlur(images[i], (3,3), 0)
        _, images[i] = cv.threshold(temp,0,255,cv.THRESH_BINARY+cv.THRESH_OTSU)
    return images
def getContFeatures(images, labels):
    X = []
    L = []
    for i in range(len(images)):
        contours, _ = cv.findContours(images[i], cv.RETR_TREE, cv.CHAIN_APPROX_SIMPLE)
        cnt = contours[0]
        if cnt is None:
            print("error, no cont detected")
            exit()
        M = cv.moments(cnt)
        cy = int(M['m01']/(M['m00'] + 1e-5))
        _ , _, width, height = cv.boundingRect(cnt)
        X.append([cv.countNonZero(images[i]),
                  float(width/height),
                  cv.contourArea(cnt),
                  cv.arcLength(cnt,True),
                  cy,
                  width])
        L.append(labels[i])
    return X, L
def printMeanAndVar(X):
    m = np.mean(X, axis=0)
    v = np.var(X, axis=0)
    plt.bar([1,2,3,4,5,6],m)
    plt.title("Mean")
    plt.show()
    plt.bar([1,2,3,4,5,6],v)
    plt.title("Var")
    plt.show()
def dimensionReduction(features, numFeaturesToKeep):
    features = np.array(features)
    mean = np.empty((0))
    mean, eigenvectors, _ = cv.PCACompute2(np.array(features), mean, maxComponents=numFeaturesToKeep)
    reducedFeatureSet = cv.PCAProject(features, mean, eigenvectors)
    return reducedFeatureSet
def trainSVM(data, labels):
    labels = np.array(labels, dtype='int32')
    data = data.astype('float32')
    clf = svm.SVC(kernel="rbf", gamma="scale", C=0.8)
    clf.fit(data, labels)
    return clf
def make_meshgrid(x, y, h=0.02):
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
    return xx, yy
def plot_contours(data, labels, clf):
    print("visualizing")
    X0, X1 = data[:, 0], data[:, 1]
    xx, yy = make_meshgrid(X0, X1, 1)
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

```python
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    plt.imshow(
        Z,
        interpolation="nearest",
        extent=(xx.min(), xx.max(), yy.min(), yy.max()),
        aspect="auto",
        origin="lower",
        cmap=plt.cm.PuOr_r,
    )
    plt.contour(xx, yy, Z, levels=[0], linewidths=2, linestyles="dashed")
    plt.scatter(X0, X1, s=30, c=labels, cmap=plt.cm.Paired, edgecolors="k")
    plt.show()
def printConfusionmatrix(actual, predicted, simple=True):
    cmatrix = metrics.confusion_matrix(actual, predicted, normalize='true')
    if not simple:
        fig, px = plt.subplots(figsize=(7.5, 7.5))
        px.matshow(cmatrix, cmap=plt.cm.YlOrRd, alpha=0.5)
        for m in range(cmatrix.shape[0]):
            for n in range(cmatrix.shape[1]):
                px.text(x=m,y=n,s="{:.3f}".format(cmatrix[m, n]), va='center', ha='center', size='xx-large')
        plt.xlabel('Predictions', fontsize=16)
        plt.ylabel('Actuals', fontsize=16)
        plt.title('Confusion Matrix', fontsize=15)
        plt.show()
    else:
        np.set_printoptions(precision=3)
        print(cmatrix)
imgFilePath = './data/train-images-idx3-ubyte'
labelFilePath = './data/train-labels-idx1-ubyte'
images = idx2numpy.convert_from_file(imgFilePath)
labels = idx2numpy.convert_from_file(labelFilePath)
images = images[labels <= 2]
labels = labels[labels <= 2]
displayRandom10(images, labels)
total = labels.size
trainSize = int(0.7*total)
validationSize = int(0.2*total)
trainingImgs = images[:trainSize]
trainingLabels = labels[:trainSize]
validationImgs = images[trainSize:validationSize+trainSize]
validationLabels = labels[trainSize:validationSize+trainSize]
testImgs = images[validationSize+trainSize:]
testLabels = labels[validationSize+trainSize:]
trainingImgs = preprocessing(trainingImgs)
imgFeatures, labels = getContFeatures(trainingImgs, trainingLabels)
printMeanAndVar(imgFeatures)
reducedFeatures = dimensionReduction(imgFeatures, 2)
plt.scatter((reducedFeatures[:,0])[np.isin(labels, 0)], (reducedFeatures[:,1])[np.isin(labels, 0)], color='green', alpha=0.3)
plt.scatter((reducedFeatures[:,0])[np.isin(labels, 1)], (reducedFeatures[:,1])[np.isin(labels, 1)], color='blue', alpha=0.3)
plt.scatter((reducedFeatures[:,0])[np.isin(labels, 2)], (reducedFeatures[:,1])[np.isin(labels, 2)], color='red', alpha=0.3)
plt.legend()
plt.show()
clf = trainSVM(reducedFeatures, labels)
plot_contours(reducedFeatures[0:1000, :], labels[0:1000], clf)
predicted = clf.predict(reducedFeatures)
printConfusionmatrix(labels, predicted, simple=False)
validationImgs = preprocessing(validationImgs)
validationImgFeatures, validationLabels = getContFeatures(validationImgs, validationLabels)
f1_0 = []
f1_1 = []
f1_2 = []
for n in range(1, 6+1):
    reducedFeatures = dimensionReduction(imgFeatures, n)
    print("Training with " + str(n) + " features")
    clf = trainSVM(reducedFeatures, labels)
    validationReducedFeatures = dimensionReduction(validationImgFeatures, n)
    predicted = clf.predict(validationReducedFeatures)
    printConfusionmatrix(validationLabels, predicted)
    f1 = metrics.f1_score(validationLabels, predicted, average=None)
```

```python
    validationReducedFeatures = dimensionReduction(validationImgFeatures, n)
    predicted = clf.predict(validationReducedFeatures)
    printConfusionmatrix(validationLabels, predicted)
    f1 = metrics.f1_score(validationLabels, predicted, average=None)
    f1_0.append(f1[0])
    f1_1.append(f1[1])
    f1_2.append(f1[2])
    print(f1)
testImgs = preprocessing(testImgs)
testImgFeatures, testLabels = getContFeatures(testImgs, testLabels)
n_opt = 4
reducedFeatures = dimensionReduction(imgFeatures, n_opt)
print("Training with " + str(n_opt) + " features")
clf = trainSVM(reducedFeatures, labels)
testReducedFeatures = dimensionReduction(testImgFeatures, n_opt)
predicted = clf.predict(testReducedFeatures)
printConfusionmatrix(testLabels, predicted, simple=False)
f1 = metrics.f1_score(testLabels, predicted, average=None)
print(f1)
```