CVE Assignment 8b Report

Question 1 (PS8b)

Support Vector Classifier

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labelled training data for each category, they're able to categorize new text. Primarily used as a binary classifier, the Support Vector Classifier is especially relevant to our requirement of whether an image has a person in it or not. The Support Vector Machine implementation of Scikit-Learn was used for this model.

Kernel Function is a method used to take data as input and transform it into the required form of processing data. "Kernel" is used due to a set of mathematical functions used in Support Vector Machine providing the window to manipulate the data. So, Kernel Function generally transforms the training set of data so that a non-linear decision surface can transform to a linear equation in a higher number of dimension spaces. For this model, the SVC classifier with the default parameters were used while changing only the kernel being used to benchmark the relative efficiency of each kernel. The following kernels were tested and the results.

1. *Linear Kernel*
   Linear Kernel is used when the data is Linearly separable, that is, it can be separated using a single Line. It is one of the most common kernels to be used. It is mostly used when there are a Large number of Features in a particular Data Set. With the Linear Kernel implemented in the SVM Classifier, the following Confusion Matrix was obtained.

	precision	recall	f1-score	support
0	0.93	0.95	0.94	830
1	0.90	0.88	0.89	483
accuracy			0.92	1313
macro avg	0.92	0.91	0.92	1313
weighted avg	0.92	0.92	0.92	1313

   The results how at least 90% accuracy for classifying the image as positive or negative. A threshold value of 0.6 provided a good classification for all the images. This kernel provided the best results of all alternatives.

2. *Poly Kernel*
   It represents the similarity of vectors in the training set of data in a feature space over polynomials of the original variables used in the kernel. In machine learning, the polynomial kernel is a kernel function commonly used with support vector machines (SVMs) and other kernelized models, that represents the similarity of vectors (training samples) in a feature space over polynomials of the original variables, allowing learning

of non-linear models. With the Polynomial Kernel implemented in the SVM Classifier, the following Confusion Matrix was obtained.

	precision	recall	f1-score	support
0	0.96	0.98	0.97	830
1	0.97	0.92	0.95	483
accuracy			0.96	1313
macro avg	0.96	0.95	0.96	1313
weighted avg	0.96	0.96	0.96	1313

The results how at least 96% accuracy for classifying the image as positive or negative. A threshold of 0.4 gave good results for all images except the person in the woods, which required a threshold of 0.1.

3. *RBF Kernel*

Radial Basis Kernel is a kernel function that is used in machine learning to find a non-linear classifier or regression line. The main motive of the kernel is to do calculations in any d-dimensional space where d > 1, so that we can get a quadratic, cubic or any polynomial equation of large degree for our classification/regression line. Since Radial basis kernel uses exponent and as we know the expansion of e^x gives a polynomial equation of infinite power, so using this kernel, we make our regression/classification line infinitely powerful too.

	precision	recall	f1-score	support
0	0.94	0.99	0.97	830
1	0.98	0.90	0.94	483
accuracy			0.96	1313
macro avg	0.96	0.94	0.95	1313
weighted avg	0.96	0.96	0.95	1313

The results how at least 94% accuracy for classifying the image as positive or negative. A threshold of 0.2 gave good results for all images except the person in the woods, which was unable to classify the image.

4. *Sigmoid Kernel*

This function is equivalent to a two-layer, perceptron model of the neural network, which is used as an activation function for artificial neurons. The sigmoid kernel was quite popular for support vector machines due to its origin from neural networks. However, as the kernel matrix may not be positive semidefinite (PSD), it is not widely used and the behaviour is unknown.

	precision	recall	f1-score	support
0	0.86	0.86	0.86	830
1	0.76	0.77	0.76	483
accuracy			0.82	1313
macro avg	0.81	0.81	0.81	1313
weighted avg	0.82	0.82	0.82	1313

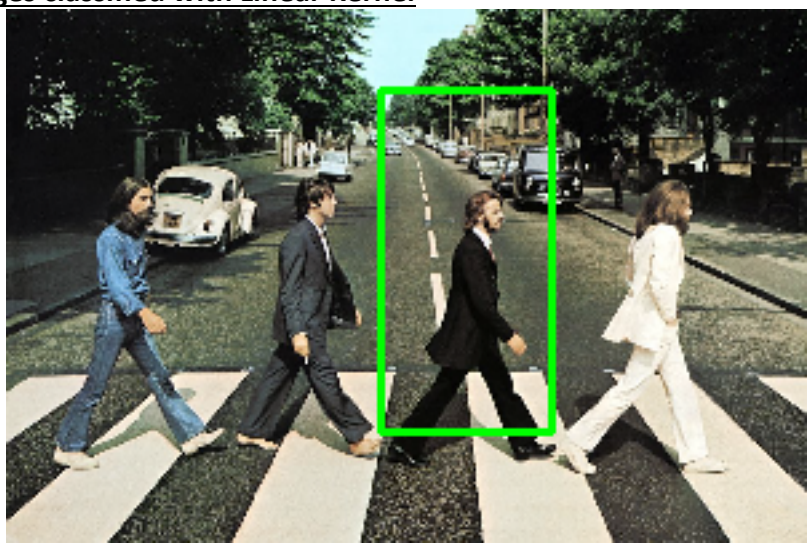
Clearly the results of classification are very poor compared to the other kernels. With the low precision in classification, a threshold of more than 1 was required to get any tangible results for the bounding box.

Histogram of Oriented Gradients

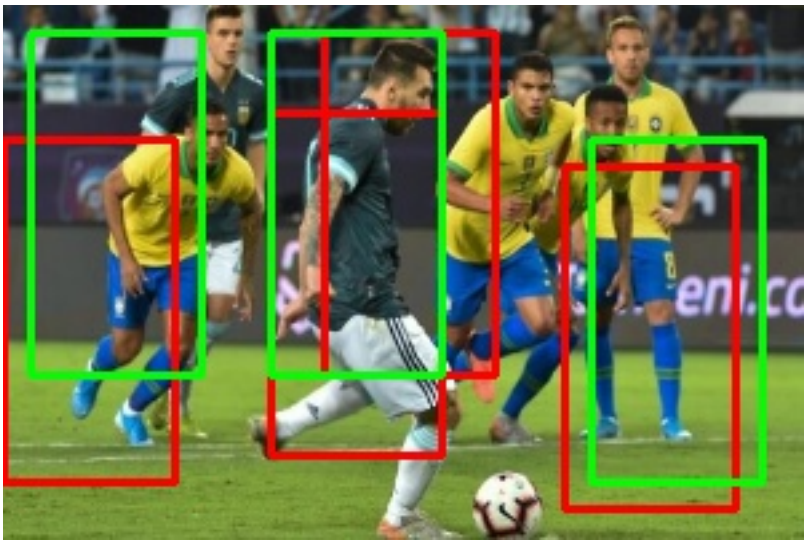
Histogram of Oriented Gradients, also known as HOG, is a feature descriptor like the Canny Edge Detector, SIFT (Scale Invariant and Feature Transform) . It is used in computer vision and image processing for the purpose of object detection. The technique counts occurrences of gradient orientation in the localized portion of an image. This method is quite similar to Edge Orientation Histograms and Scale Invariant Feature Transformation (SIFT). The HOG descriptor focuses on the structure or the shape of an object. It is better than any edge descriptor as it uses magnitude as well as angle of the gradient to compute the features. For the regions of the image it generates histograms using the magnitude and orientations of the gradient.

The HOG person detector uses a sliding detection window which is moved around the image. At each position of the detector window, a HOG descriptor is computed for the detection window. This descriptor is then shown to the trained SVM, which classifies it as either “person” or “not a person”. When using the Histogram of Oriented Gradients descriptor and a Linear Support Vector Machine for object classification you almost always detect multiple bounding boxes surrounding the object you want to detect. Instead of returning all of the found bounding boxes first apply non-maximum suppression to ignore bounding boxes that significantly overlap each other. This could be a possible reason why the Beatles image only classify a single person in the bounding box.

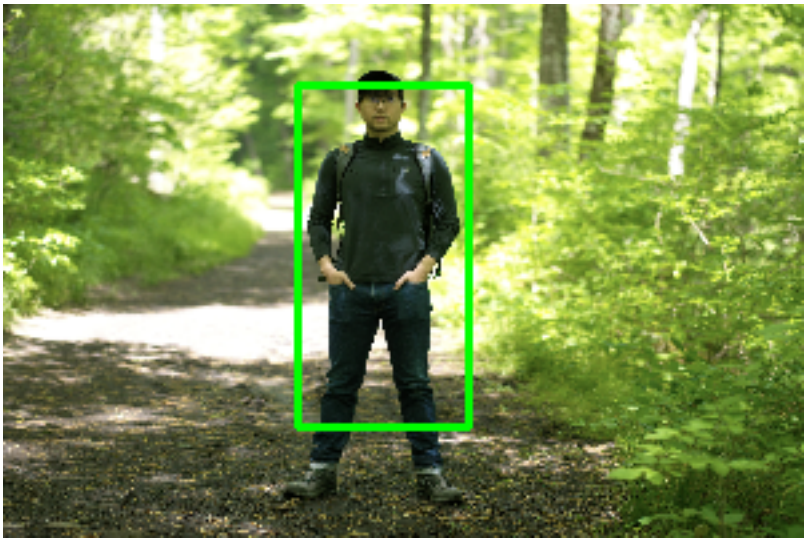
Results of images classified with Linear Kernel



Beetles.png with Linear Kernel

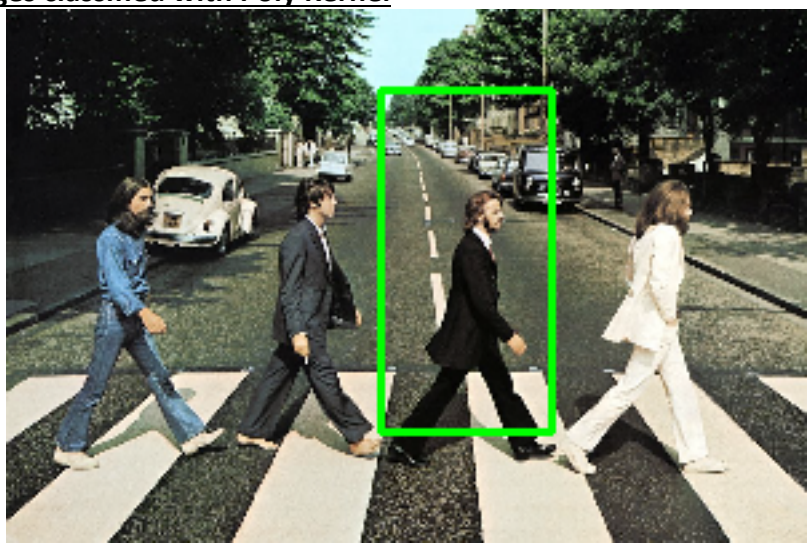


Football-field.png with Linear Kernel

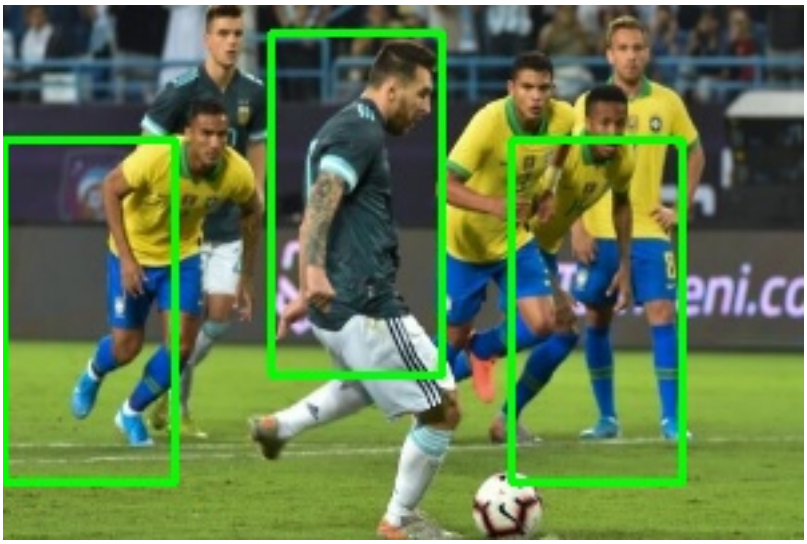


Person-in-the-woods.png with Linear Kernel

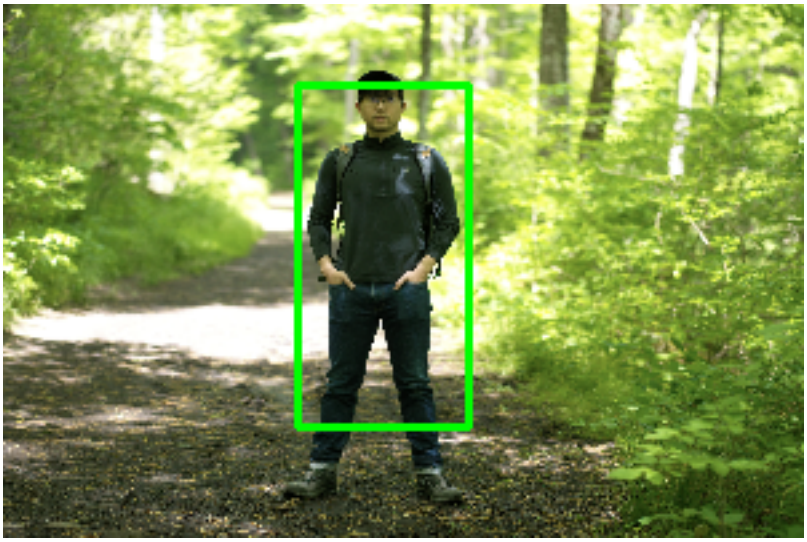
Results of images classified with Poly Kernel



Beetles.png with Poly Kernel

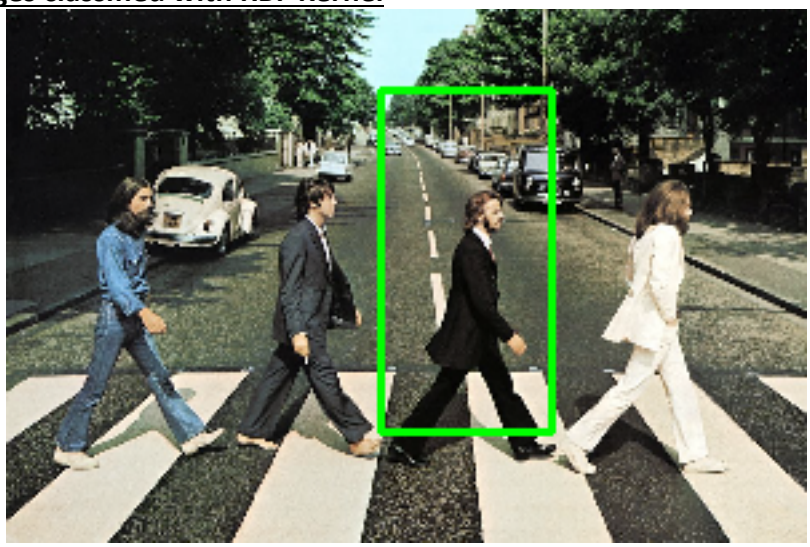


Football-field.png with Poly Kernel

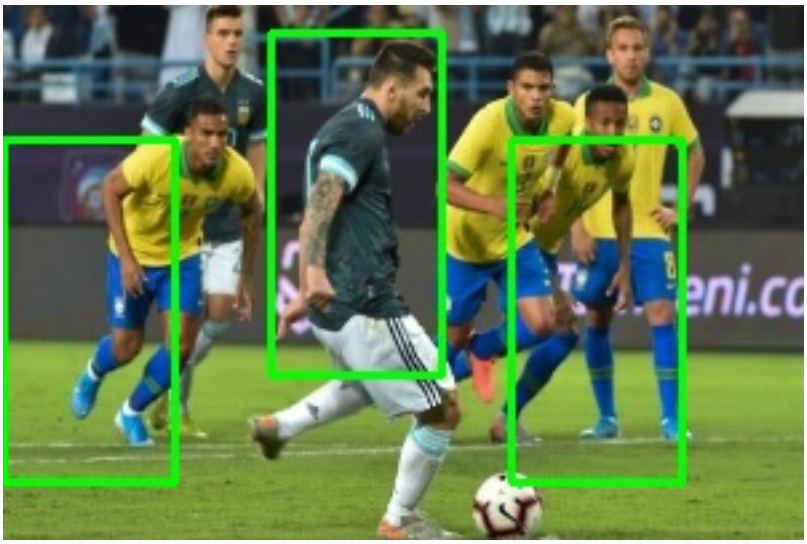


Person-in-the-woods.png with Poly Kernel

Results of images classified with RBF Kernel



Beetles.png with RBF Kernel



Football-field.png with RBF Kernel

Person-in-the-woods.png with RBF Kernel

Source Code of Training Script

```
# Importing the necessary modules:

from skimage.feature import hog
from skimage.transform import pyramid_gaussian
from skimage.io import imread
# from sklearn.externals import joblibn
import joblib
from sklearn.preprocessing import LabelEncoder
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report
from sklearn.neural_network import MLPClassifier
# from sklearn.cross_validation import train_test_split
from sklearn.model_selection import train_test_split
from skimage import color
from imutils.object_detection import non_max_suppression
import imutils
import numpy as np
import argparse
import cv2
import os
import glob
from PIL import Image # This will be used to read/modify images (can be
done via OpenCV too)
from numpy import *

# define parameters of HOG feature extraction
orientations = 9
pixels_per_cell = (8, 8)
cells_per_block = (2, 2)
threshold = .3

pos_im_path = r"./Training_Data/positive"
# define the same for negatives
neg_im_path= r"./Training_Data/negative"

# read the image files:
pos_im_listing = os.listdir(pos_im_path) # it will read all the files
in the positive image path (so all the required images)
neg_im_listing = os.listdir(neg_im_path)
num_pos_samples = size(pos_im_listing) # simply states the total no. of
images
num_neg_samples = size(neg_im_listing)
```

```
print(num_pos_samples) # prints the number value of the no.of samples
in positive dataset
print(num_neg_samples)
data= []
labels = []

# compute HOG features and label them:

for file in pos_im_listing: #this loop enables reading the files in the
pos_im_listing variable one by one
    img = Image.open(pos_im_path + '/' + file) # open the file
    #img = img.resize((64,128))
    gray = img.convert('L') # convert the image into single channel
i.e. RGB to grayscale
    # calculate HOG for positive features
    fd = hog(gray, orientations, pixels_per_cell, cells_per_block,
block_norm='L2', feature_vector=True)# fd= feature descriptor
    data.append(fd)
    labels.append(1)

# Same for the negative images
for file in neg_im_listing:
    # Compute HOG features and labels for negative images
    img = Image.open(neg_im_path + '/' + file)  # open the file
    # img = img.resize((64,128))
    gray = img.convert('L')  # convert the image into single channel
i.e. RGB to grayscale
    # calculate HOG for positive features
    fd = hog(gray, orientations, pixels_per_cell, cells_per_block,
block_norm='L2',
             feature_vector=True)  # fd= feature descriptor
    data.append(fd)
    labels.append(-1)



# Label Encoding - Conversin from string to integer
le = LabelEncoder()
labels = le.fit_transform(labels)

#%%
# Partitioning the data into training and testing splits, using 80%
# of the data for training and the remaining 20% for testing
print(" Constructing training/testing split...")
```

```
(trainData, testData, trainLabels, testLabels) =
train_test_split(np.array(data), labels, test_size=0.20,
random_state=42)
#%% Train the neural network
print(" Training...")

# Implement a SVM using the sklearn library
model = SVC(kernel = 'linear')
# Train the SVM model
model.fit(trainData,trainLabels)

#%% Evaluate the classifier

print(" Evaluating classifier on test data ...")

# check the model using the Test_images
predictions = model.predict(testData)

# Generate the report and the confusion matrix
print(classification_report(testLabels, predictions))


# Save the model:
#%% Save the Model
joblib.dump(model, 'model_linear_kernel.npy')
```

Source Code of Training Script

```
from skimage.feature import hog
from skimage.transform import pyramid_gaussian
from sklearn.svm import SVC
# from sklearn.externals import joblib
import joblib
from skimage import color
from imutils.object_detection import non_max_suppression
import imutils
import numpy as np
import cv2
import os
import glob

#Define HOG Parameters
orientations = 9
pixels_per_cell = (8, 8)
cells_per_block = (2, 2)
threshold = .3

# define the sliding window:
def sliding_window(image, stepSize, windowSize):# image is the input,
step size is the number of pixels needed to skip and windowSize is the
size of the display window
    # slide a window across the image
    for y in range(0, image.shape[0], stepSize):# this for loop defines
the sliding part and loops over the x and y coordinates
        for x in range(0, image.shape[1], stepSize):
            # yield the current window
            yield (x, y, image[y: y + windowSize[1], x:x +
windowSize[0]])

# Uncomment each line depending on the model to use
model = joblib.load('./model_linear_kernel.npy') # Path to saved model
created with linear kernel
# model = joblib.load('./model_poly_kernel.npy') # Path to saved model
created with poly kernel
# model = joblib.load('./model_rbf_kernel.npy') # Path to saved model
created with rbf kernel
# model = joblib.load('./model_sigmoid_kernel.npy') # Path to saved
model created with sigmoid kernel

# Test the trained classifier on an image below
```

```
scale = 0
detections = []

# Uncomment each line depending on which image to test this on
path = 'ps8b-test-dataset/beetles.png'
# path = 'ps8b-test-dataset/football_field.jpg'
# path = 'ps8b-test-dataset/person_in_the_woods.png'

img = cv2.imread(path)
image_name = path.split('/')
name = image_name[-1].split('.')
# you can image if the image is too big
img= cv2.resize(img,(300,200)) # can change the size to default by
commenting this code out our put in a random number

# defining the size of the sliding window (has to be the same as the
size of the image in the training data)
(winW, winH)= (64,128)
windowSize=(winW,winH)
downscale=1.5

# Apply sliding window: Do not change this code!
for resized in pyramid_gaussian(img, max_layer = 0, downscale=1.5):
    for (x,y,window) in sliding_window(resized, stepSize=10,
windowSize=(winW,winH)):
        if window.shape[0] != winH or window.shape[1] !=winW:
            continue
        window = color.rgb2gray(window)
        # Extract HOG features from the window captured, and predict
whether it is a person or not
        fds = hog(window, orientations = orientations, pixels_per_cell=
pixels_per_cell, cells_per_block= cells_per_block, block_norm='L2',
             feature_vector=True)
        #print(fds.shape)
        fds = fds.reshape(1,-1)
        pred = model.predict(fds)

        if pred == 1:
            #print('confirm')
            if model.decision_function(fds) > 0.6:  # set a threshold
value for the SVM prediction i.e. only firm the predictions above
probability of 0.6
                print("Detection:: Location -> ({}, {})".format(x, y))
```

```
                print("Scale ->  {} | Confidence Score {}
\n".format(scale,model.decision_function(fds)))

                detections.append((int(x * (downscale**scale)), int(y *
(downscale**scale)), model.decision_function(fds),

int(windowSize[0]*(downscale**scale)), # create a list of all the
predictions found

int(windowSize[1]*(downscale**scale))))
    scale+=1


clone = resized.copy()
for (x_tl, y_tl, _, w, h) in detections:
    cv2.rectangle(img, (x_tl, y_tl), (x_tl + w, y_tl + h), (0, 0, 255),
thickness = 2)
rects = np.array([[x, y, x + w, y + h] for (x, y, _, w, h) in
detections]) # do nms on the detected bounding boxes

sc = [score[0] for (x, y, score, w, h) in detections]
print("Detection confidence score: ", sc)
sc = np.array(sc)

# Non-maximal suppresion
pick = non_max_suppression(rects, probs = sc, overlapThresh = 0.3)

for (xA, yA, xB, yB) in pick:
    cv2.rectangle(img, (xA, yA), (xB, yB), (0,255,0), 2)

cv2.imshow("Detections after NMS", img)
cv2.waitKey(0) & 0xFF
cv2.destroyAllWindows()
cv2.imwrite(f'result_images/{name[0]}-output.{name[1]}',img)
```

System Specifications

Operating System: macOS Monterey Version 12.5.1
Hardware: MacBook Air 2017 (Intel Core i5)
Python: Conda environment utilizing Python 3.9.1
IDE: Visual Studio Code

Time taken: 2 hours