Kerem Şahin 24177
Ege Erdoğan 25331

1. INTRODUCTION

Object detection is one of the important problems of the Computer Vision. Object detection is about finding a specific object presence in image and its location in the image.

In this project, "Human (Pedestrian) Detection" will be implemented for image dataset. Pedestrian detection is an ongoing challenging problem as there are many different possibilities for image scenes such as various poses of humans, backgrounds, occlusions because of other objects and so on.

Object detection processes usually following 3 steps which are shown below:

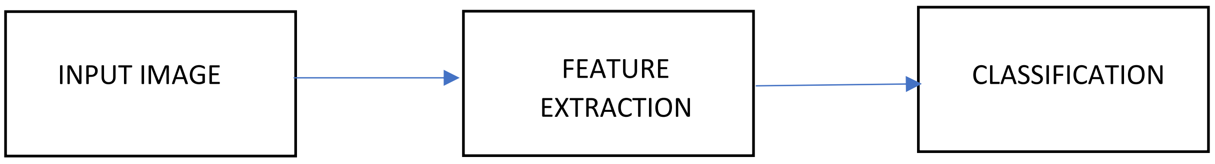


Figure 1. Object Detection Process Flow

In this project, 3 different **feature extraction** methods and **classification models** will be compared with each other for Pedestrian Detection problem. Feature extraction methods are Histograms of Oriented Gradients (HOG), Scale-Invariant Feature Transform (SIFT), and Speeded-up Robust Features (SURF) used in the feature extraction step. Classification models are Linear Support Vector Classifier (Linear SVC), polynomial degree 3 Support Vector Classifier (SVC) and Random Forest Classifier (RFC) used in the classification step.

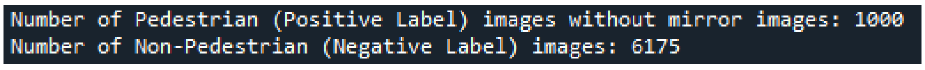
Throughout this report, each feature extraction method will be explained shortly. Then, their results will be presented for each classifier model. Then, results will be compared to each other and discussed in terms of accuracy, precision, recall and F1-score. Feature extractions' running times will be also compared with each other to see quickness of algorithms.

In addition to these, simple Convolutional Neural Network implementation will be also tried for Pedestrian Detection.

Kerem Şahin 24177
Ege Erdoğan 25331

2. DATASET AND EVALUATION METRICS EXPLANATION

Throughout the literature review, we have seen that INRIA dataset is the widely used for pedestrian detection problem by various studies [1,2]. However, we could not access to INRIA dataset for unknown reason. Therefore, we decided use CVC-01: Onboard Sequence Pedestrian Dataset. It consists of different size of png images with 1000 positive labelled images and 6175 negative labelled images. In addition to this 1000 positive pedestrian images, it has also mirrored version of these positive labelled ones.



```
Number of Pedestrian (Positive Label) images without mirror images: 1000
Number of Non-Pedestrian (Negative Label) images: 6175
```

Figure 2. Number of Positive and Negative Images

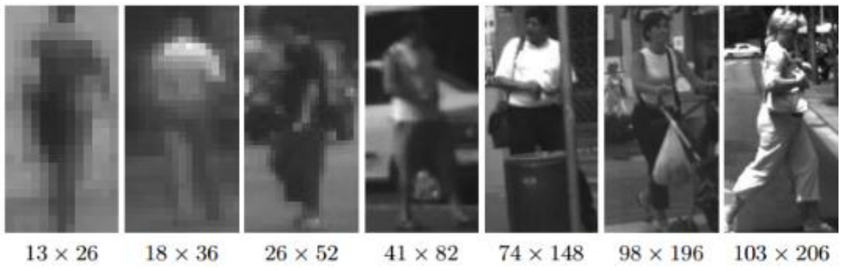
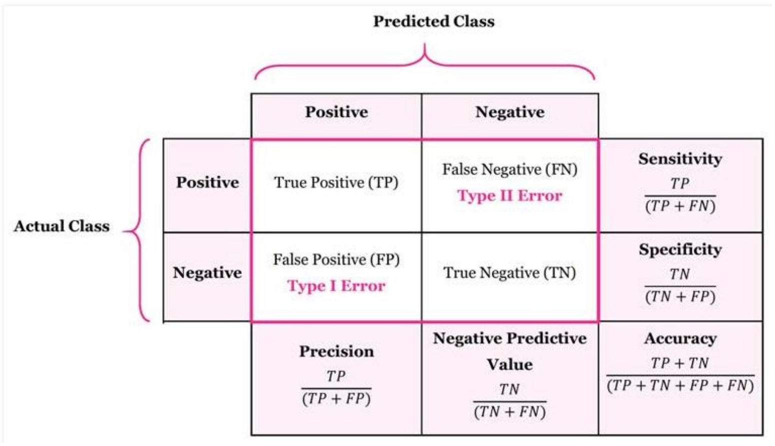


Figure 3. Example Images from Dataset

We evaluated classification results with the help of Confusion Matrix. Confusion Matrix can be seen as the summary of classification results as a tabulated manner.



		Predicted Class: Positive	Predicted Class: Negative	
Actual Class	Positive	True Positive (TP)	False Negative (FN) Type II Error	Sensitivity $\frac{TP}{(TP+FN)}$
	Negative	False Positive (FP) Type I Error	True Negative (TN)	Specificity $\frac{TN}{(TN+FP)}$
		Precision $\frac{TP}{(TP+FP)}$	Negative Predictive Value $\frac{TN}{(TN+FN)}$	Accuracy $\frac{TP+TN}{(TP+TN+FP+FN)}$

Figure 4. Confusion Matrix and Corresponding Metrics

We will consider Accuracy, Precision, Recall and F1-Score for results by utilizing this matrix.

Recall: It is the ratio of catched correct positives to all positives. For instance, if we have 50% recall rate, the model can detect 50 pedestrian among the 100 pedestrian images.

Precision: It is the ratio of catched correct positives to all positive predictions of the model. For instance, if we have 50% precision rate and the model predict 100 positive pedestrian included images, 50 of them is correctly classified. The rest of the 50 predictions are false alarms.

F1 Score: It is a special formula by combining Recall and Precision. It is calculated as below:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

Figure 5. F1-Score Formula

Kerem Şahin 24177
Ege Erdoğan 25331

3. HISTOGRAM OF ORIENTED GRADIENTS (HOG)

Firstly, we start by pre-processing the data. It is a significant step for any machine learning application. Thus, in order to extract the HOG features we need to make the width to height ratio of our images 1:2. We will make the size of our images 64x128 as it is stated in Dalal [2] to divide the image into 8x8 and 16x16 patches.

Secondly, we should calculate the gradients for each pixel in the image in x and y directions. Then, we will have two matrices which are storing the gradients, one for x direction and other for y direction. This step is actually similar to using a 3x3 Sobel Filter to detect edges as we did in the labs. After repeating the procedure for all pixels, we move on to next step.

Now that we have the gradients, we can calculate the magnitude and direction for each pixel.

These are calculated respectively by the following formulas:

$$\text{Total Gradient Magnitude} = \sqrt{[(G_x)^2+(G_y)^2]}$$

$$\Phi = \text{atan}(Gy\,/\,Gx)$$

Figure 6. Magnitude and Angle Calculation

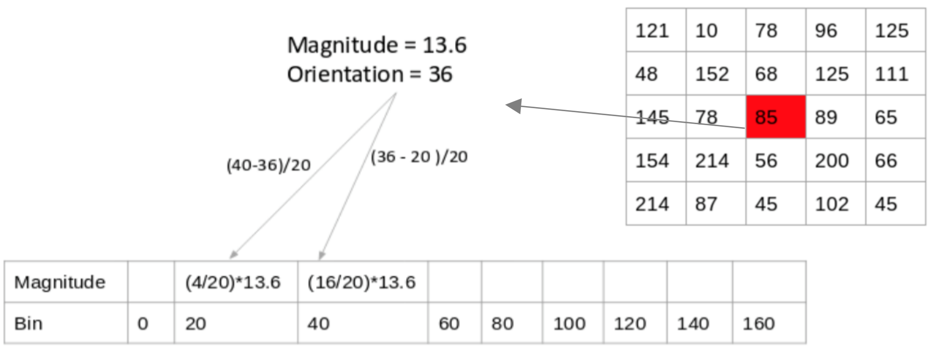
Hence, we have a gradient and an orientation for every pixel value on the image. And now we will generate histograms using the gradients and orientations.

There are different methods to generate the histogram from the magnitude of the gradients and orientation, but we will use the one which will be explained below:

For every pixel, check the orientation and store the contribution of gradient magnitude of the pixel to the bins on either side of the orientation in the matrix which has 9x1 form.

The tricky part is that contribution should be to the bin value which is closer to orientation.

Kerem Şahin 24177
Ege Erdoğan 25331



121	10	78	96	125
48	152	68	125	111
145	78	🟥 85	89	65
154	214	56	200	66
214	87	45	102	45

Magnitude		(4/20)*13.6	(16/20)*13.6						
Bin	0	20	40	60	80	100	120	140	160

Figure 7. Calculating Histograms of HOG

Now that we know how to generate a histogram from the image for extracting HOG features, but we will not generate the histograms for the whole image, instead we will divide the image to 8x8 cells and compute histogram of oriented gradients for every cell.

After generating HOG for all 8x8 cell, we will normalize the gradients. This step is to reduce the sensitivity of the gradients to effect of lighting variation. We will combine four 8x8 cell and create a 16x16 block. Remember we also had 9x1 matrix for every 8x8 cell so from 4x9 we can reshape our matrix as 36x1. Then, we also normalize this matrix.

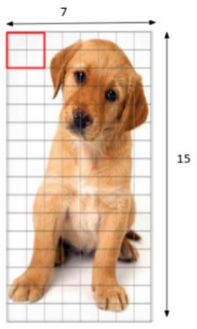


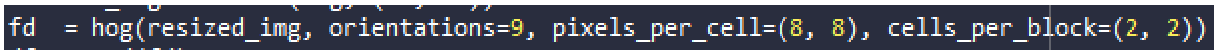
Figure 8. Combining four 8x8 cells to have 16x16 block

Lastly, to get the features for the final image we will combine features from constructed 16x16 blocks. For a 64x128 image we have 105 (7x15) blocks of 16x16 if we move the 16x16 kernel one step at a time. Hence calculating HOG matrix for 105 times and each resulting a vector of 36x1 we will finally have 105x36x1 = 3780 features. At the end, we have HOG features for a single image.

Kerem Şahin 24177
Ege Erdoğan 25331

3.1 HOG IMPLEMENTATION

We used scikit-image Python image processing library to implement HOG feature extraction method. Following one line, doing all the required steps which are explained above:



```
fd  = hog(resized_img, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2))
```

Figure 9. HOG Extraction

We resized our images as (64, 128) as it is used in Dalal [2].

We can do showcase for one example from our dataset:

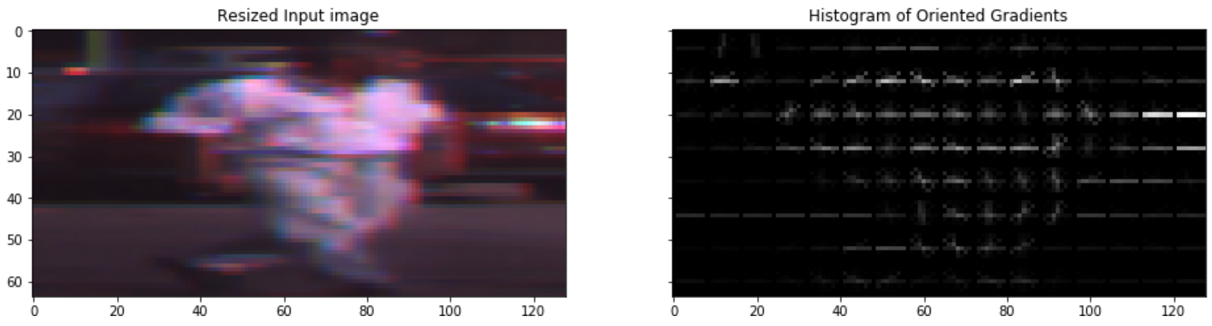
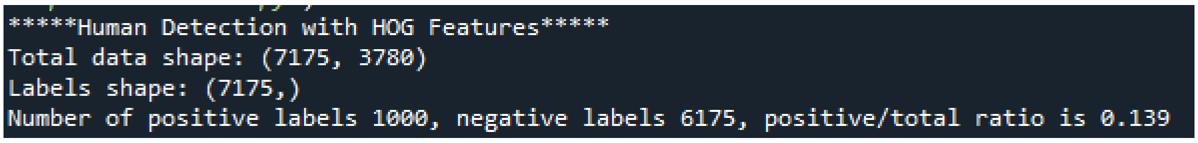


Figure 10. HOG Example for an Image

3.2 HOG RESULTS



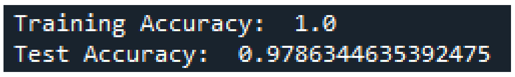
```
*****Human Detection with HOG Features*****
Total data shape: (7175, 3780)
Labels shape: (7175,)
Number of positive labels 1000, negative labels 6175, positive/total ratio is 0.139
```

Figure 11. Data after HOG Extraction

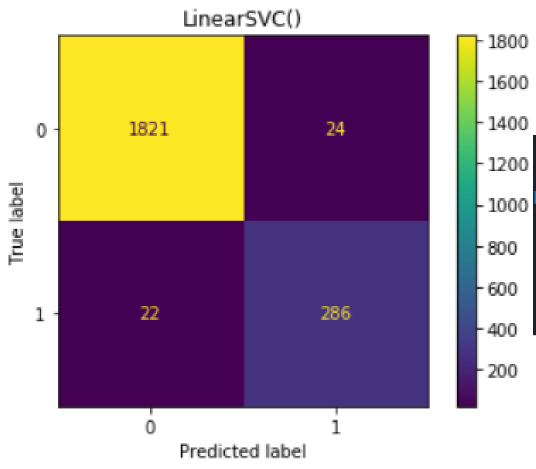
We realized that mirrored images do not increase any evaluation metrics. Therefore, we decided to not use mirrored images in our implementation.

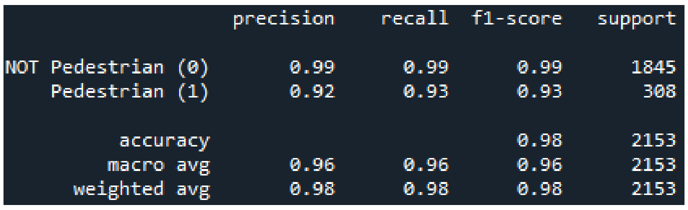
Kerem Şahin 24177
Ege Erdoğan 25331

HOG + Linear SVC:



```
Training Accuracy:  1.0
Test Accuracy:  0.9786344635392475
```

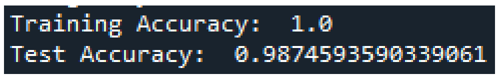




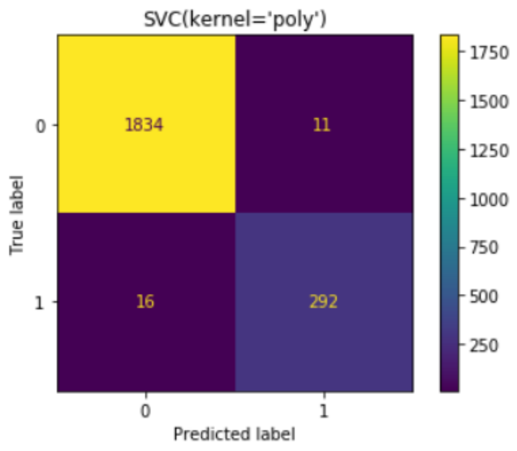
```
                  precision    recall  f1-score   support

NOT Pedestrian (0)      0.99      0.99      0.99      1845
    Pedestrian (1)      0.92      0.93      0.93       308

          accuracy                          0.98      2153
         macro avg      0.96      0.96      0.96      2153
      weighted avg      0.98      0.98      0.98      2153
```

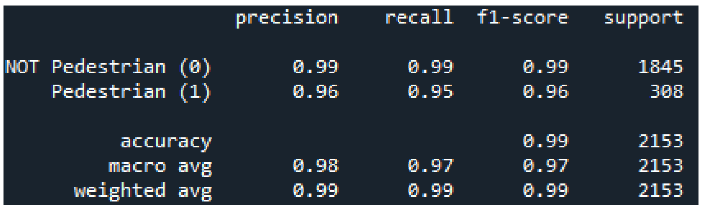
Figure 12. Results of HOG + Linear SVC

HOG + Polynomial SVC:



```
Training Accuracy:  1.0
Test Accuracy:  0.9874593590339061
```



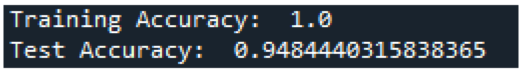


```
                  precision    recall  f1-score   support

NOT Pedestrian (0)      0.99      0.99      0.99      1845
    Pedestrian (1)      0.96      0.95      0.96       308

          accuracy                          0.99      2153
         macro avg      0.98      0.97      0.97      2153
      weighted avg      0.99      0.99      0.99      2153
```

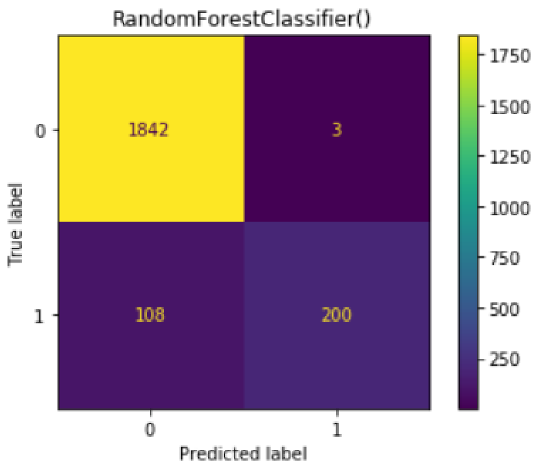
Figure 13. Results of HOG + Polynomial SVC

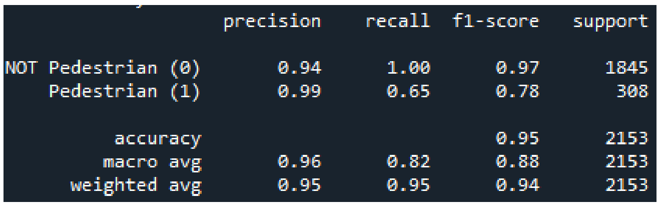
Kerem Şahin 24177
Ege Erdoğan 25331

HOG + Random Forest Classifier:



```
Training Accuracy:  1.0
Test Accuracy:  0.9484440315838365
```





```
                      precision    recall  f1-score   support

NOT Pedestrian (0)        0.94      1.00      0.97      1845
    Pedestrian (1)        0.99      0.65      0.78       308

          accuracy                            0.95      2153
         macro avg        0.96      0.82      0.88      2153
      weighted avg        0.95      0.95      0.94      2153
```

Figure 14. Results of HOG + Random Forest Classifier

3.3 HOG DISCUSSION

We can observe that HOG has really good performance for pedestrian detection problem. Surprisingly, Polynomial SVC has better results compared to Linear SVC, but it is small difference, and both are close 1. Thus, we can not say clearly that one is better than other one. However, we can say that both are better than Random Forest classifier. However, feature extraction part of the code has taken some time. This can be problematic in real world application as they need quick responses from algorithm.

Kerem Şahin 24177
Ege Erdoğan 25331

4. SCALE-INVARIANT FEATURE TRANSFORM (SIFT)

In the simplest manner, SIFT is developed to eliminate effect of scale in the image to find features. SIFT follows 4 steps to extract features from given image. First one is Scale-space extrema (peak) detection that creating a scale space of the image by convolving it with the Gaussian filter. This convolution also brings blurriness as we studied in the lecture. These blurred images are used to create Difference of Gaussians (DoG) to find keypoints in given image.

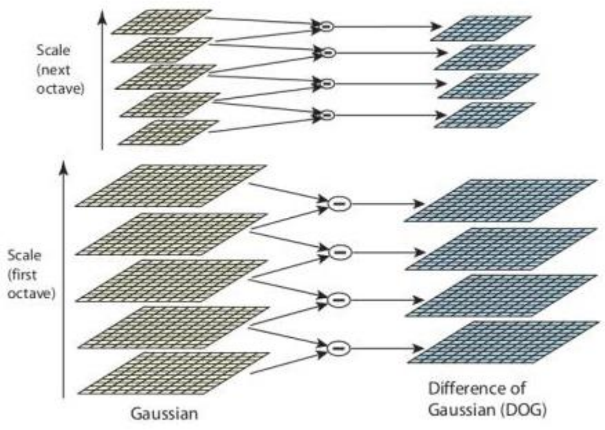


Figure 15. Obtaining DoGs from Different Scales

Then, local extrema pixels (keypoints) are searched in different scale and space in the image as it is shown below:

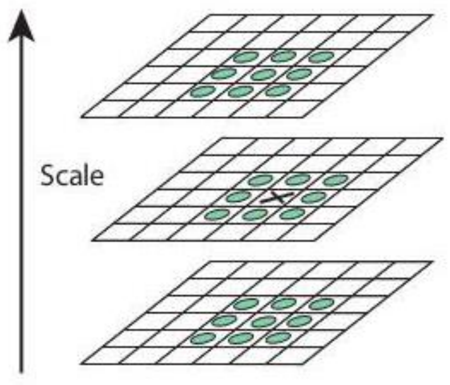
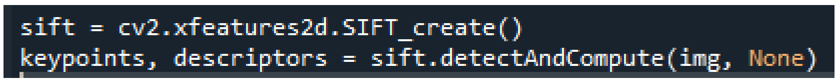


Figure 16. Search for Local Extrema Points

Second step is Keypoint Localization to filter found local extrema points for better accuracy. SIFT uses Taylor expansion to check intensity level of local extrema point with defined threshold and uses Hessian Matrix to eliminate edges. Third step is Orientation Assignment to overcome rotation problem in the images. In the last step, Keypoint Descriptor is created as a vector.

4.1 SIFT IMPLEMENTATION

It is easy to use SIFT with the help of OpenCV. However, SIFT is patented, so older community version of the OpenCV should be used for SIFT. Following two lines, doing all the required steps which are explained above:



```
sift = cv2.xfeatures2d.SIFT_create()
keypoints, descriptors = sift.detectAndCompute(img, None)
```

Figure 17. SIFT in OpenCV

We can do showcase for same example from our dataset:

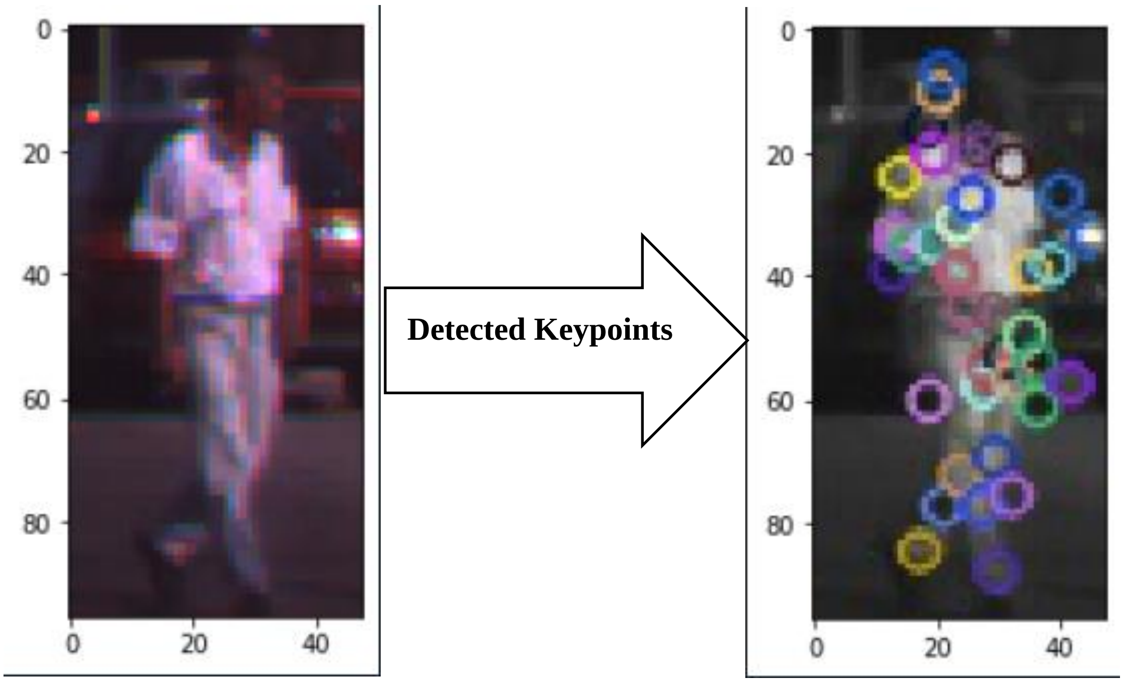
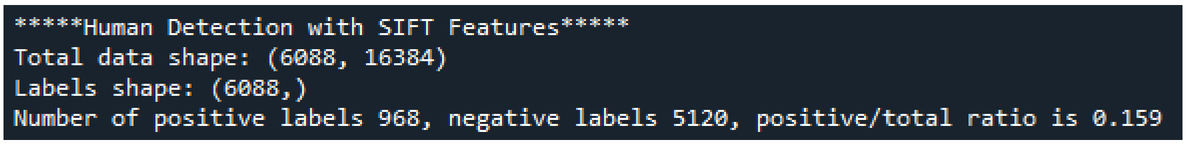


Figure 18. SIFT Keypoints Example for an Image

Each SIFT operation returns descriptor vector in size of (N, 128) where N corresponds to number of keypoints. Since each image may have different number of keypoints even 0 sometimes, we resized returned descriptor vectors as (128, 128) to train our classification models. This method may be the not correct way to do it, but we could not find a better way to handle this problem.

4.2 SIFT RESULTS

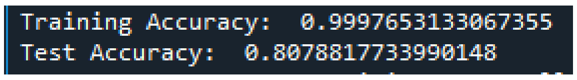


```
*****Human Detection with SIFT Features*****
Total data shape: (6088, 16384)
Labels shape: (6088,)
Number of positive labels 968, negative labels 5120, positive/total ratio is 0.159
```

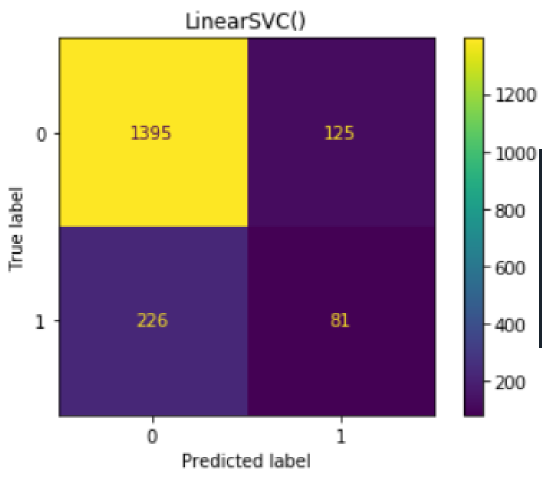
Figure 19. Data after SIFT Extraction

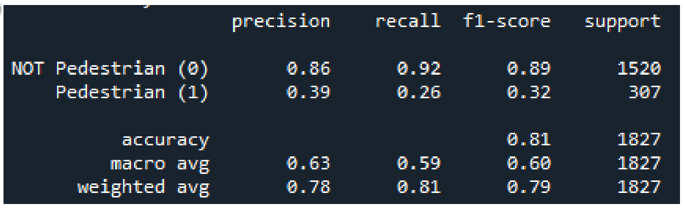
Since SIFT was unable to find any keypoints for some images, total number of images in the data is decreased.

SIFT + Linear SVC:



```
Training Accuracy:  0.9997653133067355
Test Accuracy:  0.8078817733990148
```



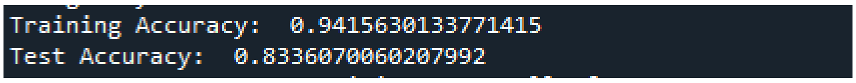


```
                    precision    recall  f1-score   support

NOT Pedestrian (0)       0.86      0.92      0.89      1520
    Pedestrian (1)       0.39      0.26      0.32       307

          accuracy                           0.81      1827
         macro avg       0.63      0.59      0.60      1827
      weighted avg       0.78      0.81      0.79      1827
```

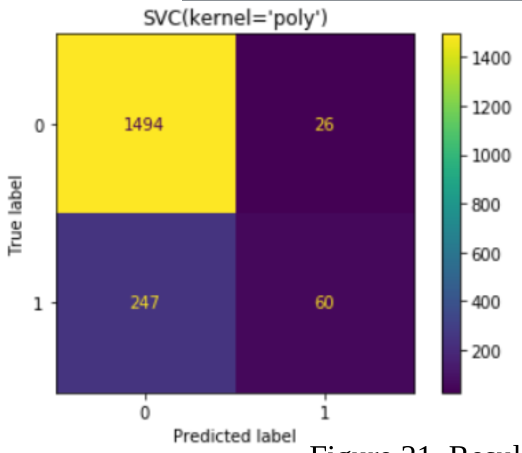
Figure 20. Results of SIFT + Linear SVC

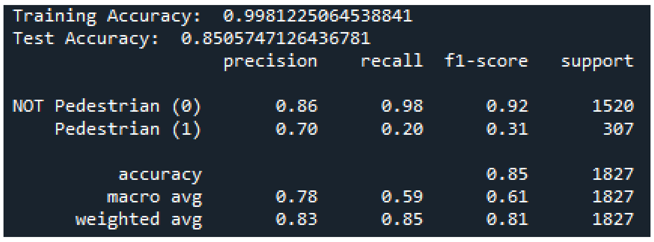
Kerem Şahin 24177
Ege Erdoğan 25331

SIFT + Polynomial SVC:



```
Training Accuracy:  0.9415630133771415
Test Accuracy:  0.8336070060207992
```

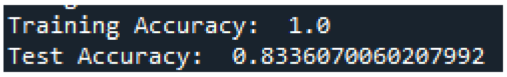




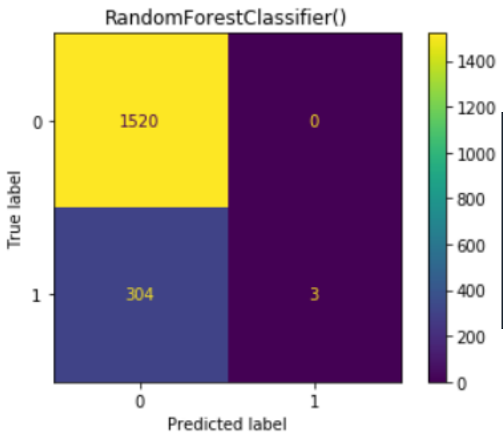
```
Training Accuracy:  0.9981225064538841
Test Accuracy:  0.8505747126436781
                     precision    recall  f1-score   support

NOT Pedestrian (0)       0.86      0.98      0.92      1520
    Pedestrian (1)       0.70      0.20      0.31       307

          accuracy                           0.85      1827
         macro avg       0.78      0.59      0.61      1827
      weighted avg       0.83      0.85      0.81      1827
```

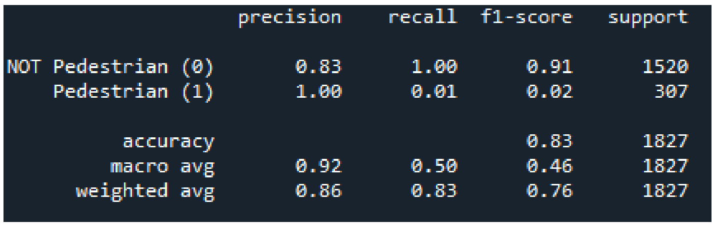
Figure 21. Results of SIFT + Polynomial SVC

SIFT + Random Forest Classifier:



```
Training Accuracy:  1.0
Test Accuracy:  0.8336070060207992
```





```
                     precision    recall  f1-score   support

NOT Pedestrian (0)       0.83      1.00      0.91      1520
    Pedestrian (1)       1.00      0.01      0.02       307

          accuracy                           0.83      1827
         macro avg       0.92      0.50      0.46      1827
      weighted avg       0.86      0.83      0.76      1827
```

Figure 22. Results of SIFT + Random Forest Classifier

4.3 SIFT DISCUSSION

We can observe that SIFT + RFC and SIFT + Polynomial SVC **do not** learn the data. They simply predict false for most of input images. However, SIFT + Linear SVC was able to learn to some extent our data. In total, we have bad performance with SIFT. The reason may be our resizing method for descriptor vector which might affect the learning phase of the model.

5. SPEEDED-UP ROBUST FEATURES (SURF)

In the simplest manner, SURF is faster version of SIFT. One of the biggest differences of SURF compared to SIFT is that it uses Box Filter rather than using Difference of Gaussian to approximate Laplacian of Gaussian. Since convolution with Box Filter is easier thanks to integral images where each pixel is the summation of the pixels above and to the left of it. Another important difference is that SURF uses determinant of the Hessian Matrix for both location and scale of the extrema pixel point. These processes can be seen in the below figure:

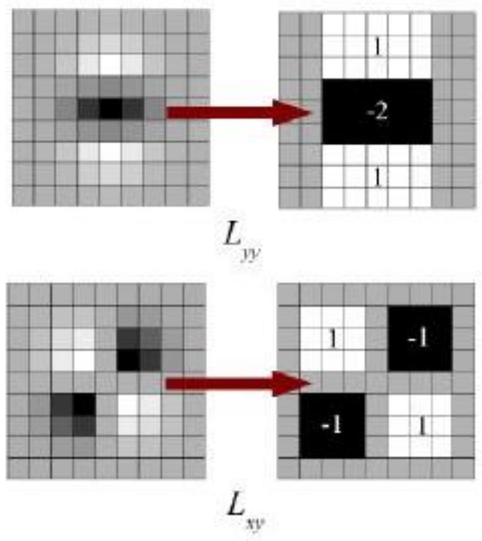


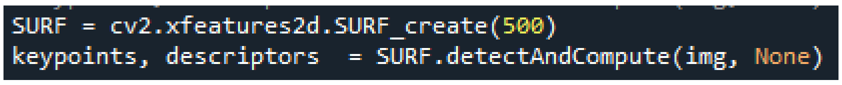
Figure 23. SURF Box Filter Approximation

Then, SURF uses wavelet responses by using integral images again which is fast and easy. After all these operations, SURF extracts descriptor feature vector without adding too much computational complexities.

Kerem Şahin 24177
Ege Erdoğan 25331

5.1 SURF IMPLEMENTATION

It is easy to use SURF with the help of OpenCV. However, SURF is also patented like SIFT, so older community version of the OpenCV should be used for SURF. Following two lines, doing all the required steps which are explained above:



```
SURF = cv2.xfeatures2d.SURF_create(500)
keypoints, descriptors  = SURF.detectAndCompute(img, None)
```

Figure 24. SURF in OpenCV

We can work on same example from SIFT part:

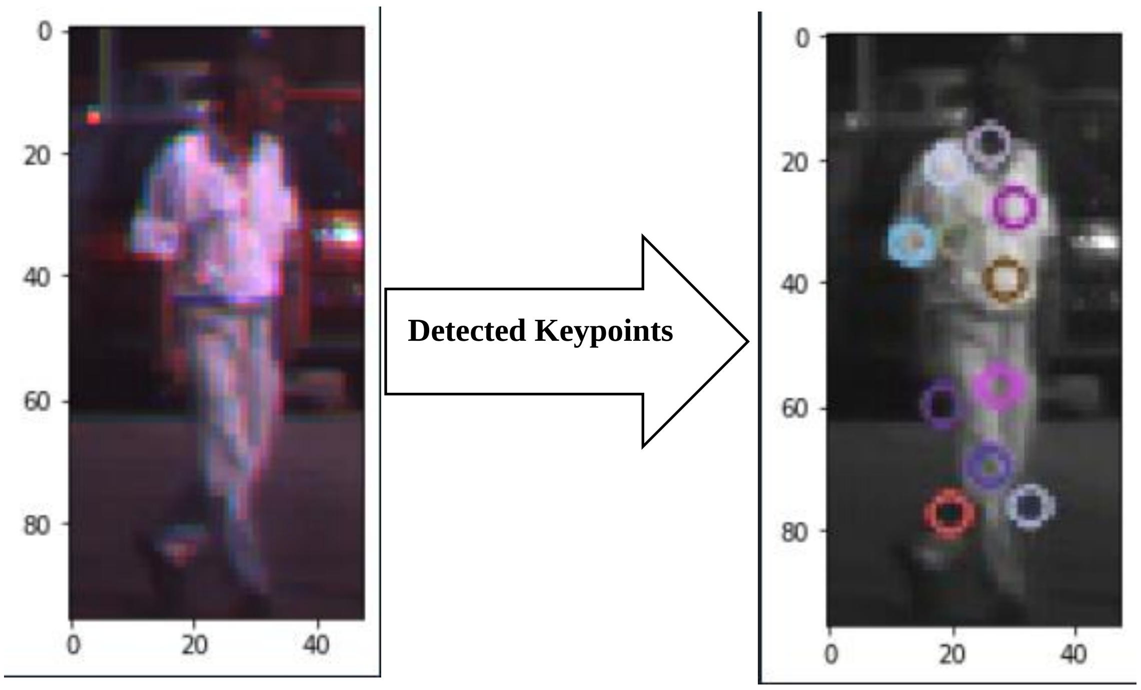
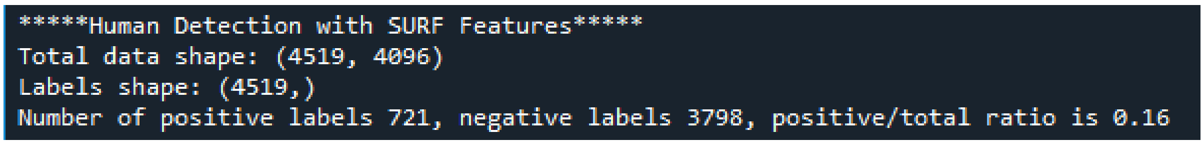


Figure 25. SURF Keypoints Example for an Image

Each SURF operation returns descriptor vector in size of (N, 64) where N corresponds to number of keypoints. Since each image may have different number of keypoints even 0 sometimes, we resized returned descriptor vectors as (64, 64) to train our classification models as we did in the SIFT part. This method may be the not correct way to do it, but we could not find a better way to handle this problem.

Kerem Şahin 24177
Ege Erdoğan 25331

5.2 SURF RESULTS

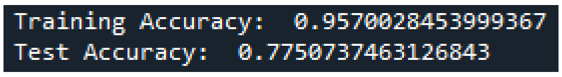


```
*****Human Detection with SURF Features*****
Total data shape: (4519, 4096)
Labels shape: (4519,)
Number of positive labels 721, negative labels 3798, positive/total ratio is 0.16
```

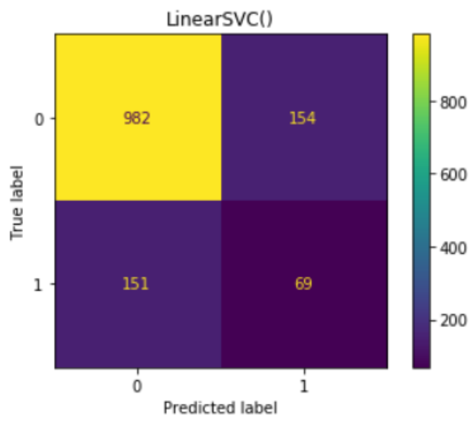
Figure 26. Data after SURF Extraction

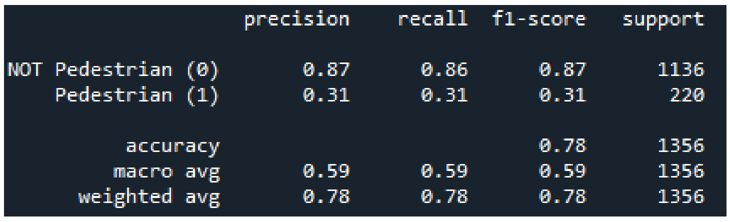
Since SURF was unable to find any keypoints for some images, total number of images in the data is decreased.

SURF + Linear SVC:



```
Training Accuracy:  0.9570028453999367
Test Accuracy:  0.7750737463126843
```

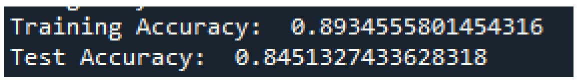




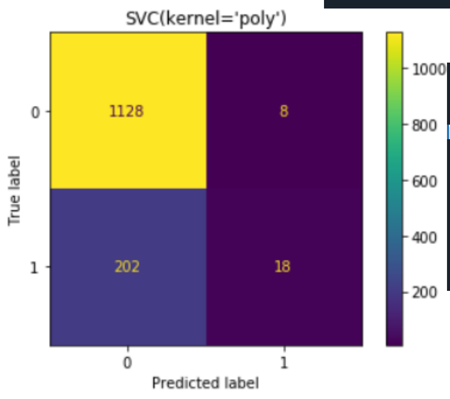
```
                    precision    recall  f1-score   support

NOT Pedestrian (0)       0.87      0.86      0.87      1136
    Pedestrian (1)       0.31      0.31      0.31       220

          accuracy                           0.78      1356
         macro avg       0.59      0.59      0.59      1356
      weighted avg       0.78      0.78      0.78      1356
```

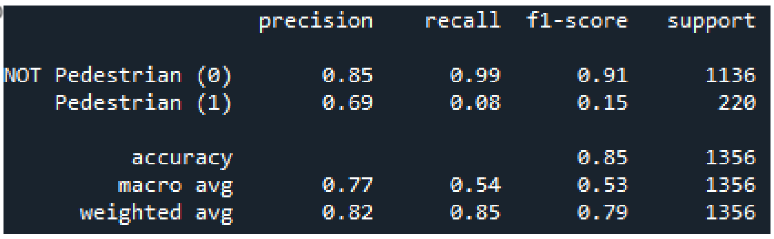
Figure 27. Results of SURF + Linear SVC

SURF + Polynomial SVC:



```
Training Accuracy:  0.8934555801454316
Test Accuracy:  0.8451327433628318
```

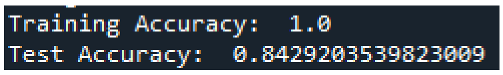




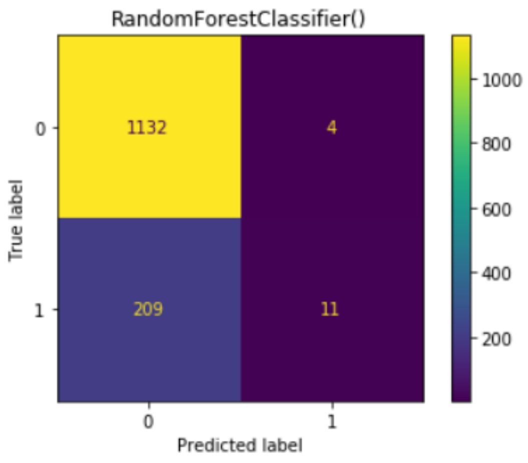
```
                    precision    recall  f1-score   support

NOT Pedestrian (0)       0.85      0.99      0.91      1136
    Pedestrian (1)       0.69      0.08      0.15       220

          accuracy                           0.85      1356
         macro avg       0.77      0.54      0.53      1356
      weighted avg       0.82      0.85      0.79      1356
```

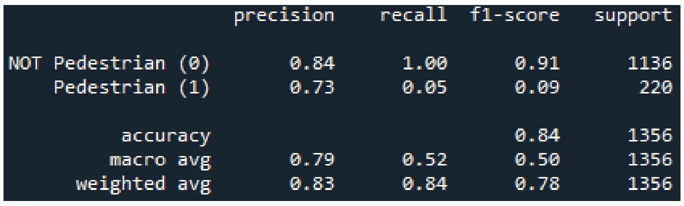
Figure 28. Results of SURF + Polynomial SVC

SURF + Random Forest Classifier:



```
Training Accuracy:  1.0
Test Accuracy:  0.8429203539823009
```





```
                    precision    recall  f1-score   support

NOT Pedestrian (0)       0.84      1.00      0.91      1136
    Pedestrian (1)       0.73      0.05      0.09       220

          accuracy                           0.84      1356
         macro avg       0.79      0.52      0.50      1356
      weighted avg       0.83      0.84      0.78      1356
```

Figure 29. Results of SURF + Random Forest Classifier

5.3 SURF DISCUSSION

Similar results with SIFT as we can observe that SURF + RFC and SURF + Polynomial SVC **do not** learn the data. They simply predict false for most of the input images. However, SURF + Linear SVC was able to learn to some extent our data. In total, we have bad performance with SURF. The reason may be our resizing method for descriptor vector which might affect the learning phase of the model as it is in SIFT case. On the other hand, we can say that SURF was the fastest extraction algorithm as we will prove this in the next section.

6. COMPARISON OF FEATURE EXTRACTION METHODS

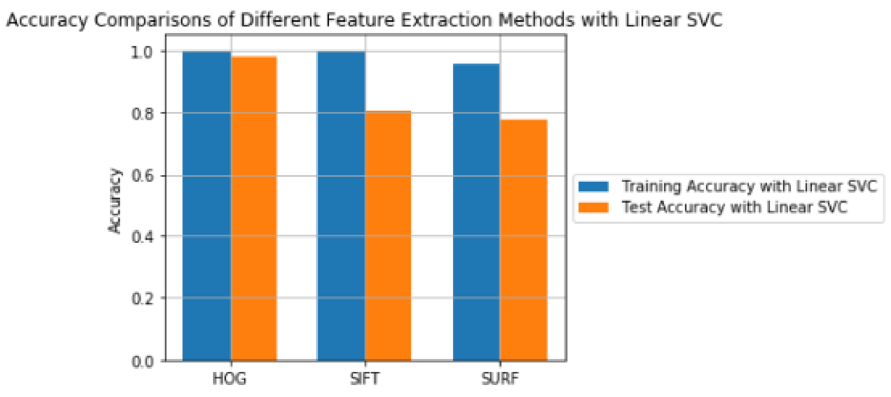


Figure 30. Train & Test Accuracies with Linear SVC

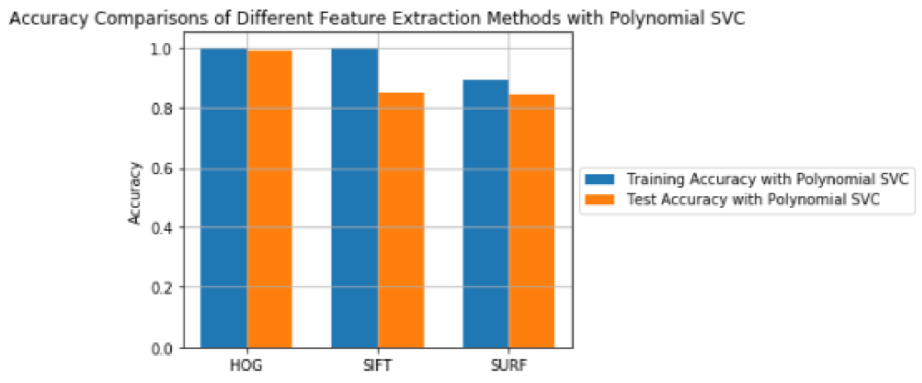


Figure 31. Train & Test Accuracies with Polynomial SVC

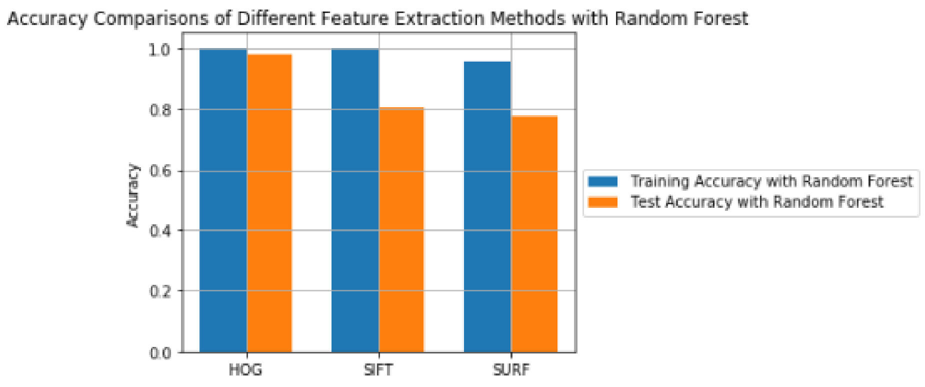
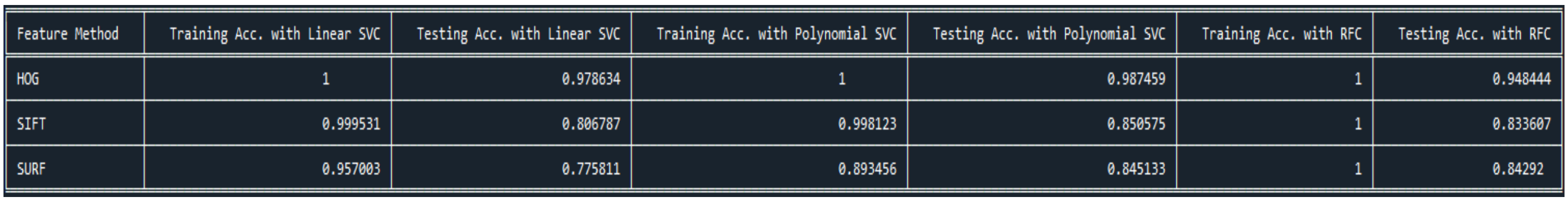


Figure 32. Train & Test Accuracies with Polynomial SVC

We can observe from Figure 30, 31 and 32 that HOG feature extraction method is better compared to both SIFT and SURF feature extraction methods. These results also can be seen below as a whole tabulated version:



Feature Method	Training Acc. with Linear SVC	Testing Acc. with Linear SVC	Training Acc. with Polynomial SVC	Testing Acc. with Polynomial SVC	Training Acc. with RFC	Testing Acc. with RFC
HOG	1	0.978634	1	0.987459	1	0.948444
SIFT	0.999531	0.806787	0.998123	0.850575	1	0.833607
SURF	0.957003	0.775811	0.893456	0.845133	1	0.84292

Figure 33. All Results of Feature Extraction Methods

Another important property of these algorithms is their quickness / running times as real time applications require low expected running time with lower complex algorithms. We also investigated needed total time for each algorithm separately for same dataset. This comparison can be seen in the below figure:

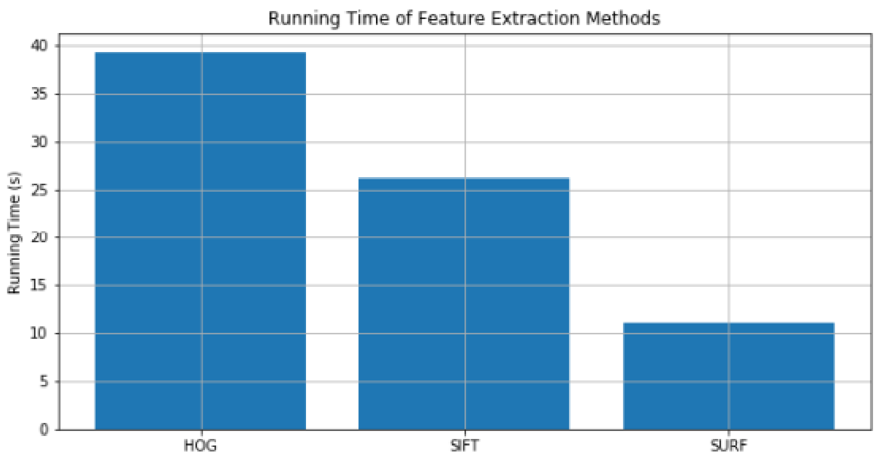


Figure 34. Running Times of Extraction Methods

We can observe that even HOG has the best performance for our problem, the required time for extracting features from input image is higher than SIFT and SURF. Thus, this is one of the trade-offs between accuracy and running time. Another observation is that SURF is the fastest as it is expected. It proves that SURF is the faster version of SIFT.

Kerem Şahin 24177
Ege Erdoğan 25331

7. CONVOLUTINOAL NEURAL NETWORK (CNN)

We also tried to implement simple CNN model to test our dataset with neural network architecture. CNN can be summarized with the below flow diagram:

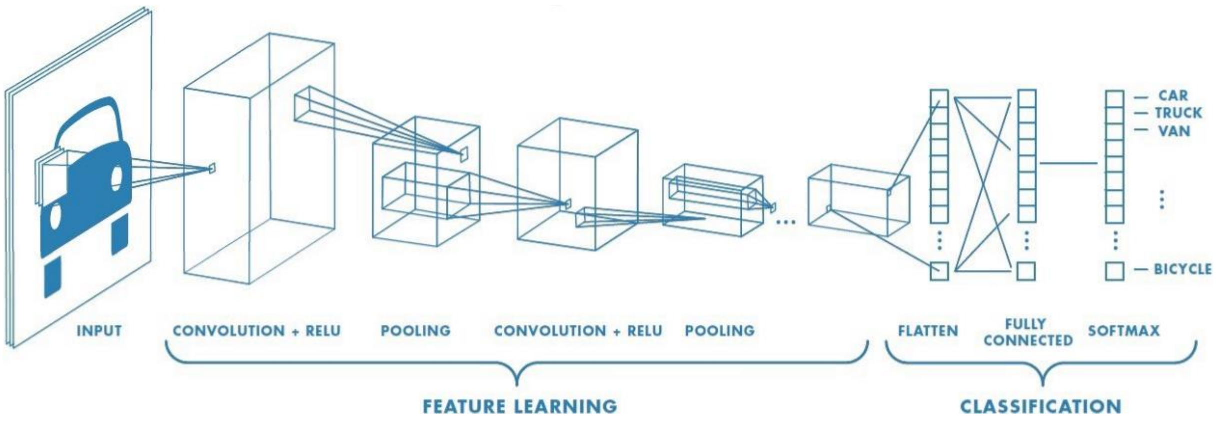


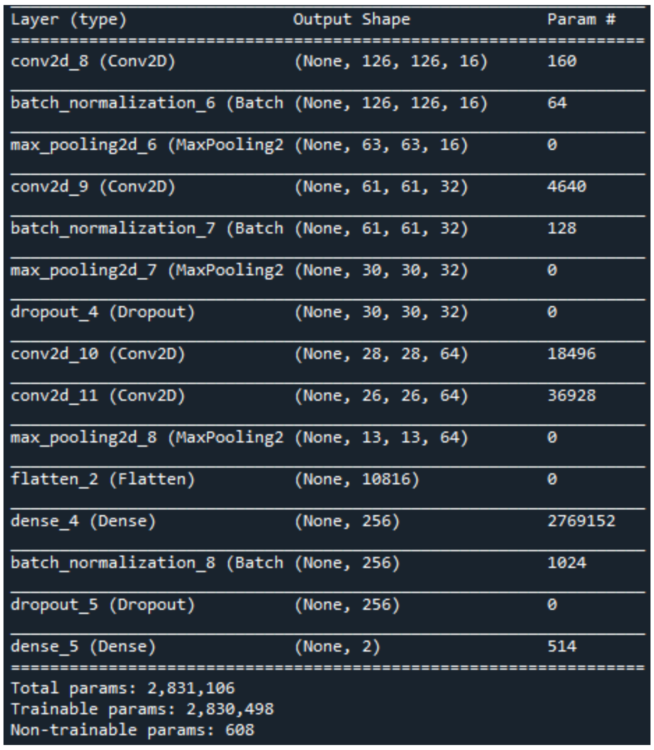
Figure 35. CNN Architecture

Each layer has own unique responsibility in the system, we will not explain them in detail, but quick summary for important parts as following:

Feature Learning Part: We sequentially convolve our input image matrix with some filter (kernel) with various sizes and using activation function as RELU. Then, we perform MaxPooling to reduce size of the feature maps. Each filter can be seen as different detectors to detect important parts of the image such as edges, corners and so on.

Classification Part: After learning features of the image, we perform usual forward neural network to classify input image. Softmax layer can be seen as the last output layer which will give the probability for each class. Output is the higher probability class.

Additionally, we have dropout which deactivates some of the neurons or parts with some probability in the system to reduce overfitting.



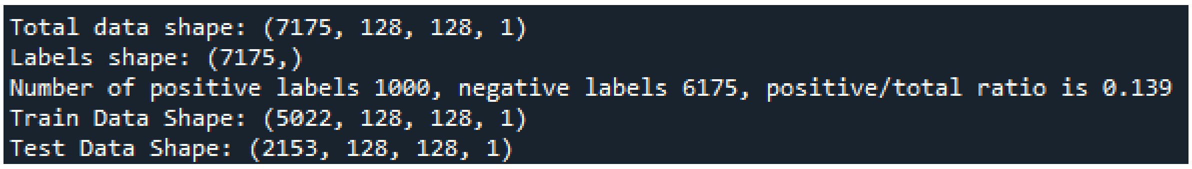
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 126, 126, 16)      160
_________________________________________________________________
batch_normalization_6 (Batch (None, 126, 126, 16)      64
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 63, 63, 16)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 61, 61, 32)        4640
_________________________________________________________________
batch_normalization_7 (Batch (None, 61, 61, 32)        128
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 30, 30, 32)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 28, 28, 64)        18496
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 26, 26, 64)        36928
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 10816)             0
_________________________________________________________________
dense_4 (Dense)              (None, 256)               2769152
_________________________________________________________________
batch_normalization_8 (Batch (None, 256)               1024
_________________________________________________________________
dropout_5 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_5 (Dense)              (None, 2)                 514
=================================================================
Total params: 2,831,106
Trainable params: 2,830,498
Non-trainable params: 608
```

Figure 36. Our CNN Architecture

7.1 CNN IMPLEMENTATION

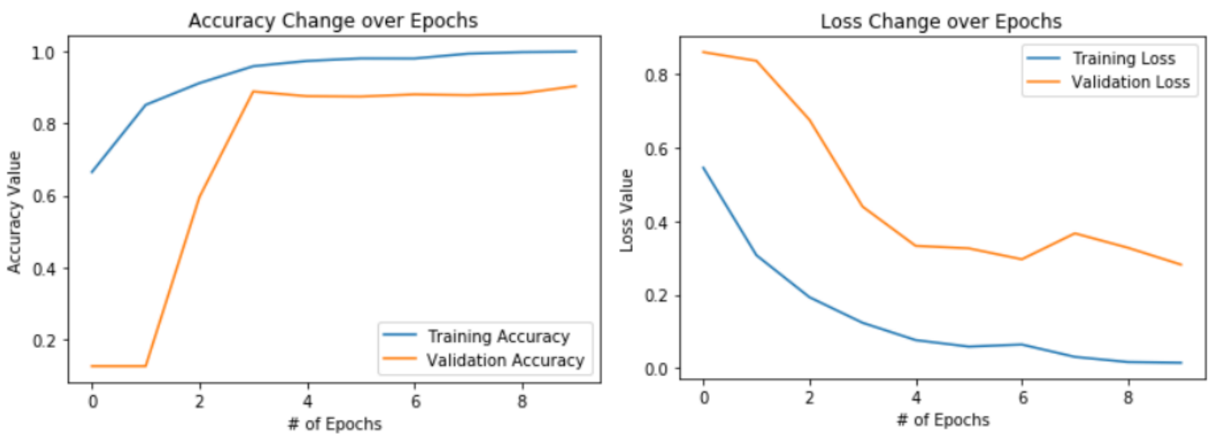
We used Keras to implement CNN approach to our dataset. Each input image converted to size of (128, 128) in a gray scale. Since we have imbalanced dataset, we calculated class weights and used them while fitting the CNN model.

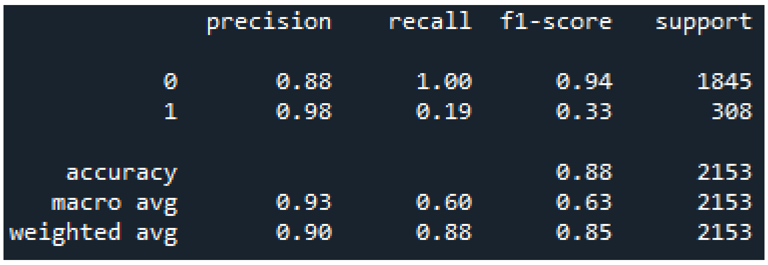
7.2 CNN RESULTS



```
Total data shape: (7175, 128, 128, 1)
Labels shape: (7175,)
Number of positive labels 1000, negative labels 6175, positive/total ratio is 0.139
Train Data Shape: (5022, 128, 128, 1)
Test Data Shape: (2153, 128, 128, 1)
```

Figure 37. Overall Data for CNN





```
              precision    recall  f1-score   support

           0       0.88      1.00      0.94      1845
           1       0.98      0.19      0.33       308

    accuracy                           0.88      2153
   macro avg       0.93      0.60      0.63      2153
weighted avg       0.90      0.88      0.85      2153
```

Figure 38. Results of CNN

7.3 SURF DISCUSSION

We can observe that our simple CNN approach is able to learn the problem in some extent, but not as much as the HOG feature extraction method. We also realized training a neural network is taking larger time compared to previous cases.

8. OVERALL DISCUSSION AND SUMMARY

We have compared different approaches for pedestrian detection problem. We realized that the best effective and useful approach is to use HOG feature extraction method with Linear Support Vector Classifier. We also observed that HOG feature extraction part consumes more time compared to other two extraction methods SURF and SIFT. We could not achieve desired performance with the CNN approach. However, this can be our fault as we do not have enough experience with CNNs.

9. REFERENCES

M Hiranmai, Niranjana Krupa B, H K Nagaraj, “Comparative Study of Various Feature Extraction Techniques for Pedestrian Detection”, Procedia Computer Science, Volume 154, 2019, Pages 622-628, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2019.06.098.

N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), 2005, pp. 886-893 vol. 1, doi: 10.1109/CVPR.2005.177.

https://docs.opencv.org/4.x/df/dd2/tutorial_py_surf_intro.html

https://docs.opencv.org/3.4/da/df5/tutorial_py_sift_intro.html

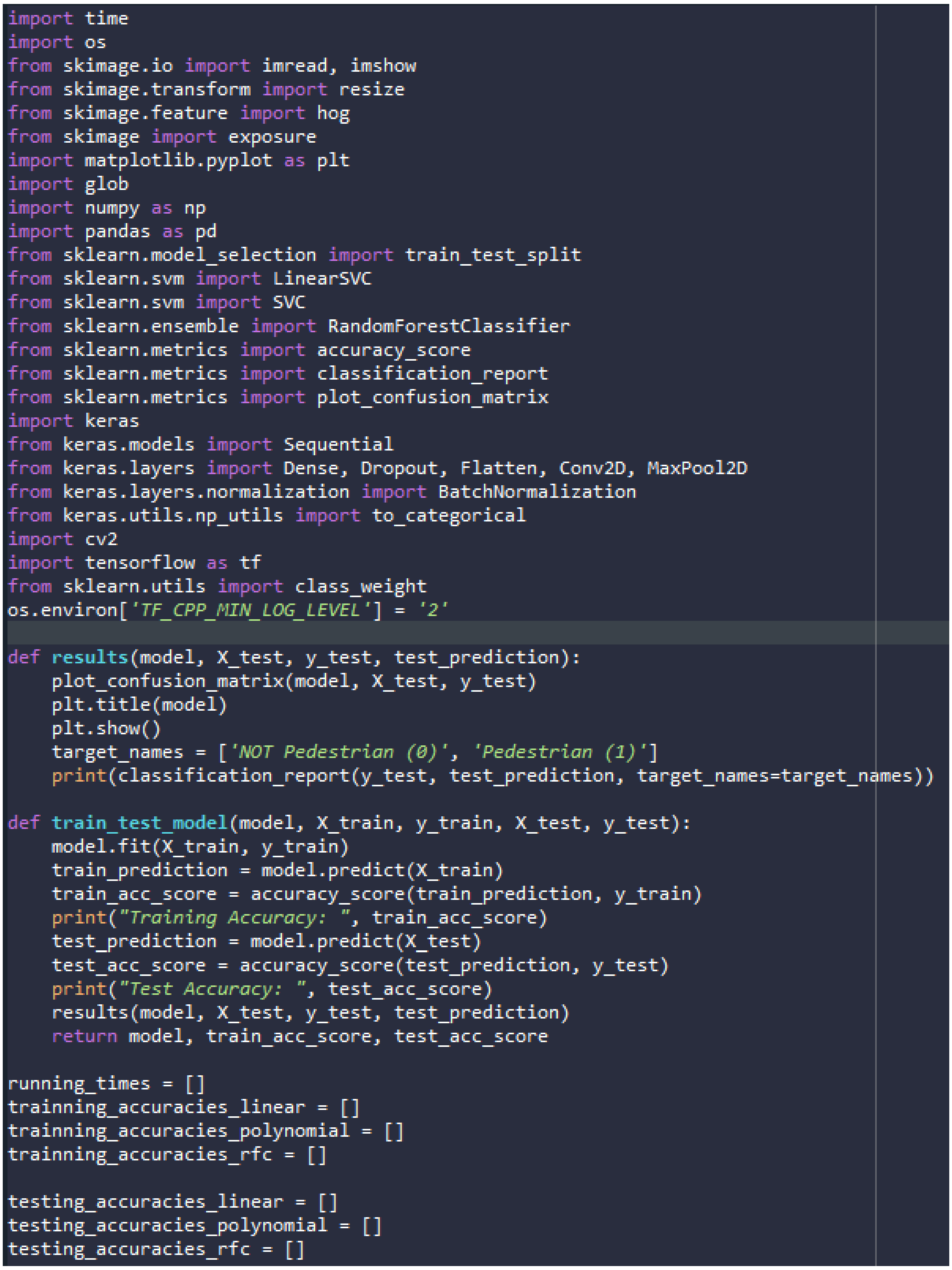
https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/

Dataset: http://adas.cvc.uab.es/elektra/enigma-portfolio/cvc-01-pedestrian-dataset/

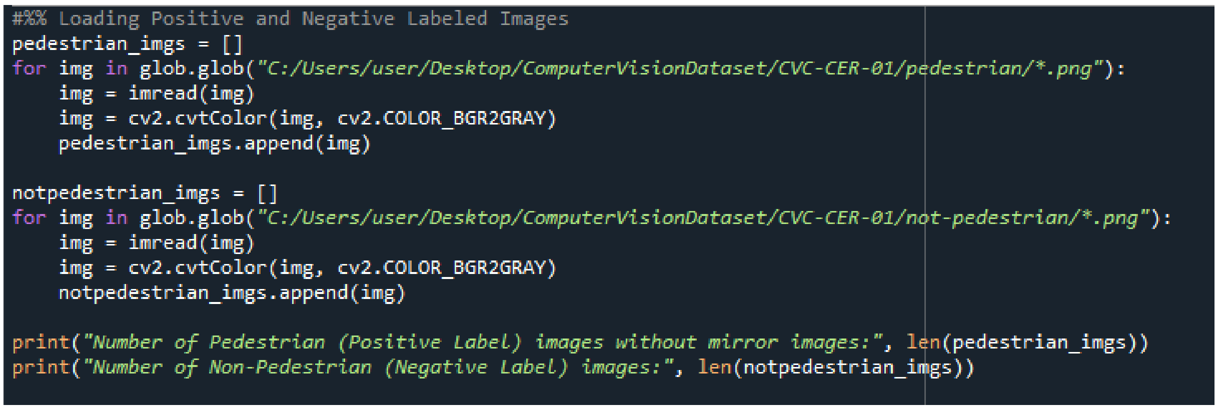
Kerem Şahin 24177
Ege Erdoğan 25331

10. APPENDIX CODES

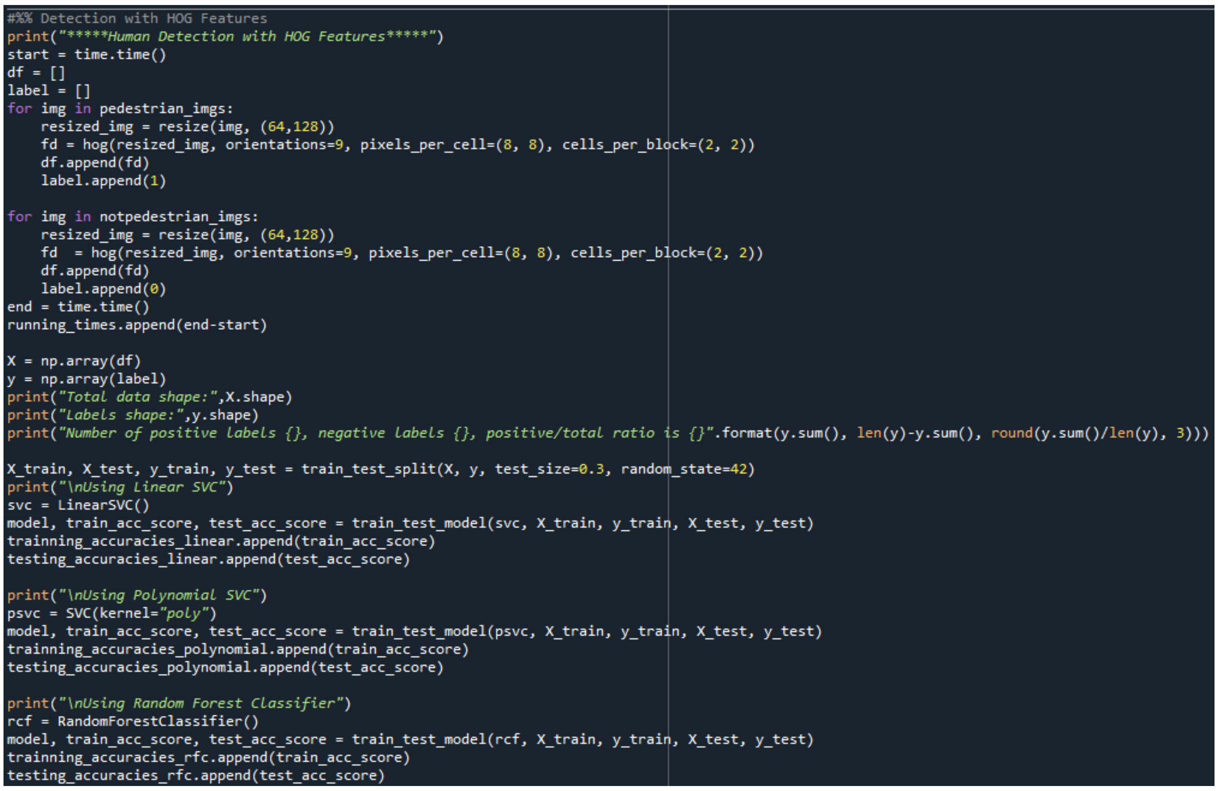


```python
import time
import os
from skimage.io import imread, imshow
from skimage.transform import resize
from skimage.feature import hog
from skimage import exposure
import matplotlib.pyplot as plt
import glob
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import plot_confusion_matrix
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPool2D
from keras.layers.normalization import BatchNormalization
from keras.utils.np_utils import to_categorical
import cv2
import tensorflow as tf
from sklearn.utils import class_weight
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

def results(model, X_test, y_test, test_prediction):
    plot_confusion_matrix(model, X_test, y_test)
    plt.title(model)
    plt.show()
    target_names = ['NOT Pedestrian (0)', 'Pedestrian (1)']
    print(classification_report(y_test, test_prediction, target_names=target_names))

def train_test_model(model, X_train, y_train, X_test, y_test):
    model.fit(X_train, y_train)
    train_prediction = model.predict(X_train)
    train_acc_score = accuracy_score(train_prediction, y_train)
    print("Training Accuracy: ", train_acc_score)
    test_prediction = model.predict(X_test)
    test_acc_score = accuracy_score(test_prediction, y_test)
    print("Test Accuracy: ", test_acc_score)
    results(model, X_test, y_test, test_prediction)
    return model, train_acc_score, test_acc_score

running_times = []
trainning_accuracies_linear = []
trainning_accuracies_polynomial = []
trainning_accuracies_rfc = []

testing_accuracies_linear = []
testing_accuracies_polynomial = []
testing_accuracies_rfc = []
```

Kerem Şahin 24177
Ege Erdoğan 25331

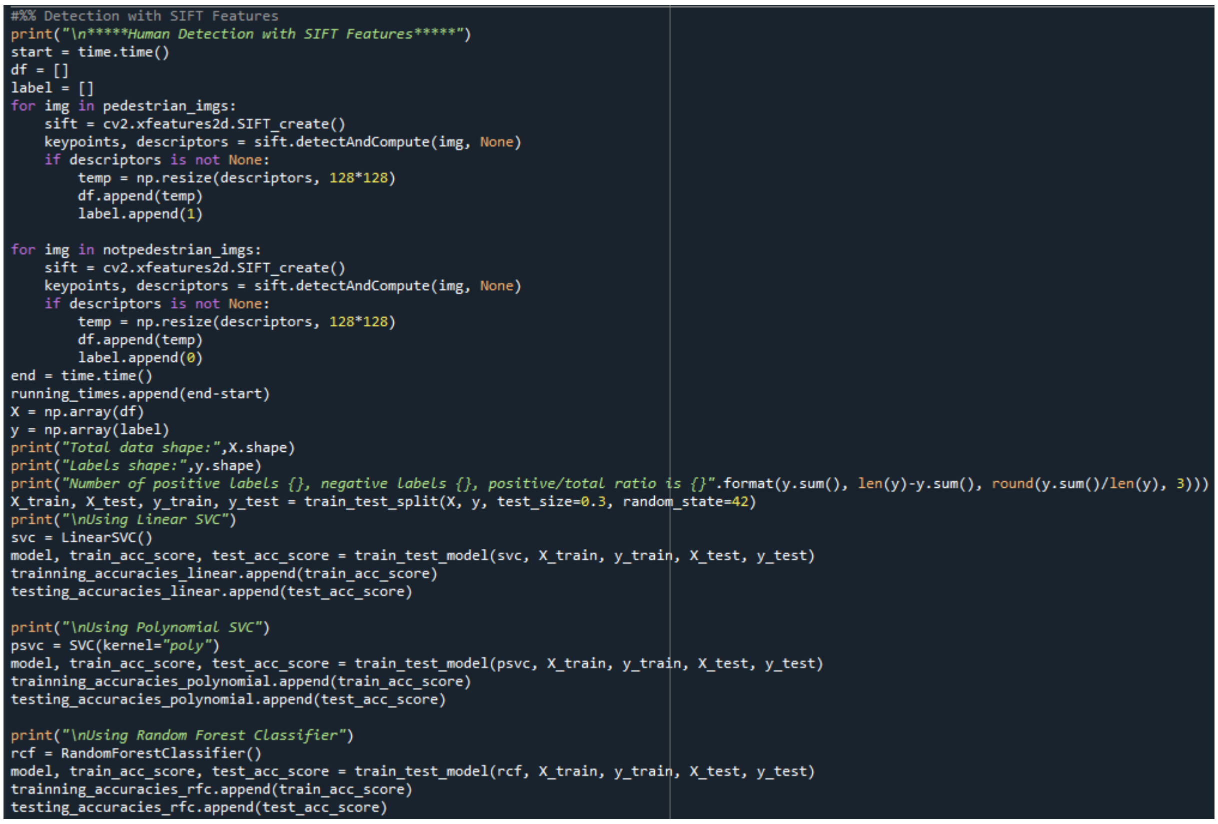


```python
#%% Loading Positive and Negative Labeled Images
pedestrian_imgs = []
for img in glob.glob("C:/Users/user/Desktop/ComputerVisionDataset/CVC-CER-01/pedestrian/*.png"):
    img = imread(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    pedestrian_imgs.append(img)

notpedestrian_imgs = []
for img in glob.glob("C:/Users/user/Desktop/ComputerVisionDataset/CVC-CER-01/not-pedestrian/*.png"):
    img = imread(img)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    notpedestrian_imgs.append(img)

print("Number of Pedestrian (Positive Label) images without mirror images:", len(pedestrian_imgs))
print("Number of Non-Pedestrian (Negative Label) images:", len(notpedestrian_imgs))
```

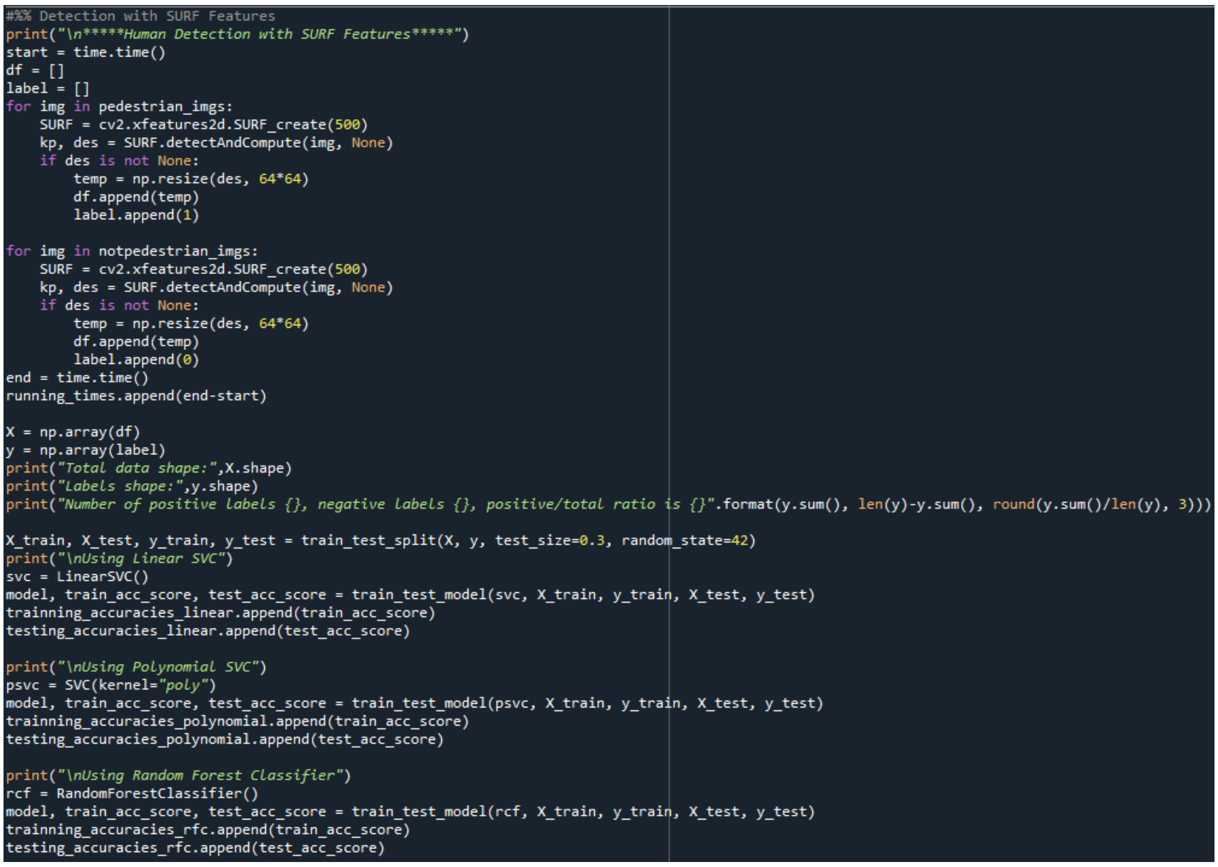


```python
#%% Detection with HOG Features
print("*****Human Detection with HOG Features*****")
start = time.time()
df = []
label = []
for img in pedestrian_imgs:
    resized_img = resize(img, (64,128))
    fd = hog(resized_img, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2))
    df.append(fd)
    label.append(1)

for img in notpedestrian_imgs:
    resized_img = resize(img, (64,128))
    fd  = hog(resized_img, orientations=9, pixels_per_cell=(8, 8), cells_per_block=(2, 2))
    df.append(fd)
    label.append(0)
end = time.time()
running_times.append(end-start)

X = np.array(df)
y = np.array(label)
print("Total data shape:",X.shape)
print("Labels shape:",y.shape)
print("Number of positive labels {}, negative labels {}, positive/total ratio is {}".format(y.sum(), len(y)-y.sum(), round(y.sum()/len(y), 3)))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print("\nUsing Linear SVC")
svc = LinearSVC()
model, train_acc_score, test_acc_score = train_test_model(svc, X_train, y_train, X_test, y_test)
trainning_accuracies_linear.append(train_acc_score)
testing_accuracies_linear.append(test_acc_score)

print("\nUsing Polynomial SVC")
psvc = SVC(kernel="poly")
model, train_acc_score, test_acc_score = train_test_model(psvc, X_train, y_train, X_test, y_test)
trainning_accuracies_polynomial.append(train_acc_score)
testing_accuracies_polynomial.append(test_acc_score)

print("\nUsing Random Forest Classifier")
rcf = RandomForestClassifier()
model, train_acc_score, test_acc_score = train_test_model(rcf, X_train, y_train, X_test, y_test)
trainning_accuracies_rfc.append(train_acc_score)
testing_accuracies_rfc.append(test_acc_score)
```

Kerem Şahin 24177
Ege Erdoğan 25331

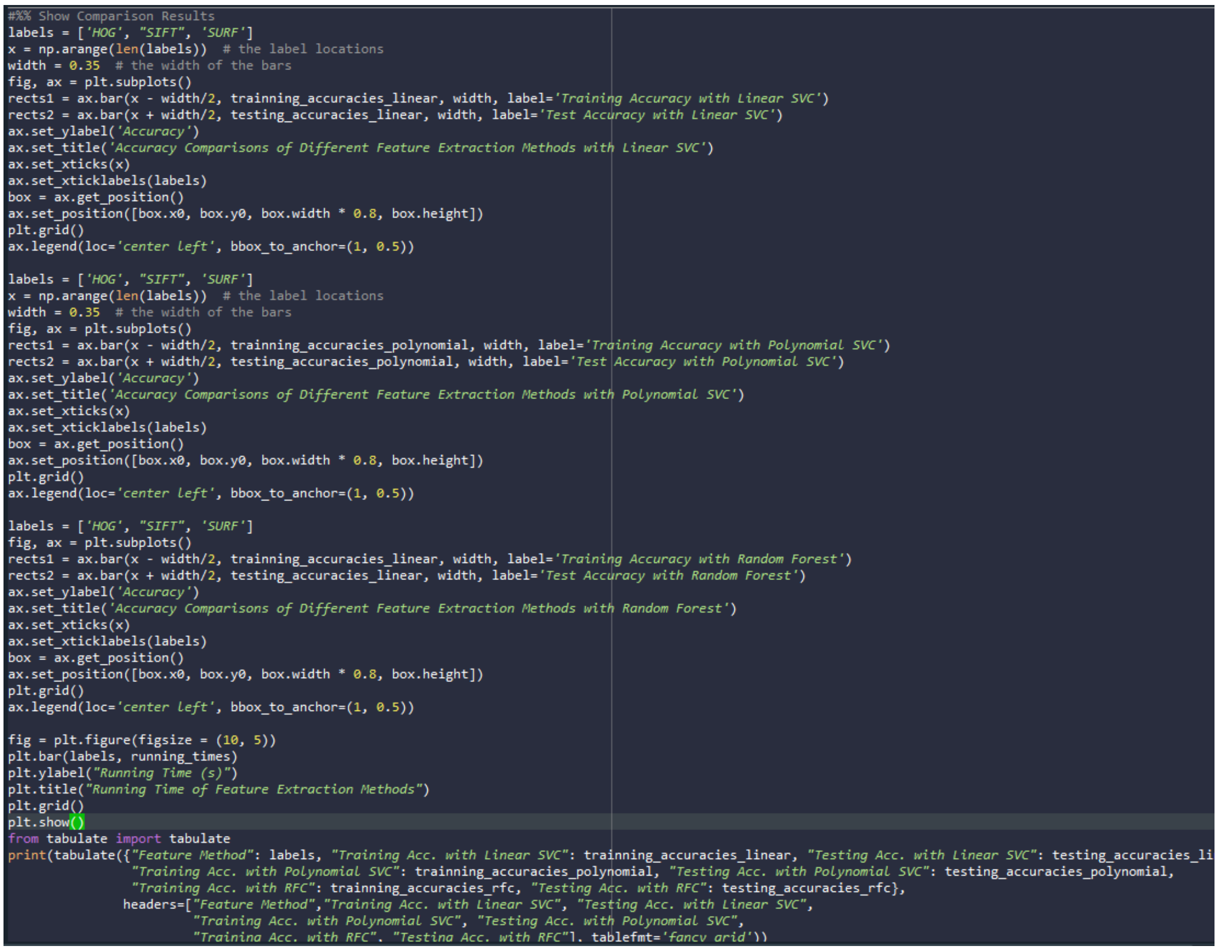


```
#%% Detection with SIFT Features
print("\n*****Human Detection with SIFT Features*****")
start = time.time()
df = []
label = []
for img in pedestrian_imgs:
    sift = cv2.xfeatures2d.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(img, None)
    if descriptors is not None:
        temp = np.resize(descriptors, 128*128)
        df.append(temp)
        label.append(1)

for img in notpedestrian_imgs:
    sift = cv2.xfeatures2d.SIFT_create()
    keypoints, descriptors = sift.detectAndCompute(img, None)
    if descriptors is not None:
        temp = np.resize(descriptors, 128*128)
        df.append(temp)
        label.append(0)
end = time.time()
running_times.append(end-start)
X = np.array(df)
y = np.array(label)
print("Total data shape:",X.shape)
print("Labels shape:",y.shape)
print("Number of positive labels {}, negative labels {}, positive/total ratio is {}".format(y.sum(), len(y)-y.sum(), round(y.sum()/len(y), 3)))
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print("\nUsing Linear SVC")
svc = LinearSVC()
model, train_acc_score, test_acc_score = train_test_model(svc, X_train, y_train, X_test, y_test)
trainning_accuracies_linear.append(train_acc_score)
testing_accuracies_linear.append(test_acc_score)

print("\nUsing Polynomial SVC")
psvc = SVC(kernel="poly")
model, train_acc_score, test_acc_score = train_test_model(psvc, X_train, y_train, X_test, y_test)
trainning_accuracies_polynomial.append(train_acc_score)
testing_accuracies_polynomial.append(test_acc_score)

print("\nUsing Random Forest Classifier")
rcf = RandomForestClassifier()
model, train_acc_score, test_acc_score = train_test_model(rcf, X_train, y_train, X_test, y_test)
trainning_accuracies_rfc.append(train_acc_score)
testing_accuracies_rfc.append(test_acc_score)
```

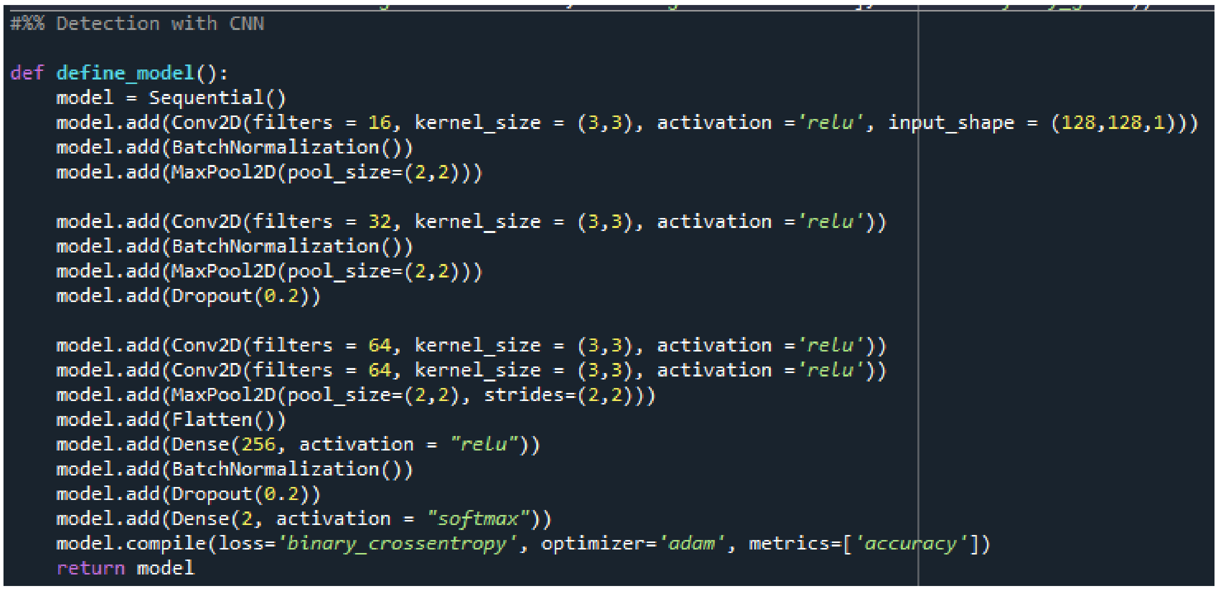


```
#%% Detection with SURF Features
print("\n*****Human Detection with SURF Features*****")
start = time.time()
df = []
label = []
for img in pedestrian_imgs:
    SURF = cv2.xfeatures2d.SURF_create(500)
    kp, des = SURF.detectAndCompute(img, None)
    if des is not None:
        temp = np.resize(des, 64*64)
        df.append(temp)
        label.append(1)

for img in notpedestrian_imgs:
    SURF = cv2.xfeatures2d.SURF_create(500)
    kp, des = SURF.detectAndCompute(img, None)
    if des is not None:
        temp = np.resize(des, 64*64)
        df.append(temp)
        label.append(0)
end = time.time()
running_times.append(end-start)

X = np.array(df)
y = np.array(label)
print("Total data shape:",X.shape)
print("Labels shape:",y.shape)
print("Number of positive labels {}, negative labels {}, positive/total ratio is {}".format(y.sum(), len(y)-y.sum(), round(y.sum()/len(y), 3)))

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
print("\nUsing Linear SVC")
svc = LinearSVC()
model, train_acc_score, test_acc_score = train_test_model(svc, X_train, y_train, X_test, y_test)
trainning_accuracies_linear.append(train_acc_score)
testing_accuracies_linear.append(test_acc_score)

print("\nUsing Polynomial SVC")
psvc = SVC(kernel="poly")
model, train_acc_score, test_acc_score = train_test_model(psvc, X_train, y_train, X_test, y_test)
trainning_accuracies_polynomial.append(train_acc_score)
testing_accuracies_polynomial.append(test_acc_score)

print("\nUsing Random Forest Classifier")
rcf = RandomForestClassifier()
model, train_acc_score, test_acc_score = train_test_model(rcf, X_train, y_train, X_test, y_test)
trainning_accuracies_rfc.append(train_acc_score)
testing_accuracies_rfc.append(test_acc_score)
```

Kerem Şahin 24177
Ege Erdoğan 25331

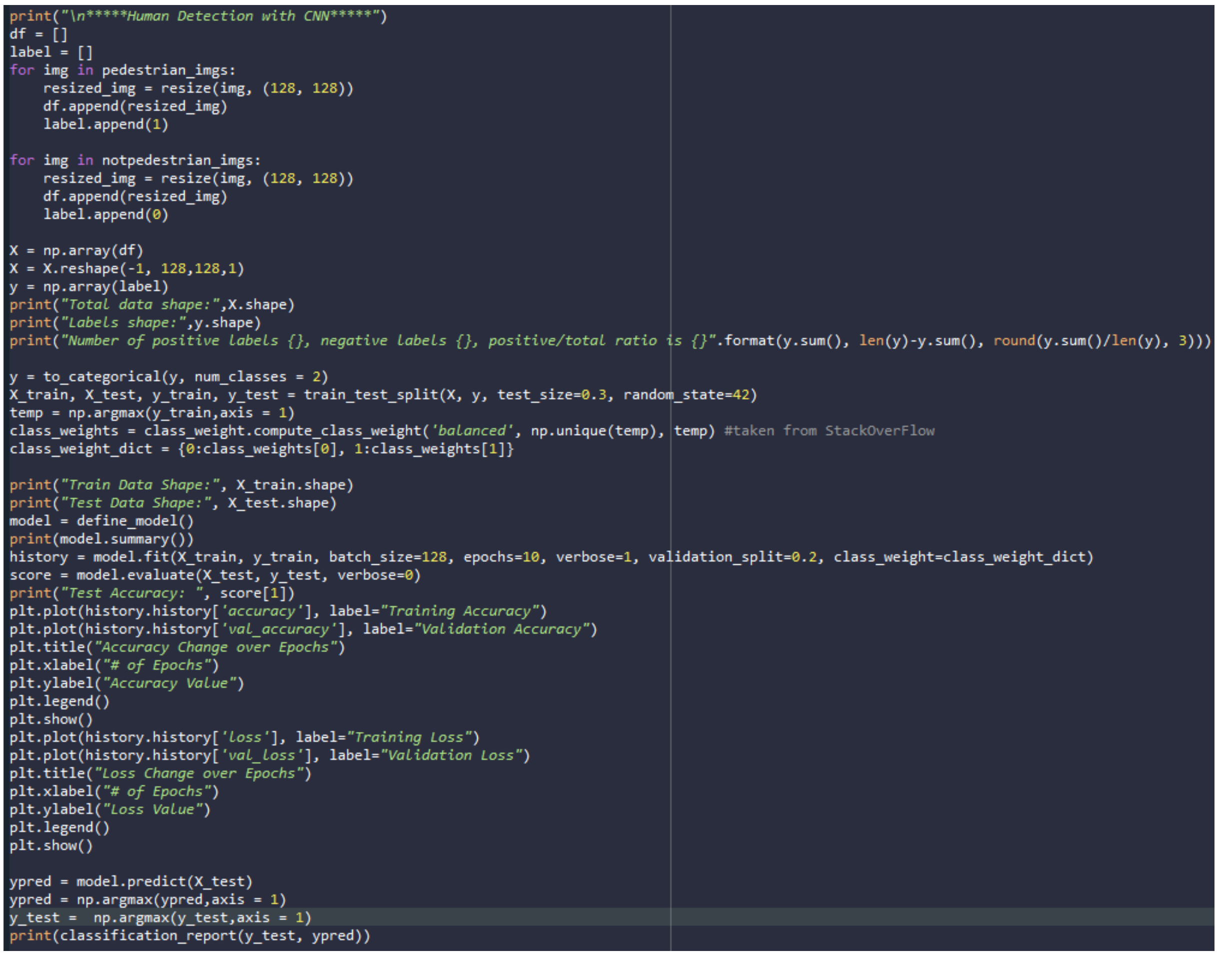


```
#%% Show Comparison Results
labels = ['HOG', "SIFT", 'SURF']
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars
fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, trainning_accuracies_linear, width, label='Training Accuracy with Linear SVC')
rects2 = ax.bar(x + width/2, testing_accuracies_linear, width, label='Test Accuracy with Linear SVC')
ax.set_ylabel('Accuracy')
ax.set_title('Accuracy Comparisons of Different Feature Extraction Methods with Linear SVC')
ax.set_xticks(x)
ax.set_xticklabels(labels)
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
plt.grid()
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

labels = ['HOG', "SIFT", 'SURF']
x = np.arange(len(labels))  # the label locations
width = 0.35  # the width of the bars
fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, trainning_accuracies_polynomial, width, label='Training Accuracy with Polynomial SVC')
rects2 = ax.bar(x + width/2, testing_accuracies_polynomial, width, label='Test Accuracy with Polynomial SVC')
ax.set_ylabel('Accuracy')
ax.set_title('Accuracy Comparisons of Different Feature Extraction Methods with Polynomial SVC')
ax.set_xticks(x)
ax.set_xticklabels(labels)
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
plt.grid()
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

labels = ['HOG', "SIFT", 'SURF']
fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, trainning_accuracies_linear, width, label='Training Accuracy with Random Forest')
rects2 = ax.bar(x + width/2, testing_accuracies_linear, width, label='Test Accuracy with Random Forest')
ax.set_ylabel('Accuracy')
ax.set_title('Accuracy Comparisons of Different Feature Extraction Methods with Random Forest')
ax.set_xticks(x)
ax.set_xticklabels(labels)
box = ax.get_position()
ax.set_position([box.x0, box.y0, box.width * 0.8, box.height])
plt.grid()
ax.legend(loc='center left', bbox_to_anchor=(1, 0.5))

fig = plt.figure(figsize = (10, 5))
plt.bar(labels, running_times)
plt.ylabel("Running Time (s)")
plt.title("Running Time of Feature Extraction Methods")
plt.grid()
plt.show()
from tabulate import tabulate
print(tabulate({"Feature Method": labels, "Training Acc. with Linear SVC": trainning_accuracies_linear, "Testing Acc. with Linear SVC": testing_accuracies_li
                "Training Acc. with Polynomial SVC": trainning_accuracies_polynomial, "Testing Acc. with Polynomial SVC": testing_accuracies_polynomial,
                "Training Acc. with RFC": trainning_accuracies_rfc, "Testing Acc. with RFC": testing_accuracies_rfc},
               headers=["Feature Method","Training Acc. with Linear SVC", "Testing Acc. with Linear SVC",
                        "Training Acc. with Polynomial SVC", "Testing Acc. with Polynomial SVC",
                        "Training Acc. with RFC", "Testing Acc. with RFC"], tablefmt='fancy_grid'))
```



```
#%% Detection with CNN

def define_model():
    model = Sequential()
    model.add(Conv2D(filters = 16, kernel_size = (3,3), activation ='relu', input_shape = (128,128,1)))
    model.add(BatchNormalization())
    model.add(MaxPool2D(pool_size=(2,2)))

    model.add(Conv2D(filters = 32, kernel_size = (3,3), activation ='relu'))
    model.add(BatchNormalization())
    model.add(MaxPool2D(pool_size=(2,2)))
    model.add(Dropout(0.2))

    model.add(Conv2D(filters = 64, kernel_size = (3,3), activation ='relu'))
    model.add(Conv2D(filters = 64, kernel_size = (3,3), activation ='relu'))
    model.add(MaxPool2D(pool_size=(2,2), strides=(2,2)))
    model.add(Flatten())
    model.add(Dense(256, activation = "relu"))
    model.add(BatchNormalization())
    model.add(Dropout(0.2))
    model.add(Dense(2, activation = "softmax"))
    model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
    return model
```

Kerem Şahin 24177
Ege Erdoğan 25331



```
print("\n*****Human Detection with CNN*****")
df = []
label = []
for img in pedestrian_imgs:
    resized_img = resize(img, (128, 128))
    df.append(resized_img)
    label.append(1)

for img in notpedestrian_imgs:
    resized_img = resize(img, (128, 128))
    df.append(resized_img)
    label.append(0)

X = np.array(df)
X = X.reshape(-1, 128,128,1)
y = np.array(label)
print("Total data shape:",X.shape)
print("Labels shape:",y.shape)
print("Number of positive labels {}, negative labels {}, positive/total ratio is {}".format(y.sum(), len(y)-y.sum(), round(y.sum()/len(y), 3)))

y = to_categorical(y, num_classes = 2)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
temp = np.argmax(y_train,axis = 1)
class_weights = class_weight.compute_class_weight('balanced', np.unique(temp), temp) #taken from StackOverFlow
class_weight_dict = {0:class_weights[0], 1:class_weights[1]}

print("Train Data Shape:", X_train.shape)
print("Test Data Shape:", X_test.shape)
model = define_model()
print(model.summary())
history = model.fit(X_train, y_train, batch_size=128, epochs=10, verbose=1, validation_split=0.2, class_weight=class_weight_dict)
score = model.evaluate(X_test, y_test, verbose=0)
print("Test Accuracy: ", score[1])
plt.plot(history.history['accuracy'], label="Training Accuracy")
plt.plot(history.history['val_accuracy'], label="Validation Accuracy")
plt.title("Accuracy Change over Epochs")
plt.xlabel("# of Epochs")
plt.ylabel("Accuracy Value")
plt.legend()
plt.show()
plt.plot(history.history['loss'], label="Training Loss")
plt.plot(history.history['val_loss'], label="Validation Loss")
plt.title("Loss Change over Epochs")
plt.xlabel("# of Epochs")
plt.ylabel("Loss Value")
plt.legend()
plt.show()

ypred = model.predict(X_test)
ypred = np.argmax(ypred,axis = 1)
y_test =  np.argmax(y_test,axis = 1)
print(classification_report(y_test, ypred))
```