Introduction

In this project, we wanted to classify two classes – pedestrian traffic lights and vehicle traffic lights. To so we used the data from Kaggle: https://www.kaggle.com/hj23hw/pedestrian-augmented-traffic-light-dataset.

The project goal was to classify pedestrian traffic lights and vehicle traffic lights, some uses we thought can be useful from this project is for automation for cars so it will know the difference between the two types of traffic lights, it is not a complete model but can be used as features extractor for a more complicated model.

Our goal was to experience the algorithms – CNN, KNN, SVM, Logistic Regression and to analyze the results, we also wanted to see what happens if we combine all the models together (kind of Adaboost).

Our data is from various sizes in format JPG, we transformed it to 128x128x3 RGB. In the CNN we transformed the images to - number_of_samplesX128X128X3, and in all other models to - number_of_samplesX49152 (128*128*3).

For better results, we also made some random transformations on the data to make it more diverse, for example, to invert the image, rotate the image, etc.

When using the Adaboost we gave each model (rule) weight according to its accuracy and predicted the label of the new image by summing all models predictions multiplied by their weight. If the sum was greater equal to – 0 then the output is pedestrian, otherwise, it is a vehicle.

Struggles & Solutions

1. Our main struggle in this project was to find enough data, we wanted equal number of images from pedestrian traffic lights and vehicle traffic lights.

We did not manage to find enough vehicle traffic lights images, so we used 1630 pedestrian traffic lights and 767 vehicle traffic lights, we trained our model with this as imbalanced data, the results were not bad – we got 99.6% accuracy on CNN, 97.8% accuracy on KNN, 98.7% accuracy on Logistic Regression, 98.8% on SVM and 99.2% on the Adaboost.

At first, we tried to find more vehicle traffic lights images, but we did not manage to find data that fit our purpose.

Secondly, we thought to duplicate images of vehicle traffic lights with transformation for each image, for example, to invert the image, rotate the image, etc.

Thirdly, we wanted to add bias to images that were classified as vehicle traffic lights due to lack of data, but we figured out it won't go well because some models can't add bias to the classes such as KNN because we didn’t even have weights, although we can add weights to the KNN such that the distance to pedestrian traffic light image will grow with some bias, so if we have two points with equivalent distance from our point and one is pedestrian and the other is a vehicle the vehicle will be chosen. But we assumed that this algorithm does not work well.

Eventually, we decided to use less data, we removed pedestrian traffic lights images such that we have 767 for each class, 1534 images in total.

We chose this solution because we noticed that our data is similar, that is a lot of images look the same. Although we used a method to transform our data in the code, we still think it is important to use a variety of images, which this solution gives us by removing images.

The results we got from the balanced data 99.2% accuracy on CNN, 97.4% accuracy on KNN, 97.8% accuracy on Logistic Regression, 97.9% on SVM, and 98.9% on the "Adaboost".

2. Another problem we had is that in the CNN model the train accuracy was too high (100%)

Most of the time, this can imply that we have overfitting.

We think that the reason for this is that the regularization is too low, and our model is too complicated.

To solve this problem we added more regularization, it is we increased the dropout layer to 0.6, so every iteration about 60% of the neurons are dead. This solution decreases the power of the model and makes it more simple, so our accuracy also decreased but we believe that the model has better generalization, meaning that for more complicated new data which is not in our dataset it will work better.

What we learned

1. How to write the algorithms in Python, we experience the Keras and Sklearn libraries and how to use them.

2. SVM and Logistic Regression misclassify the same images but Logistic Regression misclassify more, we think the reason for that is that both give us some hyperplane to separate the data so it's reasonable that they both will learn the same hyperplane and that’s why their results are very similar.

3. The Adaboost algorithm gives better results than SVM, Logistic Regression, and KNN but not better than the CNN So for images classification, CNN seemed to be the most efficient model.

4. The KNN works best for one neighbor because all the images from the same class are similar, this implies that this model does not good for image classification, also because the distance between images is ambiguous, for example, if we take one image and blacken some pixels in some position and in another image we will blacken the same amount of pixels in another position their distance from the original image will be the same.

5. We extracted every image that every model misclassified and we saw that there was one image that all models classified correctly except the CNN, we think that the CNN sometimes has constraints so he misclassifies simple things that other models can classify correctly.

6. We also tried our models on new images that were not from our dataset, and the results weren't too good, the SVM classified 8/10 images correctly the other models classified 6/10 correctly, and the Adaboost classified 7/10 correctly.

The Results weren't so good because our data isn't reflect the real world good enough, it is too small and not very diverse.

It should be noted that when we checked the new data we trained the CNN for only 15 iterations (due to lack of time), a better results might be achived if we trained it for 100 iterations.

7. All models except CNN transform our images – 128x128x3 to vector of 49152 (128*128*3), because similar images with the regular form will still be similar in the vector form which is what we want.

We also tried the same method for greyscale to see if the representation of an image as vector will be effected, the results were not good significantly on the dataset, and for the images that were not from the dataset the CNN got 5/10 compared to 6/10, SVM got 7/10 compared to 8/10, KNN got 5/10 compared to 6/10, Logistic Regression got 7/10 compared to 6/10 and Adaboost got 7/10 compared to 7/10. So we see that for our dataset it gave relatively bad results to the RGB, but it seemed to be more generalized for images from the real world, and give results similar to the test results.

```
[98]  1 cnn_preds = model.predict_classes(x)
      2 cnn_acc = metrics.accuracy_score(y, cnn_preds)
      3 print(cnn_preds)
      4 print(cnn_acc)
```

```
[1 1 0 0 0 1 1 0 0 0]
0.5
```

```
1 print("SVM" , svm_acc)
2 print("KNN" ,knn_acc)
3 print("Logistic Regression" ,logistic_acc)
```

```
SVM 0.7
KNN 0.5
Logistic Regression 0.7
```

```
1 adb = adaboost(x_, y, cnn_preds, KNN, LGR, SVM)
2 adb.run()
```

```
Accuracy: 0.7
Precision: 0.625
Recall: 1.0
              precision    recall  f1-score   support

  pedestrian       1.00      0.40      0.57         5
     regular       0.62      1.00      0.77         5

    accuracy                           0.70        10
   macro avg       0.81      0.70      0.67        10
weighted avg       0.81      0.70      0.67        10
```