Detection of Pedestrians and Vehicles to aid Advanced Driver-Assistance Systems

Aakash Ramchandani
aramc003@ucr.edu
862324774

Rahul Sailesh Wadhwa
rwadh004@ucr.edu
862309846

INTRODUCTION

Advanced Driver Assistance Systems (ADAS) were developed in order to reduce the number of accidents that take place due to human error. These systems include different types of technologies that can be used to assist drivers in driving and parking functions. The main objective of these systems is to increase car and road safety by making use of automated technology like cameras and different kinds of sensors. With the improvement of ADAS technologies, it may eventually result in vehicles that will be truly “self-driving” cars and will be able to move without any form of human supervision and assistance.

Some of the main features of ADAS include but are not limited to [1] -

- Vehicle detection
- Parking assistance
- Lane departure warning
- Traffic sign recognition
- Pedestrian detection
- Blind-spot assistance
- Navigation systems

Figure 1: Pedestrian Detection in ADAS

Object recognition is a computer vision task to identify different objects in digital images. Convolutional Neural Networks (CNNs) are an effective way of capturing different patterns in multidimensional spaces. They are a particularly useful tool in evaluating images and can be used to learn certain “features” for a set of images. A CNN comprises either one or several convolutional layers that can be used to extract useful values from the image. They can be used to act as “filters” with each filter learning a certain set of patterns and features of the image. When these layers are stacked on top of one another they can be used to effectively learn patterns in the image. “For instance, the lower layers will produce feature maps for vertical and horizontal edges, corners, and other simple patterns. The next layers can detect more complex patterns such as grids and circles. As you move deeper into the network, the layers will detect complicated objects such as cars, houses, trees, and people.”[2]

The motivation behind this project is to develop a neural network that is capable of detecting vehicles and pedestrians given an image as input by one of the cameras. Deep learning methods can be used effectively to extract the features from an image and are very useful for object detection and classification given any input image. They can be used to train a model to classify images very quickly with relatively high accuracy and with considerably less training data.

RELATED WORK

There has been plenty of work done on vehicle and pedestrian detection using different methods. A variety of different machine learning methods have been used effectively to train models for vehicle and pedestrian detection. Several methods including SVM, Fast R-CNN, pretrained models with Imagenet, Yolov3, etc have been used effectively however most of these models require large amounts of data.

For human and pedestrian detection, SVM with Histogram of Oriented Gradients (HOG)[7] descriptors have been used very effectively in creating bounding boxes around humans detected in a given image. OpenCV has a pre-trained model that makes use of HOG with linear SVM to classify humans. However, this method tends to be very sensitive with image rotations and the computation in this method is tardy so to speak. Another feature when making use of HOG descriptors that is particularly useful in our case as well could be the extraction of different regions of interest (ROIs). Each ROI can then be evaluated individually and free of other background data that could be considered to be noise. This situation can be useful when used with neural networks as it would help restrict the regions of interest for which the image has to be evaluated by the model, thereby reducing prediction time.

Another method that is particularly useful for object detection is the Yolov3[8] (You Only Look Once) classifier that uses features learned by a deep convolutional neural network to detect an object. This classifier shows how deep neural networks can be used effectively in object detection with a relatively high accuracy and speed. However, this method fails to detect objects that are close by as effectively as it only takes 2 bounding regions at a time and also struggles to detect smaller objects.

DATASET

Object detection as such can be considered to be a supervised learning task as each of the objects to be detected will have to be assigned a label. For this reason, we had to make use of different datasets for the detection of vehicles and pedestrians.

For vehicle detection, we made use of the KITTI-extracted dataset. The KITTI dataset is captured by driving around the mid-size city of Karlsruhe, in rural areas, and on highways[4]. Up to 15 cars and 30 pedestrians are visible per image in the original dataset. Our KITTI-extracted dataset contains 2 types of images: vehicle-focused images from different angles (label 1) and images of backgrounds that are commonly seen on roads while driving (label 0). Around 5966 images of vehicles were extracted from this dataset and were used to train the model.

For pedestrian detection, we made use of 2 datasets, Penn-Fudan Database for Pedestrian Detection and Segmentation[5] and the Human Detection Dataset on Kaggle[6] were used. There was a total of around 729 images containing people that were used to train the model for pedestrian detection. The images had varying dimensions however they were resized to 64*64 pixels using OpenCV, to match the other training image dimensions. In addition to this, a total of 8968 images of backgrounds were also used, from the KITTI-extracted data. All training images used had 3 channels.

We also performed standard augmentation on each of the images, to scale the size of the dataset and increase the number of training images we could make use of. This not only increased the amount of data that we had but it also increased the variation in the data and made our model more versatile and capable of identifying lossy images accurately as well.

Figure 2: Vehicle images used for training before and after Standard augmentation

Figure 3: Background images used for training before and after Standard augmentation

Figure 4: Pedestrian images used for training before and after Standard augmentation

PROPOSED METHOD

Our implementation involves various components that work in conjunction to detect pedestrians and/or vehicles in an image. It involves a data preprocessing module, data augmentation module, 2 CNN models (one for detecting pedestrians and the other for vehicles), Sliding Window for image segmentation, and the plotting of boxes around the detected objects in the input image, as visible in Figure 8. First, we import the various types of data:

- Vehicle images that consist of back, front, left and right oriented cars.
- Pedestrian images in various backgrounds, mostly outdoors
- Images of backgrounds - Highways, trees, sky, empty crosswalks, etc, which do not contain any people or vehicle objects.

Now, we normalize all the images by subtracting the mean and dividing by the standard deviation. The next step is to create 2 datasets, one for training the model that detects pedestrians and the other for training the vehicle detection model. The first dataset was built by combining an equal number of vehicle and background images using random sampling, which would train the model to differentiate between the presence of a vehicle from its surroundings. Similarly, for the second model, an equal number of pedestrian and background images were combined. After this we applied standard augmentation on both datasets. This involved horizontally flipping each image as well as performing a left/right and up/down shift of the pixels. The shifts were done as follows: pick two independent integers k1, k2 uniformly between [−K,K] range (we used K=8). Move the image upwards by k1 and right-wards by k2 pixels (negative value means downwards and leftwards), and zero pad the missing pixels. This essentially doubled the size of our datasets to 23,864 images (vehicle dataset) and 2,916 images (pedestrian dataset).

Next we had 2 separately trained CNNs, one for pedestrian detection and the other for vehicle detection. After trying different configurations, we decided to use the following setup:

- 16 Convolution Layers - 3x3 Kernel, Stride 1, Zero Padding, ReLu Activation
- Dropout with probability 0.5
- 32 Convolution Layers - 3x3 Kernel, Stride 1, Zero Padding, ReLu Activation
- Dropout with probability 0.5
- 64 Convolution Layers - 3x3 Kernel, Stride 1, Zero Padding, ReLu Activation
- MaxPooling - 8x8 Kernel, 8x8 Stride, No zero padding
- Dropout with probability 0.5
- Fully Connected Layer - Output a single logit, followed by Sigmoid Activation

Dropout helped prevent overfitting. This same configuration was used for training both datasets. The best parameters for each model were saved and later used for prediction on new input images. The total number of parameters required for the 2 models was only 27,681 each.

```
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
NeuralNetwork                            [32, 1, 1, 1]             --
├─Sequential: 1-1                        [32, 16, 64, 64]          --
│    └─Conv2d: 2-1                       [32, 16, 64, 64]          448
│    └─ReLU: 2-2                         [32, 16, 64, 64]          --
├─Sequential: 1-2                        [32, 16, 64, 64]          --
│    └─Dropout: 2-3                      [32, 16, 64, 64]          --
├─Sequential: 1-3                        [32, 32, 64, 64]          --
│    └─Conv2d: 2-4                       [32, 32, 64, 64]          4,640
│    └─ReLU: 2-5                         [32, 32, 64, 64]          --
├─Sequential: 1-4                        [32, 32, 64, 64]          --
│    └─Dropout: 2-6                      [32, 32, 64, 64]          --
├─Sequential: 1-5                        [32, 64, 64, 64]          --
│    └─Conv2d: 2-7                       [32, 64, 64, 64]          18,496
│    └─ReLU: 2-8                         [32, 64, 64, 64]          --
├─Sequential: 1-6                        [32, 64, 8, 8]            --
│    └─MaxPool2d: 2-9                    [32, 64, 8, 8]            --
├─Sequential: 1-7                        [32, 64, 8, 8]            --
│    └─Dropout: 2-10                     [32, 64, 8, 8]            --
├─Sequential: 1-8                        [32, 1, 1, 1]             --
│    └─Conv2d: 2-11                      [32, 1, 1, 1]             4,097
│    └─Sigmoid: 2-12                     [32, 1, 1, 1]             --
==========================================================================
Total params: 27,681
Trainable params: 27,681
Non-trainable params: 0
Total mult-adds (G): 3.09
==========================================================================
Input size (MB): 1.57
Forward/backward pass size (MB): 117.44
Params size (MB): 0.11
Estimated Total Size (MB): 119.12
==========================================================================
```

Figure 5: The configuration of our custom model for batch size = 32

Once we had the models trained, we implemented a Sliding window technique to convert an input image into many smaller images of varying sizes. Starting with a sliding window of size 32x32 pixels and stride 16 we divided the input image into multiple images and stored them. Then we scaled the size of the window and the stride by increasing it by 50% each time, generating many images that were all saved. Figure 6 represents the basic idea of how the sliding window size kept increasing each time.

Figure 6: Sliding windows of various sizes to extract images at different scales from input image

Also, while creating the image segments, we made sure to store their pixel positions (relative to the original image) so that they can be used later to make boxes in case an object is detected. Now, we resized all the extracted images to 64x64 pixels and applied the 2 models on all the images. This gave us 2 values of probabilities for each image segment, one for the presence of a vehicle in the image segment, and the other for the presence of a pedestrian. If this probability was higher than a fixed threshold value, then we can conclude that the image segment contained a vehicle or a pedestrian. After getting a list of all the segments that had either a pedestrian or a vehicle, we used OpenCV to draw rectangular borders for the segments that had an object detected, for which we used the previously stored segment locations. Blue boxes represent vehicles whereas red boxes represent pedestrians, as visible in Figure 7.

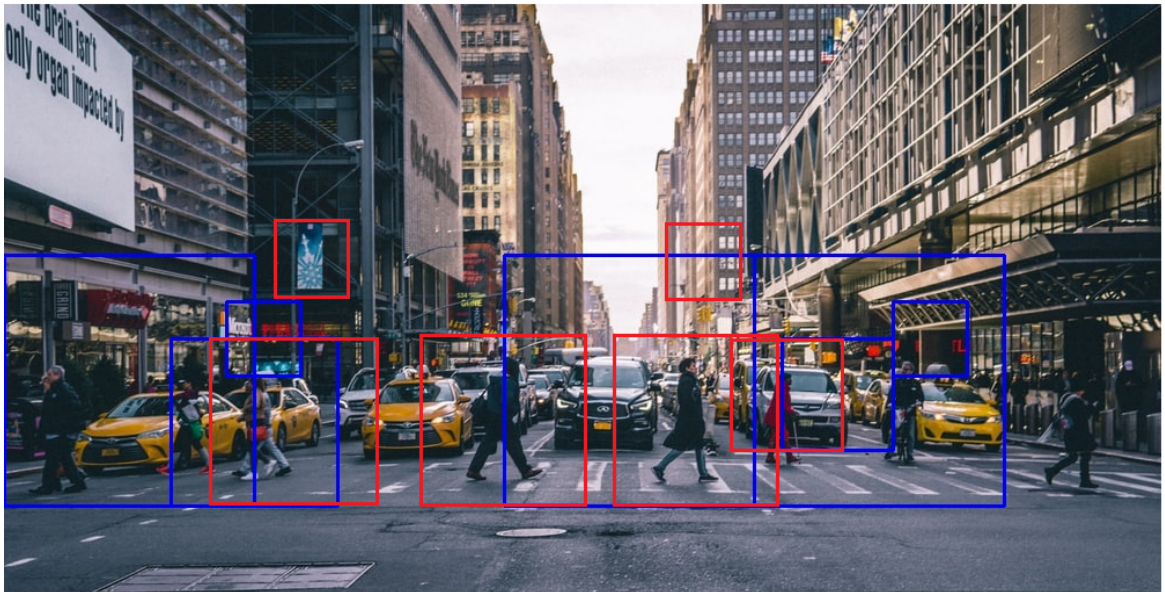

Figure 7: Final output after applying both model

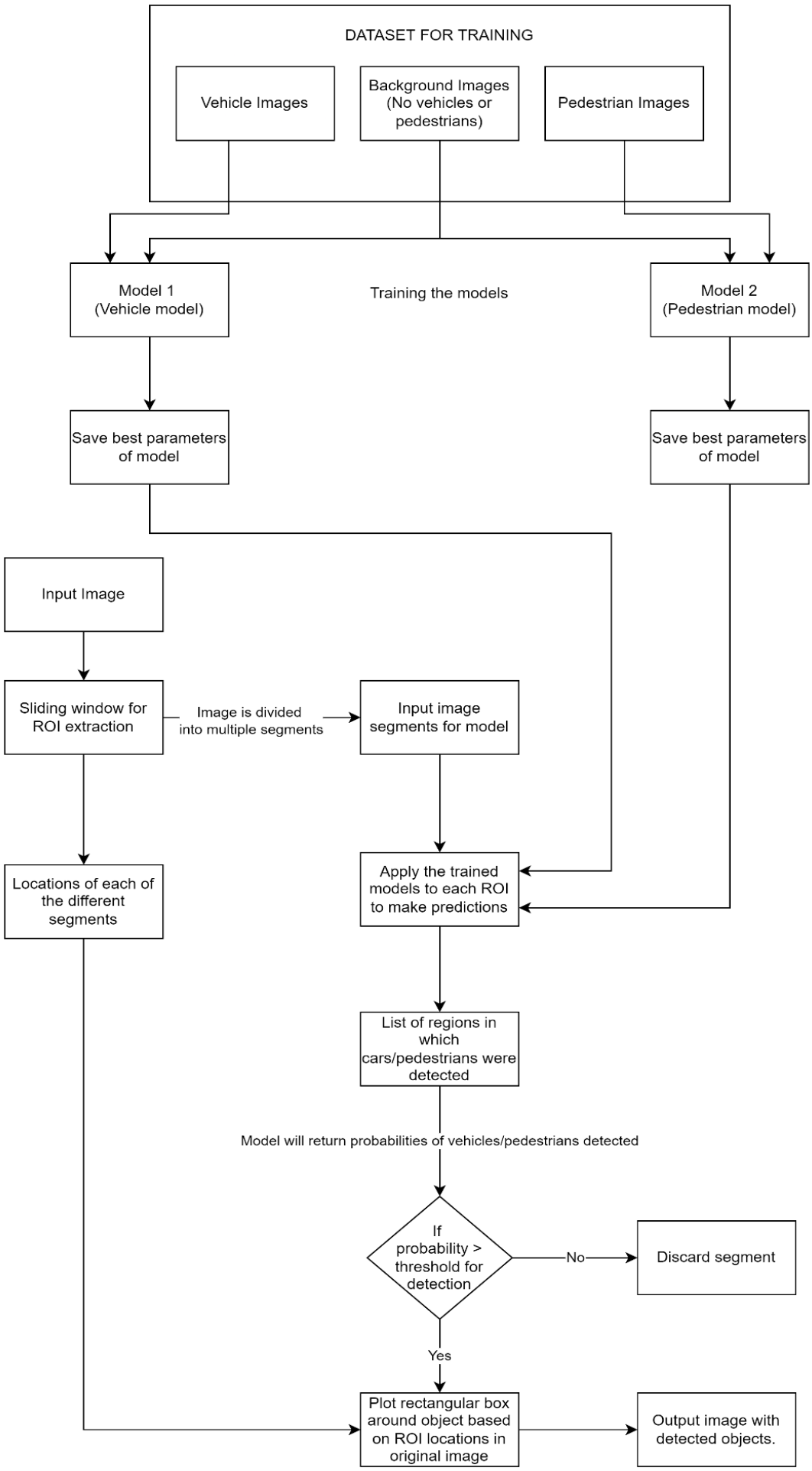

Figure 8: A flowchart of our implementation

RESULTS

For the two individual models we trained, we had near perfect accuracy on our testing data. While training, we used Adam optimizer (learning rate = 0.001) with a mini-batch size of 32 and MSE Loss. We divided the datasets into Training and testing with a 80:20 ratio, and managed to achieve ~ 99.8% test accuracy for vehicle detection and ~ 97.6% test accuracy for pedestrian detection. However, since we had less training data for pedestrians (2,916 images), it did not adapt well to new images and often mispredicted the background as a pedestrian. It also was not able to detect pedestrians that were in a dark background as visible in Figure 7 (left-most person). The vehicles on the other hand were detected quite accurately, including detection of one of the vehicles (on the left in figure 7) that was partially hidden by a car in front of it. In order to more accurately detect objects, that is, have a tighter bounding box around the object, we need to fine tune our sliding window by testing various values of stride and by how much the window scales after each iteration on the input image. The detection on average (using pretrained models) took 3 seconds. This too, could be greatly improved if we fine-tune our sliding window implementation. Also, with the use of GPU/TPU, we could reduce the prediction time exponentially, which would eventually make it efficient enough for use in real-time situations.

FUTURE WORK

Some areas in which our model could be improved to increase the accuracy and efficiency include:

- Try different sizes of sliding windows and stride to more efficiently divide the input image into multiple segments, thereby making the object-detection quicker.
- Fine-tuning the hyperparameters like the value of K in standard augmentation to make the model more versatile and improve the detection accuracy. We could also tune the values of the 2 thresholds based on which we decide whether to consider the detection or not.
- Adding mixup and cutout augmentation to the dataset will also help in detection of certain objects that are only partially visible or hidden by other objects.
- Instead of 2 models for pedestrian/vehicle detection, we can combine them into one single model that can detect both cars and pedestrians. This involves multiclass classification (3 classes - vehicle, pedestrian and background) and would reduce the number of total parameters required, resulting in faster predictions as only a single model has to be applied, instead of 2 in our case.
- We can also implement lane detection to notify a driver if he unknowingly steers out of his lane and parking assistance to detect an empty parking space.

CODE

The zipped file of our project code can be found at the below link

https://drive.google.com/drive/folders/1xZyo93s2dHKsKtAKXQcz8HjY0J4eLpfg?usp=sharing

REFERENCES

1. https://www.synopsys.com/automotive/what-is-adas.html
2. https://bdtechtalks.com/2021/06/21/object-detection-deep-learning/
3. https://www.teendriversource.org/learning-to-drive/self-driving-cars-adas-technologies#:~:text=As%20ADAS%20technologies%20improve%2C%20they,yet%20available%20to%20the%20consumer.
4. “Vision meets Robotics: The KITTI Dataset”, Andreas Geiger and Philip Lenz and Christoph Stiller and Raquel Urtasun International Journal of Robotics Research (IJRR) (2013), http://www.cvlibs.net/datasets/kitti/
5. “Penn-Fudan Database for Pedestrian Detection and Segmentation”, https://www.cis.upenn.edu/~jshi/ped_html/
6. “Human Detection Dataset”, https://www.kaggle.com/datasets/constantinwerner/human-detection-dataset
7. “HOG based fast human detection”, M. Kachouane, S. Sahki, M. Lakrouf, N. Ouadah, 24th International Conference on Microelectronics (ICM) (2012)
8. “YOLOv3: An Incremental Improvement”, Joseph Redmon, Ali Farhadi, https://arxiv.org/abs/1804.02767 (2018)
9. “A Fast Object Detector for ADAS using Deep Learning”, a presentation from Panasonic (2017)
10. “Object Recognition with Gradient-Based Learning”, Yann LeeCun, Patrick Haffner, Leon Bottou, Yoshua Bengio, http://yann.lecun.com/exdb/publis/pdf/lecun-99.pdf
11. “Convolutional Networks for Images, Speech and Time-Series”, Yann LeCun, Yoshua Bengio, http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf