2019

DATA SET GENERATION USING DEEP LEARNING ALGORITHMS AND VISUAL FEATURE TRACKING

Kusuma Pallapotu
Michigan Technological University, kpallapo@mtu.edu

DATA SET GENERATION USING DEEP LEARNING ALGORITHMS AND VISUAL FEATURE TRACKING

By
Kusuma Pallapotu

A REPORT
Submitted in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE
In Computer Science

MICHIGAN TECHNOLOGICAL UNIVERSITY
2019

This Report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Computer Science.

Department of Computer Science

Report Co-advisor:	*Dr. Nilufer Onder*
Report Co-advisor:	*Dr. Nina Mahmoudian*
Committee Member:	*Dr. Bo Chen*
Committee Member:	*Dr. Jianhui Yue*
Department Chair:	*Dr. Zhenlin Wang*

Dedication

To my family and my dearest friends who have always supported me and have been there for me when I needed them. This would not have been possible without their support.

Contents

List of Figures

List of Tables

List of Abbreviations

AR	Augmented Reality
ASEF	Average of Synthetic Exact Filters
CNN	Convolution Neural Network
COCO	Common Objects in Context
CSRT	Discriminative Correlation Filter with Channel and Spatial Reliability
CSV	Comma separated values
DCF	Discriminative Correlation Filter
FPN	Feature Pyramid Network
FPS	Frames per second
GOTURN	Generic Object Tracking Using Regression Networks
GUI	Graphic User Interface
IOU	Intersection over Union
KCF	Kernelized Correlation Filters
MIL	Multiple Instance Learning
MOSSE	Minimum Output Sum of Squared Error
ROI	Region of Interest
SLAM	visual Simultaneous Localization and Mapping
SSE	Sum of Squared Error
TLD	Tracking, learning and detection
UI	User Interface
YOLO	You Only Look Once

Abstract

Object detection and classification plays a major role in today's modern technology. The implementations of these concepts range from consumer products to self driving cars. These concepts largely reply on the data sets used for training these models. There is a considerable amount of effort in generating these data sets for every specific application of these algorithms.

In this report, a method for generating image data sets with the use of visual feature tracking and deep learning algorithms for application in autonomous vehicles has been proposed. The aim is to reduce the time and effort dedicated towards the generation of these application specific data sets.

For this purpose, a software has been developed in Python for a Linux based system using Tensorflow, Keras, Pygames and OpenCV libraries which is capable of tracking an object of interest in a given media input specified by the user along with detecting various similar objects using a pre-trained Classification neural network. This software then compiles a file containing all the annotations for the above specified objects.

Chapter 1

Introduction

Though the concepts of deep learning have existed for far longer, the increase in the processing power of the computers and the capability of processing large sets of data in the last decade has led to the increase in their popularity. A major application of deep learning algorithms in today's world is in the field of object detection, classification and segmentation. These are largely used for the development and operation of autonomous vehicles or bots. Some of the most popular algorithms in object detection include the likes of YOLO [1] and Alexnet which are trained over hundred thousand images [2].

However, the algorithms are only as good as the images they were trained on. They can only perform within the range of data that they were trained on. Hence, an algorithm trained for use with an autonomous car cannot work on a drone, unless it was trained on an image data set that included images which the drone would encounter. Therefore, it is important to use the right set of annotated images for each application to achieve high performance.

These annotated data sets are generated over years of collecting, processing and labelling. This process is long and takes a lot of effort to generate annotated data for a specific application. For example, the COCO data set has over 100,000 images which has been in the works since 2014. There is a need to make this process of generating annotated data sets simpler and faster.

1.1 Objective

To ease this process of generating annotated data sets, we developed a software. The objective of this software is to generate annotations for image data sets while optimizing time and effort using the following.

1.1.1 Develop a GUI for user interaction

The software is designed to take media inputs both in Video and Image formats which the user can use to generate his/her data set. The user is then prompted to select one or multiple ROI around the objects of interest and annotate the selected objects.

1.1.2 Suggesting objects of interest

At each Frame/Image, the user is prompted with suggestions of objects similar to the ones already selected. These suggestions are made using

† Visual Feature Tracking

† Classification Neural Network

These suggestions can then be confirmed, modified or rejected based on the need of the user. Following the user's response, the suggestions for the following frames are modified accordingly.

The visual feature tracking feature is aimed at reducing the effort of annotating the same object across different frames of video. This will enable the annotation of the same object in different angle of view or different look as long as it is gradually changing.

The object detection feature is aimed at allowing the user to annotate any object that can already be detected by a trained CNN. This reduces the effort by automatically selecting the objects and will help in enriching the data base of the user.

Chapter 2

Background

2.1 Visual Feature Tracking

Applications for Visual Feature Tracking range from the areas of AR to self driving cars [3]. This is due to its ability to track complex objects through rotations, occlusions and other distractions [4].

2.1.1 Tracking vs Detection

Visual feature tracking is the problem of continuously localizing a target in a video-sequence when provided with its appearance in one frame [5] where as detection is the problem of identifying a set of objects in a frame of image that it is trained to recognize.

In terms of identification of an object in a given set of frames, Tracking works faster

than Detection in most of the cases [6]. It is only presumable as in Tracking there is much more information available about the object's appearance and location. A good Tracking algorithm will make use of all the information available about the object in order to track it efficiently in the further frames.

Tracking algorithms can accumulate a small amount of error through multiple frames and lose track of the object at a point. This is when detection algorithms can help rectify the problem. Running a detection algorithm for every few frames will reduce the error and optimize the computation time.

On the other hand, Tracking algorithms are of a great advantage if the object is being occluded. Detection algorithms would not be able to detect that object whereas, the Tracking algorithm would still be able to detect it.

2.1.2 Tracking Algorithms

The following are the 8 Tracking algorithms implemented in Opencv

2.1.2.1 Boosting Tracker

This Tracker algorithm is based on the on-line version of the AdaBoost Algorithm [7]. This algorithm is trained on positive and negative samples and each of the user selected object is considered to be a positive sample and the background around the bounding box as a negative sample. The location of the object is used to define a search region where the algorithm is run to identify a set of pixels with the least amount of error. And the detected object is then a part of the positive samples.

2.1.2.2 MIL Tracker

For each frame of the video sequence, a set of HAAR-like features are computed for each of the image patch [8]. A discriminative classifier is then used to determine if the object of interest is present in that image patch.

The tracker maintains the object location and collects a set of image patches within a certain radius of that location. It then uses the classifier to determine if the object is present in any of those image patches collected and updates the location of the tracker using the greedy strategy.

2.1.2.3 TLD Tracker

In this algorithm, the object selected by the user is tracked using a short term tracker based on the lucas Kanade method [9]. The online model is then created by analyzing the trajectory in the feature space. The model is then pruned from samples that are wrong.

The objective of creating the online model is to represent a memory of the system and to generate the object detector which is constantly updated and evaluated. Using the online model, the object detector return a set of bounding boxes for every frame in the video sequence which represents an alternative suggestions to the result of the tracker.

The tracker result is compared to these set of bounding boxes returned by the object detector. A confidence score is then calculated to decide if the object is visibe or not.

2.1.2.4 GOTURN Tracker

GOTURN is a deep learning based tracking algorithm [6]. Unlike most of the algorithms which train in an online manner to learn the appearance of the object of interest, GOTURN learns the motion of the object in an offline manner. It is trained on multiple video sequences and does not require to be trained during runtime.

GOTURN has been trained on a pair of cropped images. The previous frame is the cropped image where the object is centered and the bounding box has been provided by the user, and the current frame is the one where the algorithm crops the image to double the size of the bounding box provided.

These two images are then sent to through a bank of convolution layers which comprise of the first five convolution layers of the CaffeNet architecture. The output if these convolution layers are then concatenated into a 4096 length vector which is then sent as an input to a 3 fully connected layers. The last fully connected layer is connected to an output layer which returns 4 nodes containing the bottom and top points of the bounding box.

2.1.2.5 MOSSE Tracker

Mosse algorithm is designed to produce ASEF [10] like filters from fewer training images. It needs training images and training outputs where the training outputs are generated from the ground truth such that it has a 2D Gaussian shaped peak centered on the target in the training image. Training is performed in a Fourier domain which has an advantage of having a simple element -wise relationship between the input and output. [4]

MOSSE finds a filter that minimizes the SSE between the actual and desired output of the convolution to map the training images to their output. In this optimization every training output is customized unlike most other algorithms where it is assumed that the target is always centered in the training images and their outputs.

2.1.2.6 CSRT Tracker

In DCF-CSR we use spatial reliability map for adjusting the filter support to the part of the object which is suitable for tracking. This helps overcome the problem of circular shift and the limitations related to the rectangular space assumptions. This is done by using the output of a graph labeling problem which is solved for each of the frames [5].

The second feature is the channel reliability which is estimated using the properties of the constrained least square solution to filter design. The channel reliability scores are used for the weights of the per-channel filter responses in localization [5].

2.2 Classification Neural Network

Image recognition has boomed over the past decade due to the desire to move towards automation and the many services that become possible with it's use. To this end, CNN have been able to deliver good and consistent results. This has been made possible due to the availability of hardware with high computational capacity and the annotated data sets that are comprehensive [11, 12].

However, the performance of these CNNs is highly dependant on the quality and the

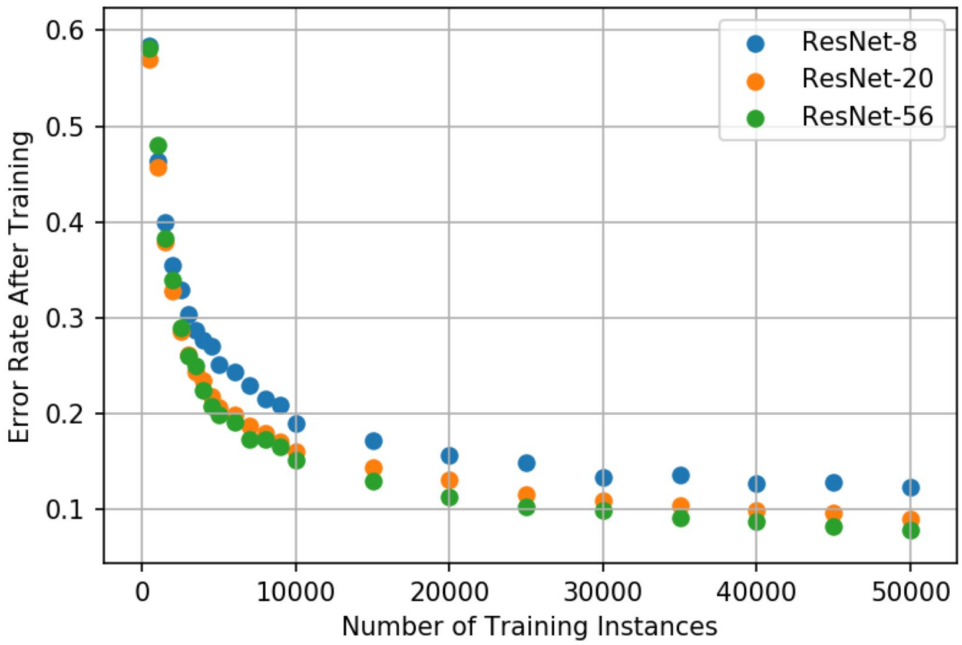


Figure 2.1: Error vs Size of training data set[13]

size of the data sets that are used to train them [13]. Figure 2.1 shows the Error rate vs Size of training dataset. As can be seen, the error rate reduces exponentially as the number of the training images increase. Therefore, it is important to choose the right set of data and the right size of data for the right application to optimize the performance and the computation of the CNN.

2.2.1 Retinanet

Retinanet [14] is based on a one stage object detector that matches the accuracy of more complex two stage detectors such as FPN [15] or Mask R-CNN [16] variants of faster R-CNN[17].

Retinanet is composed of an off-the-shelf convolutional backbone network which is responsible for the computation of a convolutional feature map over the input image

and two task-specific subnetworks namely classification subnet and box regression subnet.

The classification subnet is responsible for the classification of the objects from the backbone's output and the box regression subnet is responsible for the convolutional bounding box regression.

The backbone of the retinanet is adopted by the FPN described in [15]. This is built on top of the RestNet architechture in [18]. [14] shows the architecture of the RetinaNet detector.

Chapter 3

Software Design and Implementation

3.1 UI Design

The software is designed to facilitate in the process of labelling and annotating images for data set generation. The aim is to build a user friendly UI for the proposed flow of algorithm in Figure 3.5 which would enable the user to select multiple objects of interest and annotate them subsequently in following frames without having to select the ROI again. The use of CNN further reduces the need to select objects which the CNN can detect.

3.1.1 Main Window

The main window consists of a large image display pane which is opened once a media input has been selected by the file browser display.

3.1.2 Image display pane

The user interacts for the most amount of time with the image display pane. It is designed to select, display and edit the bounding boxes as necessary. This pane is designed to perform all the tasks necessary without having to move to another pane. The image display pane covers the entire window and the user is allowed to interact with the images using the mouse to drag and draw bounding boxes and use keystrokes to give commands such as next frame, delete bounding box or entering the label name.

3.2 Implementation

This section describes the method of implementation and integration of the UI with the algorithm flow of the proposed software.

The UI is designed in python using the OpenCV library. The UI flow of the software in Figure 3.1 is implemented using the algorithm flow discussed in 3.2.4. In this section, each step of the UI's flow is described along with its code.

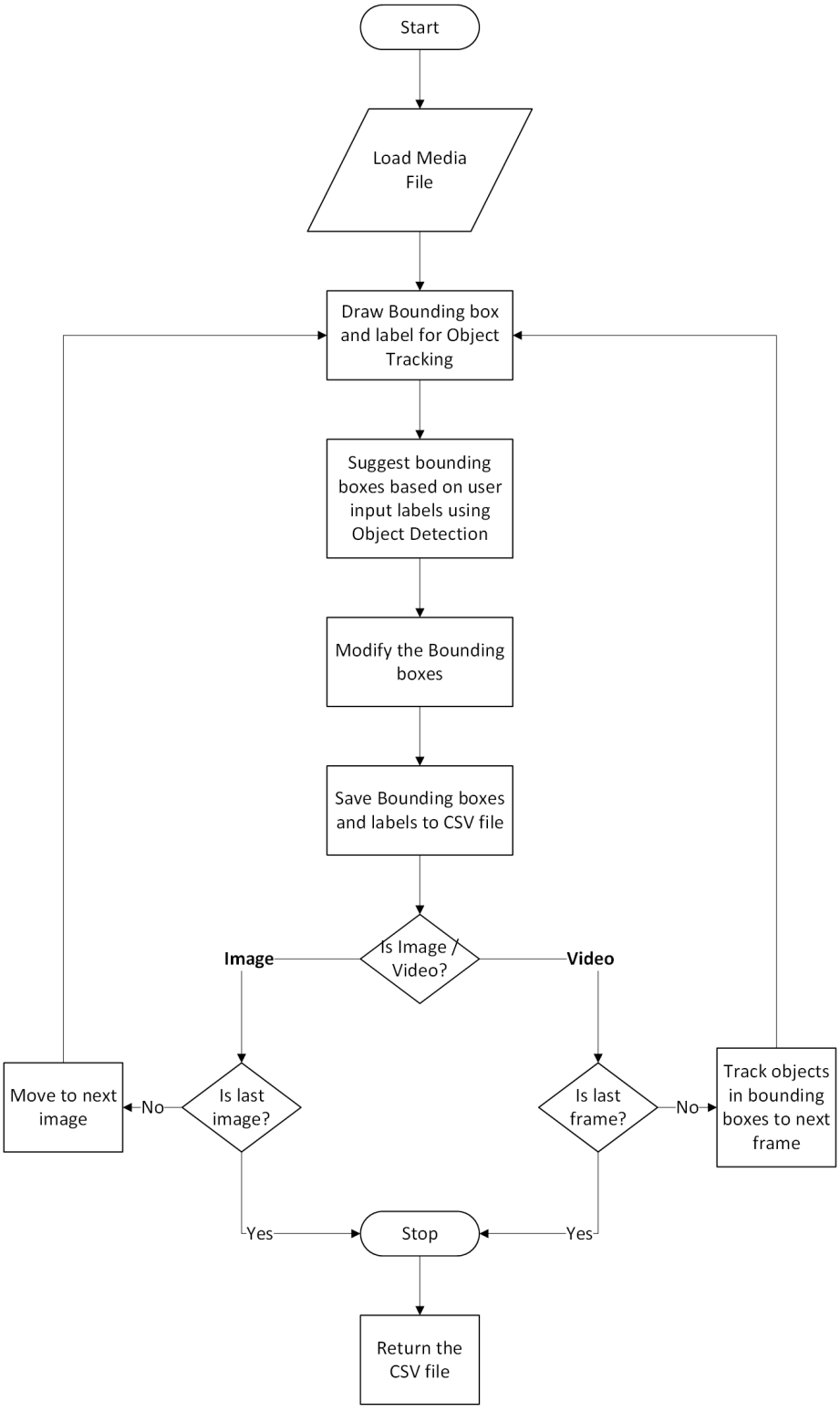


Figure 3.1: UI flow of the software

3.2.1 Load media File

The first step in the process of data generation is to load the files to be used. There are two types of files that can be used, Images and Videos. The user can select if they are opening a video or an image from the file menu on the menu bar.

Upon selecting the type of file the user would like to load, they are given a file browser window to select the directory in case of an Image, and for a video, the user is required to select the video file to be used.

3.2.2 Select Bounding Boxes for Object Tracking

After specifying the file directory, the first image/frame is displayed on the image display pane. The user then needs to follow the following steps to select the objects of interest for tracking.

† Draw bounding box by dragging the mouse from one corner vertex to the opposite corner vertex of the desired box.

† As soon as the user releases the mouse button, they are prompted to enter the label of the object.

† After entering the label name, the user needs to press enter to complete annotating the object.

† The user can repeat the above process to select multiple bounding boxes within the same image or frame by entering any value other than 'q' using the keyboard.

† After selecting all the bounding boxes the user needs to enter 'q' on the keyboard to start tracking the objects.

For our application, we require a tracking algorithm that has a higher accuracy in terms of tracking as well as reliably reporting the failure of the tracking. In terms of accuracy, CSRT Tracker is more accurate than all the other tracking algorithms mentioned in 2.1.2.

Also, To create a diverse data set, we not only need complete images of the objects of interest, but we also need images where only a part of the object is visible. While Boosting does not handle occlusion well, MIL and KCF Tracker do not recover from full occlusion where as CSRT handles occlusion very well.

Computation time and fps throughput are also some of the features to be considered while considering an algorithm for an application. Although, MOOSE is very fast and KCF has a faster FPS throughput, in our application we will not be processing real time data and hence can accommodate lower FPS throughput.

Due to the better accuracy and better handling of occlusions in CSRT, we prefer CSRT over MOOSE.

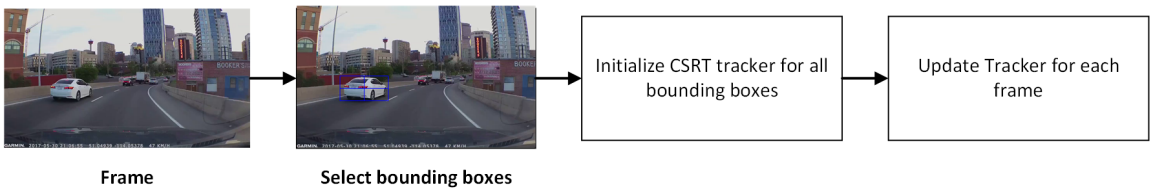


Figure 3.2: Object Tracking Algorithm

To track an object of interest, the CSRT tracker requires a bounding box enclosing the object. This bounding box is then used to initialize the tracker. This tracker is

then used to update the location of the object in the subsequent frames. Figure 3.2 shows the flow of the algorithm for object tracking.

```
#Select the bounding boxes
Select_bounding_box(frame)

#Creating a multitracker to track all the objects ↩
   specified in the bounding boxes above
multiTracker = cv2.MultiTracker_create()
    for select_box in bounding_boxes:
        multiTracker.add(cv2.TrackerCSRT_create(),frame,↩
           select_box)
```

3.2.3 Bounding box suggestions using Object Detection

The RetinaNet Detector described in 2.2.1 is used as the object detection algorithm as it shows a considerable amount of increase in accuracy compared to some of the existing detection algorithms.The comparision between RetinaNet and other one-stage and two-stage detectors is shown in [14].

After the first frame/image is annotated, the algorithm moves to the next frame/image. However, since the user has already listed some labels of interest, the CNN analyses the current frame/image to suggest bounding boxes for objects from the labels already created by the user.

This is achieved by first training the RetinaNet model as shown in Figure 3.3 on a subset of COCO database containing cars, trucks, buses, motorcycles, bicycles, traffic lights, people and road signs, as this software was designed for a specific application in the field of object detection for vehicles to begin with. Although, this

can be used for other applications suited to the user by training the model with the entire COCO data set. ResNet50 was used as the backbone for the training model.

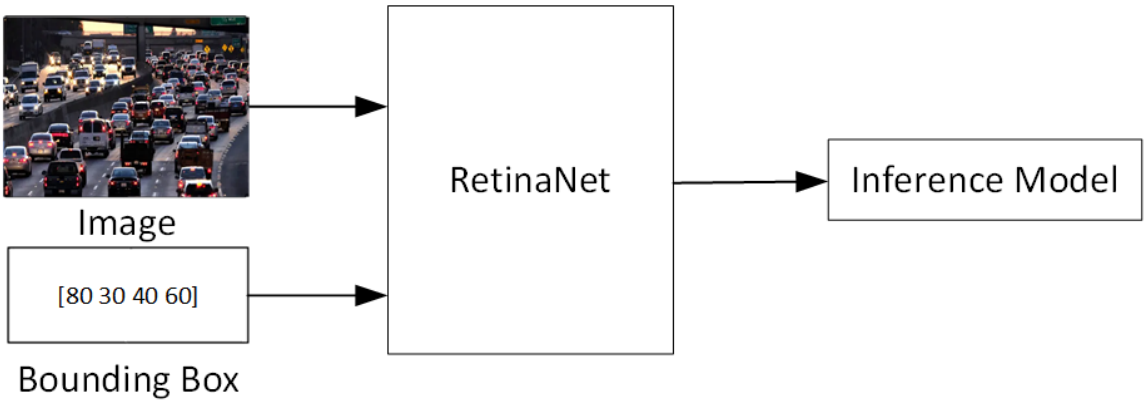


Figure 3.3: Training the CNN [19]

After the training has been complete, the training model is converted to an inference model that can be used for performing the object detection as shown in Figure 3.4.

The inference model returns the coordinates of the bounding box, the detected label id for the object enclosed and a confidence score for the detected label as shown in Figure 3.4. This information can be then visualized by using OpenCV library.

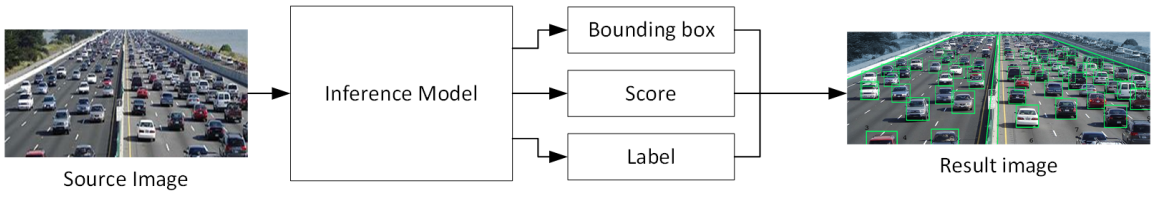


Figure 3.4: Object Detection with the Inference Model [20] [21]

```
def cnn_keras(detect_frame,out_queue):
    # preprocess image for network
    image = cv2.cvtColor(detect_frame, cv2.COLOR_RGB2BGR↩
       )
```

```
image = preprocess_image(image)
image, scale = resize_image(image)

# process image
start = time.time()
cnn_boxes, cnn_scores, cnn_labels = model.↩
   predict_on_batch(np.expand_dims(image, axis=0))
print("processing time: ", time.time() - start)

# correct for image scale
cnn_boxes /= scale

out_queue.put([cnn_boxes, cnn_scores, cnn_labels])
```

3.2.4 Integration of Object Detection and Object Tracking

The results from both the object tracker module in Figure 3.2 and the object detection module in Figure 3.4 are compared to see if there is any redundancy. If the object tracker and the object detector give two overlapping bounding boxes, then we use the Intersection Over Union (IOU) calculation to decide whether to combine both the boxes or to keep them as separate boxes.

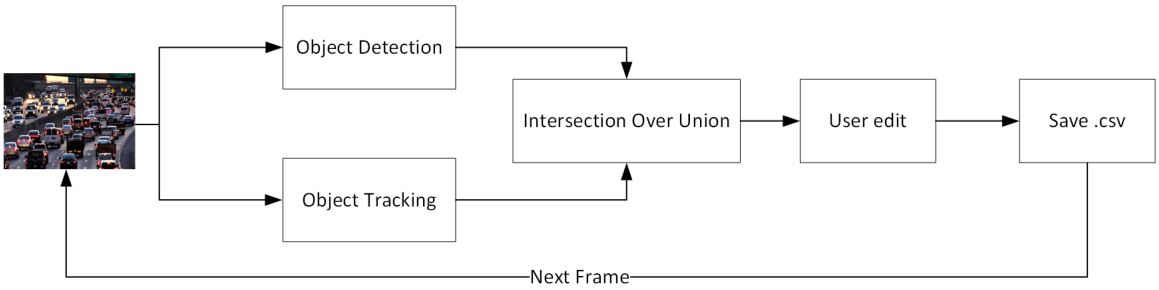


Figure 3.5: Algorithm flow of the software [19]

After computing both the modules, an IOU is computed between all the bounding boxes from the RetinaNet with the ones in object tracking. Any two bounding boxes with IOU greater than 0.75 is considered as the same object and is removed from the RetianNet set of bounding boxes. Then we check the scores of the bounding box and the ones with a score less than 0.5 are also eliminated.

```
def IOU(box_cnn,box_object):
    int1_x = max(box_cnn[0],box_object[0])
    int1_y = max(box_cnn[1],box_object[1])
    int2_x = min(box_cnn[2],(box_object[0]+box_object[2]))
    int2_y = min(box_cnn[3],(box_object[1]+box_object[3]))

    intersect_area = max(0, int2_x - int1_x + 1) * max(0, int2_y - int1_y + 1)

    area_box_cnn = (box_cnn[2] - box_cnn[0] + 1) * (box_cnn[3] - box_cnn[1] + 1)
    area_box_object = box_object[2] * box_object[3]

    iou_area = intersect_area/float(area_box_object + area_box_cnn - intersect_area)
    return iou_area
```

Then we make our final comparison of the class labels with the labels annotated by the user for object tracking and only the labels that have been annotated by the user are provided as suggestions for user convenience.

3.2.5 User Modifications

The user is then presented with the suggested bounding boxes with their labels in the current frame/image. The user can then follow the following steps to edit/delete any of the bounding boxes.

† To delete a bounding box, select the box of interest by double clicking inside the box and enter 'w' using the keyboard.

```
if user_input == ord('w') and active:
        bounding_boxes.remove(bounding_boxes[↩
           box_number])
        bounding_boxes_color.remove(↩
           bounding_boxes_color[box_number])
        reset()
        if(len(bounding_boxes) == 0):
            flag=0
        else:
            edited = True
        active = False
```

† To move a bounding box, double click within the box of interest, then move the box to the desired location and double click to confirm the position and deselect the bounding box.

† To edit the shape of a bounding box, double click on any of the 4 vertices or the center of the 4 edges. Then, move the selected edge or vertex to its desired location. Double click once again to confirm the location and deselect the bounding box [22].

```
def mouseMove(x,y,param):
    global hold,TL,TM,TR,LM,RM,BL,BR,BM,↩
       bounding_boxes,box_number

    if hold:
        print("moving")
        new_x = x - anchorx
        new_y = y - anchory
        new_wid = bounding_boxes[box_number][2]
        new_hig = bounding_boxes[box_number][3]
        createObjectBox(new_x,new_y,new_wid,new_hig)
        RedrawRect(param)
        return
    # endif

    if TL:
        new_x = x
        new_y = y
        new_wid = (bounding_boxes[box_number][0] + ↩
           bounding_boxes[box_number][2]) - x
        new_hig = (bounding_boxes[box_number][1] + ↩
           bounding_boxes[box_number][3]) - y
        createObjectBox(new_x,new_y,new_wid,new_hig)
        RedrawRect(param)
        return
    # endif
    if BR:
        new_x = bounding_boxes[box_number][0]
        new_y = bounding_boxes[box_number][1]
        new_wid = x - bounding_boxes[box_number][0]
        new_hig = y - bounding_boxes[box_number][1]
        createObjectBox(new_x,new_y,new_wid,new_hig)
        RedrawRect(param)
        return
```

```
# endif
if TR:
    new_x = bounding_boxes[box_number][0]
    new_y = y
    new_wid = x - bounding_boxes[box_number][0]
    new_hig = (bounding_boxes[box_number][1] + ↩
       bounding_boxes[box_number][3]) - y
    createObjectBox(new_x,new_y,new_wid,new_hig)
    RedrawRect(param)
    return
# endif
if BL:
    new_x = x
    new_y = bounding_boxes[box_number][1]
    new_wid = (bounding_boxes[box_number][0] + ↩
       bounding_boxes[box_number][2]) - x
    new_hig = y - bounding_boxes[box_number][1]
    createObjectBox(new_x,new_y,new_wid,new_hig)
    RedrawRect(param)
    return
# endif

if TM:
    new_x = bounding_boxes[box_number][0]
    new_y = y
    new_wid = bounding_boxes[box_number][2]
    new_hig = (bounding_boxes[box_number][1] + ↩
       bounding_boxes[box_number][3]) - y
    createObjectBox(new_x,new_y,new_wid,new_hig)
    RedrawRect(param)
    return
# endif
if BM:
    new_x = bounding_boxes[box_number][0]
```

```
            new_y = bounding_boxes[box_number][1]
            new_wid = bounding_boxes[box_number][2]
            new_hig = y - bounding_boxes[box_number][1]
            createObjectBox(new_x,new_y,new_wid,new_hig)
            RedrawRect(param)
            return
        # endif
        if LM:
            new_x = x
            new_y = bounding_boxes[box_number][1]
            new_wid = (bounding_boxes[box_number][0] + ↩
               bounding_boxes[box_number][2]) - x
            new_hig = bounding_boxes[box_number][3]
            createObjectBox(new_x,new_y,new_wid,new_hig)
            RedrawRect(param)
            return
        # endif
        if RM:
            new_x = bounding_boxes[box_number][0]
            new_y = bounding_boxes[box_number][1]
            new_wid = x - bounding_boxes[box_number][0]
            new_hig = bounding_boxes[box_number][3]
            createObjectBox(new_x,new_y,new_wid,new_hig)
            RedrawRect(param)
            return
        # endif
```

† Press 'return' to complete and save the annotation and move to the next frame/image.

The user can also draw any new bounding boxes of interest by pressing the 'f' key on the keyboard and following the steps mentioned in the previous section. 3.2.2

3.2.6 Save Annotations

Video Name	Frame Number	Bounding Box	Label
Testing.mp4	1	[80,50,20,30]	Car
Testing.mp4	1	[10,40,20,40]	Bike
Testing.mp4	1	[30,60,5,4]	Person
Testing.mp4	2	[12,38,20,40]	Bike

Table 3.1
CSV File structure

While the user selects and annotates the objects of interest, the program saves the bounding boxes and its labels into a CSV file. The file is updated regularly after each frame/image. The structure of the CSV file is shown in the table 3.1.

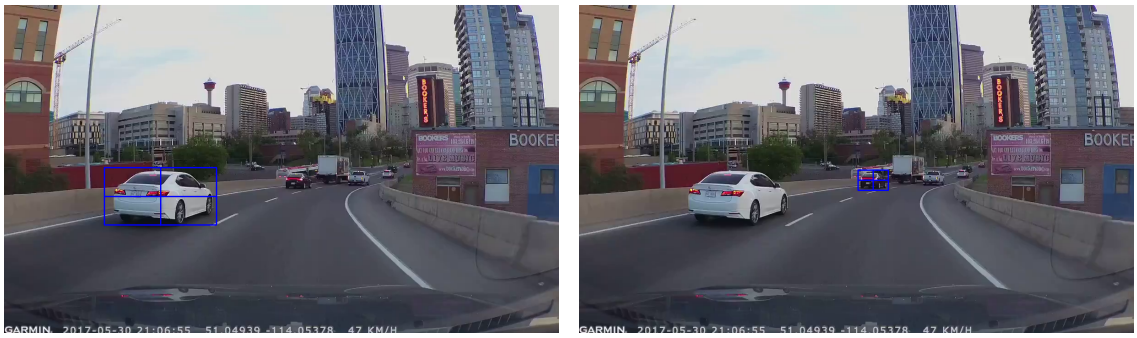
```
with open('Object_Detection.csv', 'a') as csvFile:
    writer = csv.writer(csvFile)
    for i, box in enumerate(bounding_boxes):
        row = [vid_path, frame_number,box,↩
           bounding_box_labels[i]]
        writer.writerow(row)

csvFile.close()
```

Chapter 4

Results and Discussion

Figure 4.1 shows how ROI can be selected in an image or a frame. Figure 4.1(a) shows the first object that was selected in the frame and Figure 4.1(b) shows the second object that was selected in the frame.



(a) Select 1st object

(b) Select 2nd object

Figure 4.1: Select objects [23]

Figure 4.2 shows the objects detected in the frame by the CNN along with the two objects selected by the user. The objects in pink and lime green (the colors are assigned randomly) are the objects selected by the user and are being tracked. The

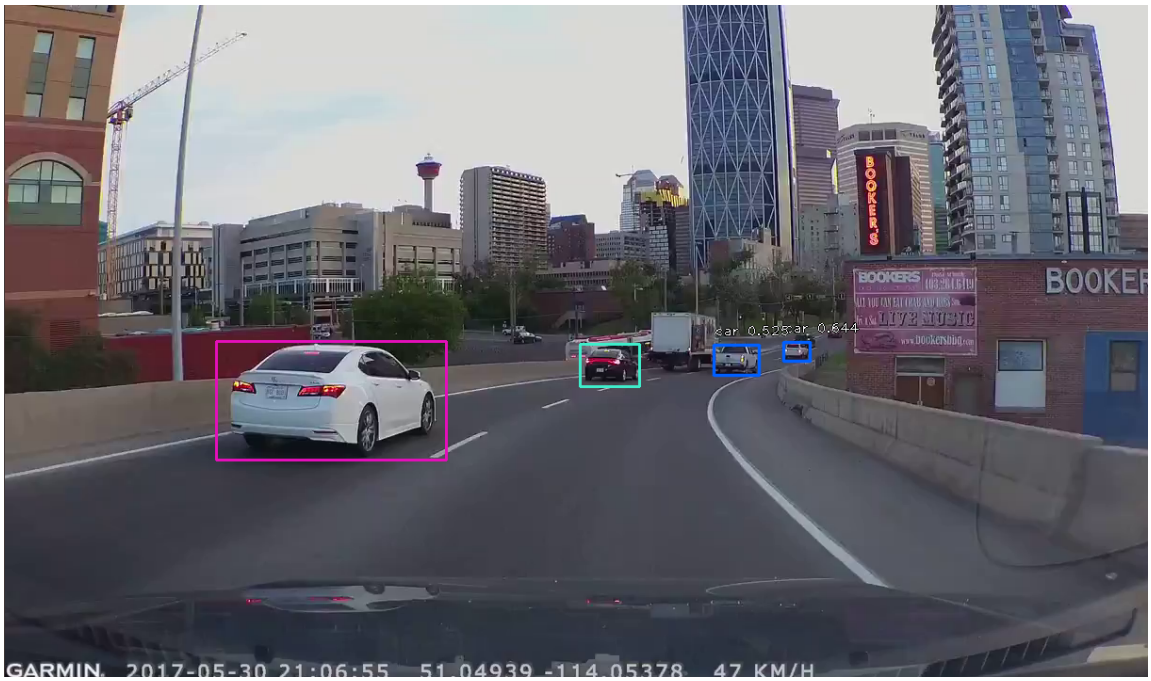
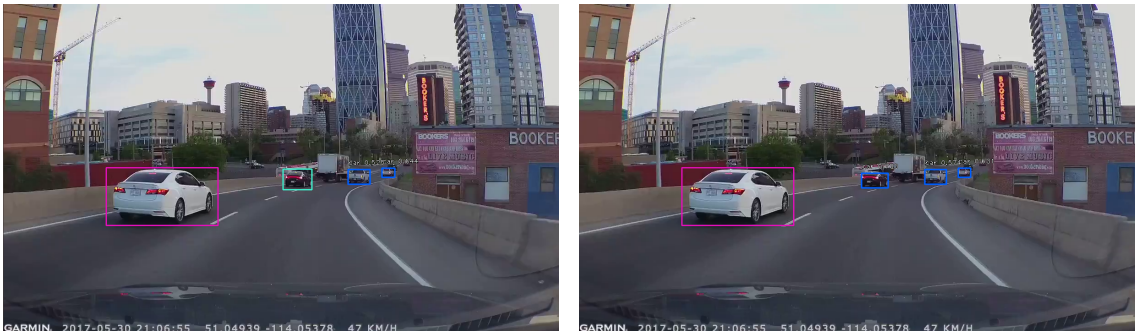


Figure 4.2: Object Detection [23]

objects in the blue boxes are the ones that were detected by the CNN. These also have the label of detection and the confidence score on top of the boxes.



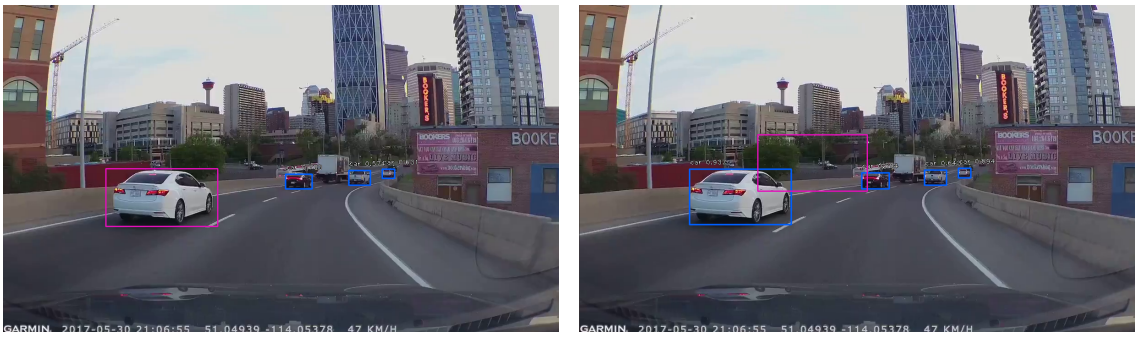
(a) Before Deletion (b) After Deletion

Figure 4.3: Deleting bounding boxes [23]

If the user chooses to use these boxes they can move to the next frame. However, if they would like to not use some or all of the boxes detected in the frame, they may remove them, as shown in Figure 4.3. In this figure, the bounding box in lime

green color in Figure 4.3(a) is deleted. When the user moves to the next frame after deleting the bounding box, the CNN once again detects the object and displays the label and the confidence score along with it.

Figure 4.4 shows an image of a bounding box that was moved. As we can see that the pink bounding box in Figure 4.4(a) has been moved to a different location in Figure 4.4(b). When the user moves to the next frame, the moved bounding box starts tracking the object within it while the car that was earlier enclosed in the bounding box in Figure 4.4(a) is now detected by the CNN.



(a) Before Moving (b) After Moving

Figure 4.4: Moving a bounding box [23]

4.1 Software performance evaluation

The software that has been created needs to be evaluated to analyze the improvement in the effort of the user. The following assumptions are made during this analysis.

† The quality of the media input is acceptable, meaning the objects of interest are clearly distinguishable.

† The media has multiple instances of the objects of interest.

† The change in visual features from frame to frame is not very high.

For this, we performed an evaluation where the user defined a bounding box for the object of interest in each frame. An object tracker was then initialized using the bounding box in the first frame. The goal was to compute the percentage of overlap of the bounding boxes defined by the user and the ones returned by the tracker. An 80% overlap was considered to be an acceptable suggestion from the tracking algorithm. With this criteria, we were able to see an acceptable performance from the tracker for 4 frames on an average. The performance of the object detection algorithm is given in [14].

4.1.1 Case 1

The most conservative case would be when the number of objects of interest is 1 in each frame. Considering this case, let us assume a video at 60 fps. The user has to draw a bounding box around the object of interest every 5th frame as the tracker can track the object for 4 frames. Hence,

Number of frames in 1 second = 60
Number of frames the tracker can track an object = 4
Number of frames the object is annotated = 1 (user defined) + 4 (tracker)
Number of frames the user has to define the bounding box in 1 second = 60/5 = 12
Number of times the user has to define the bounding box without the tracker in 1 second = 60
Reduction in effort = 60/12 = 5

The effort of the user in this case is reduced by 5 times which is 80%.

4.1.2 Case 2

Considering a case where the number of instances of the object of interest is more than 1. In such a case, once the user has labelled the first instance of the object of interest, the object detector will detect the rest of the instances. Hence, the efforts of the user for defining 1 instance remains the same as in case 1. However, the effort for defining the rest of instances reduces proportionally to the number of instances.

Therefore, ideally the software should be able to reduce the user's effort by at least 80%.

Chapter 5

Conclusion and Future Work

A software has been developed to facilitate in the process of image data set generation and annotation. This has been achieved with the help of deep learning algorithms and Visual Feature Tracking. The use of RetinaNet CNN and CSRT Tracker in this process has helped reduce the time and effort required in generating an application specific data set.

(a) Frame 1 (b) Frame 2

Figure 5.1: Variation in consecutive video frames

Currently, the software is saving the annotations at the end of every frame. This can lead to a lot of redundancy in the images saved as the variation of the data within a

video frame is not that significant in all the cases. Figure 5.1 shows a comparison of two consecutive frames in a video.

Hence, it would be beneficial to remove the redundancy in the video files by compressing them based on the variation of the image from frame to frame. This would lead to a compressed video where the variation from frame to frame is significant and can lead to distinct images as opposed to similar images as shown in Figure 5.1.

References

[1] Redmon, J.; Divvala, S.; Girshick, R.; Farhadi, A. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[2] Krizhevsky, A.; Sutskever, I.; Hinton, G. E. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] Gauglitz, S.; Höllerer, T.; Turk, M. *International journal of computer vision* **2011**, *94*(3), 335.

[4] Bolme, D. S.; Beveridge, J. R.; Draper, B. A.; Lui, Y. M. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2544–2550. IEEE, 2010.

[5] Lukezic, A.; Vojir, T.; Cehovin Zajc, L.; Matas, J.; Kristan, M. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6309–6318, 2017.

[6] Mallik, S. *Object Tracking using OpenCV (C++/Python);* Learn OpenCV: https://www.learnopencv.com/object-tracking-using-opencv-cpp-python/, 2017.

[7] Grabner, H.; Grabner, M.; Bischof, H. In *Bmvc*, Vol. 1, page 6, 2006.

[8] Babenko, B.; Yang, M.-H.; Belongie, S. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 983–990. IEEE, 2009.

[9] Kalal, Z.; Matas, J.; Mikolajczyk, K. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1417–1424. IEEE, 2009.

[10] Bolme, D. S.; Draper, B. A.; Beveridge, J. R. *2009 IEEE Conference on Computer Vision and Pattern Recognition* **2009**.

[11] Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; Fei-Fei, L. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[12] Lin, T.-Y.; Maire, M.; Belongie, S.; Hays, J.; Perona, P.; Ramanan, D.; Dollár, P.; Zitnick, C. L. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[13] Soleyman, S. *Effect of Dataset Size on Image Classification Accuracy;* Word Press: http://seansoleyman.com/effect-of-dataset-size-on-image-classification-accuracy/, 2018.

[14] Lin, T.-Y.; Goyal, P.; Girshick, R.; He, K.; Dollár, P. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[15] Lin, T.-Y.; Dollar, P.; Girshick, R.; He, K.; Hariharan, B.; Belongie, S. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* **2017**.

[16] He, K.; Gkioxari, G.; Dollar, P.; Girshick, R. *2017 IEEE International Conference on Computer Vision (ICCV)* **2017**.

[17] Ren, S.; He, K.; Girshick, R.; Sun, J. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **2017**, *39*(6), 11371149.

[18] He, K.; Zhang, X.; Ren, S.; Sun, J. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] Vogel, R. *Traffic Image;* NewYork Times: https://www.nytimes.com/2019/01/21/upshot/stuck-and-stressed-the-health-costs-of-traffic.html, 2019.

[20] Vision, T. *Traffic bounding box image;* Traffic Vision: http://www.trafficvision.com/, 2015.

[21] Minesweeper. *Traffic bounding box original image;* Wikipedia: https://en.wikipedia.org/wiki/Traffic, 2015.

[22] Chavan, A. *opencvdragrect;* GitHub: https://github.com/arccoder/opencvdragrect, 2016.

[23] cam videos, L. V. D. *Testing Video;* Youtube: https://www.youtube.com/watch?v=8aFkNRrj8n8—&t=5s, 2015.

[24] Sharif Razavian, A.; Azizpour, H.; Sullivan, J.; Carlsson, S. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[25] Shin, H.-C.; Roth, H. R.; Gao, M.; Lu, L.; Xu, Z.; Nogues, I.; Yao, J.; Mollura, D.; Summers, R. M. *IEEE transactions on medical imaging* **2016**, *35*(5), 1285–1298.