Traffic Accident Prediction

Anvesh Mateti and Sarthak Gupte

CSCI 631

Introduction

Accident prediction is the ability to predict if and when an accident will happen. The ability to predict accidents effectively is a much sought after goal for futuristic technologies. In the real world, self-driving cars will require software to prevent accidents long before they have a chance of happening. The objective of this paper is to use computer vision to predict accurately and in a timely manner of a crash occurring between vehicles. Sequential video frames will be provided to be used to create a learning system that will help model an accident prediction framework. These sequential video frames will come about in the form of the dataset provided called the crash-1500 dataset. This dataset contains 1500 accident clips in the .mp4 format, where each clip is composed of 50 frames. Another dataset called the normal dataset will also be provided which contains normal video clips that do not contain accidents. These two datasets will be used to be trained on our model to predict accidents accurately.

To reach the final goal of creating an accurate accident prediction model, we must develop a computer vision algorithm. This requires knowing the dataset provided, building a model, training our model, and finally evaluating the model. In the following sections we will go through how a computer vision algorithm can be implemented to generate an accident prediction framework.

Dataset Description

The dataset consists of two types of the videos.
1. Positive - videos that contain accidents.
2. Negative - videos that do not contain accidents.

The positive video data set was provided on the drive under the folder CRASH-1500. This folder consists of 1500 crash videos. Each video is of approximately around 50 frames and for every video annotation was provided. In annotation for a video an array of 50 is created with 0 and 1. 0 indicates that the frame does not have an accident and 1 indicates that it does have an accident in it.

The negative video data set also provided on the drive under the folder Normal. This folder consists of 3500 videos which does not include accidents in them. It was acquired from the famous BDD-100K dataset.

Methodology

For prediction of accidents in the given video-sets, we used HRNN(Hierarchical Recurrent Neural Network). Before diving into HRNN, some methods were implemented to create input for the model.

The first step performed is object identification of the videos. To perform this, we used a pre-trained python library named “tensornets”. It is an open source model which can be implemented using tensorflow. It provides models which are pre trained on MS COCO dataset. Which is a large-scale object detection, segmentation, and captioning dataset having 90 pre trained objects. Using this first we read the videos, then break down the images into frames and using tensornets, we try to identify the object. If found we draw a box around the object and store the frame as an image.

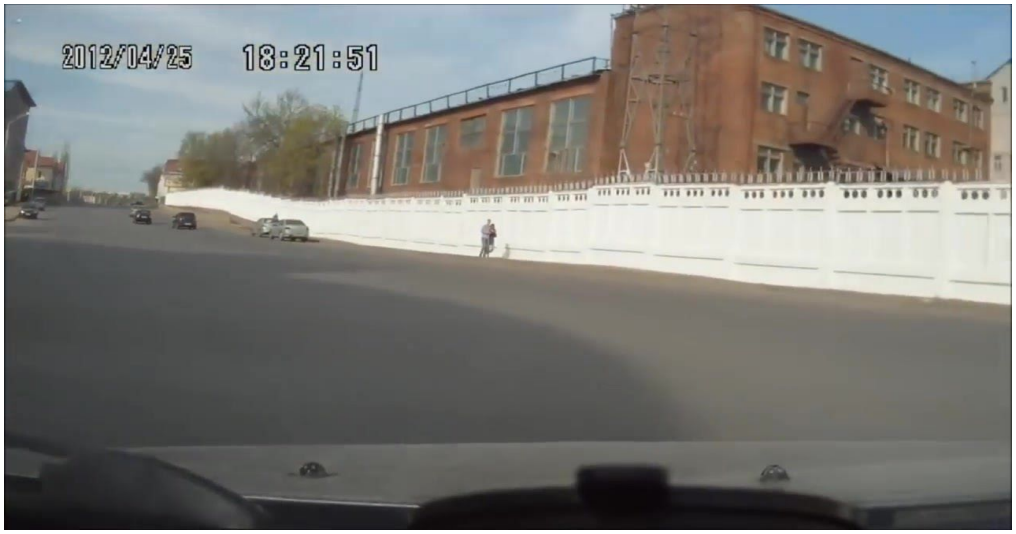

Fig. 1 Before Annotation.

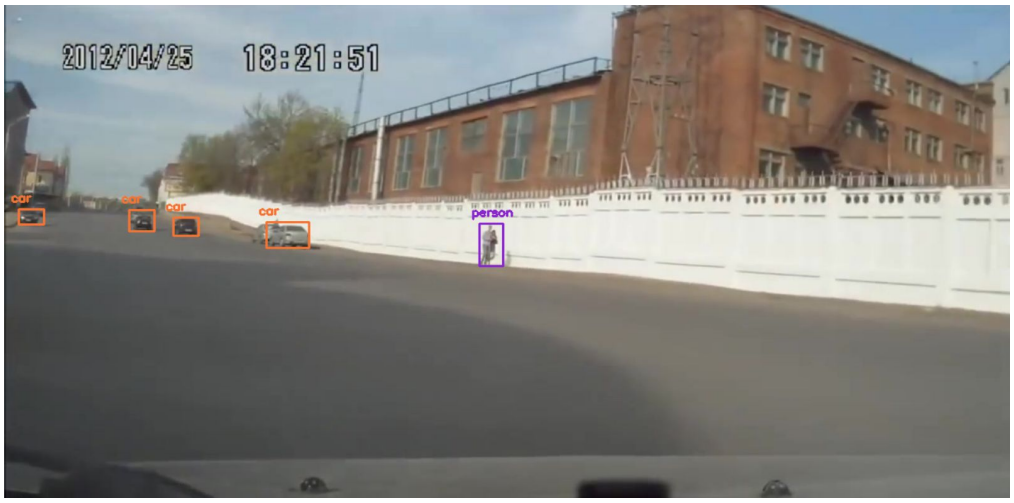

Fig. 2 After Annotation.

After object identification the next step is creating the dataset for the model. For that, we read and stored the name of the files from both the folder(positive and negative). Then merged them together to create a single dataset. The data set was then downsampled. Down sampling means that the size of the data was reduced by reducing features. So that we can load the data into the memory together for training and testing purposes. For downsampling, the image was converted from rgb to grayscale to get rid of one dimension. Otherwise multiple dimensions can lead us to the curse of dimensionality, which occurs because of the sparseness of the data. Next we converted the size of the image to 144x256 such that most of the property is stored but some is lost to make the program run.

Then we created the HRNN model. HRNN is basically a hierarchy of RNN, where one RNN is wrapped in another RNN. It is like layers of RNN. It consists of multiple layers, the first layer is a time distributed layer; this is a wrapper layer used to process input in a sequential manner for RNN. Second layer used is a special type of RNN, that is, LSTM(Long Short Term Memory). It can easily process an entire sequence of data for example a video. LSTM can process the data and keep updating the hidden states in it sequentially.

Implementation

We converted our images to video to fit in our code, to do so we used the code given below.

```
import cv2

import os
```

```
img_array = []

for filename in sorted(os.listdir('/Users/sarthakgupte/Desktop/Intro to CV/Annotation/Team 1/0000')):

  print(filename)

  img = cv2.imread(filename)

  height, width, layers = img.shape

  size = (width, height)

  img_array.append(img)

out = cv2.VideoWriter('/Users/sarthakgupte/Desktop/Intro to CV/Annotation/Team 1/project.avi',
cv2.VideoWriter_fourcc(*'DIVX'), 15, size)

for i in range(len(img_array)):

  out.write(img_array[i])

out.release()
```

For annotation the videos we created a tensornet model. Here model is of the tensenots which is taking input with the help of tensflows. This model helps to read video and annotate the car and person if found. This function also helps to create an updated video with all the annotations.

```
def annotation(path, filename):

  input_m = tf.compat.v1.placeholder(tf.float32, [None, 416, 416, 3])

  model = tensonets.YOLOv3COCO(input_m, nets.Darknet19)
```

For loading annotation data we used load_annotated data(). The function reads the updated video and then downscale the image, such that we can load the data into the memory while training the dataset. It collects all the images into a list and then returns the list. This function also used to label the data. If the path is of the positive dataset we label it as 1 and if the path is of negative dataset we label it as 0.

```
if filename.startswith("updated") and filename.endswith(".mp4"):

  cap = cv2.VideoCapture(os.path.join(path, filename))

  for i in range(50):
```

```
    print(len(frames))

    ret, frame = cap.read()

    frames.append(down_scale(frame))
```

The function down_scale() takes the image in the form of an array and convert it into an image and then resizes it. Here the image is also converted from rgb to grayscale

```
        cv2.imwrite('color_img.jpg', image)

        img = cv2.imread('color_img.jpg')

        img = color.rgb2gray(img)

        tmp = transform.resize(img, (144, 256))

        return tmp
```

.

Dividing training and testing data. For this we used the sklearn package which helped us to split the training and the testing data into 70 and 30% respectively.

```
clip_train, clip_test, label_train, label_test = train_test_split(clips, labels, test_size=0.3, random_state=130)
```

Since the data was too much to handle we had to keep it for two nights to train it and for testing we were getting errors.

To handle this we tried to save the model so that we do not have to re train the model again and again.

```
filepath = 'models.hdf5'

checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=False, mode='max')

callbacks_list = [checkpoint]
```

This is how we manage saving the trained model.

Lesson learned

In this project we learned some key concepts in computer vision.

One important concept is image processing. To process an image, we started by first converting the images from rgb to grayscale to reduce dimension. Next, we resize the images,followed by downscaling the images to further reduce the size of the image to extract useful information from it. Conversion of images to video and video to images. What information can be obtained from just an image.

Another important concept we learned about is developing a model using neural networks. We learned about different types of neural networks. Perceptron is a single layer neural network, it is also considered as a building block of neural networks. We first separate the data set into training and testing data. Then, we can split up the dataset into inputs and targets. Then we create a model, by generating a neural network (HRNN) and adding full connected layers such as softmax layers. This neural network will be trained to develop predicted accuracy of an accident occurring.

References Used -

https://github.com/tensorflow/tensorflow

https://github.com/ankush17100/Accident-Detection-Using-Deep-Learning/blob/master/HRNN_updated.py

https://www.scipy.org/

Lecture Notes and paper.