ENPM673 Project 3

Prannoy Namala (pnamala)

May 2021

Introduction

This project is divided into two sections. The first section is to compute optical flow for a video footage of cars on an highway. The second section is to implement a classifier to which can identify different types of seafood. A brief of process, challenges and results will be explained in this report.

Optical Flow

Process

There are many ways to achieve the optical from a video. For this project, we will be looking at sparse and dense implementations of Lucas Kanade algorithm. In the sparse implementation a fixed number of features are tracked. Inbuilt functions are used to detect the features to track. These features are tracked for all of the frames. The path of the features in the image is drawn by keeping the data of the pixel location and highlighting them on the output video feed.

Now discussing the dense implementation. Using the inbuilt functions, the flow matrix is obtained. This flow matrix is of the size of the frames but has 2 channels. These channels represent the before and after value for each pixel. By subtracting the values in the channel, one can obtain the magnitude of distance travelled by the pixel in both the x and y directions. Now this information is represented in two ways. The first is by building a vector field. This vector field can be obtained getting the magnitude from the flow matrix. The output is shown for a grid with a space of 25. The second method of representation is a color based representation. In this representation, the magnitude value from flow are normalized in the range of [0,255]. Using the normalized values an HSV representation is generated. This is then converted to RGB colorspace. This results in a visualization for each pixel which is more appealing than the vector field. The last part in this section is to filter the background from the video feed. This is done by implementing a background subtraction functionality in opencv. This object when implemented on one frame of the image produces a near binary image. This output is made completely binary and bitwise and operation is used to get the objects and removing the background.

Challenges

For sparse implementation, the new features tracked were not being detected. Since the image is continuously changing, the features need to be updated. However, the features are not updated and the new features are not tracked.

The dense implementation seems to have an issue with computational expense. With just the dense implementation running, my computer got stuck several times while attempting to save the video using opencv functionality. This issue is rooted to memory management in the end and was solved. Adding to that, sometimes while looking at individual frames in the second implementation, the brightness of the color contrast is varying. In the vector field there are some random vectors observed which are not in the direction of the general flow of the whole frame. These can be related to the assumptions of the optical flow algorithms which are difficult to br true in a real world scenario.

The background subtraction implementation output has a lot of noise. The noise can be seen as some part of the background is visible in some frames. This can be reduced by implementing image morphing techniques on the mask image.

Some of the images 1, 2, 3 and **??** frames from the obtained results. The links to output videos are:

Sparse - `https://youtu.be/Cvu1ZFZjRX0`

Dense Colored- `https://youtu.be/9Z0r8uFwrx4`

Dense Vector field - `https://youtu.be/b65vbtwSBYc`

Background Subtraction - `https://youtu.be/F8p_y2UBzNE`

Image Classifier

Process

The first step in the process was to segregate the data into training and validation data. The training and validation need to have labels for the model built to learn from or to test against. This done by coding a segregation program which can split the images in each folder. This program outputs the split dataset which can be used by the model. In addition to segregation,the program also resizes the image into a (224,224) sized image.

After resizing the dataset is ready for running through the model. Keras is used to implement the model. I tried using two models and the performances of them are compared in the results section. The first model is just built using random layers. The second model is the VGG 16 model. Both the models are built and trained against the segregated data. For each model, the model summary helps to show how accurately the robot is learning. For optimizing, Adam optimizer has been implemented with learning rate of $10e^{-}5$ and decay rate of $10e^{-}7$.

Challenges and Observations

As mentioned earlier, the segregation program resizes the images before the dataset is used for training. The model was also trained without resizing. This did not give promising results. By modifying batch size, the model is able to train quicker and gives fractionally better results when compared to a bigger batch size. One key observation is that if the model is not having a higher rate of accuracy changes for the first few epochs, it is not going to give good accuracy. For example, if after first 10 epochs, the accuracy is close to 0.1, it is better to stop the run and try changing some of the parameters. When the same process is applied to groundtruth files, there is an eror which states 'no algorithm found'. After some searching on google, there is no clear solution found to solve this problem.

Results

The graphs for accuracy v epochs and loss v epochs is shown in figures 5, 6, 7 and 8. The hyper parameters for the graph is mentioned in the caption of the figure.

1 References

- Background Subtraction — OpenCV-Python Tutorials 1 documentation. (2021). Retrieved 17 May 2021, from `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html#background-subtraction`
- Rosebrock, A. (2021). Image Masking with OpenCV - PyImageSearch. Retrieved 17 May 2021, from `https://www.pyimagesearch.com/2021/01/19/image-masking-with-opencv/`
- Sharma, V., Matplotlib, D. (2021). Deep Learning Model Data Visualization using Matplotlib — Pluralsight. Retrieved 17 May 2021, from `https://www.pluralsight.com/guides/data-visualization-deep-learning-model-using-matplotlib`
- Step by step VGG16 implementation in Keras for beginners. (2021). Retrieved 17 May 2021, from `https://towardsdatascience.com/step-by-step-vgg16-implementation-`
- O.Ulucan, D.Karakaya, and M.Turkan.(2020) A large-scale dataset for fish segmentation and classification. In Conf. Innovations Intell. Syst. Appli. (ASYU)

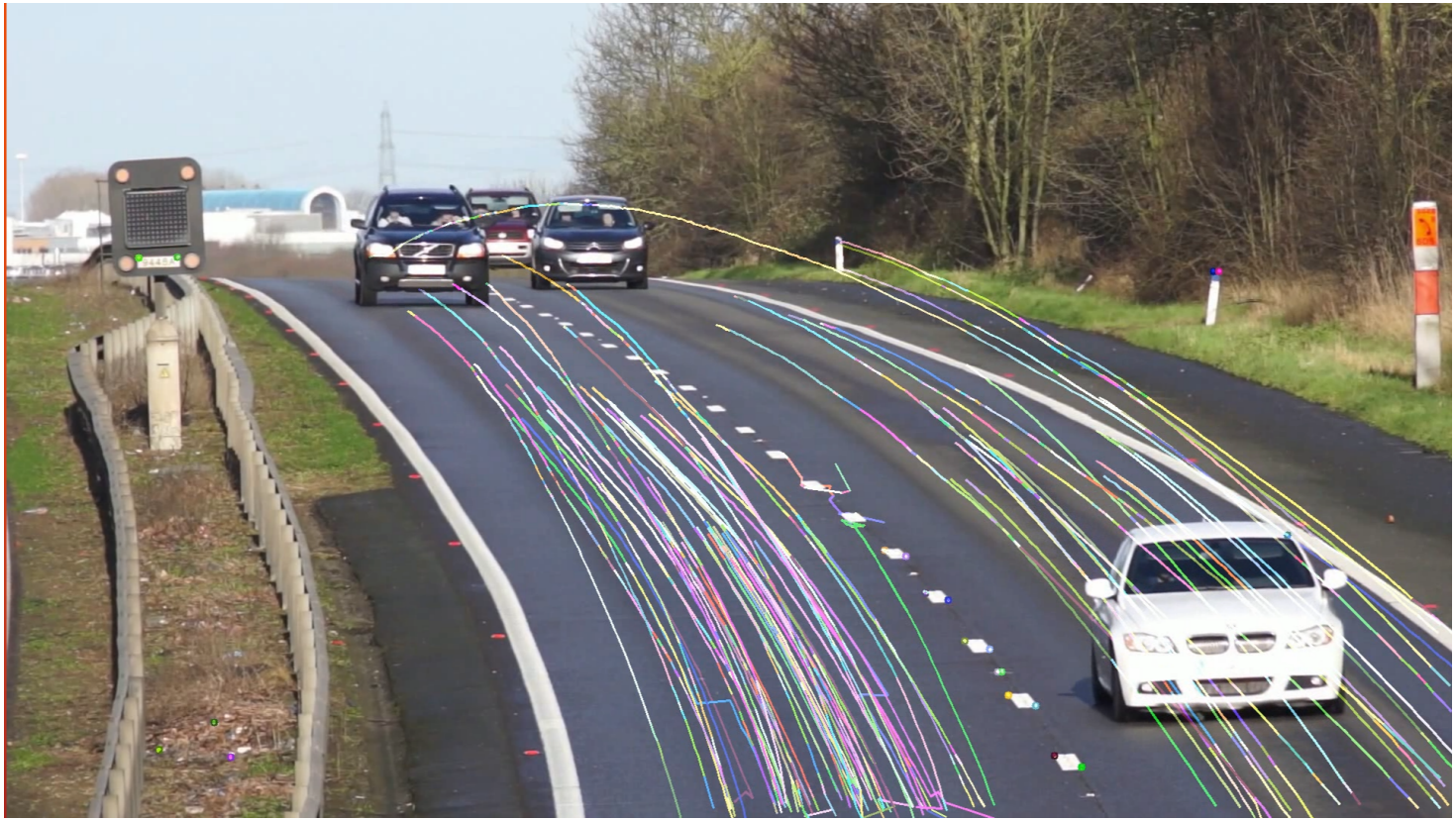

Figure 1: One frame in Sparse implementation

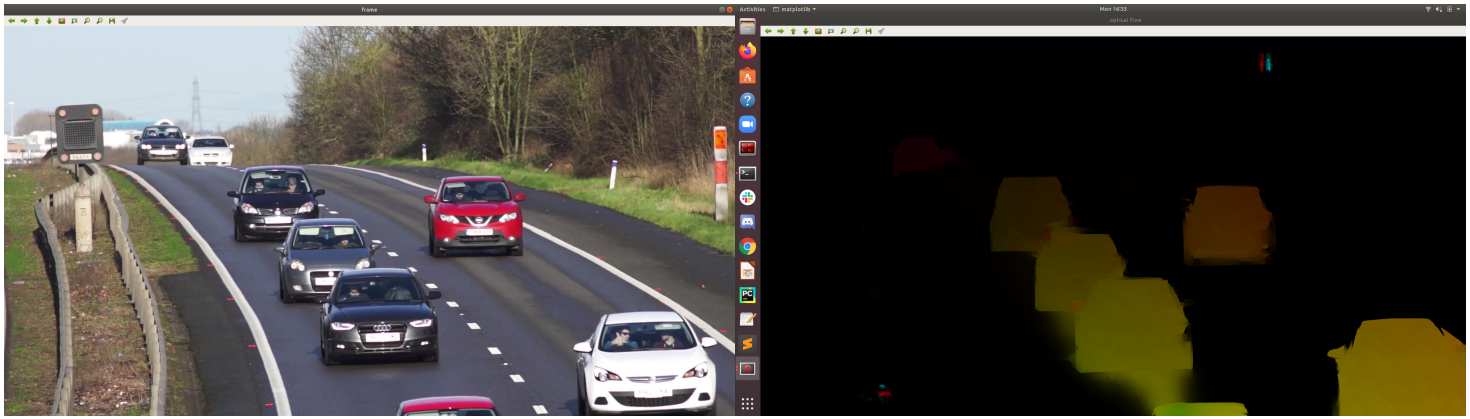

Figure 2: Colored representation of Dense implementation

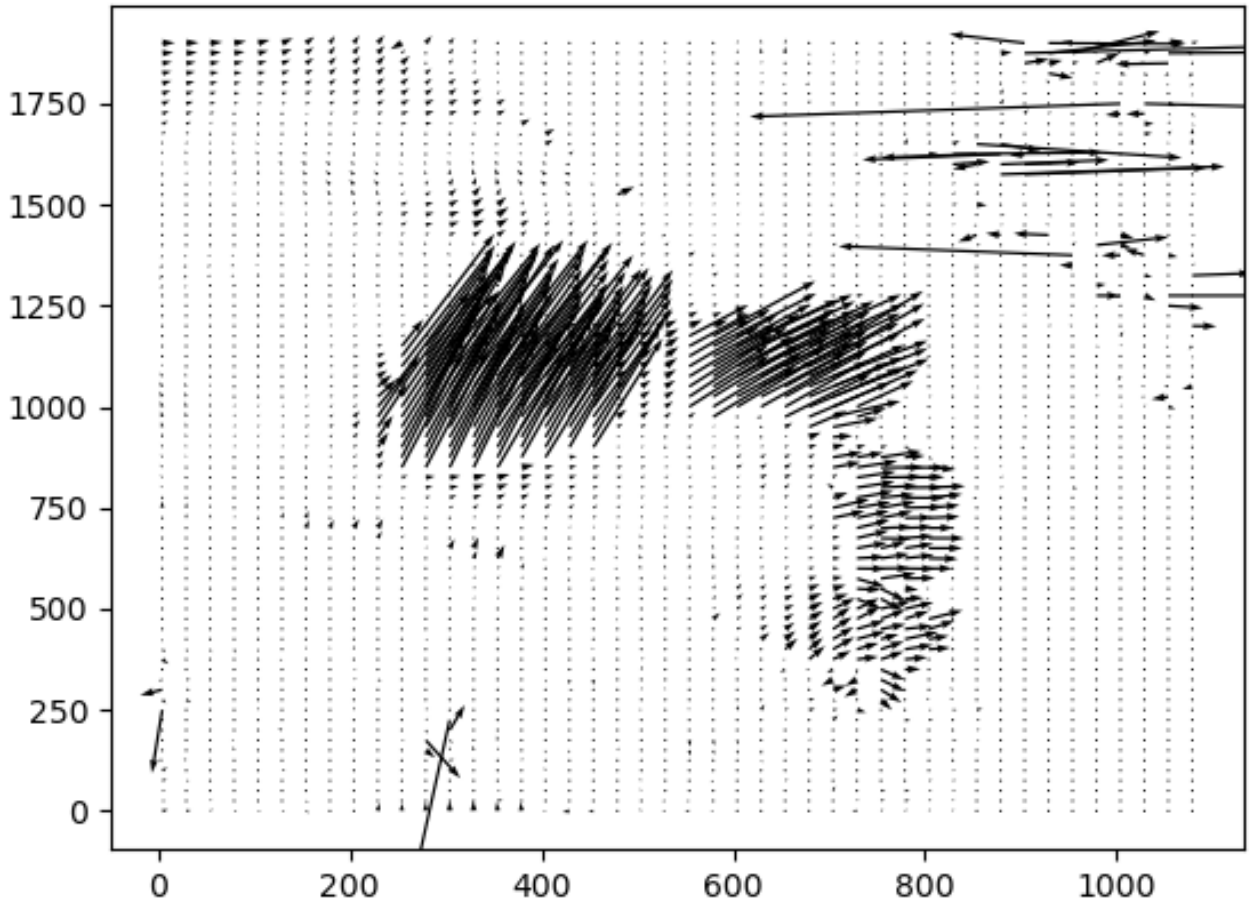

Figure 3: One frame of Vector Field representation

Figure 4: One frame of Background seperation

Figure 5: Custom model with regular data set. Batch size 40. Learn rate is $10e^{-}5$ and decay rate is $10e^{-}7$

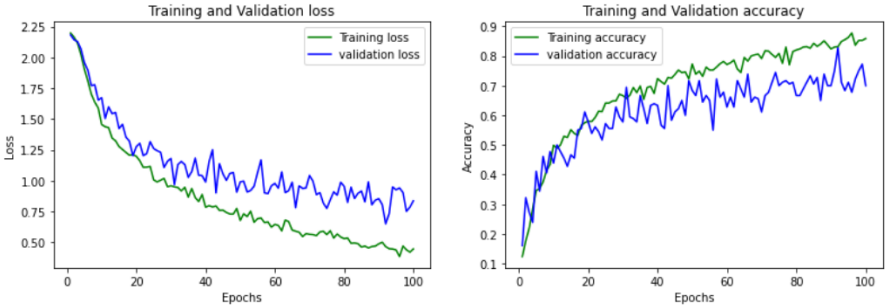

Figure 6: Custom model with regular data set. Batch size 20. Learn rate is $10e^{-}5$ and decay rate is $10e^{-}7$

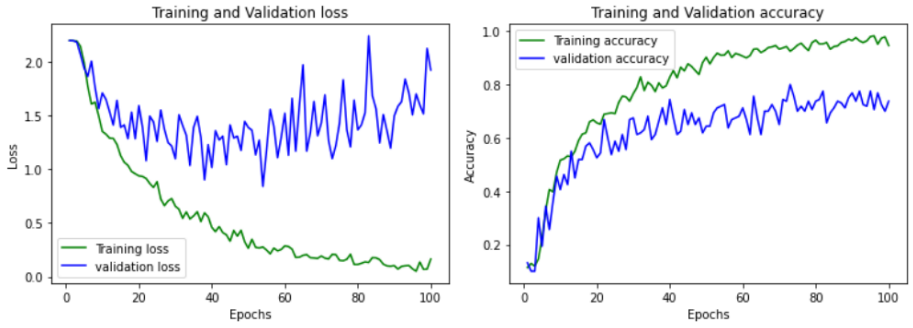

Figure 7: VGG16 with regular data set. Batch size 40. Learn rate is $10e^{-}5$ and decay rate is $10e^{-}7$

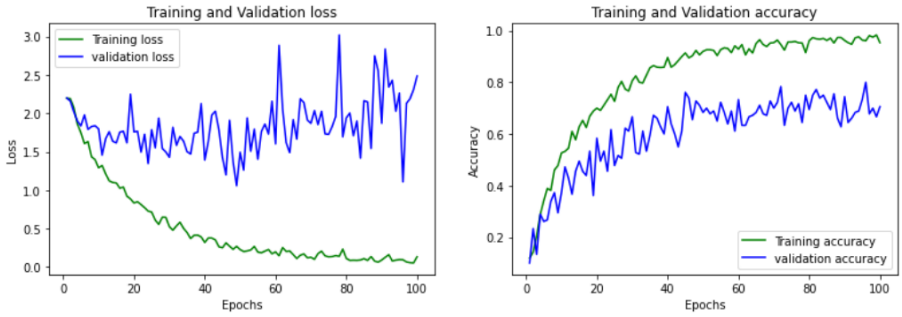

Figure 8: VGG16 with regular data set. Batch size 20. Learn rate is $10e^{-}5$ and decay rate is $10e^{-}7$