Case Studies
Assignment 1: Object Detection and Classification

Khadija Sitabkhan-20236001

Aim:

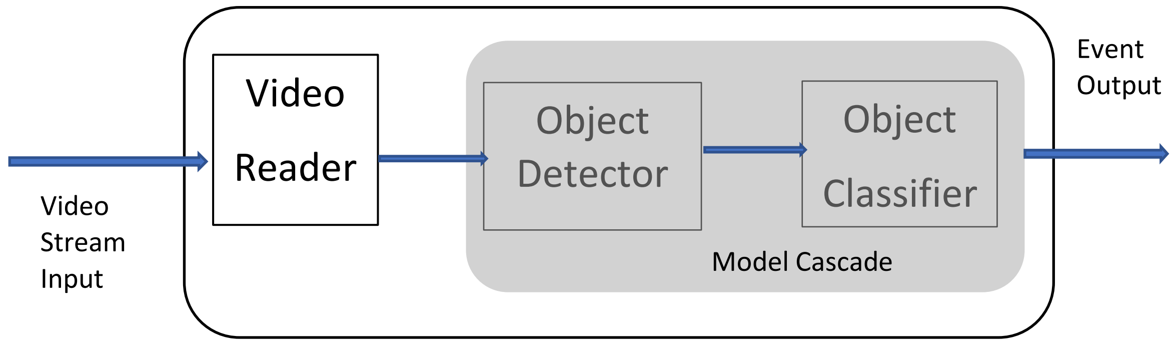
The goal of this assignment is to detect cars in any given video and classify the cars into two classes, one of them being a sedan and the other being an SUV.

Implementation and Design:

● Pipeline Design:

A Pipeline is considered with the sequence of events that happens when a flow takes place. Here data is flowing, we give a video as an input to the pipeline and get classified data on that particular video as output. Our main aim is to detect the car types whether sedan or SUV that are present in the video.

I have used the Tiny-YOLO Model for object detection and then passed the bounding boxes to a keras trained model (trained using transfer learning using various sedan and SUV images) and predict the class of the car.



The Pipeline is as shown in the figure above. We Give some input and the processing takes place within a black box and we see the event output. The input to the system is a video of cars on a street, whereas the output gives us details about the class of cars whether sedan or SUV present in the video.

OpenCV:

We have used the OpenCV library in python.This was created to provide a common platform for computer vision applications and to ease the process of machine learning. The algorithms present are used to read images or videos and detect faces and objects [1]

- The VideoCapture object is used to read the video in the python file Following functions are used:
    - The read() function returns 2 parameters, one being Boolean whether the frame is read correctly or not and the other is the frame received.
    - The release() function is used to release the captured video once the processing of all frames is complete.
- Object detector and classifier.

The working of the system is divided into 2 main classes.

- One class is used to train the classifier. I have implemented the concept of Transfer learning to retrain the already existing Keras model according to the classes that we want to detect in out cars.

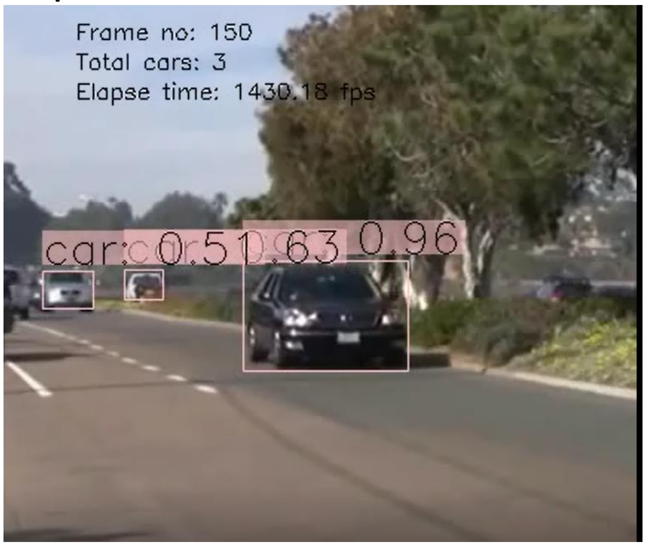
    - Transfer learning helps to use the previously acquired knowledge and enhance it to support our requirements. Here, a set of images of sedan cars and SUV cars is given to the model along with the class id. The data is divided into training and testing. The model is then tested using the test images that how many cars it can correctly classify.
- The other Class (Object_detection) is where we have used Tiny-YOLO to capture the video and detect the objects in the car. After the Video is read using the VideoCapture() class the following functions are used to process the data and detect the objects:
    - The imread() function loads an image from the specified file. If the image cannot be read then this method returns an empty matrix.
    - The resize() function is used to resize the height and width of an image. Since we are capturing the frames from a video, they can be assumed to be of one size. When we are capturing the object in the image after detection, we are resizing that as well to maintain the aspect ratio. We can give the desired size or scaling factors of the image in order to keep the size aligned.
    - The **dnn.readNetFromDarknet**() function is used to read the model from the Darknet model file and store it in the memory. Here we give the path of the config file of our model (Tiny YOLO v4 in our case) and the weights associated with the model. It returns a network object that is ready to process the frames and also perform error handling. [1]
        - Using the object retrieved by the above function we call the **getLayerNames**() function. This function is used to get the layer names of the network. To read the image in a numpy array we need to divide into layers. These layers are then processed simultaneously to train the model
        - We also use the network model to run the **getUnconnectedOutLayers**() function. It is used to obtain position of the layers.
        - The network model is also used in the **forward**() function. Here we extract feed from the output and create a list of layered outputs. If we do not specify the layers it will return the list of predictions from the final layer only. We go through each and every output layer retrieved to get the actual list of objects present. [2] This function is where the exact object detection happens.
    - From the scores obtained, the class_id of the object. There are 80 different classes that are present in the coco.names file. This means we can detect 80 different kinds of objects that are present in the list. Few of the items present in the list are: cars, person, train, toothbrush etc. Hence these can be detected using the tiny-YOLO model. We give the label of the class_id with maximum score to the object detected. A confidence id is also obtained. This value indicates with how much confident the object detecte belongs to the class it says. If the confidence value is greater than the threshold set, It will create a rectangle around the object.
    - dnn.NMSBoxes() is used to perform non-max suppression. It is used to select one entity out of the many overlapping entities. For example if there is a horizontal and vertical bounding box that are overlapping each other, this technique is used to select only the common area between the overlapping rectangles. [2]

- Based on the class id of the objects, the bounding boxes are then passed to the trained classifier to detect whether the car is a sedan or a SUV depending on the check_car_type parameter passed when calling the function.

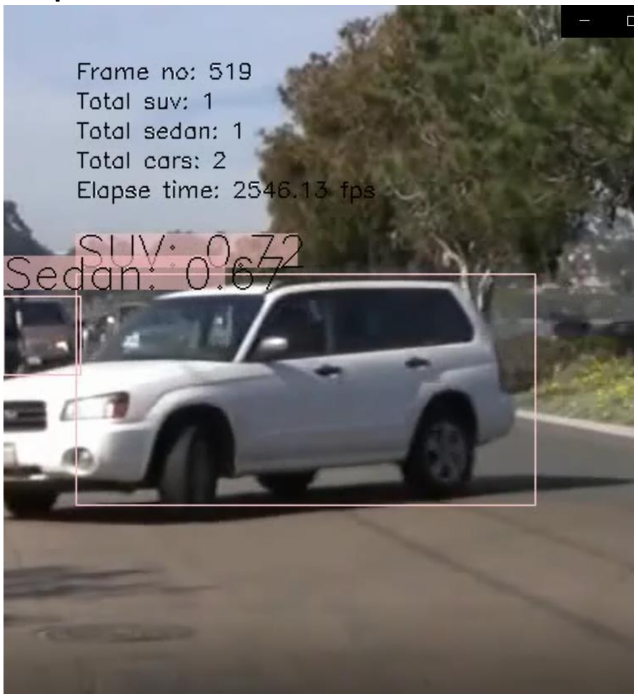
● Pipeline Output:

Throughput: Throughput is usually calculated as the goods per unit time. For example, how many frames processed per unit of time. Hence, in this case we can calculate as how much time did the model take to process 900 frames. (30 seconds * 30 frames/sec).
Throughput of the detection model is ~ **8.48** fps.
Throughput of the classification model is ~**2.62** fps.
Average processing time the Tiny-YOLO Model takes to detect all objects in a frame is about **0.07**s

Output of the Detector Model:

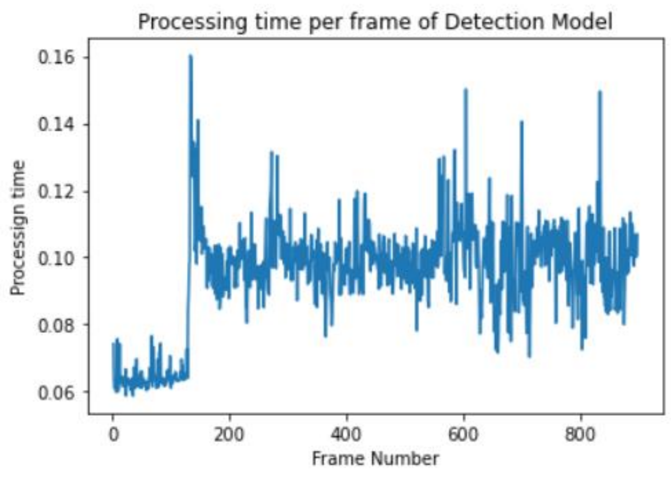


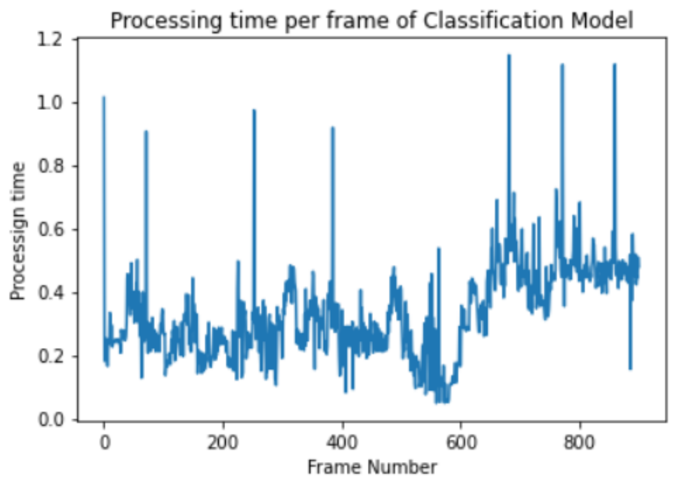
Output of the classifier:



The graphs below show the processing time per frame.

We see that on an average , the detection takes less time as compared to classification. This can also be concluded from the throughput mentioned above.





● Model Training Evaluation:

dataset preparation: The bing-image-downloader library is used in Python to download bulk images from bing.com. Since it uses asyn URL, it is very fast. The downloader.download() function is used. Important Parameters:
query_string: Sedan or SUV (depending on the type of images required)
limit: number of images to download.
output_dir: To determine where to store the downloaded images. It creates a directory with the name given as the query string in this directory
adult_filter_off: To filter the kind of images we need

● Transfer learning approach:

The entire concept if transfer learning is about Taking a network pre-trained on a dataset and then utilizing it to recognize image/object categories it was not trained on. The Keras sequential model is used since this model is used for a plain stack of layers and each layer has eactly one input tensor and one output tensor. [3]
Functions used:

- **tf.keras.Sequential:** There are many models stored in keras tensorflow library, we are calling the Sequential API which is used to build a simple feed forward Neural network.

- **feature_extracter_model:** feature extractor allows the input image to propagate forward, stopping at pre-specified layer, it then takes the *outputs* of that layer as our features [3].
- **hub.KerasLayer()**: We can use it to wrap a callable object from the Keras layer. This is the preferred API to load a TF2-style Saved Model from TF Hub into a Keras model.
- ***summary ():*** This function is used to print the summary of the model so far. It is useful when an incremental model is used
- **compile():** The model is compiled to set the optimizer and loss function. The binary_crossentropy function could be used to calculate loss since we have only two classes, but the categorical_crossentropy provides better results. Compiling a model could be summarised as setting the default values for the model before running the fit function.
- **fit():** This function is where the actual model is trained. We give the training data to the model. Here the weights for each feature are calculated based on data and stored in the model object. When we run the predict() function, the model uses the calculated weights and can predict the class or the response variable(here it is sedan or SUV)
- Since we are implementing the transfer learning there are some predecided weights to start with, we do not have random weights associated. Hence the models intend to train faster since it has some base to start from.

- Accuracy of the validation set is around **97.10%**
- The accuracy of the trained model is about **95.65%**

● Pipeline Optimisation:

We have optimized the system by using the producer consumer model. Here the job of the producer is to read the video file and produce frames (30 frames per second) and keep adding to the queue. About 900 frames are added for our 30 second video. The consumer then reads the frames from the queue and runs TINY YOLO in it. There are no changes to our object_classifier class. In the detection class we have made certain changes like:

1. we do not read the video in the detection class
2. only the frame is passed for processing
3. no further computation is happening there.

We are not calculating the accuracy of the producer consumer model. Only throughput optimisation is the idea here.

Throughput of the classification model with producer consumer implemented is **2.754 frames per second**

● Design Strengths and Weaknesses:

The main idea of the assignment is to perform object detection and classification by getting familiar with the concepts of transfer learning, understanding Keras and Tesorflow libraries, YOLO Algorithm for object detection. Following are the strengths and weakness of the developed system:

- Strengths:
  - By using the existing sequential model, we are able to train a model as per our requirements. This trained model has an accuracy of about 98.9%.
  - With minimal data (like 400 images of each class) such high accuracy is obtained.
  - A lot of time is saved in developing the neural networks from scratch.

    - YOLO Algorithm: This is a widely used fast algorithm. It is used in critical applications where the cost of error is very high for example, self-driven cars. This is because of its high processing speed about 45 frames per second.
    - Selective search using bottom-up optimization is used to involve bottom-up hierarchy to classify from small to large images.
    - Non-max suppression is used to reduce error. It is helpful in selecting the right bounding box, when multiple boxes are selected on the same object. Using NMS we can reduce the duplicate count of objects and hence improve accuracy also reducing the processing time by skipping overlapping objects.
- Weakness:
    - Detection of the cars on the other side is not very accurate. It is because of the spatial constraints of the algorithm. This is a known limitation of the YOLO algorithm. It cannot detect very small objects.
    - The training data set does not include a lot of real-life images. It includes filtered and picture-perfect images. Getting a dataset similar projecting images similar to the actual examples, is a challenge.
    - YOLO tends to detect many false positives. For example, it detects a car even when it is not present, thereby reducing accuracy of the model.
    - Tiny-YOLO is implemented to work faster but it is much less accurate than YOLO. [4]
    - Lack of architecture is the reason for less accuracy. Tiny YOLO is created for domestic applications where speed is more of a requirement as compared to accuracy.
    - The accuracy of the classification is affected by the accuracy of the detection algorithm. This is because If the detection algorithm is not able to detect a car with a confidence higher than the threshold, then the classification algorithm will not be able to classify the car, hence the false negatives are affected.

- Download Link for Submission Files:

https://nuigalwayie-my.sharepoint.com/:u:/g/personal/k_sitabkhan1_nuigalway_ie/EVYhqR7aDXNNluz9LOUo9eYBmNKbHzXn9S8x6E4gofKvjw?e=GZFpkN

References:

Report References

[1] “OpenCV,” OpenCV, [Online]. Available: https://opencv.org/about/.

[2] “kite,” [Online]. Available: https://www.kite.com/python/docs/cv2.dnn.NMSBoxes.

[3] “pyimagesearch,” pyimagesearch, [Online]. Available: https://www.pyimagesearch.com/2019/05/20/transfer-learning-with-keras-and-deep-learning/.

[4] P. Adarsh, P. Rathi and M. Kumar, “YOLO v3-Tiny: Object Detection and Recognition using one stage improved model,” 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), no. 10.1109/ICACCS48705.2020.9074315, pp. 687-694, 2020.

[5] J. C. Redmon, “pjreddie,” [Online]. Available: https://pjreddie.com/darknet/yolo/.

[6] “Analytics India Magazine,” [Online]. Available: https://analyticsindiamag.com/top-8-algorithms-for-object-detection/.

Code References (Also included in Jupyter Notebook)

[1] Object Detection using YOLO: https://www.thepythoncode.com/code/yolo-object-detection-with-opencv-and-pytorch-in-python

[2] Download bulk Images: https://github.com/ostrolucky/Bulk-Bing-Image-downloader

[3] TransferLearning using Keras: https://github.com/codebasics/deep-learning-keras-tf-tutorial/blob/master/18_transfer_learning/cnn_transfer_learning.ipynb

[4] Object Detection using YOLO & OpenCV: https://www.mygreatlearning.com/blog/yolo-object-detection-using-opencv/

[5] Implementation of producer consumer model: https://www.agiliq.com/blog/2013/10/producer-consumer-problem-in-python/#:~:text=The%20problem%20describes%20two%20processes,adds%20it%20to%20the%20queue.