Real-Time Object Detection and Classification

Task Description:
Detect the number of cars, number of SUV cars, and Sedan cars per frame of any given input video at a real-time constraint.

Introduction:
To design and implement a computer vision pipeline to perform the following task:

1. **Video Reading:** For this task, we have used OpenCV to read the video frame by frame and then finally displaying the original video to the user and then passing a queue of frames for the next task
2. **Object Detection:** Used a pretrained state of art object detection model TinyYOLO which is trained on the COCO dataset. The role of TinyYOLO is to detect cars in frames which is one of the classes present in the COCO dataset.
3. **Car Type Classifier:** Once we get the cars with their bounded box for every frame we need to classify them as either SUV or Sedan. For this, we have used the MobileNetV2 classifier and applied transfer learning concepts to modify the model according to our use case.

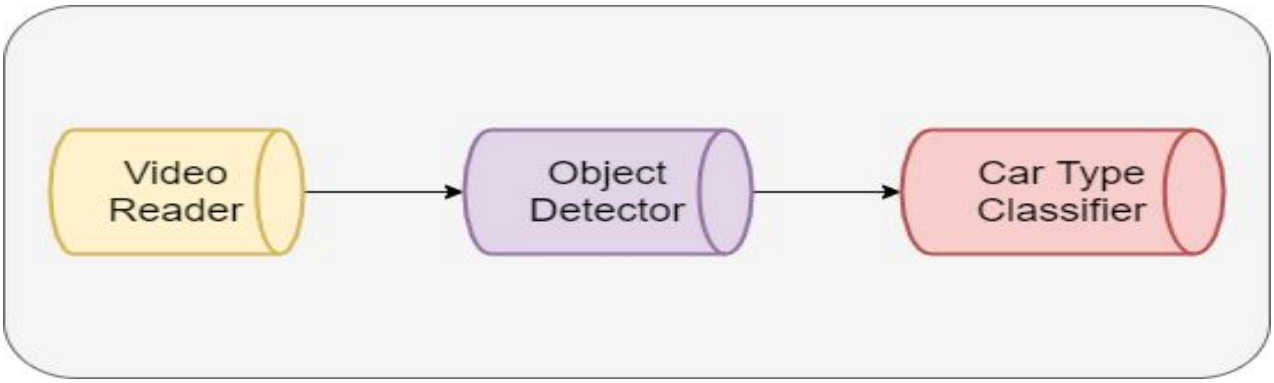

Figure 1. Description of computer vision pipeline

Project Setup:

- Clone the git repo: https://github.com/prakhargurawa/Vehicle-Detection-Classification-YOLO-MobileNet.git
- Download TinyYOLO or YOLO weights from https://pjreddie.com/darknet/yolo/
- To use TinyYOLO use command A else command B:
    A. python convert.py yolov3-tiny.cfg yolov3-tiny.weights model_data/yolo_tiny.h5
    B. python convert.py yolov3.cfg yolov3.weights model_data/yolo.h5
- To test YOLO working: python yolo_video.py --image. Then provide the path to any test image.
- It's recommended to create a separate python environment to avoid any dependency clashes.
  conda env create -f test\dependecies.yml car_env
- Activate the environment and upgrade the Pillow library
  conda activate car-project-env
  pip inststall --upgrade Pillow

- To train your own MobileNet model with modified configuration do changes in a python file and run. Default configuration have Adam optimizer with a learning rate of 0.0001 and trained for 20 epoch
  python MobileNet_TransferLearning.py
- Start the video reading process and overall object detection and classification pipeline
  python VideoReader.py

Pipeline Design, Model Configurations, and Working:

- For **video reading**, we have used OpenCV which reads the video frame by frame at a rate of 30 FPS.
- **Dataset Preparation:** The dataset includes images from two classes SUV and Sedan, with **1540 images of Sedan** and **1519 images for SUV**. The images have been collected using google images web scraping and Stanford Car Dataset[1] which contains images of multiple kinds of cars including SUVs and Sedans. All the images collected from web scraping and Stanford dataset were manually checked once to remove all the ones with were irrelevant or highly dubious. This step is a part of data preparation
- For **Object Detection**, we have preferred TinyYOLO over YOLO as it's lightweight and faster as compared to later. TinyYOLO can process at 220 FPS whereas YOLO processed 20-40 FPS. Due to the requirement to detect in real-time better to use TinyYOLO.
- Once all the frames are read using OpenCV we pass a queue of frames to the detection function which uses the **YOLO's detect_image** function to detect classes in any given frame.
- After we get all the detected classes and their position using YOLO we consider only those detected objects whose class is **the car**.
- Finally, the detected car with their positions in any particular frame is passed to a function that **classifies** any car's image in either SUV or Sedan. This function basically uses the model trained on thousands of SUV and Sean images using the MobileNet model which can be found in MobileNet_TransferLearning.py

MobileNet Model and Transfer Learning:

- The basic structure of MobileNet models looks as depicted in figure 1 which is reused using the concept of transfer learning.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 1. Structure of original MobileNetV2 Model

- MobileNet model is used as a starting point where we discarded the last layer and added our own layers as shown in figure 2 with two additional hidden layers with 512 neurons, one

hidden layer with 256 neurons and finally a single output layer with a single node as this is a binary classification task

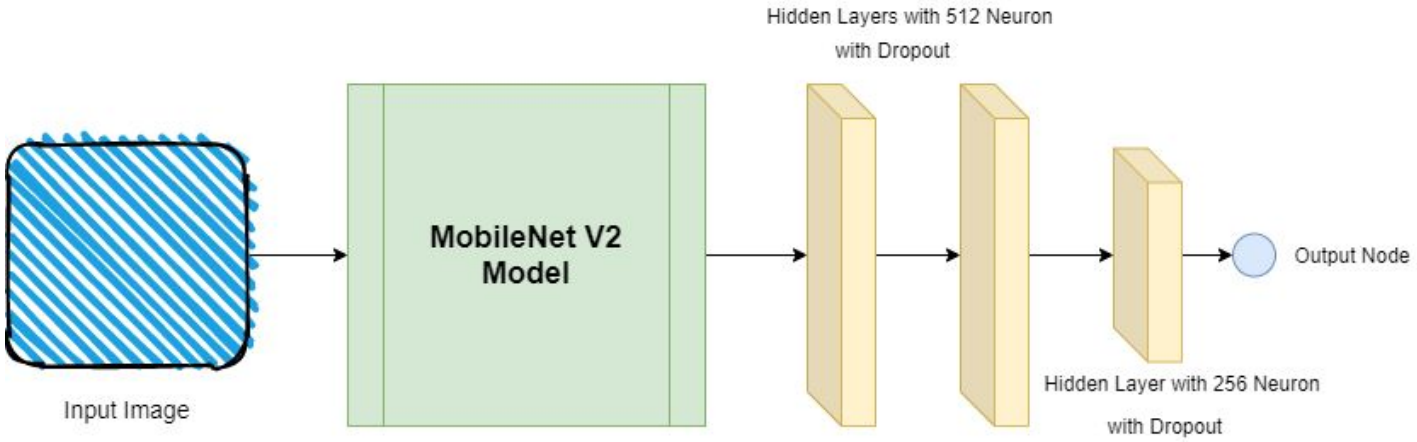

Figure 2. Structure of modified model by the addition of extra hidden layers

- **Hyperparameters and Model Tuning:**
  A. **Dropout:** Used a regularization method dropout on new layers with a probability of 0.2 which helped to reduce overfitting in the model which can be studied due to less gap between training and validation accuracies and errors.
  B. **Data Augmentation:** Tried using data augmentation which performs a different operation like zooming, shifting, etc but was not giving satisfactory results so dropped this operation.
  C. **Epochs:** The number of epoch set for this model was 20 as a higher number of epoch was not giving satisfactory results on ground truth and the rate of increase of accuracy was also low, could be they were overfitted.
  D. **L1/L2 Regularization:** Tried experimenting with L1 and L2 weight regularization but was giving worse results with very high validation errors and low accuracy.
  E. **Model compilation:** The loss function used is binary cross-entropy with accuracy as a metric to progress. Finally, the model weights are saved with the proper name in the folder saved_models.
  F. **Gradient descent optimization algorithms:** Experimented with two optimization algorithms RMSProp and Adam optimizer. For our use case, Adam was giving satisfactory results so used that. The learning rate used was 0.0001 as a lower learning rate was resulting in slower learning with a demand of higher epochs.

Accuracy and Cross-Entropy vs Number of the epoch:
The variation of accuracy and cross-entropy with respect to time/epoch for optimization Adam and RMSProp is depicted in figure 3 below. Also, we have preferred Adam over RMSProp due to the less overfitted model as the difference between training accuracy and validation accuracy is less as compared to RMSProp. As expected it was giving better results on ground truth with better F1 scores of the number of sedan and SUV cars which also proved that overall it's a better model.

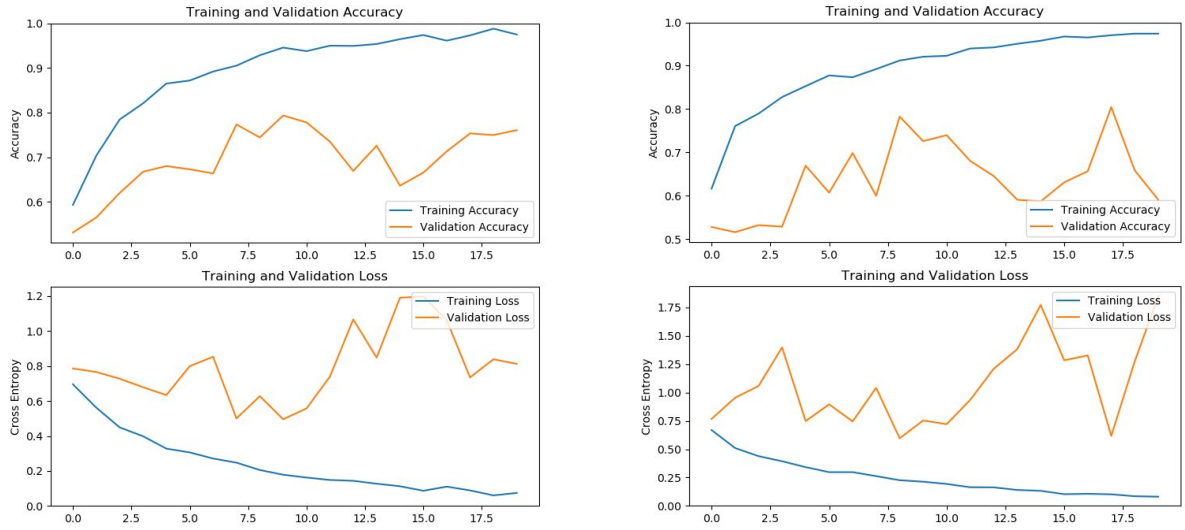

Figure 3. Variation of accuracy and cross entropy(loss) wrt to time/ epoch for Adam(left) and RMSProp(right)

Final Accuracy for deep learning models:

- Adam optimizer (lr=0.0001)

Training loss = 0.0772	Training accuracy = 0.9733
Validation loss = 0.8120	Validation accuracy = 0.7605

- RMSProp optimizer (lr=0.0001)

Training loss = 0.0887	Training accuracy = 0.9721
Validation loss = 1.8487	Validation accuracy = 0.5905

Pipeline Optimization:

- Performed experiments with **Non-Maximal Suppression (NMS)** and **Intersection over Union Threshold (IOU)** parameters of YOLO model to give better results. These parameter changes the threshold of a selection of objects and removes all boxes with low probability and intersection probability. For our use case with TinyYOLO, an NMS Score of 0.2 and IOU of 0.2 were giving better results which finally improves the F1 Score on ground truth.
- Experiments with different hyperparameters for deep learning models such as epochs, optimizer, regularization, dropout, data augmentation with a focus on providing clear and accurate input dataset to the training model.
- Integrated a faster approach of processing using **Thread Pool (Executor framework)** to parallelly process the frames which reduced the execution time by factor. Will be more useful in the future if need to process videos of longer length. But still, there is a need to make the pipeline a little faster as currently, it takes around 3 min to process (detect and classify) a video with 900 frames.

- We tried detecting cars first using harr cascades but it was not giving satisfactory results so we used YOLO for object detection whose performance was much better as compare to first method. The implementation of first method can be found in Car_Detection_Using_Harr_Cascade.py

Pipeline Output:
The input of pipeline is any given video that is displayed to the user and output is video with detected cars and their type (SUV/Sedan) as below in figure 4. The output of the system is also an excel sheet with columns Frames, Sedan, SUV, Total Cars which is then compared with ground truth to calculate the F1 Score of the number of cars per frame, F1 Score of sedan cars, and F1 scores of SUV cars which are shown in figure 5.

Figure 4. Display snapshot out output video with the number of cars with their type attached to cars

```
**************************************************************
      Accuracy/F1 Score with respect to Ground Truth
**************************************************************
Accuracy :  0.5433333333333333
F1 Score for Total Cars in each frame:  0.5601001854715687
F1 Score for SUV in each frame:  0.3164389897441949
F1 Score for Sedan in each frame:  0.47090479319590045
```

Figure 5. The F1 Score for Total Cars, SUV and Sedan compared with ground truth on 900 frames in total

Although the F1 Scores might look low because comparing with this ground truth is a little harsh but overall I feel the model performs really good and can be adjusted to any real-world condition with a few tweaks in NMS, IOU for TinyYOLO model.

Design Strengths and Weaknesses:

- Even though it's a computer vision pipeline the implementation of code has been code on **object-oriented programming standards** with separate classes for video reading, object detection, car classification, etc which promotes reusable programming and will help extend this project.
- Even with multithreading and thread pool the time to process video is significant which can be improved by running this application on **a high-performance system** with CUDA-enabled GPUs.

- Even though the dataset collected from web scraping and Stanford dataset is checked manually there is a number of images that reduced the quality of the dataset and thus reduces the overall performance of the system but at the same time cannot be removed from the dataset as their count is significant.

Download Link:

The code can be downloaded from:
https://github.com/prakhargurawa/Vehicle-Detection-Classification-YOLO-MobileNet
In the same repository Car_Results.xls is the output excel which contains numbe rof cars,sedan,SUV detected and classify per frame with Output.avi as output video.

References:

1. https://ai.stanford.edu/~jkrause/cars/car_dataset.html
2. https://machinethink.net/blog/mobilenet-v2/
3. https://keras.io/api/applications/mobilenet/
4. https://www.researchgate.net/publication/337339754_VidCEP_Complex_Event_Processing_Framework_to_Detect_Spatiotemporal_Patterns_in_Video_Streams
5. https://medium.com/analytics-vidhya/you-only-look-once-yolo-implementing-yolo-in-less-than-30-lines-of-python-code-97fb9835bfd2#:~:text=YOLO%20uses%20Non%2DMaximal%20Suppression,this%20NMS%20threshold%20to%200.6.
6. https://tutorialedge.net/python/concurrency/python-threadpoolexecutor-tutorial/
7. https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/
8. https://towardsdatascience.com/transfer-learning-using-mobilenet-and-keras-c75daf7ff299
9. https://www.kaggle.com/vasantvohra1/transfer-learning-using-mobilenet
10. https://keras.io/api/layers/regularizers/
11. https://machinelearningmastery.com/how-to-reduce-overfitting-in-deep-learning-with-weight-regularization/
12. https://medium.com/analytics-vidhya/you-only-look-once-yolo-implementing-yolo-in-less-than-30-lines-of-python-code-97fb9835bfd2#:~:text=YOLO%20uses%20Non%2DMaximal%20Suppression,this%20NMS%20threshold%20to%200.6.
13. https://github.com/qqwweee/keras-yolo3
14. https://pjreddie.com/darknet/yolo/
15. https://keras.io/api/applications/mobilenet/
16. https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5
17. https://github.com/prakhargurawa/Facial-Emotion-Detection-Neural-Net-OpenCV/blob/main/facial_emotion_detection.py
18. https://ruder.io/optimizing-gradient-descent/