Assignment 3

Prem Shah — IMT2020044
Mayank Chadha — IMT2020045
Shridhar Sharma — IMT2020065

Instructor: Prof. Dinesh Babu *Date: March 22, 2023*

1 Play with CNNs

1.1 Model Architecture

Our model has a simple architecture with 2 convolutional layers followed by max pooling, 2 more convolutional layers followed by max pooling again, a fully connected layer with dropout, and a final fully connected layer with softmax activation.

Here is the model summary of our architecture for deeper understanding:

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 flatten (Flatten)           (None, 4096)              0

 dropout (Dropout)           (None, 4096)              0

 dense (Dense)               (None, 128)               524416

 dense_1 (Dense)             (None, 10)                1290

=================================================================
Total params: 591,274
Trainable params: 591,274
Non-trainable params: 0
_________________________________________________________________
```

Model Architecture

1.2 ReLU vs tanh vs sigmoid

In this assignment, we compared the performance of three different activation functions using the Adam optimizer. We trained the model for 10 epochs and did not perform any hyperparameter optimization.

Overall, our findings suggest that the choice of activation function can significantly impact the performance of the model. Further studies with hyperparameter optimization could potentially improve the performance of the model even further.

Below are the obtained results (values approximated for better comparison):

	ReLU	tanh	sigmoid
Training Time	89.74	84.29	83.84
Classification Performance	0.770	0.710	0.610

Based on the results presented in the table, we can interpret that ReLU (Rectified Linear Unit) performs equally than tanh (Hyperbolic Tangent) and sigmoid in terms of training time. However, when it comes to classification performance, ReLU achieves the best results, followed by tanh and sigmoid, respectively. Therefore, we can conclude that ReLU is the most suitable activation function for classification tasks.

To summarize:

- **Training Time:** ReLU $\approx$ tanh $\approx$ sigmoid
- **Classification Performance:** ReLU $>$ tanh $>$ sigmoid

It is important to note that the results may vary depending on the specific dataset and model architecture used. Nonetheless, based on the findings of this experiment, **ReLU appears to be the best choice for achieving high classification performance while minimizing training time.**

1.3 Momentum and Adaptive Learning

After experimenting with different activation functions and optimization algorithms, we found that using ReLU as the activation function gives the best results. We compared three different optimization algorithms: SGD without momentum, SGD with momentum, and Adam. The results are summarized in the table below.

	SGD(w/o momentum)	SGD (with momentum)	Adaptive Learning (Adam)
Training Time	76.64	77.41	89.74
Classification Performance	0.504	0.758	0.770

We found that the model with momentum outperformed the model without momentum in terms of classification performance. Furthermore, the model with momentum achieved this improvement in accuracy with only a slight increase in training time. However, the Adaptive Learning model outperformed both SGD and SGD with momentum, achieving the highest classification performance and having a considerable training time increment.

Based on these results, we recommend using an architecture with ReLU as the activation function and Adam or SGD(with momentum), according to the requirement, as an optimizer for this task.

2 CNN as a feature extractor

2.1 DATASET : Multi-class Weather Dataset

The Multi-class Weather Dataset is a collection of images of various weather conditions that can be used for image classification tasks. This dataset contains a total of 1125 images of four different weather categories, namely Sunny, Cloudy, Rainy, and Snowy.

Each category contains an approximately equal number of images, i.e., around 275 images per category. The images are taken from various locations and angles, including aerial shots and ground-level images, and were captured by different cameras and photographers.

2.1.1 Assumption:

You are adviced use this Dataset for working with this code.

2.2 Model Architecture

We utilized the ResNet18 architecture as a feature extractor in our study. ResNet18 is a deep neural network architecture that comprises 18 layers, including convolutional layers, pooling layers, fully connected layers, and an output layer. One of the key features of the ResNet18 architecture is the use of residual connections, which enable the network to learn identity mappings and enable deeper networks to be trained while maintaining accuracy. These residual connections allow gradients to flow more efficiently through the network, resulting in more efficient training. Although the model summary is lengthy, the ResNet18 architecture's use of residual connections has made it a popular choice for a variety of computer vision tasks.

Model summary of the architecture on the next page for deeper understanding.

2.3 Classification Accuracies for the Multi-class Weather Dataset

The following table presents the classification performance of three models applied after removing the last softmax layer, which was used in Assignment-2, on the Multi-class Weather Dataset:

	Support Vector Machine	Logistic Regression	K Nearest Neighbour
Classification Performance	0.924	0.884	0.768

Based on the results presented in the table, SVM has the highest accuracy, followed by LR, and then KNN.

2.4 Classification Accuracies for Bike VS Horses Dataset

Dataset given by Sir can be found here.

The following table presents the classification performance of three models applied after removing the last softmax layer, which was used in Assignment-2, on Bike VS Horse Dataset:

	Support Vector Machine	Logistic Regression	K Nearest Neighbour
Classification Performance	1.0	0.972	0.944

Based on the results presented in the table, SVM has the highest accuracy, followed by LR, and then KNN.

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 64, 112, 112]	9,408
BatchNorm2d-2	[-1, 64, 112, 112]	128
ReLU-3	[-1, 64, 112, 112]	0
MaxPool2d-4	[-1, 64, 56, 56]	0
Conv2d-5	[-1, 64, 56, 56]	36,864
BatchNorm2d-6	[-1, 64, 56, 56]	128
ReLU-7	[-1, 64, 56, 56]	0
Conv2d-8	[-1, 64, 56, 56]	36,864
BatchNorm2d-9	[-1, 64, 56, 56]	128
ReLU-10	[-1, 64, 56, 56]	0
BasicBlock-11	[-1, 64, 56, 56]	0
Conv2d-12	[-1, 64, 56, 56]	36,864
BatchNorm2d-13	[-1, 64, 56, 56]	128
ReLU-14	[-1, 64, 56, 56]	0
Conv2d-15	[-1, 64, 56, 56]	36,864
BatchNorm2d-16	[-1, 64, 56, 56]	128
ReLU-17	[-1, 64, 56, 56]	0
BasicBlock-18	[-1, 64, 56, 56]	0
Conv2d-19	[-1, 128, 28, 28]	73,728
BatchNorm2d-20	[-1, 128, 28, 28]	256
ReLU-21	[-1, 128, 28, 28]	0
Conv2d-22	[-1, 128, 28, 28]	147,456
BatchNorm2d-23	[-1, 128, 28, 28]	256
Conv2d-24	[-1, 128, 28, 28]	8,192
BatchNorm2d-25	[-1, 128, 28, 28]	256
ReLU-26	[-1, 128, 28, 28]	0
BasicBlock-27	[-1, 128, 28, 28]	0
Conv2d-28	[-1, 128, 28, 28]	147,456
BatchNorm2d-29	[-1, 128, 28, 28]	256
ReLU-30	[-1, 128, 28, 28]	0
Conv2d-31	[-1, 128, 28, 28]	147,456
BatchNorm2d-32	[-1, 128, 28, 28]	256
ReLU-33	[-1, 128, 28, 28]	0
BasicBlock-34	[-1, 128, 28, 28]	0
Conv2d-35	[-1, 256, 14, 14]	294,912
BatchNorm2d-36	[-1, 256, 14, 14]	512
ReLU-37	[-1, 256, 14, 14]	0
Conv2d-38	[-1, 256, 14, 14]	589,824
BatchNorm2d-39	[-1, 256, 14, 14]	512
Conv2d-40	[-1, 256, 14, 14]	32,768
BatchNorm2d-41	[-1, 256, 14, 14]	512
ReLU-42	[-1, 256, 14, 14]	0
BasicBlock-43	[-1, 256, 14, 14]	0
Conv2d-44	[-1, 256, 14, 14]	589,824
BatchNorm2d-45	[-1, 256, 14, 14]	512
ReLU-46	[-1, 256, 14, 14]	0
Conv2d-47	[-1, 256, 14, 14]	589,824
BatchNorm2d-48	[-1, 256, 14, 14]	512
ReLU-49	[-1, 256, 14, 14]	0
BasicBlock-50	[-1, 256, 14, 14]	0
Conv2d-51	[-1, 512, 7, 7]	1,179,648
BatchNorm2d-52	[-1, 512, 7, 7]	1,024
ReLU-53	[-1, 512, 7, 7]	0
Conv2d-54	[-1, 512, 7, 7]	2,359,296
BatchNorm2d-55	[-1, 512, 7, 7]	1,024
Conv2d-56	[-1, 512, 7, 7]	131,072
BatchNorm2d-57	[-1, 512, 7, 7]	1,024
ReLU-58	[-1, 512, 7, 7]	0
BasicBlock-59	[-1, 512, 7, 7]	0
Conv2d-60	[-1, 512, 7, 7]	2,359,296
BatchNorm2d-61	[-1, 512, 7, 7]	1,024
ReLU-62	[-1, 512, 7, 7]	0
Conv2d-63	[-1, 512, 7, 7]	2,359,296
BatchNorm2d-64	[-1, 512, 7, 7]	1,024
ReLU-65	[-1, 512, 7, 7]	0
BasicBlock-66	[-1, 512, 7, 7]	0
AdaptiveAvgPool2d-67	[-1, 512, 1, 1]	0
Linear-68	[-1, 1000]	513,000

ResNet-18 Model Architecture

3 YOLO V2

Five additional Features of YOLO V2:

1. Firstly, YOLO V2 uses a softmax classifier to determine the class probabilities of the items in each grid cell. This is more precise and trustworthy than the logistic regression classifier used in the previous version.
2. Secondly, YOLO V2 uses Darknet-19 architecture, which is smaller and faster, and offers superior accuracy and detection performance with fewer convolutional layers.
3. Thirdly, anchor boxes were introduced in YOLO V2, which are pre-defined bounding boxes used to detect objects of various sizes and aspect ratios. This increases the flexibility and accuracy of object detection compared to the fixed size and aspect ratio of grid cells used in the previous version.
4. Fourthly, YOLO V2 uses batch normalisation to normalise the inputs to each layer of the neural network. This reduces the impacts of internal covariate shift and increases the stability and speed of the training process.
5. Finally, YOLO V2 uses multi-scale training, which involves training the neural network on images of various sizes. This enables YOLO V2 to identify objects of varied sizes and scales more accurately.

Overall, YOLO V2 is a significant improvement over the previous version, with a range of features that make it more precise, flexible, and accurate.

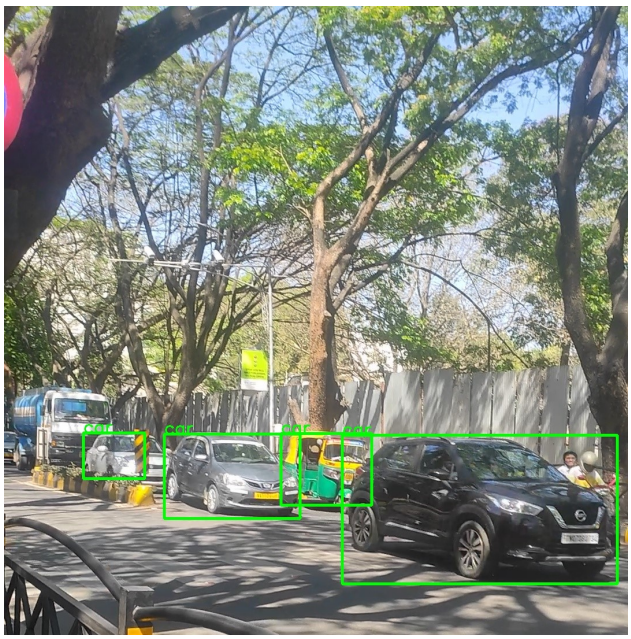
---

4 Object tracker: SORT + DeepSORT

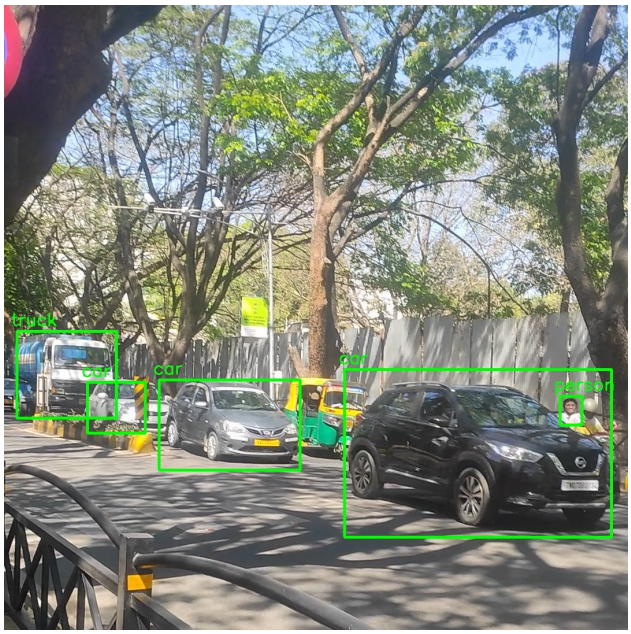
4.1 Results and Observations

4.1.1 Faster R-CNN and YOLO

As per the first video shot near Axis Bank ATM (Electronics City), Object Detection was performed using two popular deep learning models - Faster R-CNN and YOLO. The results obtained are as follows:

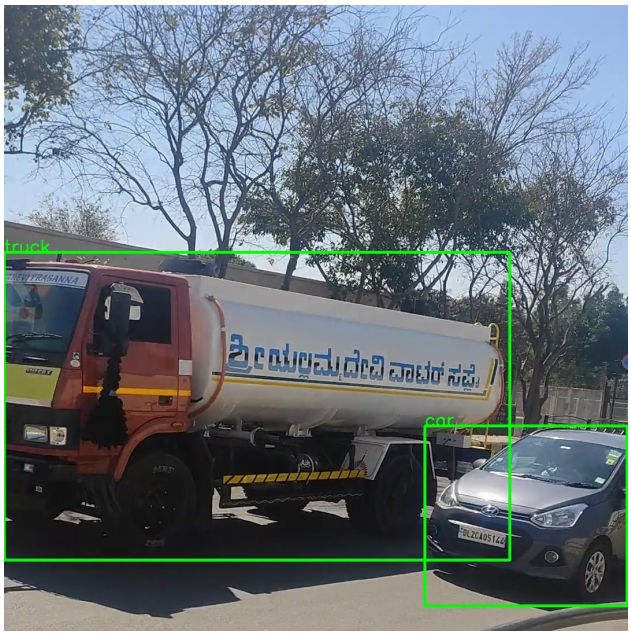


Faster R-CNN: Axis Bank ATM

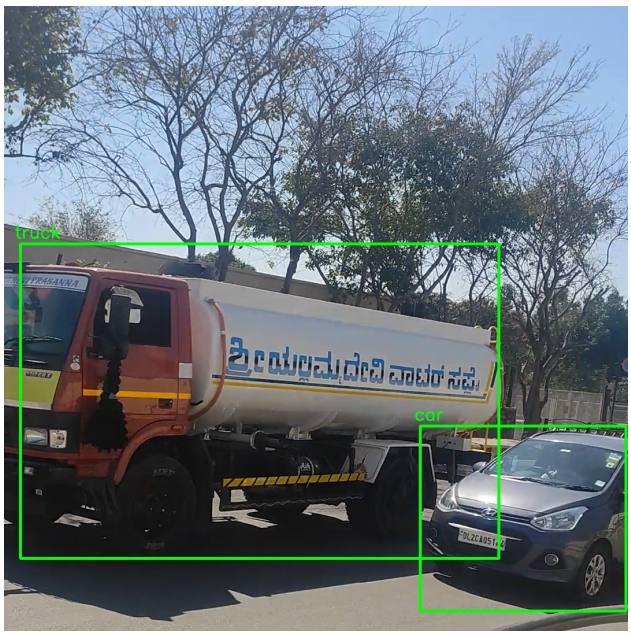


YOLO: Axis Bank ATM

As per the second video shot near Niladri, Object Detection was performed using two popular deep learning models - Faster R-CNN and YOLO. The results obtained are as follows:

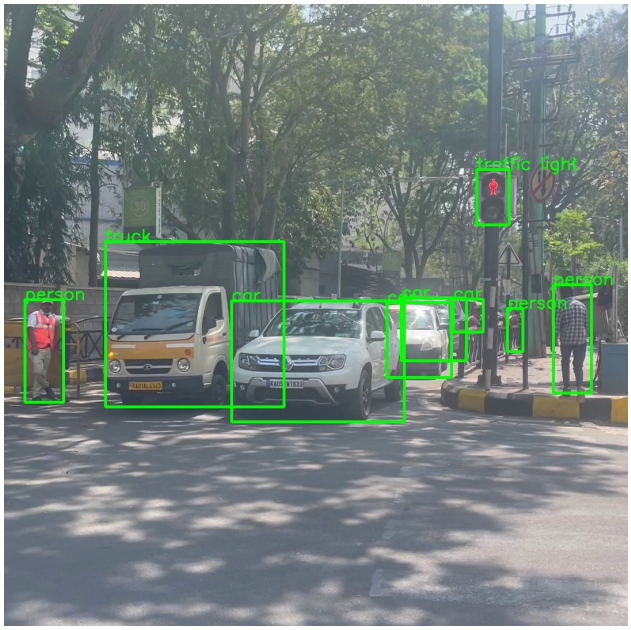


Faster R-CNN: Niladri

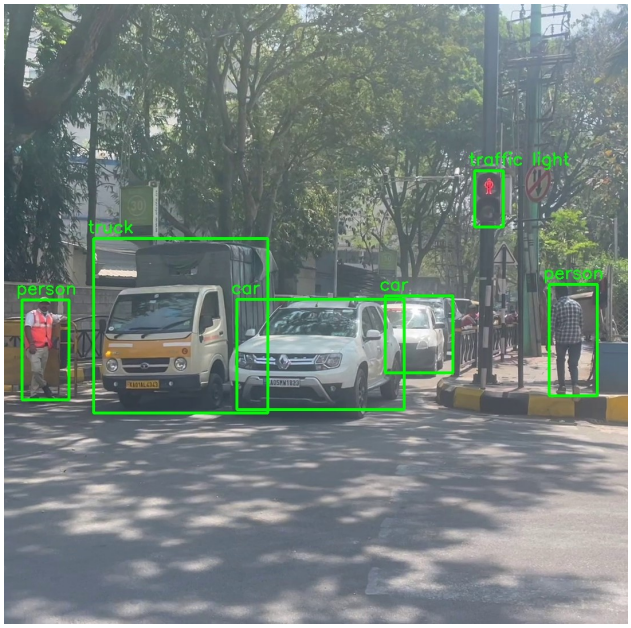


YOLO: Niladri

As per the third video shot at RS Electronics, Object Detection was performed using two popular deep learning models - Faster R-CNN and YOLO. The results obtained are as follows:



Faster R-CNN: RS Electronics



YOLO: RS Electronics

These results indicate that both models were able to detect and classify objects in the video successfully.

Main Observations :

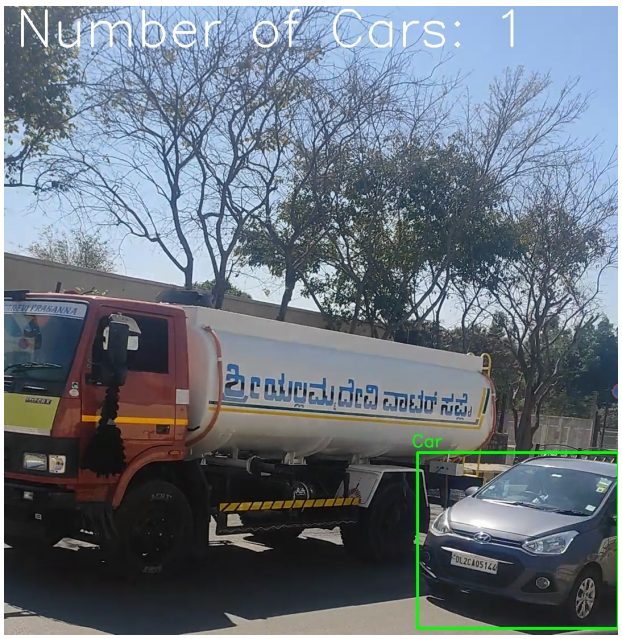
- YOLO has a faster training time and faster inference speed whereas Faster R-CNN has a longer training time and slower inference speed.
- YOLO has lower accuracy compared to Faster R-CNN.

In summary, YOLO is a faster single-shot object detection model, which is easy to implement and offers faster training and inference speeds. However, its accuracy is slightly lower than Faster R-CNN and it is not as good as Faster R-CNN at detecting small objects. On the other hand, Faster R-CNN is a region-based detection model that offers higher accuracy, but is slower and more complex to implement.

It's important to note that the choice between YOLO and Faster R-CNN depends on the specific requirements of the application, and neither algorithm is universally better than the other.

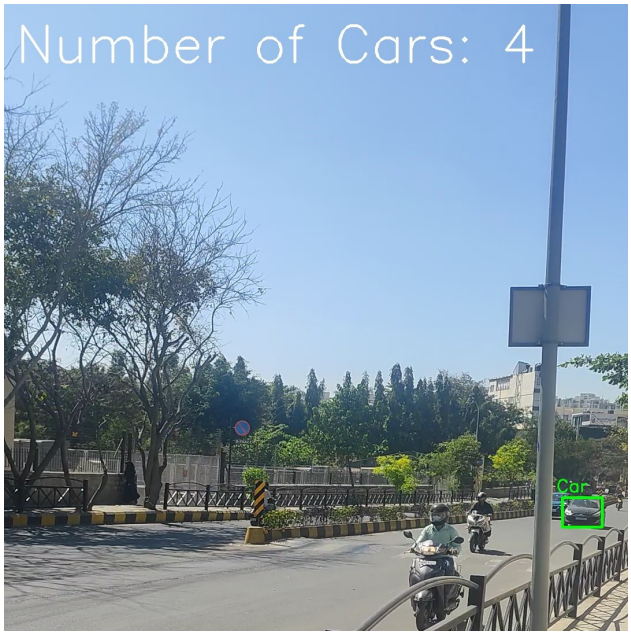
4.1.2 Car Counter: SORT and DeepSORT

For displaying the results obtained through SORT, we will be utilizing selected frames from the second video captured at Neeladri.



SORT: Neeladri (1)

SORT: Neeladri (2)



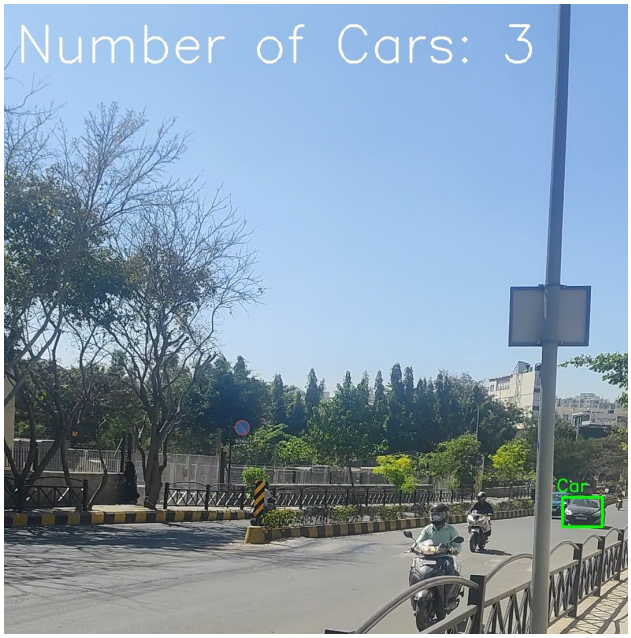
SORT: Neeladri (3)

SORT: Neeladri (4)

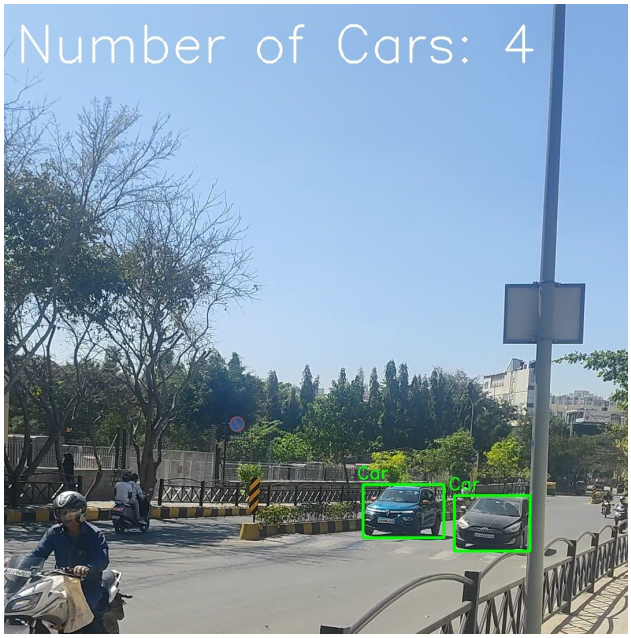
For displaying the results obtained through DeepSORT, we will be utilizing selected frames from the second video captured at Neeladri.

DeepSORT: Neeladri (1)

DeepSORT: Neeladri (2)



DeepSORT: Neeladri (3)



DeepSORT: Neeladri (4)

The findings of the study indicate that both SORT and DeepSORT algorithms were effective in accurately detecting and counting cars in the video. However, a notable difference was observed between the two algorithms in terms of their ability to reidentify cars and avoid counting them multiple times. Specifically, SORT had a tendency to count some cars more than once due to its reidentification process, whereas DeepSORT did not exhibit this behavior. Additionally, it was found that DeepSORT took slightly longer to perform the task compared to SORT. These findings have implications for the practical application of these algorithms in scenarios where accurate car counting is important.

4.2 Difference between SORT and DeepSORT

1. **Tracking method:**

- SORT uses a combination of a Kalman filter and Hungarian algorithm to track objects in real time. The Kalman filter is used to estimate the object's state (position, velocity, etc.), while the Hungarian algorithm is used to match the object's predicted state with the observed state in the next frame.
- DeepSORT uses a deep neural network to extract features from objects and create a similarity metric that is used to associate detections across multiple frames. The network is trained on a large dataset of annotated video data to learn discriminative features that can be used to track objects over time.

2. **Feature extraction:**
   - SORT uses simple handcrafted features, such as color, size, and location, to describe each object in the scene. These features are used to estimate the object's state and make predictions about its future position.
   - DeepSORT uses deep-learned features, which are learned automatically from the data using a deep neural network. These features can capture more complex relationships between the object's appearance and motion, making them more effective for tracking objects over time.
3. **Re-identification:**
   - SORT does not support re-identification, which means that if an object disappears from the scene and reappears later, it may be tracked as a new object.
   - DeepSORT supports re-identification by using a deep neural network to learn a feature representation that is invariant to changes in appearance due to lighting, pose, or occlusion. This allows it to track objects more accurately over longer periods of time, even when they disappear and reappear in the scene.
4. **Accuracy:**
   - SORT has moderate accuracy compared to more advanced algorithms. It may struggle with complex scenarios, such as crowded scenes or objects with similar appearances.
   - DeepSORT has higher accuracy due to its ability to learn more discriminative features and handle more complex scenarios. It is particularly effective for tracking objects with complex appearances or behavior.
5. **Speed:**
   - SORT is fast and can process video data in real-time. It is suitable for applications that require real-time tracking, such as surveillance systems.
   - DeepSORT is slower than SORT due to the computational overhead of deep learning. However, it is still fast enough to process video data in near real-time, and it can be optimized to run more efficiently on specialized hardware.

4.3 Assumptions

- In order to perform object detection on a video, we employed a method that involves converting each frame of the video into an image and then running object detection on each of these images. Due to hardware limitations, we relied on Kaggle to perform the computations. Once we had completed object detection on all the video frames, we stitched them back together to create a new video. It should be noted that we were unable to use the cv2.imshow function in real-time as it caused a pop-up window, which is not supported on the Kaggle platform, which is why this methodology was adopted. Note this is a time consuming process.
- To run the notebook, you need to provide the path of your video file in the following line of code:

  ```
  vidcap = cv2.VideoCapture('/path/to/your/video/file.mp4')
  ```

  Replace `/path/to/your/video/file.mp4` with the actual file path of your video.

  The output video file will be saved as `.avi` file.

- For the SORT (Simple Online and Realtime Tracking) algorithm, we are using the official implementation available on Github. To use this implementation, it is necessary to install all the requirements as listed in the readme file on the Github page.

4.4 Input Videos and YOLO Files

For running the code, you will need access to the video files used for illustration, which can be found at the following link: Link.

In addition, there are some useful YOLO files that can be found at the following link: Link.

4.5 Output Videos

Finally, all the output videos generated by running the code can be found at the following link: Link.
