VR Assignment-3

Team Name:Raven Claw

SINCE 1998

SILVER JUBILEE YEAR

iiit-b

ज्ञानमुत्तमम्

Pioneering Excellence in Education,
Research & Innovation

Team Members:
Rittik Panda(MT2022090)
Soumya Chakraborty(MT2022162)

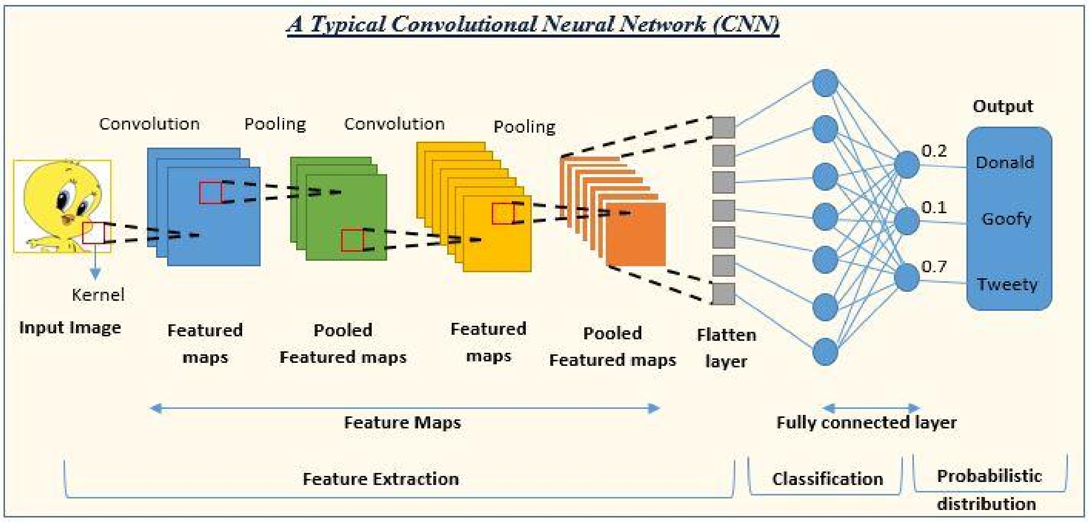
VR MINI PROJECT

Assignment 3a: Play with CNNs

Ans:

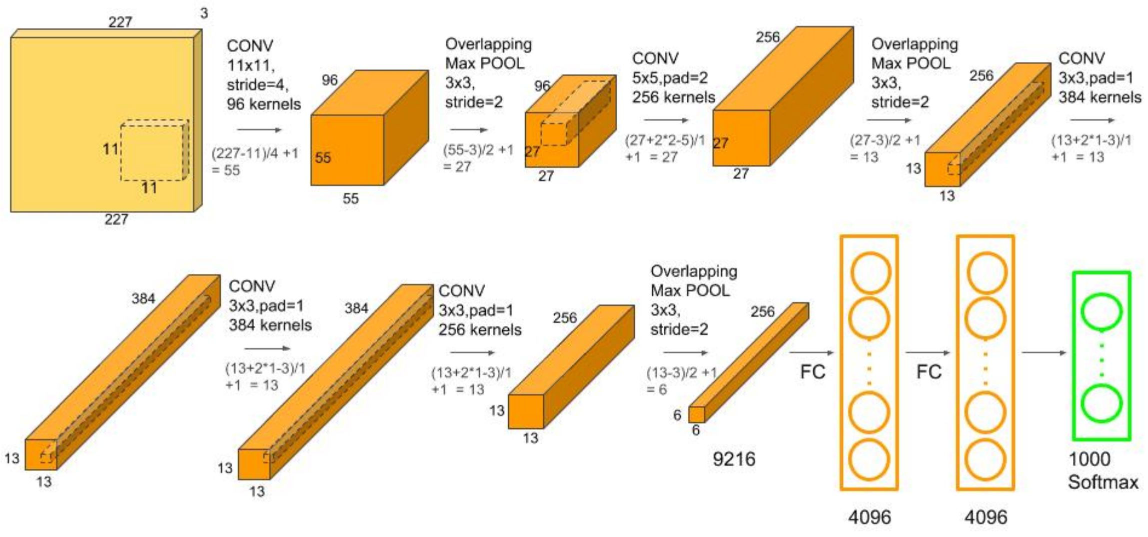
What is CNN: CNN stands for Convolutional Neural Network, which is a type of neural network commonly used in computer vision tasks such as image and video recognition, object detection, and segmentation.

The key difference between a CNN and a traditional neural network is the way in which it processes data. A CNN applies a series of convolutional filters to the input data, which helps to extract features and patterns from the input image.These filters typically start by detecting simple features such as edges, lines, and corners, and then gradually progress to more complex features such as textures and shapes.After the convolutional layers, a CNN usually includes pooling layers, which downsample the feature maps produced by the convolutional layers and help to reduce the dimensionality of the data and get the global perspective of data.Finally, the output from the convolutional and pooling layers is fed into one or more fully connected layers, which perform classification or regression tasks based on the extracted features.



AlexNet: AlexNet is a deep convolutional neural network (CNN) architecture designed by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton in 2012. It won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 by achieving a top-5 error rate of 15.3%, which was significantly better than the second-best performance of 26.2%.

The architecture of AlexNet consists of 5 convolutional layers, followed by 3 fully connected layers and a softmax output layer. The network has a total of 60 million parameters and 650,000 neurons. The convolutional layers use a rectified linear unit (ReLU) activation function, and the network uses dropout regularization to prevent overfitting.



Architecture Of Alexnet

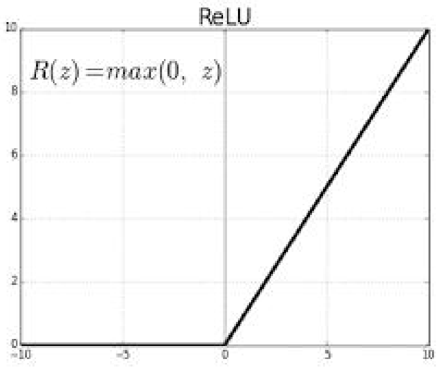
AlexNet was significant in advancing the field of computer vision because it demonstrated the power of deep learning for image recognition tasks and paved the way for subsequent breakthroughs in the field.

Training A CNN: While training a RCNN there are few things we have to keep in our minds like: what is the best architecture of CNN for our task , which loss function to use according to our task , which activation function to use in our architecture , which optimizer to use in our gradient descent optimization algorithm for faster convergence and less training time.

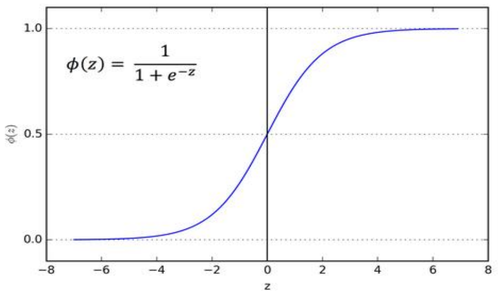
For classification task we generally use cross-entropy loss and for regression task we generally use mean squared error loss.

The 3 most popular activation functions are ReLU, Sigmoid & tanh.

ReLU: The rectified linear activation function or ReLU for short is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

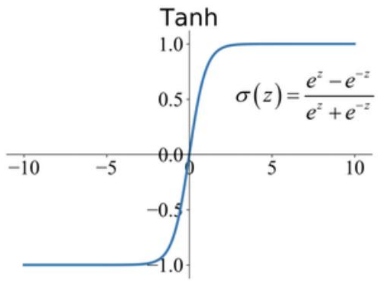


Sigmoid :The Sigmoid Function curve looks like a S-shape.The main reason why we use sigmoid function is because it exists between (0 to 1). Therefore, it is especially used for models where we have to predict the probability as an output.



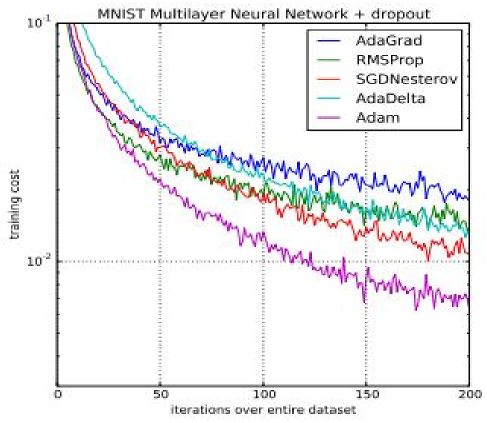
Sigmoid

Tanh:
Tanh is also like logistic sigmoid but better. The range of the Tanh function is from (-1 to 1). Tanh is also sigmoidal (s - shaped).The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the Tanh graph.



All the functions are differentiable and monotonic.

There are several optimizers available for deep learning, each with its own advantages and disadvantages. Some of the commonly used optimizers are:
Stochastic Gradient Descent (SGD): SGD is the most widely used optimizer in deep learning. It updates the parameters in the direction of the negative gradient of the loss function for each mini-batch of data.
Adam: Adam is a popular optimizer that uses a combination of adaptive learning rates and momentum. It adapts the learning rate for each parameter based on the magnitude of its gradient and past gradients.
Adagrad: Adagrad adapts the learning rate of each parameter based on the sum of the squares of its past gradients. It is well-suited for sparse data.
RMSProp: RMSProp is similar to Adagrad but it uses a moving average of the squared gradient instead of the sum of the squares. It is also well-suited for sparse data.
Adadelta: Adadelta is similar to RMSProp but it uses a moving average of the squared gradient and the squared parameter update.
Each optimizer has its own hyperparameters that need to be tuned to obtain the best performance on a given dataset.Today Adam is the most popular optimizer in deep learning.



Implementation Details:

Dataset Details: We used CIFAR-10 dataset for our project.The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.The dataset is divided into five training batches and one test batch, each with 10000 images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class.The 10 classes are airplane, car, bird, cat, deer, dog, frog, horse, ship, and truck.

Preprocessing: The following preprocessings were done on CIFAR-10 dataset:
Random Crop: The first preprocessing step is to randomly crop the input image to a size of 32x32 with padding of 4 pixels on each side. This is done to augment the data and create more diverse images for training.
Random Horizontal Flip: The second preprocessing step is to randomly flip the image horizontally with a probability of 0.5. This is another form of data augmentation to create more diverse images and help the model generalize better to unseen data.
ToTensor: The third preprocessing step is to convert the image to a PyTorch tensor. PyTorch uses tensors as the main data structure for its deep learning models, and this step converts the image from its original format to a tensor that can be processed by the model.
Normalize: The final preprocessing step is to normalize the tensor values using mean and standard deviation values specific to the CIFAR-10 dataset. This helps to reduce the range of the input data and make it easier for the model to learn the patterns in the data.
While training we divided our dataset into batches of size 64 and we used cross-entropy loss for our classification task.

Used Model Architectures:

We tried several model architectures for our task:

1.PandaNet1: It takes a 32x32 pixel input image with 3 color channels (RGB) and outputs a probability distribution over 10 classes.
The architecture consists of 3 convolutional layers followed by max-pooling layers to reduce the spatial dimensions of the output. Each convolutional layer uses a 3x3 kernel with padding of 1 to maintain the spatial resolution. The first layer has 32 output channels, the second has 64, and the third has 128.
After the final pooling layer, the output is flattened and passed through two fully connected layers (or linear layers) with a ReLU activation function applied to the first layer and a dropout regularization with a probability of 0.5. The first fully connected layer has 512 hidden units and the final layer has 10 output units, corresponding to the 10 classes of the classification problem. Finally, a softmax activation function is applied to the output layer to obtain the class probabilities.
The network has a total of **1,147,466** trainable parameters, which include the weights and biases of the convolutional layers, the weights and biases of the fully connected layers, and the probabilities for each class in the output layer.

2. PandaNet2: This CNN takes a 3-channel image as input and consists of 3 convolutional layers followed by max-pooling layers, and 3 fully connected layers. ReLU activation function is used after each layer except the last fully connected layer which uses log-softmax activation function. The first convolutional layer has 64 filters, the second layer has 128 filters, and the third layer has 256 filters. Each convolutional layer uses a 3x3 kernel with stride 1 and padding 1 to preserve the spatial dimensions of the input. The max-pooling layers have a 2x2 kernel with stride 2 to downsample the spatial dimensions by a factor of 2. After the convolutional and max-pooling layers, the output is flattened and passed through two fully connected layers with 256 and 128 units respectively, each followed by a ReLU activation function. The final layer is a fully connected layer with 10 units (corresponding to the 10 output classes) and uses log-softmax activation function to normalize the output probabilities.This model has **930826** total trainable parameters.

3. **PandaNet3**: It consists of six convolutional layers followed by max-pooling layers, three fully connected (linear) layers, and ReLU activation functions between layers. Batch normalization is used to improve the convergence speed and regularization. The number of feature maps (depth) increases from 32 to 256 gradually, while the spatial resolution is reduced by the max-pooling layers. The output of the last linear layer is passed through a log-softmax activation function, which is commonly used for multi-class classification. This architecture is not a specific famous model, but it follows the general design principles of popular CNNs like VGG, ResNet, and Inception. It has total **5,852,234** trainable parameters.

4. **PandaNet with Skip Connections**: It consists of several layers of convolutional and residual blocks, followed by a classifier. The architecture takes 3-channel input images and outputs a class prediction among 10 possible classes.The CNN architecture consists of the following layers:
The first layer is a convolutional block with 64 output channels.
The second layer is another convolutional block with 128 output channels and a max-pooling layer with a kernel size of 2.
The third layer is a residual block consisting of two convolutional blocks with 128 output channels.
The fourth layer is another convolutional block with 256 output channels and a max-pooling layer with a kernel size of 2.
The fifth layer is another convolutional block with 512 output channels and a max-pooling layer with a kernel size of 2.
The sixth layer is a residual block consisting of two convolutional blocks with 512 output channels.
The seventh and final layer is the classifier which consists of a max-pooling layer with a kernel size of 4, a flattening layer, and a linear layer that outputs the class scores.
The model uses the ReLU activation function after each convolutional block, and it also uses batch normalization to improve training stability. It has total **6,575,370** trainable parameters.

Training Details & Results:

Sr. NO.	Network	Activation	Optimizer	No. Epochs	Training Time (s)	Val. Accuracy	Test Accuracy
1	PandaNet-1	ReLU	SGD	50	912.53	24.86	25.55
2	PandaNet-1	Sigmoid	SGD	50	401.80	10.28	11.56
3	PandaNet-1	Tanh	SGD	50	404.41	25.3	27.63
4	PandaNet-1	ReLU	Adam	20	372.99	62.74	63.12
5	PandaNet-1	Tanh	Adam	20	384.69	66.88	67.59
6	PandaNet-1	Sigmoid	Adam	20	564.71	53.32	54.02
7	PandaNet-1	ReLU (He initialized)	Adam	20	567.67	10.10	10.0
8	PandaNet with Skip Connections	ReLU	SGD with momentum	40	1320.63	84.04	83.64
9	PandaNet-2	ReLU	SGD with momentum	40	1118.97	78.36	78.34
10	PandaNet-2	Tanh	SGD with momentum	40	326.46	76.88	77.51
11	PandaNet-2	Sigmoid	SGD with momentum	40	325.72	10.20	10.00

12	PandaNet-2	ReLU	Adam	40	325.29	82.12	82.6
13	PandaNet-2	Tanh	AdaDelta	40	324.02	65.04	65.51
14	PandaNet-3	ReLU	SGD with momentum	50	1532.57	77.08	77.95
15	PandaNet-3	ReLU	NAG	50	1112.52	80.74	80.46
16	PandaNet-3	ReLU	AdaDelta	50	1120.35	78.36	78.57
17	PandaNet-3	ReLU	AdaGrad	50	1102.49	77.94	77.27
18	PandaNet-3	ReLU	RMS_Prop	50	1099.29	84.04	84.67
19	PandaNet-3	ReLU	Adam	50	1091.60	85.32	85.81
20	PandaNet-3	Tanh	Adam	50	423.98	69.94	69.69
21	PandaNet-3	Sigmoid	RMS_Prop	50	409.40	85.02	84.96

Obseravations:

- We observe that among all the activation functions ReLU performs the best with all the models.
- We also notice that among all the activation functions, Sigmoid performs the worst in most of the cases.
- Adam optimizer performs well as it can converge the model in less number of epochs in less amount of time.
- We also noticed that adding skip-connection to our model, improves the model's performance.
- PandaNet2 (with ReLU and Adam) performed really well even though having the least number of parameters among the 3 models that we have tested. It's best performance was with an accuracy of 82.6 on the test data in 40 epoch as compared to our best model PandaNet3 in 50 epochs with 85.81 accuracy.
- PandaNet1's best performance was 67.59 accuracy with Tanh activation and Adam optimizer.

Recommended architecture:

Based on our exploration on all the types of activation functions and optimizers and model architectures, **PandaNet-3** is our recommended architecture with **Adam** optimizer and **ReLU** activation.

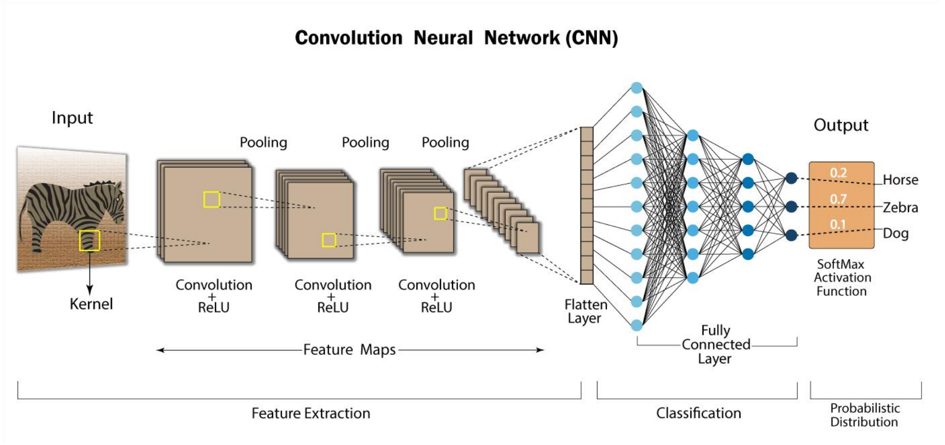
VR MINI PROJECT REPORT

Q3b. CNN as feature extractor

Ans:

<u>What is CNN</u>: CNN stands for Convolutional Neural Network, which is a type of neural network commonly used in computer vision tasks such as image and video recognition, object detection, and segmentation.

The key difference between a CNN and a traditional neural network is the way in which it processes data. A CNN applies a series of convolutional filters to the input data, which helps to extract features and patterns from the input image.These filters typically start by detecting simple features such as edges, lines, and corners, and then gradually progress to more complex features such as textures and shapes.After the convolutional layers, a CNN usually includes pooling layers, which downsample the feature maps produced by the convolutional layers and help to reduce the dimensionality of the data and get the global perspective of data.Finally, the output from the convolutional and pooling layers is fed into one or more fully connected layers, which perform classification or regression tasks based on the extracted features.



<u>CNN As A Feature Extractor</u>:To use a pre-trained CNN as a feature extractor, we can remove the fully connected layers from the network and take the output of the last pooling layer and flatten it and use it as features. These features typically represent high-level representations of the input image or data. These features can be thought of as a compressed representation of the most important information in the input that is relevant to the specific task being performed.These features can then be fed into a separate classifier or regression model to perform a specific task like image classification , object detection etc.

Using a pre-trained CNN as a feature extractor can be especially useful in scenarios where you have limited labeled data or limited computational resources to train a CNN from scratch. By leveraging a pre-trained CNN's learned features, you can save time and resources while still achieving good performance on your task.

Some popular pre-trained CNNs that are commonly used as feature extractors include AlexNet,VGG, ResNet.

<u>Implementation Details</u>:

<u>Feature Extraction Module</u>:

In our project we used **alexnet()** model from **torchvision** module of **PyTorch** library,which is previously trained on **1000 class imagenet** dataset. Then we removed the fully connected layers or classification module in front of it and only took the feature extraction module using **alexnet().features** function and added a **flatten layer** in front of it. Finally we got our feature extraction sequential model using **model = torch.nn.Sequential(features, flatten)** which had total **2,469,696** trainable parameters & **0** non trainable parameters.

<u>Dataset Description & Preprocessing:</u>

Then we used our feature extraction module to extract features from 5 different datasets namely

1. **Cats vs Dogs Classification Dataset**:
 We got this dataset from kaggle. It contains two folders dogs and cats , dog folder contains 12500 dog images inside it and cat folder contains 12500 cat images inside it .We first merged those two folders and assigned corresponding labels with respect to the images.Then suffled the data and spiltted the data in train and test data while train data contains 20000 images and test data contains 5000 images.
2. **Birds Species Classification Dataset:** This dataset we also got from kaggle. It contains images of 500 different bird species.It contains total 80085 training images and 2500 test images of 500 different species of birds.
3. **Horse vs Bike Classification**: This dataset was provided by our course instructor.It contains total 80 bike images and 81 horse images. We mixed them ,suffled them and divided into train test.
4. **CINIC-10 Dataset:**CINIC-10 is an augmented extension of CIFAR-10. It contains the images from CIFAR-10 (60,000 images, 32x32 RGB pixels) and a selection of ImageNet database images (210,000 images downsampled to 32x32). It was compiled as a 'bridge' between CIFAR-10 and ImageNet, for benchmarking machine learning applications. It is split into three equal subsets - train, validation, and test - each of which contain 90,000 images. For our project we used 90000 images from train set for training and 1000 images from test set for testing.

Training:

In training phase we resized our each image into 224*224 size and passed them through our feature extractor module and got a feature vector of 9216 imension. For whole training data we got a 2d numpy array whose one row corresponds to one datapoint and one column corresponds to one feature. Then we trained **svm** & **xgboost** classifier from **sklearn** on training data.

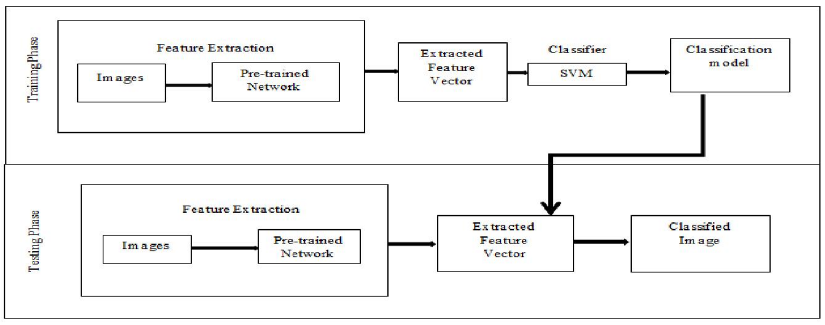


Fig. 2. Classification using Pre-trained Neural Network + SVM

Testing: In testing phase we extracted features of 9216 dimension from each test image using the same process as the training phase.Then used our trained classifier to predict on test features.Then we measured the accuracy.

Initially we tried our approach on **CIFAR-10** data as well.

Results:

Dataset	**Classifier**	**Accuracy**
CIFAR-10	SVM	61.20%
	XGBoost	56.66%
Cats Vs Dogs	SVM	73.96%
Birds Species	XGBoost	46.07%
	SVM	.16%
Horse Vs Bike	SVM	100%
CINIC-10	SVM	48.80%

<u>Observations</u>:

1. We saw that our method performs fairly well in case of Cats vs Dogs and Horse vs Bike dataset as they are only two class classification problem and those classes were included in imagenet dataset on which our feature extractor was pretrained. Moreover in case of Cats Vs Dogs we had a huge amount of data to train our classification model as compared to our no. of classes.

2. In case of Birds species classification we had a huge no. of classes. Moreover this classification task is domain specific. So our imagenet pretrained alexnet feature extractor could not extract the right features for this task. So our xgboost classifier gave a low accuracy of 46.07% and SVM classifier did not learn anything at all and gave a accuracy of 0.16%.

Assignment3C
YOLO
(You Only Look Once)

YOLO (You Only Look Once) is a state-of-the-art object detection algorithm that uses a neural network to detect objects in an image. The YOLO architecture is designed to be fast and accurate, allowing it to process images in real-time.
The YOLO architecture consists of a series of convolutional layers followed by fully connected layers. The convolutional layers are used to extract features from the input image, while the fully connected layers are used to make predictions about the objects in the image. YOLO uses a single convolutional network to predict both the class probabilities and the bounding boxes of the objects in the image.
YOLO divides an image into a grid and predicts bounding boxes and class probabilities for each grid cell.One of the unique features of YOLOv2 and its subsequent models is the use of anchor boxes. Anchor boxes are predefined boxes of different sizes and aspect ratios that are used to predict the bounding boxes of objects in the image. Instead of predicting the coordinates of the bounding box directly, YOLO predicts the offsets between the anchor box and the true bounding box. This allows YOLO to handle objects of different sizes and aspect ratios.

The anchor boxes are represented as vectors, with each vector containing the width and height of the box,the aspect ratio,probability of the object classes and confidence score.
An anchor box B can be represented as:
[x1,y1,x2,y2,w1,h1,.......pc1,pc2,pc3...]
The vectors are learned during training, allowing the model to adapt to the specific characteristics of the objects in the dataset.
The YOLO algorithm uses a unified detection approach, which means that it predicts the class probabilities and bounding boxes of all objects in the image simultaneously through one single network and training. This is in contrast to other object detection algorithms that use a sliding window approach or 2-step ,3-step training as in RCNNs which can be slower and less accurate.

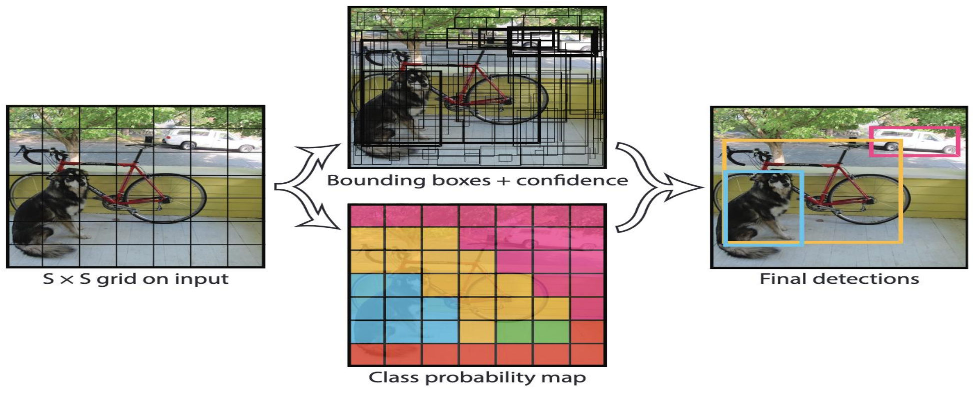


Fig: YOLO algorithm

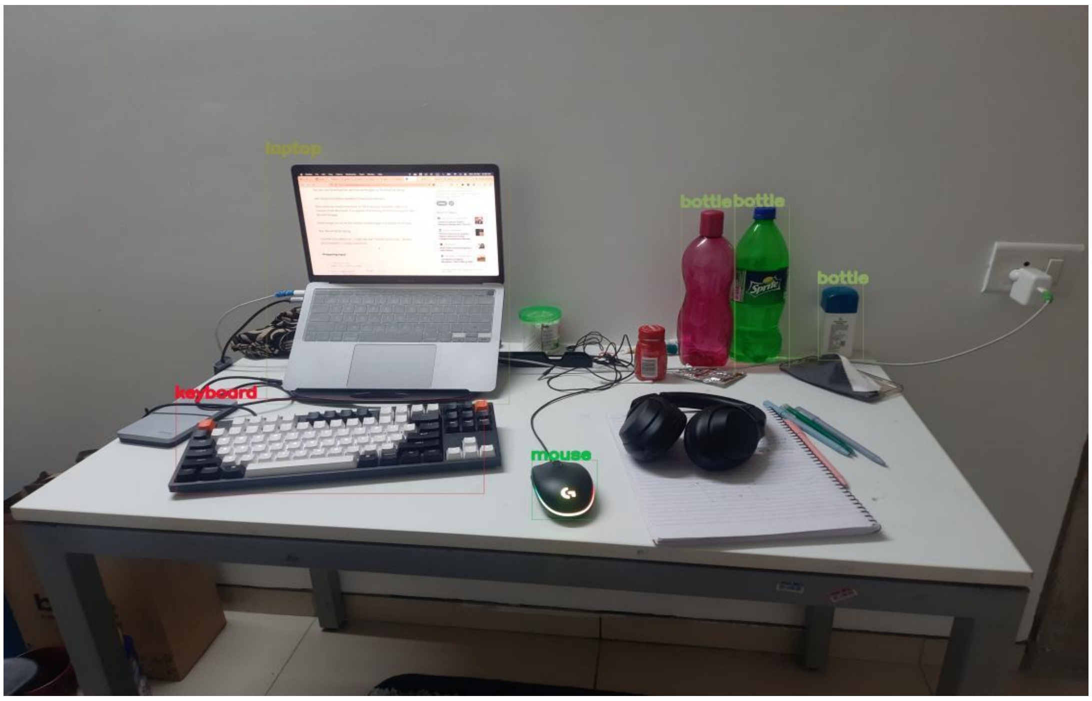


Fig: Object detection with YOLO

The algorithm has evolved over time and has been updated in different versions. Here are some of the differences between YOLO v1, and v2:

Architecture: The detection layer of YOLOv1 contains 24 convolutional layers and 2 fully connected layers. YOLOv2's architecture , the Darknet-19 has 19 convolutional layers and 5 max-pooling layers.YOLOv2 removes all fully connected layers and uses anchor boxes to predict bounding boxes. One pooling layer is removed to increase the resolution of output.

Batch Normalization: YOLOv1 only had convolutional layers in the feature extraction network, YOLOv2 introduced Batch normalization which improves the convergence of the model.

Accuracy: YOLO v1 was the fastest of the available detection models when it was first published but it lacked in accuracy a lot. It suffered from localizing smaller objects and objects very close to one another. YOLO v2 the 2nd version of the model is faster and also more accurate than its predecessor.

Anchor boxes: YOLO v2 introduced the concept of anchor boxes, which are predefined bounding boxes of different sizes and aspect ratios that are used to detect objects of different shapes and sizes. YOLOv2 removed the fully connected layers of YOLO and replaced them with anchor boxes to predict bounding boxes.

Resolution: YOLOv2 is trained on 416x416 images as compared to 448x448 of YOLOv1. This decreased resolution and the addition of anchor boxes has increased the mAP(mean average precision) from 63.4 at 45fps to 76.8 at 67fps. YOLOv2 trained on 288x288 images gives mAP of 69 at 91fps. At 544x544 it gives 78.6 mAP at 40fps.

The most recent official release of YOLO is YOLOv7 which gives a good amount of accuracy over 286fps.

To summarize, we can say that although YOLO is a state-of-the-art model for object detection, it still suffers from accuracy challenges compared to RCNNs and other models. It also suffers form challenges of Occlusion and detection of small objects.But when it comes to speed, it is the fastest model available. That's a trade-off of speed over accuracy that YOLO offers us and it is the most preferred technique in most of the object detection tasks especially in real-time scenarios.

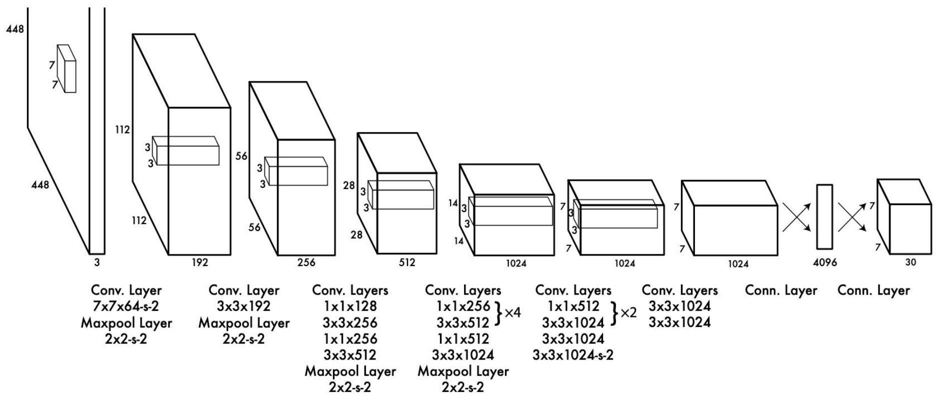


Fig: Architecture of YOLOv1

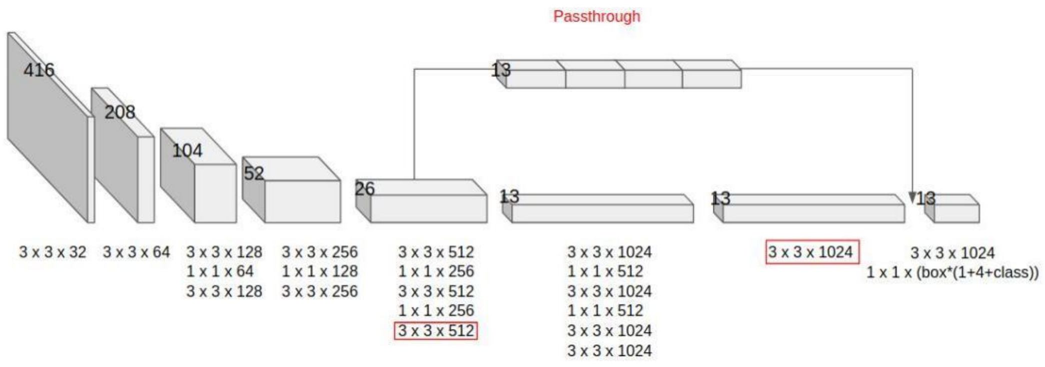


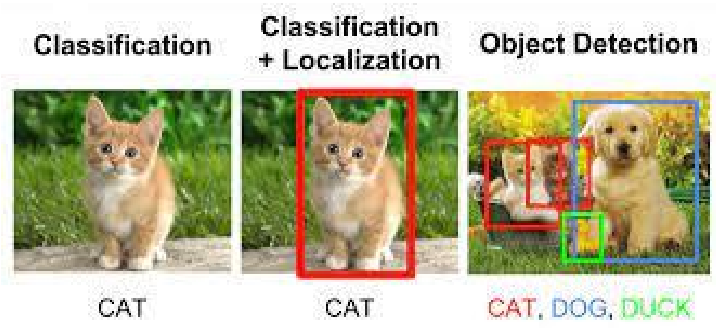
Fig: Architecture of YOLOv2

VR MINI PROJECT REPORT

Q3d: Object Detection(Faster RCNN + YOLO V2) & Object Tracking(SORT + Deep SORT)

Ans:

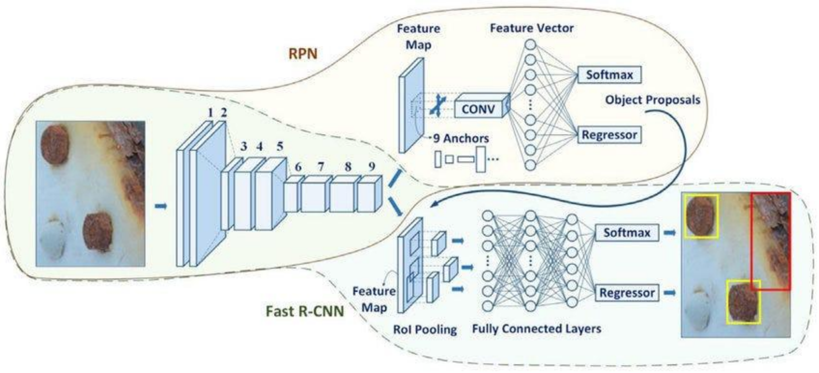
Object Detection: Object detection is a computer vision technique that involves detecting and locating objects within an image or video. The goal of object detection is to identify and localize objects within an image or video and provide information about their class and position. This can be used for a variety of tasks, including surveillance, autonomous driving, image and video analysis, and robotics.



Here in this project for object detection we used two specific methods,namely **Faster RCNN(Region Proposal Based) & YOLO(Dense Sampling Based).**

Faster RCNN:Faster R-CNN is a popular object detection algorithm that was introduced by Ross Girshick in 2015. It is an extension of the earlier R-CNN and Fast R-CNN algorithms and is considered to be one of the state-of-the-art object detection models.

Faster R-CNN is a two-stage detection algorithm, based on a region proposal network (RPN) that shares convolutional layers with the detection network. The RPN generates region proposals by sliding a small network over the convolutional feature map. The network predicts whether each anchor box should be classified as an object or background, and also generates refined bounding box coordinates for each proposal.



Architecture Of Faster RCNN

After generating region proposals, Faster R-CNN passes them through a detection network, which performs object classification and bounding box regression on the proposed regions. The detection network uses features generated by the shared convolutional layers and combines them with the proposals to classify and localize objects.

Compared to its predecessors,(R-CNN & Fast RCNN) Faster R-CNN is faster and more accurate, thanks to the use of a single, unified network for region proposal and object detection. This architecture also allows for end-to-end training, which further improves performance. As a result, Faster R-CNN has become a popular choice for a wide range of applications, including autonomous vehicles, robotics, and surveillance systems.

For being a two stage detection framework, faster RCNN is more robust in performance but slower.

YOLO: YOLO (You Only Look Once) is a one-stage state-of-the-art real-time object detection system. The first version of YOLO was released in 2015 and quickly gained popularity due to its high speed and accuracy.YOLOv2 was released in 2016 and improved upon the original model by incorporating batch normalization, anchor boxes, and dimension clusters. YOLOv3 was released in 2018 and further improved the model's performance by using a more efficient backbone network, adding a feature pyramid, and making use of focal loss.In 2020, YOLOv4 was released which introduced a number of innovations such as the use of Mosaic data augmentation, a new anchor-free detection head, and a new loss function.

In 2021, Ultralytics released YOLOv5, which further improved the model's performance and added new features such as support for panoptic segmentation and object tracking.

YOLO uses a single neural network that simultaneously predicts the bounding boxes and class probabilities for each object in an image by integrating region proposals and detection by acting on a dense sampling of possible locations. It divides the input image into a grid of cells and predicts object bounding boxes and class probabilities for each cell.This means that YOLO can detect multiple objects in a single image in a single forward pass through the neural network, making it very fast.

For being a one-stage detection framework it is simple and fast but performance wise not as good as Region Proposal-based methods like Faster R-CNN.

Object Tracking : Object tracking is a computer vision technique that involves identifying and following the movement of objects in a video sequence or a series of images. The goal of object tracking is to locate and track the position, orientation, and motion of one or multiple objects over time, even as they move across the camera's field of view or change appearance due to lighting, occlusion, or other factors.Object tracking is a critical component in many applications, including surveillance, robotics, autonomous vehicles, human-computer interaction, and sports analysis, among others.The prerequisite of object tracking is object detection. Tracking can be accomplished using a variety of methods. Here in this project we basically used two methods **SORT and DeepSORT.**

SORT : SORT (Simple Online Realtime Tracking) is a popular algorithm for multi-object tracking in videos. SORT is a simple and efficient algorithm that is capable of handling large numbers of objects in real-time. The SORT algorithm first detects objects in each frame of the video sequence using an object detector. Then, it assigns a unique identity to each object based on its location and appearance. Then, it uses a Kalman filter to predict the location of each object in the next frame, and then associates the predicted locations with the detected objects to track them over time if the overlap (IOU) between predictions (based on last frame) and actual detections crosses a certain thresold.Finally, a matrix holds the IOUs and a Hungarian algorithm associates tracks and detections.When a new detection has an IOU below a specific threshold it is not assigned to an existing track but classified as a new object. Tracks are removed when no detection is assigned for a certain number of frames. That helps to avoid unbound growth.

DeepSORT: Deep Learning based SORT is an extension of SORT that incorporates deep learning techniques to improve the tracking performance.The authors of DeepSORT argue that SORT creates too many “identity switches” when the sight of objects is blocked. Therefore, they propose to enrich the motion model (Kalman filter) with a deep learning component that incorporates the visual features of an object.DeepSORT uses a deep neural network to extract feature embeddings of each object in the video sequence, which are then used to associate objects across frames. The feature embeddings are learned using a Siamese network that is trained to distinguish between objects based on their appearance. DeepSORT also uses a gating mechanism to filter out noisy detections and improve the accuracy of the tracking. The combination of SORT and deep learning techniques in DeepSORT has shown to improve the tracking performance significantly in complex scenarios, such as occlusions, crowded scenes, and long-term tracking.

The main differences between SORT and DeepSORT are:

- **Feature extraction**: SORT uses hand-crafted features, such as color histograms, HOG features, and deep features, to represent objects. In contrast, DeepSORT uses a deep neural network to extract feature embeddings that capture more fine-grained details of the object's appearance.
- **Data association**: SORT uses a simple distance-based matching approach to associate object detections across frames, whereas DeepSORT uses a learned metric to associate objects based on their feature embeddings.
- **Robustness**: DeepSORT is generally more robust than SORT because it can handle occlusions, re-appearances, and appearance changes more effectively.
- **Accuracy**: DeepSORT is generally more accurate than SORT because it can capture more subtle differences in object appearance and can better handle cluttered scenes.
- **Computational complexity**: DeepSORT is more computationally expensive than SORT because it involves training a deep neural network, which requires significant computational resources.

Implementation Details:

In our project we used pretrained(on 80 class COCO dataset) Faster R-CNN from pytorch , which uses RESNET50 as a backbone for object detection. We combined it with SORT algorithm (official implementation in '**https://github.com/abewley/sort**') and DeepSORT (from **deep-sort-realtime** library). We applied them on 3 different videos taken by us in Electronic City.
We also used YOLO V2 for object detection. We downloaded the configuration file, pretrained weights(on coco dataset) and classes file (which is a .txt file containing 80 classes same as coco classes) from '**https://pjreddie.com/darknet/yolo/**' website and used OpenCV's DNN module to configure the YOLO V2 architecture ran inferences on our videos by extracting each frame at a time and prepared them as a blob using:

blob = cv2.dnn.blobFromImage(frame, scale, (416,416), (0,0,0), True, crop=False)
Where scale = 0.00392 or 1/255

and applying the algorithms on them. After detecting Objects we used SORT and DeepSORT to track the objects in our videos.While drawing bounding boxes on detected objects we used a thresold value of 0.8 to ignore weak detections.Even though we ignored weak detections, there will be lot of duplicate detections with overlapping bounding boxes. We used **Non-max suppression** to remove boxes with high overlapping using **cv2.dnn.NMSBoxes()** function **with nms thresold = 0.4** .

As per the requirement of the project we implemented **car counter** for our videos. At first we manually created **ground truths** (i.e. the no. of cars present in our videos) for our videos. Then we ran inference using above detection and tracking combinations . Using each combination we only detected and tracked cars in our videos and created a list which contains the all tracking ids of the various cars detected in our videos. Then we converted that list into a set in python and took the length of that set which basically tells us the no. of unique car ids or cars present in the video.
We also tried object detection using YOLOV3 and applied tracking DeepSORT on top of it.

- ❖ We attached all the ipython notebooks, which are self explanatory.
- ❖ We also attached input and output video files.

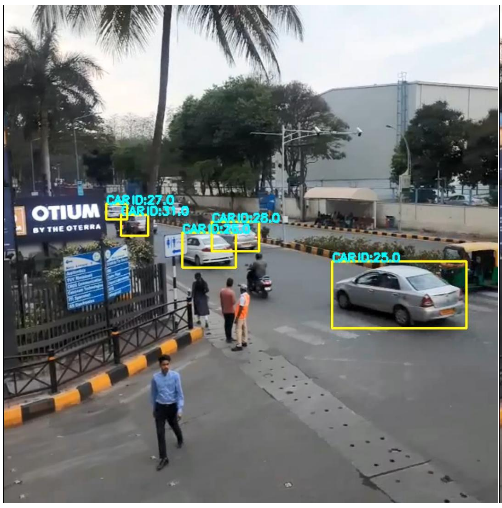
Results:

Input	length	FPS	Groud Truth (#car)	Faster R-CNN & SORT		Faster R-CNN & DeepSORT				YOLO V2 & SORT		YOLO V2 & DeepSORT			
				Inferen Time	Predicted #cars	Inferenc eTime n_iter 5	10	Predicted #cars n_iter 5	10	Inference Time	Predicted #cars	Inference Time n_iter 5	10	Predicted #cars n_iter 5	10
Video1	18s	30	16	24mins	52	44 m	26 m	59	32	4m23s	31	12m 25s	6m	36	24
Video2	15s	30	14	15m34s	41	22 m	15 m	47	36	1m21s	14	8m	4m4 5s	15	14
Video3	16s	30	5	22m58s	13	25 m	18 m	13	7	1m 37s	24	8m1 4s	5m	20	7

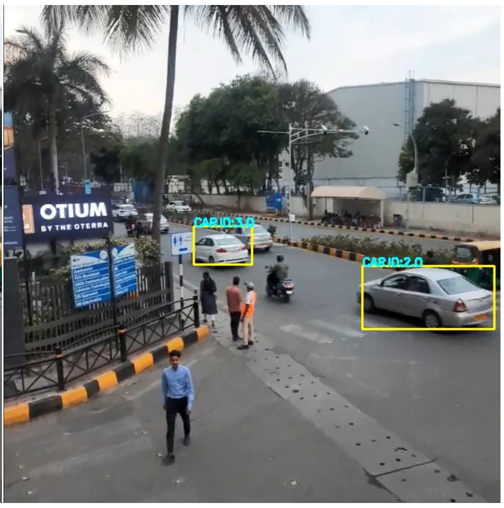
Observations:

1. We can see that Faster R-CNNs generally taking way longer time for inference than YOLO V2 based approaches. This is because YOLOv2 is a single-shot detector, meaning that it performs both object localization and classification in a single forward pass through the network. In contrast, Faster R-CNNs use a two-stage approach where region proposals are first generated before object classification and localization is performed. This two-stage process can be computationally expensive and result in slower processing times. So in most of the real time applications we prefer YOLO more than faster R-CNN.

2. On the other hand,Faster R-CNN is more accurate than YOLO, especially in scenarios where there are small objects or multiple objects in close proximity. In contrast, YOLO may struggle to accurately detect small objects or objects that are close together.



Car Detection With Faster R-CNN



Car Detection With YOLO V2

Here we can see that Faster R-CNN can accurately detect small objects and objects that are close together, as the RPN can generate proposals at different scales and aspect ratios to capture a wide range of object sizes and shapes. Where as single stage object detection algorithm, YOLO struggles to detect small objects or objects that are close together, as it relies on the grid cells to capture the objects. Moreover, YOLO also has a fixed grid size, which can limit its ability to detect objects of different sizes.That’s why Faster R-CNN is more preferable when accuracy is the concern.

3. We can also see that deepsort based tracking approaches generally taking longer time than sort based ones for with same detection algorithms as DeepSORT involves training a deep neural network, which requires significant computational resources and time.

4. For video1 , we can see that faster R-CNN based approaches predict high values for no of cars like 52 and 59 where YOLO V2 based approaches predict lower values like 31 and 36 where ground truth is much smaller 16.As faster R-CNN's two-stage approach can lead to higher accuracy, but it can also be more prone to false positives, which could be one reason why the predicted number of cars is higher.On the other hand, the YOLO V2 based approaches are detecting fewer false positives but also missing some of the true positives which results into fewer car detections than faster RCNN.

5. Faster R-CNN + SORT predicts less cars (52) than faster R-CNN + DeepSort(59) method for video1. DeepSORT is an extension of SORT that adds deep appearance features to the tracking algorithm. It uses a deep neural network to extract appearance features of objects and uses these features to match detections across frames. This allows DeepSORT to handle more complex scenarios such as occlusions and track objects more accurately.Therefore, it's possible that the Faster R-CNN + DeepSORT method is able to detect and track more objects accurately than the Faster R-CNN + SORT method, leading to a higher predicted number of cars. **However, different hyperparameters and different implementations of these algorithms can affect their performance on this specific video.**Like in our DeepSort tracker we used max_age = 5 i.e. maximum no .of frames a object could disappear before reappearing on the frame with same id , making it a higher value could result into detection of less no of cars which discards the possibility of false positives but can also miss out true positives and n_init =1
i.e. minimum no. of frames a object has to appear before getting assigned a track_id. Making it lower can result into more unique car ids. When we made n_iter =10 we got no. of cars = 24 which is closer to ground truth for video 1 using YOLOV2 + DeepSort.

6. When our detectors and tracker faced occlusion multiple times (like in our first video) it resulted into assigning more unique ids to the same object appearing in different frames with a gap of frames, which resulted into huge no. of predicted cars in case of our video1 & video2. But in our video 3 there wasn't a lot of occlusion so the predicted no. of cars is lot closer to the ground truth.

<u>Conclusion</u>: We clearly see that tuning hyperparameters like max_age, n_iter of DeepSORT Tracker give us better result in terms of inference time and no. of predicted cars. Another importtant thing we noticed that when there is less occlusion in our video we get better result in terms of car counting.

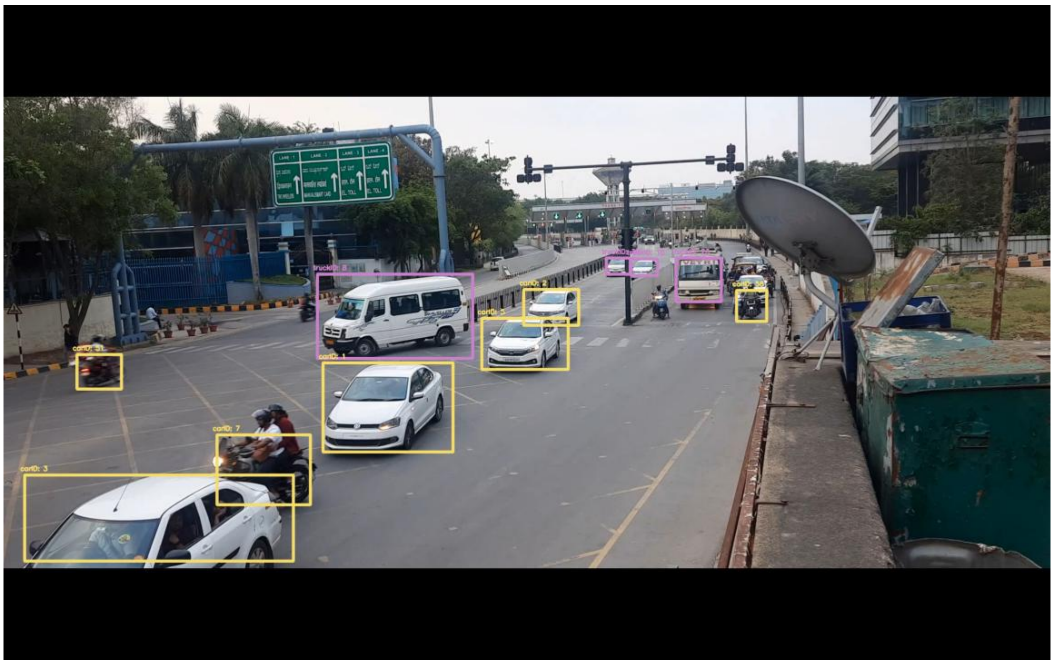
Today YOLO is the state of the art object detection algorithm and DeepSORT is the state of the art object tracking algorithm. Ultralytics YOLOv8 is the latest version of the YOLO (You Only Look Once) object detection and image segmentation model developed by Ultralytics. It provides backward compatibility to all the previous versions of YOLO. YOLOv8 include a new backbone network, a new anchor-free detection head, and a new loss function. YOLOv8 is also highly efficient and can be run on a variety of hardware platforms, from CPUs to GPUs.

So today the best possible combination of detection and tracking algorithm is as per our knowledge YOLO V8 + DeepSORT. We will explore that in future.

<u>Link for i/p and o/p video files</u>: Videos

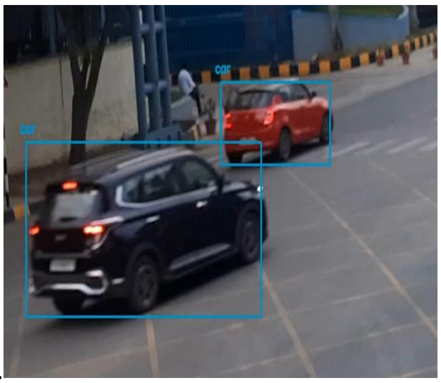
Some more observations on YOLO V3 & SORT

YOLOv3 on Traffic video:



Observations:

We observe that YOLO is fast as compared to other algorithms but the results suggest that it’s accuracy hampers in the case of occlusions and similar feature of 2 different objects. When there are 2 or more objects very nearby, we see that the object labels jump from one bounding box to other as it fails to distinguish between the 2 in 2 different frames. Below 2 images show how the label of truck and car changes from frame to frame for the same object



.

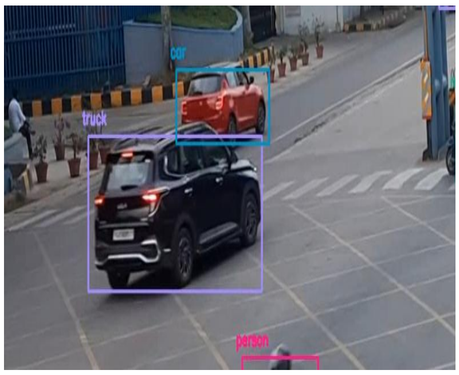


Fig: Same object is classified as car and truck in 2 different frames

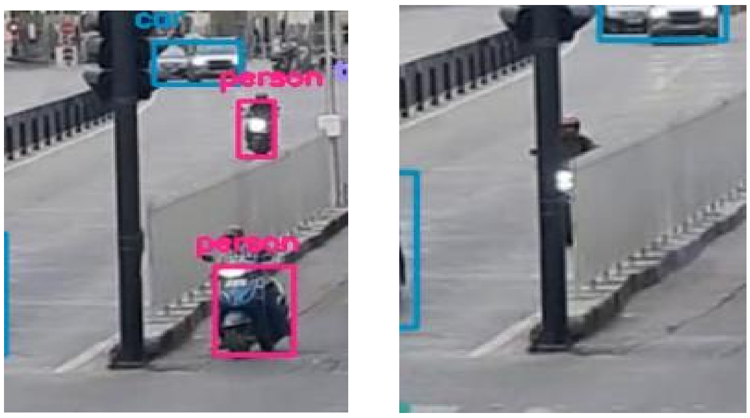


Fig: Person detected in the first scene is not detected due to occlusion in the second scene.

This flaw is more visible when we apply SORT on the detection of YOLO. We can see how the class labels jumps from one box to the other due to vicinity of similar feature object nearby.

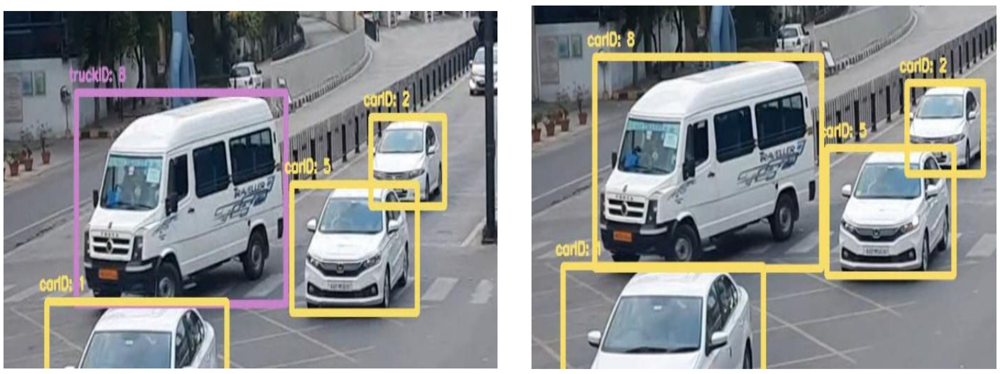
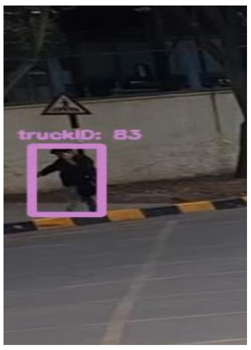


Fig: truck in first frame classified as car in the second.

We also observe that misclassification is a big issue with SORT. Though a ingenious algorithm, it lacks in accuracy due to various reasons. It uses Kalman filter which assumes the movement of the objects to be linear. The SORT implementation we used(**abewley/sort** from github) didn't give similar output length as the input of the detected bounding boxes but the number of classes remain the same which makes is prone to misclassification. In the below image, a person is misclassified as a truck.


truckID: 83