International Institute of Information Technology
Bangalore

Visual Recognition
AI 825

Assignment 3 - Mini Project

March 22, 2023

Team Members		
Name	***Roll No***	***Email***
Kedar Deshpande	IMT2020523	Kedar.Deshpande@iiitb.ac.in
Aakash Khot	IMT2020512	Aakash.Khot@iiitb.ac.in
Darpan Singh	IMT2020133	Darpan.Singh@iiitb.ac.in

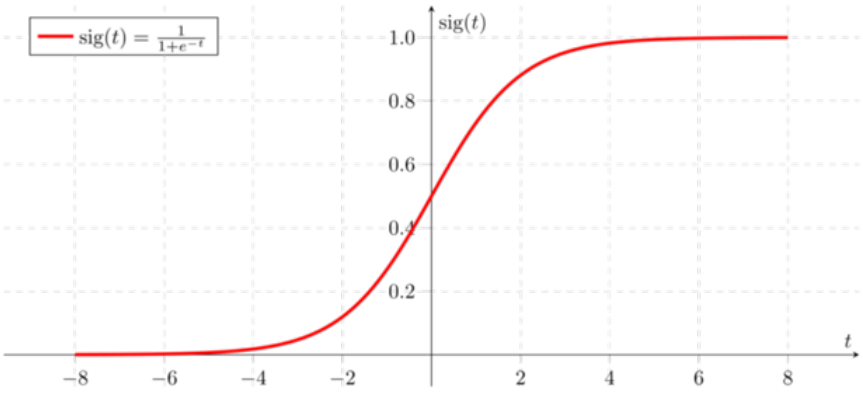
Assignment Part 3a :-

As per question, we will be using the CIFAR-10 dataset for our CNN architecture.

The CIFAR-10 dataset is a collection of 60,000 (50,000 Training images and 10,000 Test images) - 32x32 color images in 10 classes, with 6,000 images per class. The 10 classes are Airplane, Automobile, Bird, Cat, Deer, Dog, Frog, Horse, Ship, and Truck.

Activation functions:- Activation function is used to generate or define a particular output for a given node based on the input is getting provided. We will be using 3 main types of activation functions used – Sigmoid, Tanh and ReLU.

1) **Sigmoid function** – The sigmoid function is known as the logistic function which helps to normalize the output of any input in the range between 0 to 1.

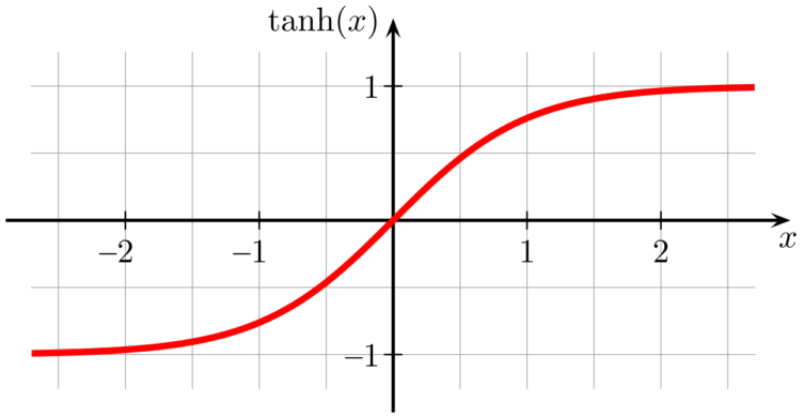


The equation of the sigmoid activation function is given by:

$$y = 1/(1 + e^{(} - x))$$

Drawbacks
- This is the Non-zero Centered Activation Function.
- Model Learning rate is low.
- Create a Vanishing gradient problem.

2) **Tanh function** – Tanh Activation function is superior then the Sigmoid Activation function because the range of this activation function is higher than the sigmoid activation function i.e. -1 to 1. (Zero centered activation function)



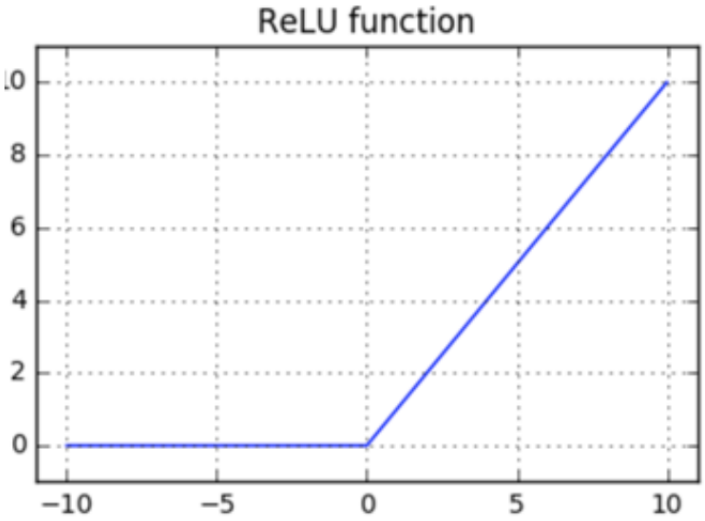
Equation of the tanh activation function is given by:

$$y = tanh(x)$$

Drawbacks
- Create a Vanishing gradient problem.

3) **ReLu (Rectified Linear Unit) function** –
Equation –

$$Y = x \text{ if } x > 0 \text{ else } 0$$



The range is 0 to infinity so Maximum Threshold values are Infinity, so there is no issue of Vanishing Gradient problem so the output prediction accuracy and their efficiency are maximum. Speed is fast compared to other activation functions.

We will be trying out all of these three activation functions, results of these will be given further in the report.

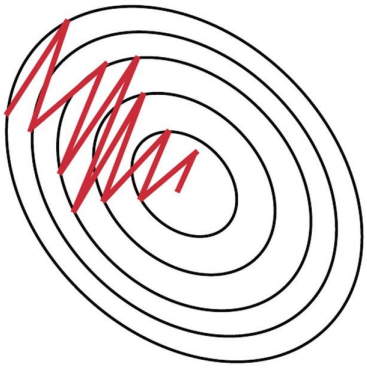
SGD with Momentum :-

Momentum is a technique used in stochastic gradient descent (SGD) to speed up the training process and avoid getting stuck in local minima.
In SGD with momentum, the update rule for each weight parameter is given by:

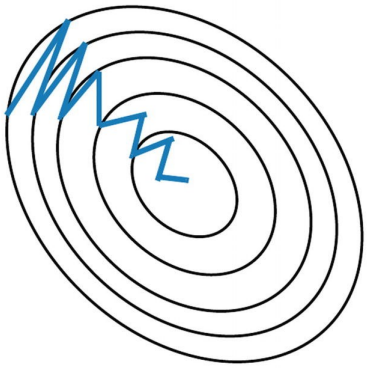
$$V_t = \beta V_{t-1} + \alpha \nabla_w L(W, X, y)$$
$$W = W - V_t$$

Here beta is the momentum coefficient, alpha is the learning rate which is multiplied by the gradient of loss function L with respect to weights, and Vt is the velocity of the weight update at time step t.

The momentum coefficient controls the weight given to the previous velocity compared to the current gradient. A value of gamma close to 1 gives more weight to the previous velocity, while a value close to 0 gives more weight to the current gradient.
When the gradient gets computed every iteration, it can have totally different direction and the steps make a zigzag path, which makes training very slow. To prevent this from happening, momentum stabilizes this movement.



Stochastic Gradient Descent **withhout** Momentum



Stochastic Gradient Descent **with** Momentum

Drawback –

This can also introduce overshooting, where the optimizer oscillates around the minimum instead of settling down. So for this, we have to choose the momentum coefficient and learning rate carefully.

Adaptive Learning rate:-

Adaptive learning rate methods in stochastic gradient descent (SGD) adjust the learning rate during training based on the behavior of the loss function.

One of the most popular and effective methods for this is the Adam optimizer which is a combination of SGD with momentum and RMSProp. It is an adaptive learning rate optimization algorithm that combines the benefits of both momentum-based optimization and root mean square propagation (RMSProp) optimization.
The algorithm calculates an exponential moving average of the gradient and the squared gradient and then adjusts the learning rate based on these moving averages.
Update rule –

$$\theta t+1 = \theta t - \alpha * (mt / (\sqrt{vt} + \varepsilon))$$

θt is a vector of parameters at time step t, alpha is the learning rate, mt is the exponentially weighted average of the gradient at time step t, vt is the exponentially weighted average of the squared gradient at time step t,ϵ is a small constant to avoid division by zero. The exponentially weighted moving averages of the gradient and squared gradient are calculated as –

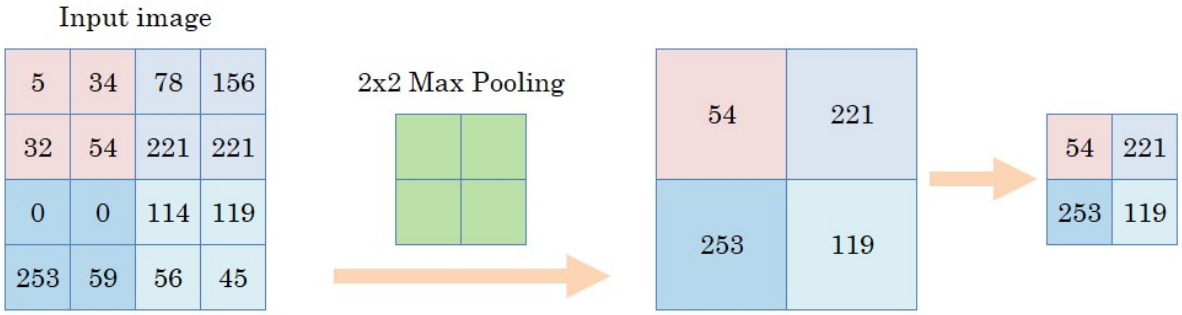
$$mt = \beta 1 * mt-1 + (1-\beta 1) * gt$$
$$vt = \beta 2 * vt-1 + (1-\beta 2) * gt\text{^}2$$

gt is the gradient at time step t, β_1 and β_2 are the exponential decay rates for the moving averages of the gradient and squared gradient. The values of β_1 and β_2 are typically set to 0.9 and 0.999, respectively.

Max-pooling :-

Pooling – Enhancing the power of convolutions.
This operation is used in convolutional neural networks (CNNs) used to downsample feature maps.
Max pooling is typically applied after one or more convolutional layers in a CNN. The operation divides the input feature map into a set of non-overlapping rectangles, called pooling regions, and outputs the maximum value within each pooling region. (The one in which we take an average of samples is called average pooling).

Most common size – 2*2



Batch Normalisation :-

Batch Normalization (BN) is a technique in deep learning used to normalize the inputs of each layer in a neural network. BN introduces an additional layer to the neural network that performs operations on the inputs from the previous layer.

The operation standardizes and normalizes the input values. Batch normalization was performed as a solution to speed up the training phase of deep neural networks through the introduction of internal normalization of the input values within the neural network layer.

Batch normalization has several benefits, including improving the training speed, making the model more robust to different initialization methods, and reducing the need for regularization techniques such as dropout.

Results :-

We have used a Neural Network with 3 convolution layers, 3 fully connected layers, and max pooling, and batch normalization. (Exact architecture can be seen in the code)
Reference -Pytorch CIFAR10 tutorial
Learning rate=0.001, momentum (if used) =0.9, batch size=4, epochs=20
The following table describes our results:

Activation	Momentum	Training Time	Test Accuracy
ReLu	No	1327 sec	69%
ReLu	Yes	1461 sec	70%
Tanh	No	1316 sec	64%
Tanh	Yes	1382 sec	66%
Sigmoid	No	1311 sec	64%
Sigmoid	Yes	1390 sec	66%

Without batch normalization but with momentum, ReLu gave 61%, tanh gave 63%, and sigmoid gave 10%. Hence sigmoid could not learn in just 20 iterations.
With adaptive learning rates (Adam optimizer):

Activation	Training Time	Test Accuracy
ReLu	1696 sec	69%
Tanh	1685 sec	62%
Sigmoid	1693 sec	60%

As seen even with using Adam optimiser there is not much of a difference.

We tried a different types of architectures and the best we got which takes adequate time with 5 convolution layers and 3 fully connected layers. One can view the architecture in the ipynb file of this question. Briefly, there is a Maxpool layer after the first convolution layer and there is a Batch Normalization layer between two consecutive convolution layers. Updated batch size = 128 and the rest values are the same.

Activation	Momentum	Training Time	Test Accuracy
ReLu	No	245 sec	73%
ReLu	Yes	245 sec	77%
Tanh	No	254 sec	65%
Tanh	Yes	267 sec	66%
Sigmoid	No	245 sec	63%
Sigmoid	Yes	247 sec	73%

We can observe from the above table that ReLu gives the best accuracy of 77% than all other activation function and momentum give a significant change of accuracy. With respect to Training time, all have approximately similar finishing times though ReLu took less time than others.
With Adam Optimiser -

Activation	Training Time	Test Accuracy
ReLu	251 sec	78%
Tanh	253 sec	61%
Sigmoid	253 sec	65%

The highest Accuracy got from the ReLU activation function using Adam optimizer on the given Architecture is **78%**.

Assignment Part 3b :-

The dataset used - Agricultural crops image classification from kaggle
Link to dataset - https://www.kaggle.com/datasets/mdwaquarazam/agricultural-crops-image-classification

About Alexnet -
AlexNet is a convolutional neural network (CNN) architecture consisting of eight layers, including five convolutional layers, two fully connected layers, and a softmax layer.

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

AlexNet can be used as a feature extractor by removing its last fully connected layer and using the output of the previous layer as a fixed-length feature vector for classification. This approach is also called as transfer learning.

Since AlexNet was trained on the ImageNet dataset, which contains a large number of diverse images from 1,000 different classes, the features learned by the CNN layers can be useful and used as feature extractor for different classification tasks.

By using the pre-trained AlexNet as a feature extractor, we can avoid the need to train a large neural network from scratch on a new dataset, which can be computationally expensive and require a large amount of labeled data. Instead, we can use the pre-trained AlexNet to extract features from the new dataset and train a classifier on top of these features such as Logistic regression or SVM etc.

In Kaggle, We found a dataset called agricultural crops consisting of 30 type of Agriculture Product Crop images of maize, rice(Paddy),sugarcane, etc with approximately 30 images per class.

Firstly we extracted the data from the agricultural crop dataset converting each image to RGB and transforming it to the appropriate standards given in pytorch documentation.

Next we loaded a pretrained alexnet model from pytorch with pretrained parameter set to True. Then we extracted the features using the pre-trained model and ran different classification models on top of it to report the accuracies. Here Training data is 80% of total data and Testsing data is 20%.

Model	Training Accuracy	Testing Accuracy
Logistic Regression	100%	69.3%
SVM	100%	68%
KNN	28.6%	29.5%
Gaussian NB	99.6%	34.3%

As observed, the highest testing accuracy is around 70%. This is because of the high number of classes i.e. 30 and the low number of images per class. We also tried to fine tune the alex net model but there were no significant increase of accuracy.

Now Let us check this feature-extracting method on Bike vs Horse dataset. Same procedure is followed for to find the accuracies by extracting features using alexenet and putting it on top of different classifiers.

Model	Training Accuracy	Testing Accuacy
Logistic Regression	100%	100%
SVM	100%	100%
KNN	98.6%	100%
Gaussian NB	100%	75%

As observed, we are getting maximum 100% accuracy. This higher percentage is because of only 2 categories and total of 179 images as training or testing data.

Assignment Part 3c :-

YOLO V1 - "You Only Look Once: Unified, Real-Time Object Detection," is a popular object detection algorithm introduced by Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi in 2016.

It is a real time object detection algorithm that can detect multiple objects in an image in a single pass through a convolutional neural network. The algorithm divides the input image into a grid of cells and for each cell, predicts bounding boxes and class probabilities of objects within that cell.

YOLO v1 uses a CNN architecture that is composed of 24 convolutional layers followed by

two fully connected layers. The network predicts bounding boxes using a combination of a grid cell.

The grid cell that contains the centre of the object is responsible for detecting that object. Each grid cell predicts 2 bounding boxes. The predicted bounding boxes are then mapped to the original image space.
The algorithm also uses a technique called non-maximum suppression to eliminate duplicate bounding boxes and to select the best bounding box for each object.

Some of the Features of YOLO v1 –
1. **Fast Processing** of an entire image with a single forward pass through the network.
2. High Accuracy in object detection.
3. Ability to detect multiple objects in a single image.
4. End-to-end training.

YOLO v2 – version 2 is an improved version of the original YOLO object detection algorithm.

Additional features/Improvements from YOLO v1 :-

1. **CNN Architecture** - One of the main improvements of YOLO v2 over the original YOLO is its deeper neural network architecture, which consists of 19 convolutional layers and 5 max-pooling layers(Use of Darknet-19). This deeper architecture helps to capture more complex features of objects in an image and leads to higher accuracy.

2. Use of anchor boxes - YOLO v2 uses anchor boxes to better localize objects in an image and to handle objects of different sizes and shapes.
Anchor boxes - a set of predefined bounding boxes of a certain height and width. These boxes are defined to capture the scale and aspect ratio of specific object classes you want to detect and are typically chosen based on object sizes in your training datasets.

3. **Batch Normalization** – to normalize the input of each layer, improving training speed and reducing overfitting.

4. **Multi-scale Detection** - YOLO v2 processes images at different scales and incorporates features from each scale to improve detection accuracy.

5. Softmax replacement with logistic activation to improve detection accuracy for overlapping objects.

Assignment Part 3d :-

Object Detection and Tracking using Faster RCNN and YOLO with SORT/DeepSORT.

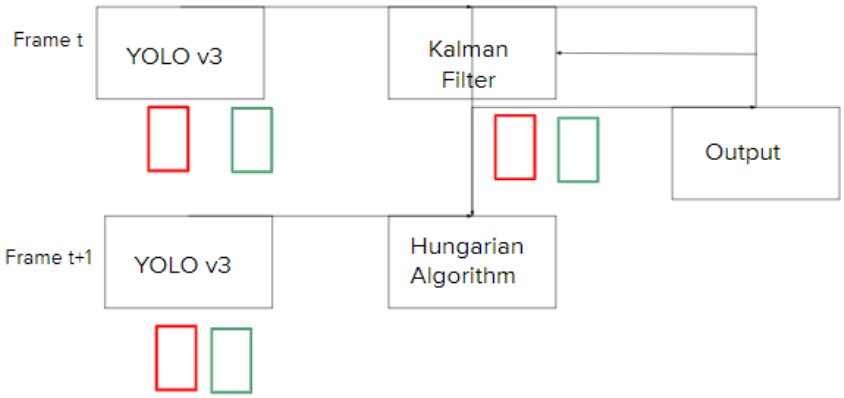
Object detection is a task that involves identifying and localizing objects within an image or a video. For this question, we have recorded some videos from Electronic city and will be depicting results based on that.
Faster R-CNN (Region-based Convolutional Neural Network) and YOLO (You Only Look Once) are two popular deep learning-based approaches for object detection.

Faster R-CNN is a two-stage object detection model that uses a region proposal network (RPN) to generate potential object regions and a region-based CNN to classify and refine those regions. It is known for its high accuracy and robustness, but can be slower than some other object detection models.
The Region Proposal Network (RPN) is a key component of the Faster R-CNN object detection model. It is responsible for generating a set of potential object regions or proposals from an input image. The RPN is a fully convolutional neural network that takes an image feature map as input and outputs a set of object proposals, each represented by a bounding box and a confidence score.

We already know about YOLO from part-3c, briefly YOLO is a one-stage object detection model that simultaneously predicts the bounding boxes and class probabilities of objects in an image or a video. It is known for its speed and real-time performance, but may sacrifice some accuracy compared to more complex models like Faster R-CNN.

Now for Object tracking/detection, we will be using SORT/DeepSORT.

Simple Online Real-Time Tracker (SORT):-
SORT is a real-time object tracking algorithm that is commonly used in computer vision applications. SORT is a tracking-by-detection algorithm, which means that it first detects objects in an image or video frame and then tracks them over time.

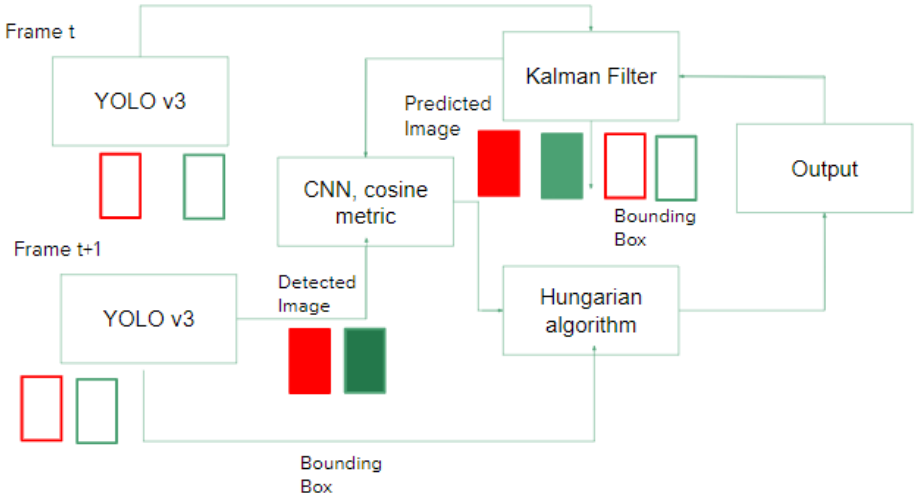


SORT uses a **Kalman filter** to track objects, which estimates the state of the object based on its previous position, velocity, and acceleration.
The Kalman filter is a mathematical model that predicts the future state of an object based on its previous state and the measurements obtained from sensors.

SORT also uses a Hungarian algorithm to match detections to tracks.
The **Hungarian algorithm** is a combinatorial optimization algorithm used in SORT (Simple Online Real Time Tracker) to associate detected objects with existing tracks. The Hungarian algorithm is used to find the optimal match between the detected objects and existing tracks based on a cost function.
In SORT, the cost function is based on the distance between the predicted location of a track and the detected location of an object. The Hungarian algorithm assigns the detected objects to the tracks that minimize the total cost of the associations.

However, there are some limitations of SORT-
-It does not handle occlusions well and can lose track of objects that are temporarily hidden from view.
-Also, SORT does not account for object appearance changes, which can lead to incorrect matches between detections and tracks.

Deep Simple Online Real Time Tracker (DeepSORT):-
DeepSORT (Deep Simple Online Real-Time Tracker) is an extension of the SORT algorithm. The key contribution of DeepSORT is the use of a deep neural network to compute a more discriminative association metric between the existing tracks and the new detections. The DNN takes as input the appearance features of the objects in the bounding boxes, which are extracted using a pre-trained DNN such as ResNet or VGG.



The deep association metric improves the accuracy of data association, especially when dealing with occlusions, similar objects, and fast-moving objects.

In addition to the deep association metric, DeepSORT also introduces a track management module that is designed to handle track fragmentation and merging. The track management module is responsible for maintaining the coherence of the tracks over time by detecting and resolving track fragmentation and merging events.

Key Differences between SORT and DeepSORT:-

1. **Data association metric**: SORT uses intersection over union (IOU) as the data association metric, while DeepSORT uses a deep neural network to learn a more discriminative association metric based on appearance features.

2. SORT does not explicitly model the appearance of objects, while DeepSORT extracts appearance features using a deep convolutional neural network to improve the accuracy of data association.

3. SORT does not require any training, while DeepSORT requires pre-training of a deep neural network such as VGG.

4. DeepSORT includes a **track management module** that is responsible for handling track fragmentation and merging events, which is not present in SORT.

5. DeepSORT has been shown to achieve better tracking performance than SORT, especially in scenarios involving occlusions, similar objects, and fast-moving objects.

6. DeepSORT is more computationally expensive than SORT due to the use of deep learn-

ing techniques and the additional track management module.

Approach

Approach is simple. We detect cars using YOLO/Faster-RCNN and then track those detections using SORT/DeepSORT.
These tracking algorithms will match the detections in one frame with those of the previous frame. For new detections, new IDs are assigned. So with these unique IDs, we can count the number of cars detected. We can see the videos also which shows the IDs and detected cars.

The detections from YOLO/Faster-RCNN will consist of prediction, score, and boxes. For DeepSORT it will also have a feature vector. We will inspect the video frame by frame. After getting detections in a frame, we will do **non-maximum suppression** to get only the best detections. Now these detections will be passed to the tracker and the tracker will be updated. Now we draw the boxes on each frame for each detection, specifying the predicted class and the unique object id. At last we will write all these new frames to a output video.

Observations

All the input and output videos, and the code can be found in this drive link.
Drive Link: https://drive.google.com/drive/folders/1INWduQ9S5O7B7OOGItBIykuaomaOXBDK?usp=share_link
We observed that the car count in each case in not accurate but very close. For the yolov8 and DeepSORT combination, the count is closest(in some videos accurate).

Videos	YOLO+DeepSORT	YOLO+SORT	Faster RCNN+DeepSORT	Faster RCNN+SORT
Video0	33	44	50	60
Video1	7	12	13	16
Video2	5	8	10	14
Video3	3	6	9	11

Actual values: video0-28 video1-7 video2-4 video3-3
These results show that DeepSORT is better than SORT. YOLO performed better at detection than FasterRCNN, as we noticed some false positives for car in our videos with FasterRCNN. More number of cars are getting detected in some cases because the tracker identified the same object as two different objects in different frames. Cars' speed determines the distance between the car in two frames. This can happen due to more distance between the same object in two different frames. Playing around with the thresholds may give better results.

References:-

1. https://www.aitude.com/comparison-of-sigmoid-tanh-and-relu-activation-functions/#:~:text=ReLu%20is%20the%20best%20and,compare%20to%20other%20activation%20function

2. https://datascience.stackexchange.com/questions/84167/what-is-momentum-in-neural-network

3. https://towardsdatascience.com/batch-normalization-explained-algorithm-breakdown-23d2794511c

4.https://www.kaggle.com/code/yogeshrampariya/cifar10-classification-with-alexnet-on-pytorch/notebook

5.https://wikidocs.net/167705#:~:text=In%20YOLO%20v1%2C%20there%20is,made%20in%20parallel%20and%20connected. 6.https://medium.com/@venkatakrishna.jonnalagadda/object-detection-yolo-v1-v2-v3-c3d5eca2312a 7. https://github.com/abewley/sort

8. https://towardsdatascience.com/object-detection-and-tracking-in-pytorch-b3cf1a696a98

9. https://debuggercafe.com/object-detection-using-pytorch-faster-rcnn-resnet50-fpn-v2/

10. https://github.com/nwojke/deep_sort