International Institute of Information Technology, Bangalore

VR 825 - Visual Recognition

Mini Project

(Assignment 3)

Nikhil Singh(MT2023070)

Ankush Patil (MT2023101)

Aakash Bhardwaj(MT2023143)

Github Link :- [VR Assignment3]

Question 1) CNN

Introduction:

Deep learning models have achieved remarkable success in image classification tasks, with AlexNet setting the benchmark with its five convolutional and three fully connected layers. However, for applications with computational constraints, exploring medium deep networks with fewer layers is essential. In this assignment, we focus on the CIFAR-10 dataset and investigate the impact of different activation functions and optimization techniques on training time and classification performance. We compare ReLU, tanh, and sigmoid activation functions, along with variations in optimization methods such as momentum and adaptive learning rates. Our goal is to identify the most effective combination of these factors and propose a recommended architecture that balances model complexity, training time, and classification accuracy.

Model Training Time and Classification Performance

1. **Training Parameters:**

   Activation functions used:

   ReLu, tanh and Sigmoid.

   Training dataset:

   CIFAR-10 batch size 32.

   Wait Initialization:

   xavier_uniform.

   Classification layer activation function:

   logsoftmax.

   GPU Used:

   Kaggle GPU - P100.

   Optimizers:

   Stochastic gradient decent(SGD),

   Stochastic gradient decent(SGD) with momentum (0.9),

   RMSprop,

   RMSprop with momentum(0.9).

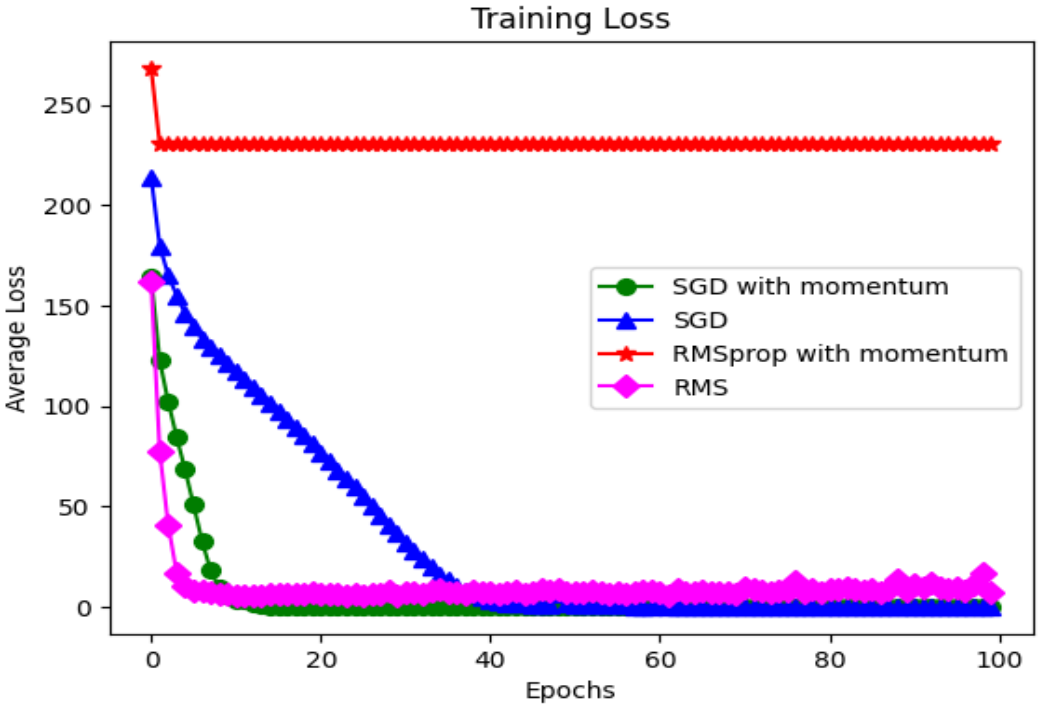
Epoch : 100

2. **Model Summary:**

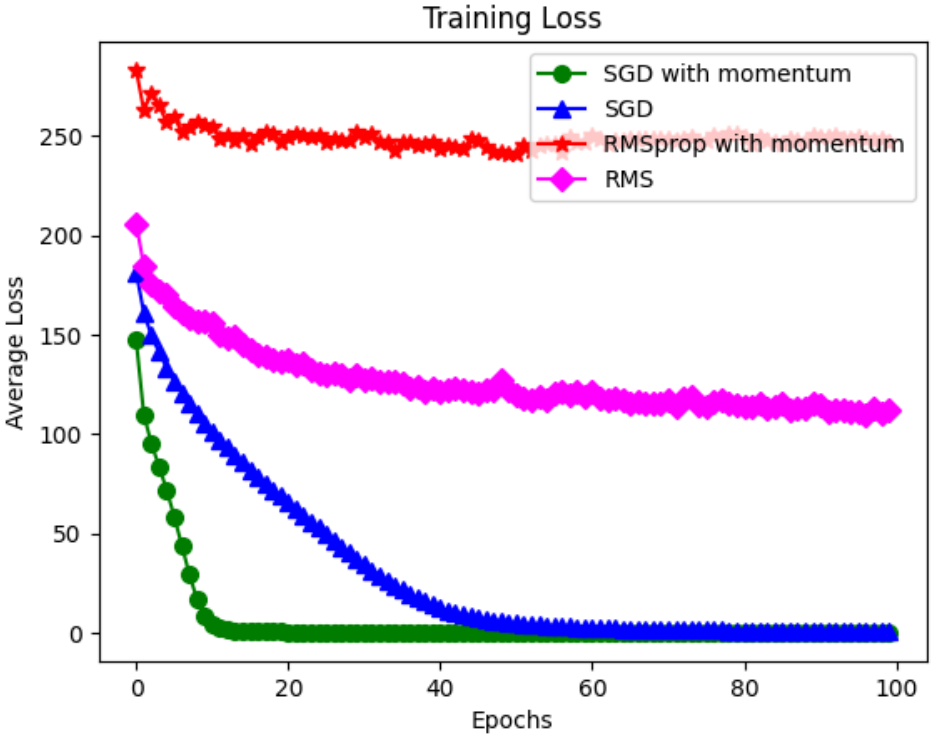
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 32, 32]             896
              ReLU-2          [-1, 32, 32, 32]               0
            Conv2d-3          [-1, 64, 32, 32]          18,496
              ReLU-4          [-1, 64, 32, 32]               0
            Conv2d-5         [-1, 128, 32, 32]          73,856
              ReLU-6         [-1, 128, 32, 32]               0
         MaxPool2d-7         [-1, 128, 16, 16]               0
           Flatten-8              [-1, 32768]               0
            Linear-9               [-1, 1024]      33,555,456
             ReLU-10               [-1, 1024]               0
           Linear-11                [-1, 512]         524,800
             ReLU-12                [-1, 512]               0
           Linear-13                 [-1, 10]           5,130
       LogSoftmax-14                 [-1, 10]               0
================================================================
Total params: 34,178,634
Trainable params: 34,178,634
Non-trainable params: 0
----------------------------------------------------------------
```

3. **Loss For Different Activation**

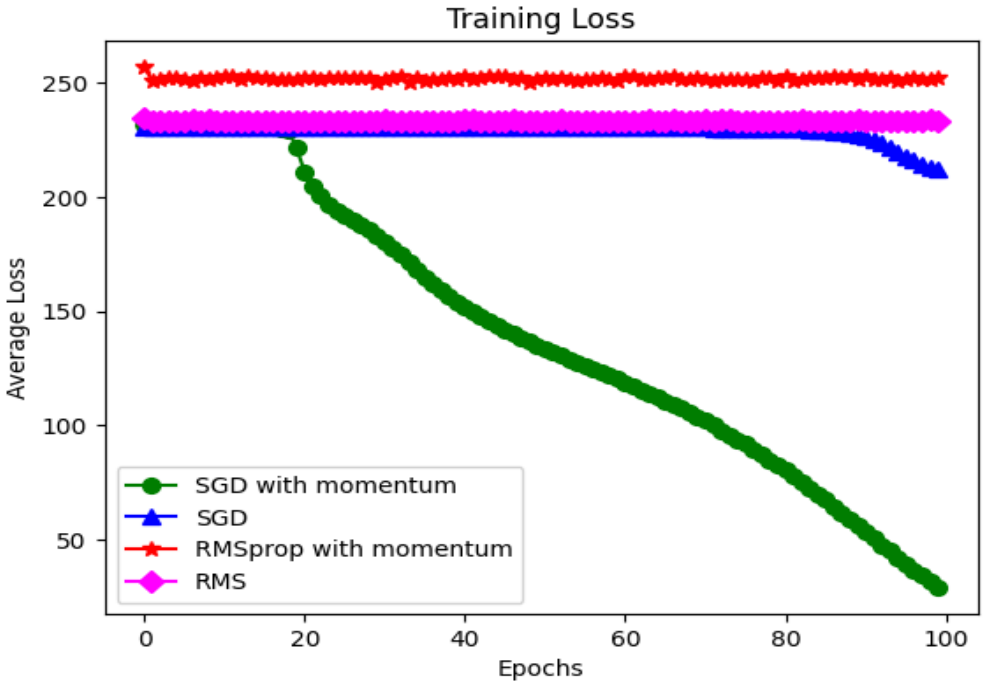
Activation as ReLu:



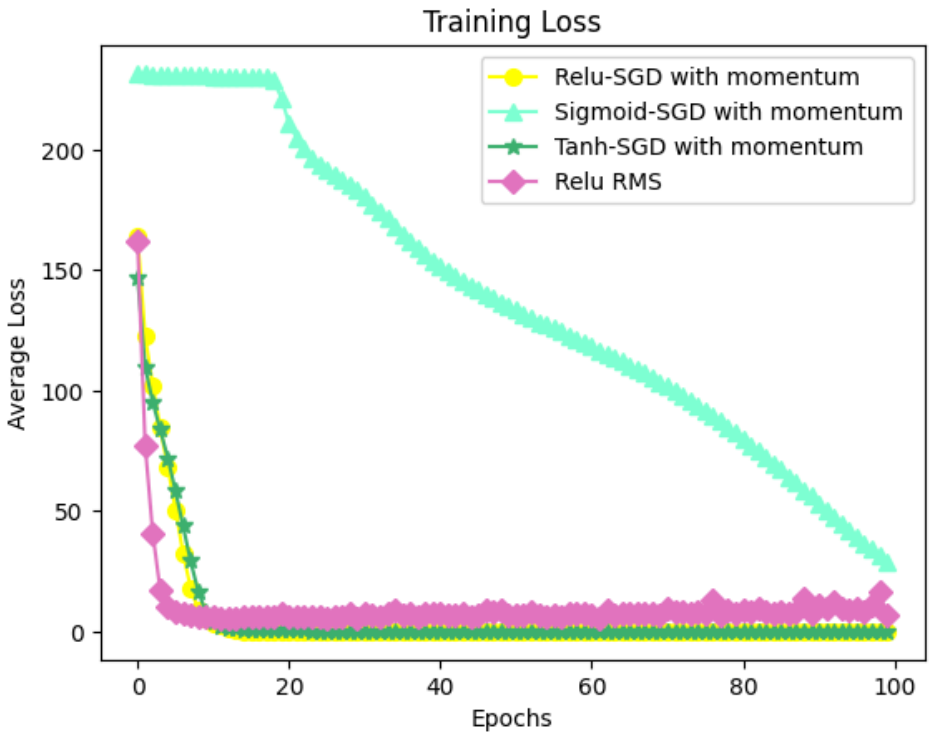
Activation as tanh:



Activation as Sigmoid:



4. Model comparison between different Activation



Classification performance on Test Data :

Activation function	Optimizers	Accuracy
ReLU	Stochastic gradient decent(SGD)	65.84%
	Stochastic gradient decent(SGD) with momentum (0.9)	71.92%
	RMSprop	65.53%
	RMSprop with momentum(0.9)	10.00%
Tanh	Stochastic gradient decent(SGD)	69.98%
	Stochastic gradient decent(SGD) with momentum (0.9)	71.10%
	RMSprop	55.32%
	RMSprop with momentum(0.9)	22.94%
Sigmoid	Stochastic gradient decent(SGD)	25.15%
	Stochastic gradient decent(SGD) with momentum (0.9)	59.22%
	RMSprop	10.00%
	RMSprop with momentum(0.9)	10.00%

Recommended architecture

For the given model recommended architecture would be activation function as ReLu and optimizer as RMSprop which is slightly better than activation as sigmoid or tanh with optimizer as SGD with momentum.

Question 2)CNN as a feature extractor

Introduction:

In this question, we leverage the Kaggle dataset containing seven classes, including images of bikes, cars, horses, cats, dogs, flowers, and humans, to perform object recognition. We employ a transfer learning approach, utilizing AlexNet as a feature extractor, followed by training support vector machine (SVM) and logistic regression classifiers on top of the extracted features. Our aim is to evaluate the classification accuracies of these models and assess their performance in recognizing objects from the given dataset.

Methodology:

Data Preparation: We load the Kaggle dataset and apply transformations to prepare it for classification tasks. The dataset comprises images of seven different classes, each representing a distinct object category.

Feature Extraction with AlexNet: We utilize a pre-trained AlexNet model to extract high-level features from the dataset. AlexNet is a deep convolutional neural network renowned for its effectiveness in image classification tasks.

Training SVM Classifier: We split the extracted features into training and testing sets and train an SVM classifier using the training data. SVM is a powerful machine learning algorithm known for its ability to handle high-dimensional data and perform well in classification tasks.

Evaluation of SVM Classifier: We evaluate the performance of the SVM classifier by predicting labels for the test set and computing the classification accuracy. Additionally, we generate a confusion matrix to analyze the classifier's performance in detail.

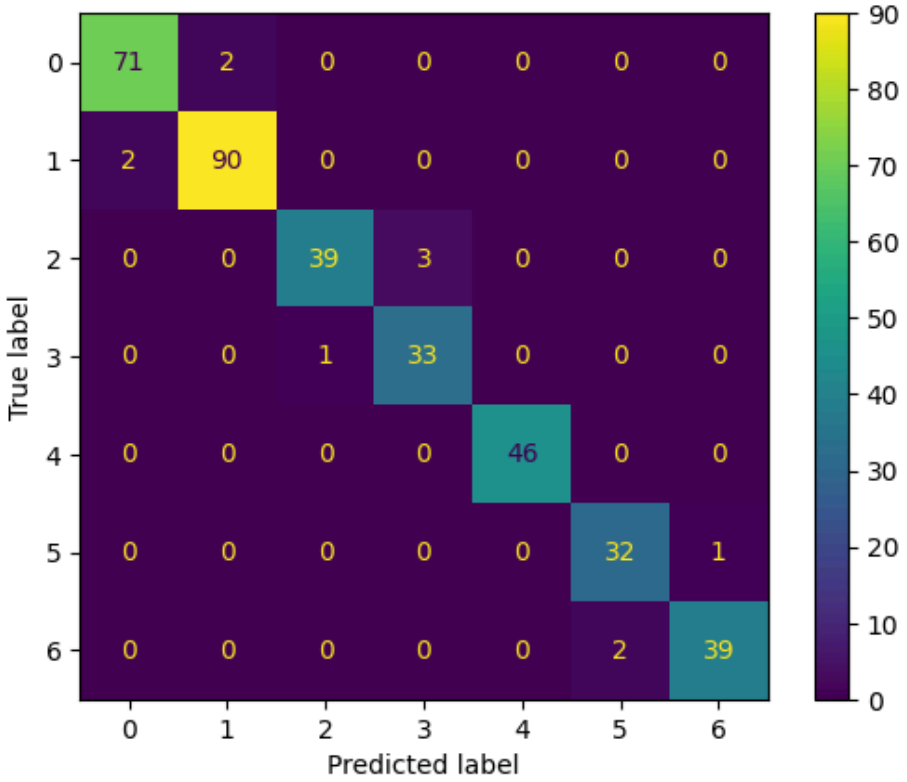
Training Logistic Regression Classifier: Similarly, we train a logistic regression classifier using the extracted features and evaluate its performance in object recognition.

Evaluation of Logistic Regression Classifier: We assess the classification accuracy of the logistic regression classifier and visualize its performance using a confusion matrix.

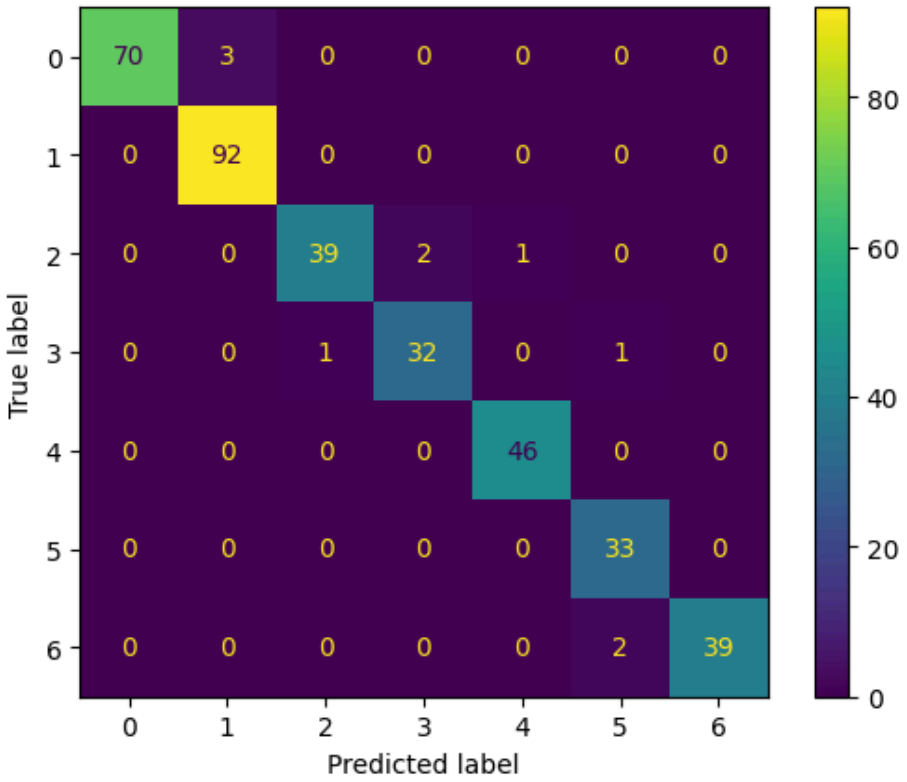
Results and Discussion:

Our experiments gave correct results in object recognition using the Kaggle dataset. The SVM classifier achieves a classification accuracy of 0.96, while the logistic regression classifier achieves 0.97. The confusion matrices provide insights into the classification performance for each class. Overall, the results indicate that both SVM and logistic regression classifiers perform reasonably well in recognizing objects from the given dataset.

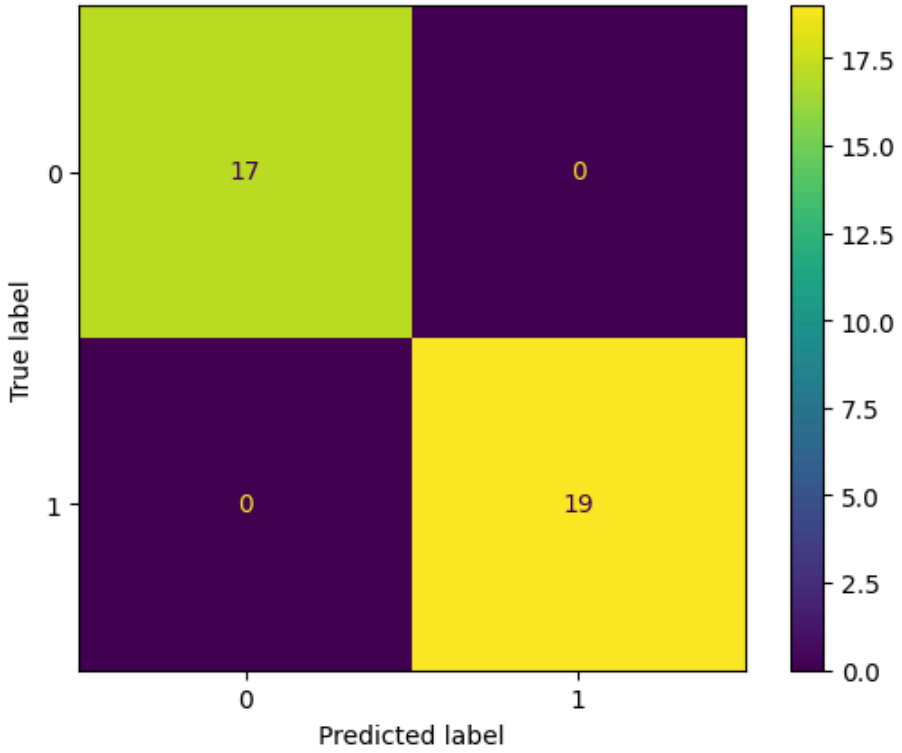
SVM Classifier on Kaggle Dataset



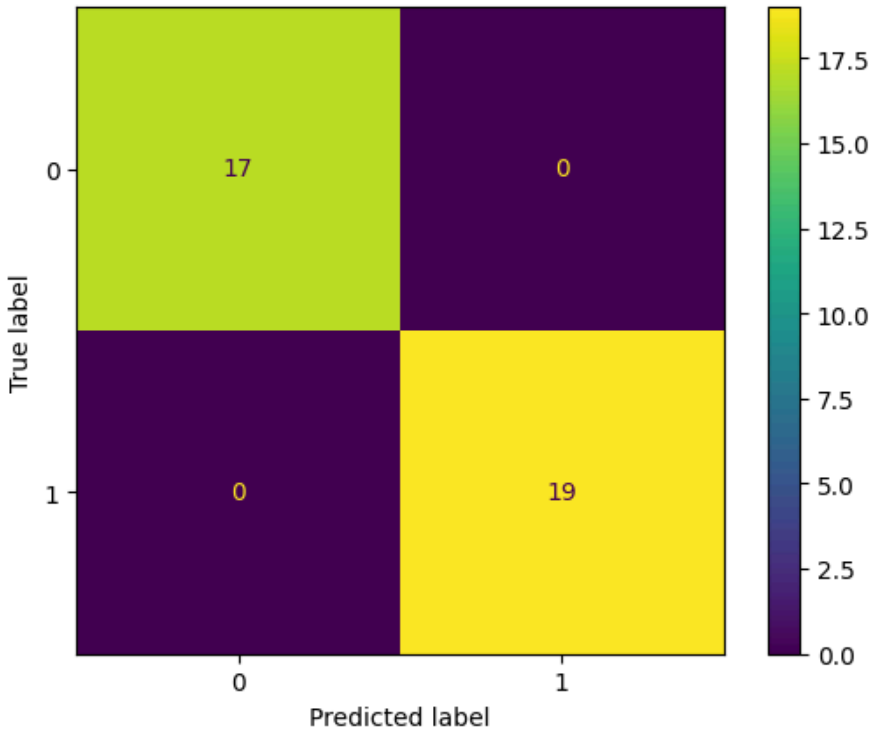
Logistic classifier on Kaggle Dataset



SVM Classifier(Bike, Horse Dataset)



Logistic Regression(Bike, Horse dataset)



Classification Accuracy:

Kaggle Dataset

	Accuracy
SVM	96.95%
LogisticRegression	97.22%

Bike-Horse Dataset

	Accuracy
SVM	100%
LogisticRegression	100%

Conclusion:

In conclusion, our study showcases the effectiveness of using transfer learning with AlexNet features for object recognition tasks. The SVM and logistic regression classifiers trained on top of these features demonstrate competitive performance in classifying objects from the Kaggle dataset

Question 3) Additional features of YOLO v2 compared to YOLO v1

Introduction:

The field of object detection in computer vision has witnessed significant advancements in recent years, with algorithms like YOLO (You Only Look Once) playing a pivotal role in pushing the boundaries of real-time object detection. YOLO v2, an improved version of the original YOLO, introduced several innovative features to enhance accuracy, speed, and robustness in object detection tasks. In this report, we delve into five key additional features of YOLO v2,which impact on the performance of the algorithm.

Now, let's explore five additional features of YOLO v2 and their contributions:

1) **Batch Normalization:**

YOLO v2 incorporates batch normalization layers into its architecture, which helps in stabilizing and accelerating the training process. By normalizing the activations of each layer, batch normalization reduces internal covariate shift, making the network more robust and less sensitive to initialization. This results in faster convergence during training and improved generalization performance.

2) **High-Resolution Classifier:**

Unlike its predecessor, YOLO v2 employs a high-resolution classifier during training. By using a classifier with a higher input resolution, YOLO v2 can capture finer details of objects and learn more discriminative features, leading to improved detection accuracy, especially for small objects in the image.

3) **Anchor Boxes:**

YOLO v2 introduces the concept of anchor boxes to better handle object localization. Instead of predicting bounding boxes directly, the algorithm predicts offsets with respect to predefined anchor boxes of different aspect ratios and scales. This allows YOLO v2 to effectively localize objects of various shapes and sizes, enhancing its versatility and robustness in object detection.

4) **Dimension Clusters for Anchor Boxes:**

In YOLO v2, anchor boxes are generated using k-means clustering on the training dataset to determine appropriate bounding box priors. By clustering the ground truth bounding box shapes, YOLO v2 adapts its anchor boxes to the specific distribution of object sizes and aspect ratios in the dataset, leading to more accurate localization and improved detection performance.

5) **Direct Location Prediction:**

YOLO v2 refines the way bounding box coordinates are predicted by directly regressing the coordinates relative to the grid cell locations. This eliminates the need for explicit anchor box adjustments, simplifying the prediction process and improving localization accuracy. Additionally, YOLO v2 predicts object confidence scores based on Intersection over Union (IoU), providing more accurate objectness estimation.

6) **Fine-Grained Features:**

Incorporating fine-grained features from higher-resolution layers allowed YOLOv2 to detect small objects more effectively, contributing to a 1% increase in mAP. By concatenating these features with the original feature maps, the model gained better object detection capabilities.

7) **Multi-Scale Training:**

YOLOv2 employed multi-scale training, where image dimensions were randomly varied every 10 batches. This technique improved the model's ability to generalize to objects of different sizes, leading to better performance across various scales.

8) **Darknet-19 Architecture:**

YOLOv2 utilized the Darknet-19 classification network, known for its efficiency and balance between accuracy and model complexity. By leveraging 1×1 convolutions to reduce parameters, YOLOv2 achieved faster detection speeds while maintaining competitive accuracy.

9) **YOLO9000 with WordTree:**

YOLO9000 expanded upon YOLOv2 by combining datasets from COCO and ImageNet using a hierarchical tree-based structure called WordTree. This allowed for the integration of diverse classes and improved classification accuracy.

These are the enhancements collectively contributed to better accuracy, stability, and efficiency in object detection tasks compared to YOLOv1.

Question 4) Object tracker: SORT + DeepSORT

Introduction:

In this study, we leverage advanced computer vision techniques to enhance traffic management at junctions. Utilizing collected traffic junction videos, we had employ object detection models such as Faster RCNN and YOLO to identify and localize vehicles accurately. Implemented a car counting system using SORT (Simple Online and Realtime Tracking) and DeepSORT (Deep Learning for Object Tracking).

SORT vs DeepSort

SORT (Simple Online and Realtime Tracking) and DeepSORT (Deep SORT) are both algorithms used for object tracking in videos, particularly in scenarios where multiple objects need to be tracked simultaneously. However, they have different approaches and capabilities:

If you have multiple objects overlapping and moving on similar paths tracking becomes more difficult. SORT only matches detections based on the bounding box parameters, multiple objects moving similarly are easily confused because of that. deepsort can also distinguish object based on their visual appearance, so clearly if one person wears a blue jacket and another is wearing a red jacket deepsort should be able to keep them apart.

SORT (Simple Online and Realtime Tracking):

SORT is a straightforward, real-time object tracking algorithm that doesn't require complex neural networks for feature extraction.

It typically utilizes simple features like bounding box coordinates and appearance descriptors (e.g., color histograms) to track objects.

SORT maintains tracks by associating detections across frames using a Hungarian algorithm-based approach.

While effective in many scenarios, SORT may struggle with occlusions, cluttered scenes, and variations in object appearance.

DeepSORT (Deep SORT):

DeepSORT extends SORT by incorporating deep learning techniques for feature extraction.

It uses a deep neural network to generate embeddings (feature vectors) representing the appearance of detected objects.

By embedding objects in a feature space, DeepSORT can better handle variations in appearance and track objects more reliably, even in challenging conditions.

DeepSORT typically employs a combination of a detection model (like YOLO or Faster R-CNN) and a deep appearance descriptor model (such as a Siamese network or a triplet loss network).

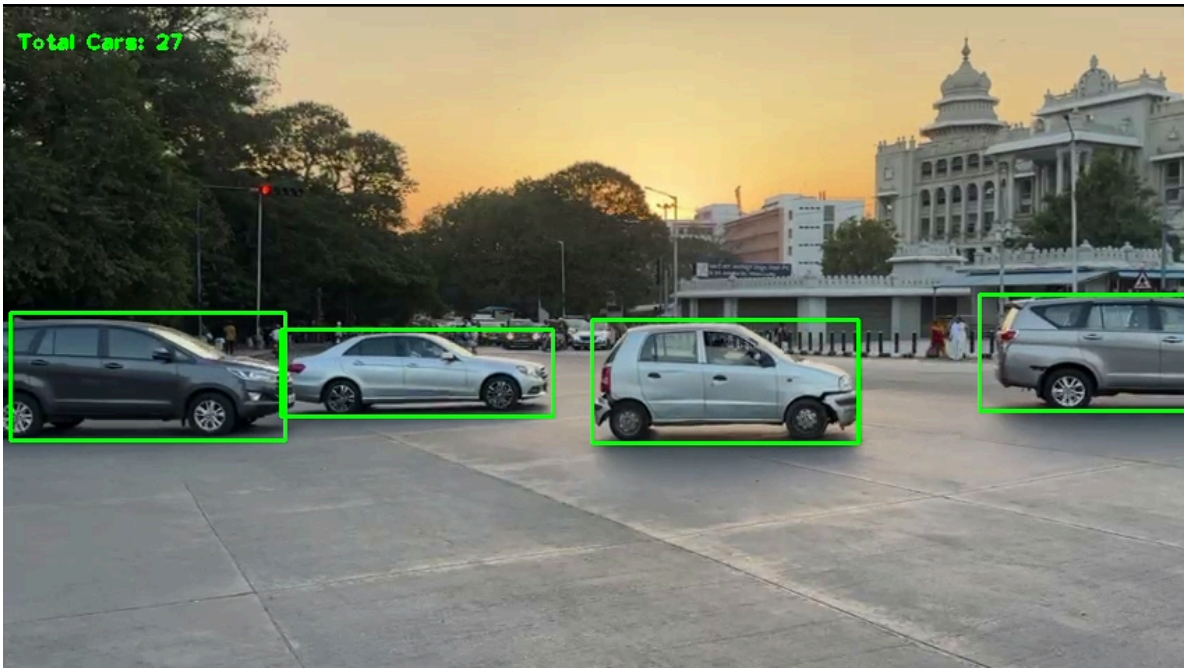
The association step in DeepSORT often combines the appearance similarity with motion predictions to assign detections to existing tracks.

DeepSORT tends to perform better than SORT in scenarios with complex motion patterns, occlusions, and variations in object appearance.

Criteria	SORT	DeepSORT
Feature Extraction	Relies on simple features like bounding box coordinates and appearance descriptors (e.g., color histograms)	Incorporates deep learning techniques for feature extraction using a deep neural network
Handling Variation	May struggle with variations in object appearance, occlusions, and cluttered scenes	Better equipped to handle variations in appearance and challenging conditions
Association Approach	Utilizes a Hungarian algorithm-based approach for associating detections across frames	Combines appearance similarity with motion predictions for more accurate detection-to-track association
Performance	Generally performs well in simple scenarios with less variation and clutter	Performs better in complex scenarios with complex motion patterns, occlusions, and variations in object appearance

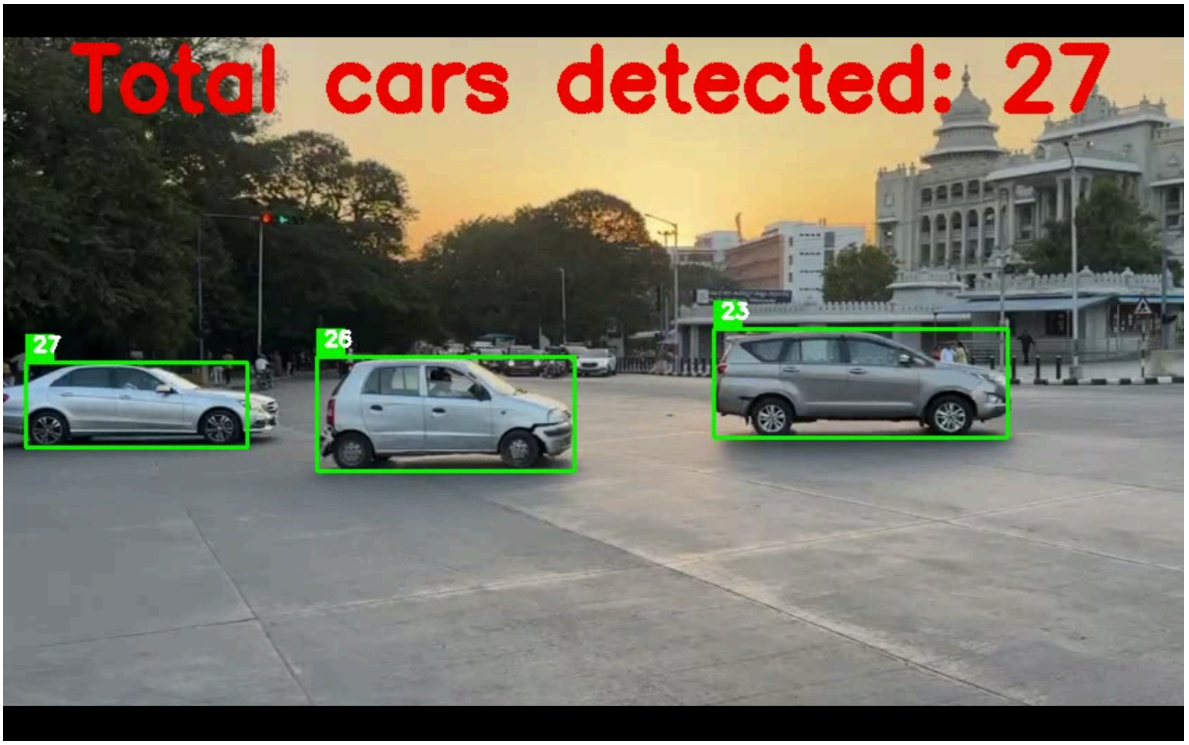
Car counting using SORT + YOLO

[Link to video]



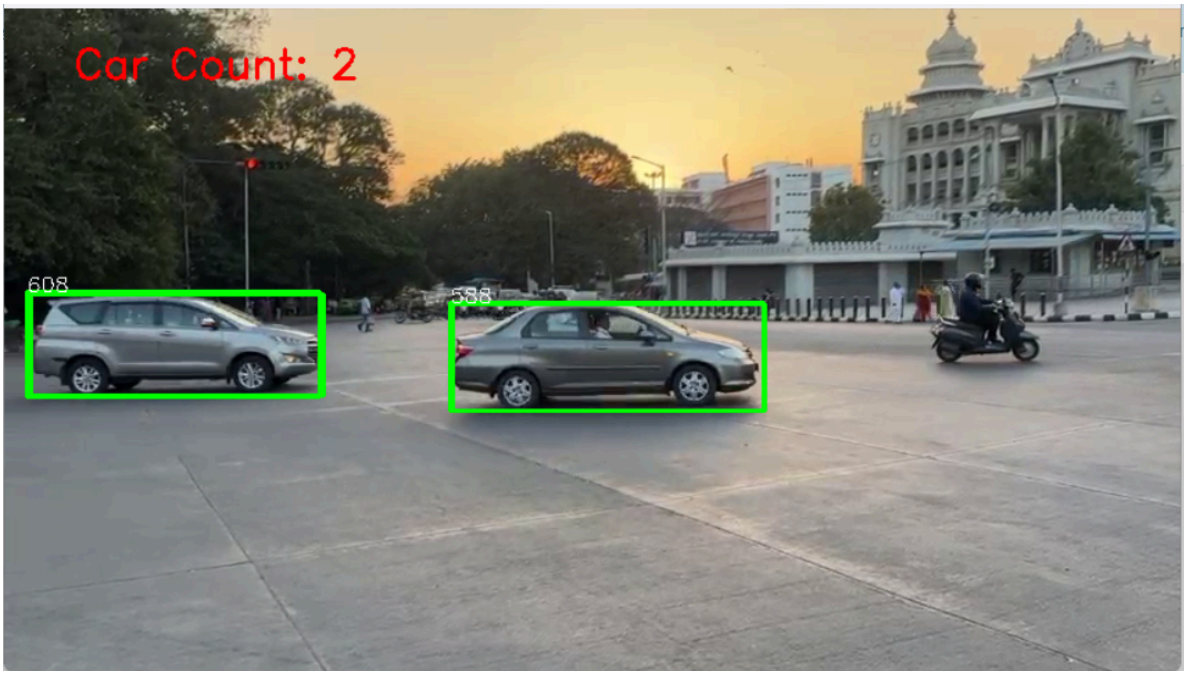
Car counting using DeepSort + YOLO

[Link to video]



Car counting using SORT+ Faster RCNN

(Counting cars in current video frame)



Observations and analysis of above algorithms.
YOLO + SORT:

- YOLO provides high-speed object detection.
- SORT provides efficient object tracking.
- Overall, this combination would be fast and suitable for real-time applications.
- However, it is the most accurate of deep learning-based tracking methods for our test video.

DeepSORT + YOLO:

- DeepSORT improves tracking accuracy using deep learning techniques.
- YOLO provides fast and accurate object detection.
- This combination would offer both high accuracy and reasonable speed, making it suitable for various applications.

Faster R-CNN + SORT:

- Faster R-CNN provides accurate object detection but might be slower compared to YOLO.
- SORT provides efficient tracking.
- This combination not that accurate also it was much slower compared to combinations involving YOLO due to the computational complexity of Faster R-CNN.

1.) Installation and working of Sort

SORT (Simple Online and Realtime Tracking) Algorithm

SORT is an algorithm designed for simple online and real-time object tracking. It is especially effective for multi-object tracking in scenarios like video surveillance, autonomous driving, and robotics. Below is the installation process and a brief overview of how SORT works:

Installation:

To use SORT, follow these steps:

1. Download the SORT Implementation:

SORT is available on the official GitHub repository: (https://github.com/abewley/sort)

Clone or download the repository to your local machine.

2. Import SORT into Your Project:

`sort.py` file is accessible within your project directory.

We have import SORT by using `from sort import *` in your Python script.

Working Principle:

SORT operates based on the following principles:

Detection and Prediction:

It starts by detecting objects in each frame of a video stream using a suitable object detection model such as YOLO.

Then, it predicts the locations of these objects in the next frame based on their current positions and velocities.

Data Association:

SORT uses a combination of Intersection over Union (IoU) and the Kalman filter to associate detections from consecutive frames with existing object tracks.

The Kalman filter helps in estimating the state (position and velocity) of each object, while IoU measures the overlap between bounding boxes to determine the likelihood of association.

Track Maintenance:

It employs a threshold-based approach to determine whether a new detection should initiate a new track or be associated with an existing one.

Track Termination:

Tracks are terminated if an object disappears for an extended period or if it fails to meet certain criteria, such as minimum detection confidence or maximum predicted velocity.

2.) Installation of YOLO (You Only Look Once) Object Detection

You Only Look Once (YOLO) algorithm:

YOLO is a popular real-time object detection system known for its speed and accuracy. It divides the input image into a grid and predicts bounding boxes and class probabilities directly from full images in a single evaluation. Below is the installation process and a brief overview of how YOLO works:

Installation:

To use YOLO, follow these steps:

1. Download the YOLOv8 Model:

YOLOv8 is one of the variants of the YOLO architecture. It offers improvements in terms of accuracy and speed.

You can obtain the YOLOv8 model weights from reliable sources or train your own model based on your dataset.

2. Using the Ultralytics YOLO Library:

(used this for our assignment)

Ultralytics provides a user-friendly Python library for implementing YOLO models.

```
Pip install ultralytics

from ultralytics import YOLO

model = YOLO("yolo_weights/yolov8n.pt")
```

3.) Installation of Faster RCNN:

PyTorch and torchvision Installation:

```
pip install torch torchvision
```

Additional Dependencies:

```
pip install numpy matplotlib
```

Download Pre-trained Model Weights:

PyTorch provides pre-trained Faster R-CNN models.

Load the model with torchvision's `fasterrcnn_resnet50_fpn()` function.

Model Configuration:

Set the model to evaluation mode using `model.eval()`.

4.) Installation of DeepSort

```
!pip install deep-sort-realtime
```

Conclusion:

We can drew conclusion from these question is we explored various object detection models like YOLO and Faster RCNN and implemented car counting systems using SORT and DeepSORT algorithms for multi-object tracking on the video of traffic junction.