VR - MINI PROJECT
Report

Team Members	Roll Number
Mukul Gupta	IMT2020083
Heet Vasani	IMT2020088
Shaurya Agrawal	IMT2020539

Answer-3a

General Introduction:-

Using the Python Pytorch toolkit, the CNN model was trained on the CIFAR-10 dataset for image categorization. Convolutional layers and fully linked layers were explored in different combinations using activation functions as ReLU, tanh, and sigmoid.

60000, 32X32 colour images make up the CIFAR-10 dataset. These 60000 photos come from ten classes, each having 6000 photographs. Aerial vehicle, automobile, bird, cat, deer, dog, frog, horse, ship, and truck are among the 10 classes in the dataset. These 60000 photographs are split into a train set and a test set for classification purposes, with 50000 images in the train set and 10,000 images in the test set.

CNN is a deep learning technique that is frequently used to evaluate visual data. While in conventional approaches, the image features are hard-engineered, CNN gives us the freedom to provide feedback on the image as-is and let the image discover its own features and traits. Several artificial neuronal layers make up CNN. The output of one CNN layer feeds the input of the following one. CNN's first layer often recognises fundamental properties like edges that are horizontal or vertical. The layer starts recognising

higher-level features, such as objects and faces, as you go deeper. Convolutional, pooling, and fully-connected layers are all possible for CNN.

What code does?:-

This code is an implementation of Convolutional Neural Network (CNN) for image classification using PyTorch library. The code performs the following steps:

- Imports necessary libraries - torch, torchvision, transforms, matplotlib, and numpy.
- Defines a set of transformations to be applied to the input image data, including converting it to a tensor, and normalizing it.Sets batch size to 20.
- Loads the CIFAR-10 dataset from the torchvision.datasets module, applies the transformations, and creates a data loader for training and testing data.
- Defines the classes of objects present in the dataset.
- Defines a function to display a batch of images from the training data loader.
- Defines a neural network architecture, consisting of convolutional layers and fully connected layers.
- Defines a loss function - cross-entropy loss, and an optimizer - stochastic gradient descent (SGD).

- Trains the neural network for 10 epochs using the training data loader, and records the loss values for each epoch.
- Evaluates the performance of the network on the test dataset after each epoch, and records the loss values for each epoch.
- Plots the training and validation losses.
- Evaluates the accuracy of the trained network on the test dataset

The CNN model consists of three convolutional blocks, each containing two convolutional layers followed by a max pooling layer. The first block has 32 filters of size 3x3, while the second and third blocks have 64 and 128 filters of size 3x3, respectively. The fully connected layers at the end of the model have 1024 and 512 hidden units.

The loss function used in this code is the cross-entropy loss, which is a standard loss function used for multi-class classification problems. The optimizer used is stochastic gradient descent (SGD) with a learning rate of 0.001 and momentum of 0.9.

The code uses two sets of data loaders (trainloader and testloader) to load batches of images and labels into the model during training and evaluation. Each batch contains 20 images.

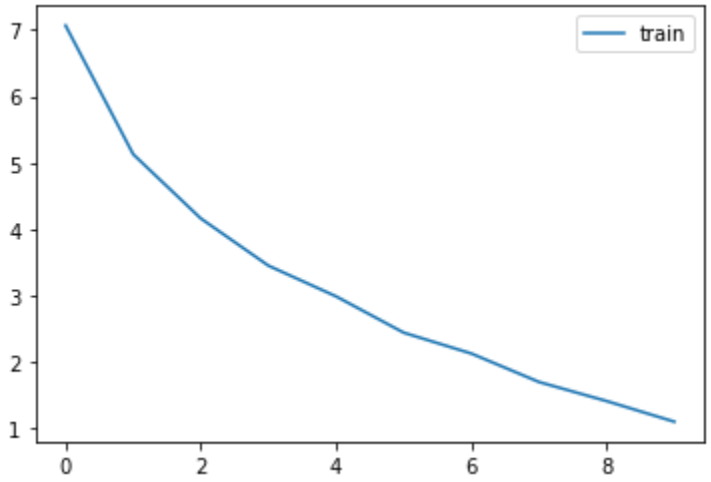
Finally, the code computes the accuracy of the trained model on the test dataset by iterating through the test data and comparing the predicted labels with the true labels. The final accuracy is not printed, however, and must be computed separately.

Results:-

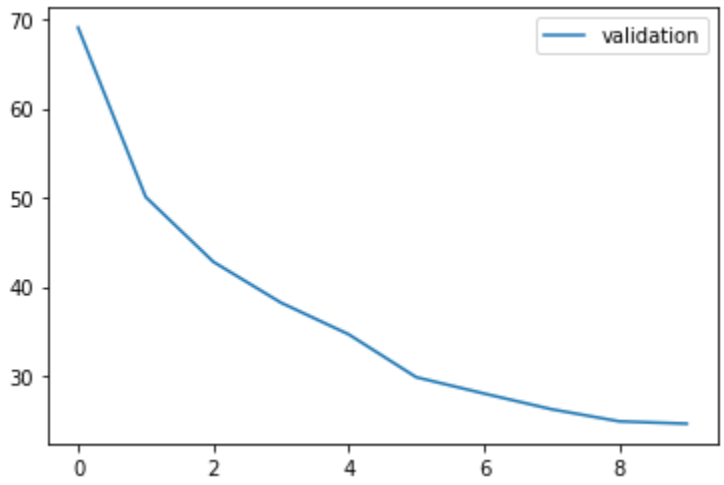
- When we used **Relu** function as our activation function in our architecture(momentum = 0.9, lr = 0.001):
    - Accuracy on test data: 82.47%

    - Accuracy for class: plane is 88.8 %
    - Accuracy for class: car is 89.6 %
    - Accuracy for class: bird is 75.7 %
    - Accuracy for class: cat is 68.8 %
    - Accuracy for class: deer is 86.3 %
    - Accuracy for class: dog is 67.1 %
    - Accuracy for class: frog is 87.2 %
    - Accuracy for class: horse is 85.2 %
    - Accuracy for class: ship is 90.5 %
    - Accuracy for class: truck is 87.7 %

Loss on the training data:

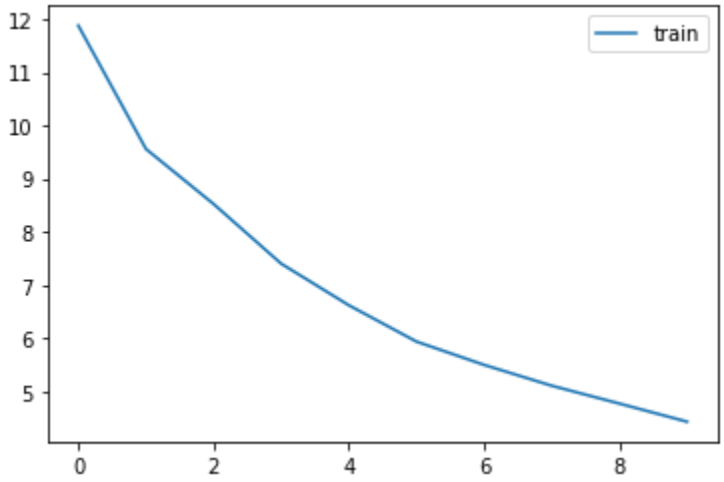


Loss on the validation data:

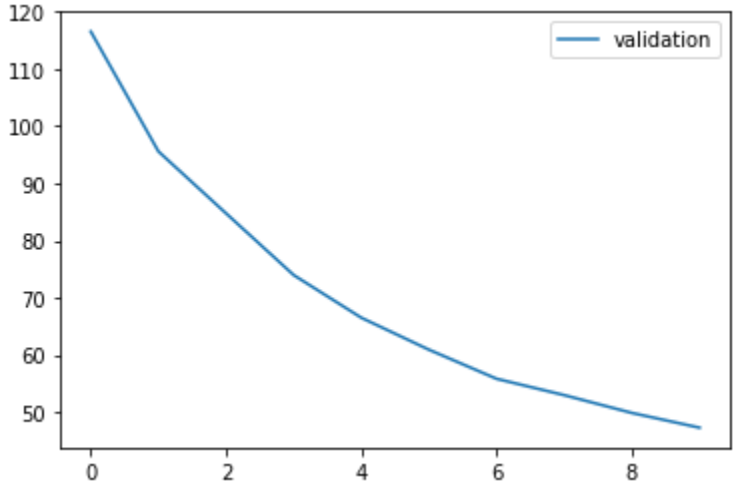


- When we used **Relu** function as our activation function in our architecture without using momentum in stochastic gradient descent:
  - Accuracy on test data: 72.74%

Loss of training data:



Loss of validation data:

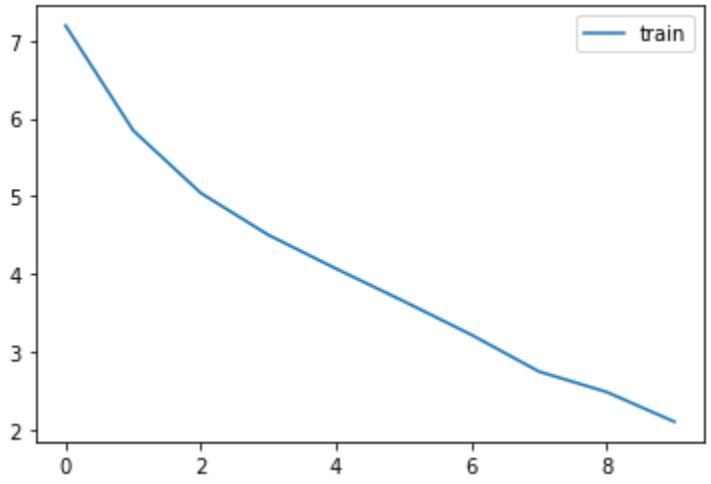


- When we used **Tanh** function as our activation function in our architecture(momentum = 0.9, lr = 0.001):

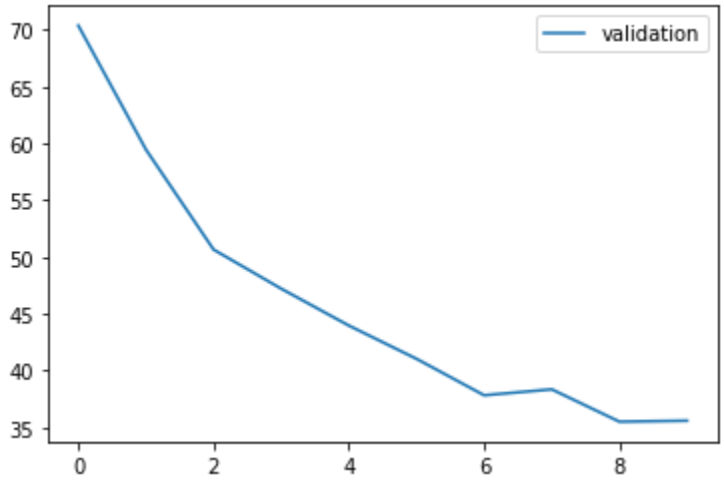
- Accuracy on test data: 75.83%

- Accuracy for class: plane is 84.2 %
- Accuracy for class: car is 88.2 %
- Accuracy for class: bird is 79.7 %
- Accuracy for class: cat is 59.2 %
- Accuracy for class: deer is 63.3 %
- Accuracy for class: dog is 56.4 %
- Accuracy for class: frog is 81.8 %
- Accuracy for class: horse is 81.5 %
- Accuracy for class: ship is 81.6 %
- Accuracy for class: truck is 83.6 %

Loss of training data:

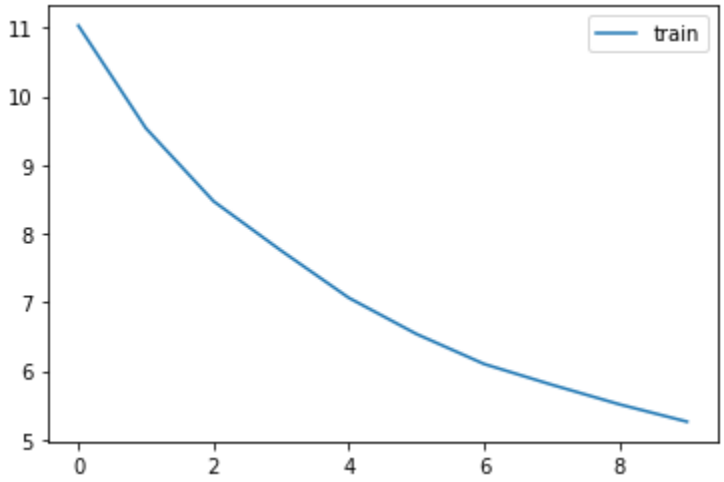


Loss of validation data:

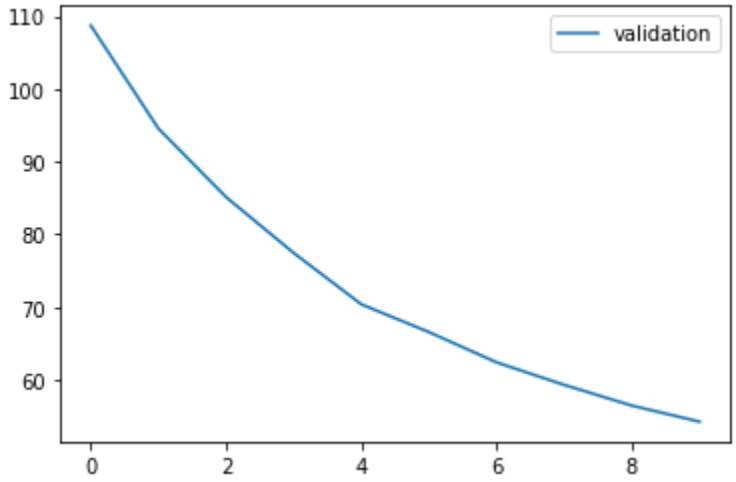


- When we used **Tanh** function as our activation function in our architecture without using momentum in stochastic gradient descent:
  - Accuracy on test data: 69.61%

Loss of training data:



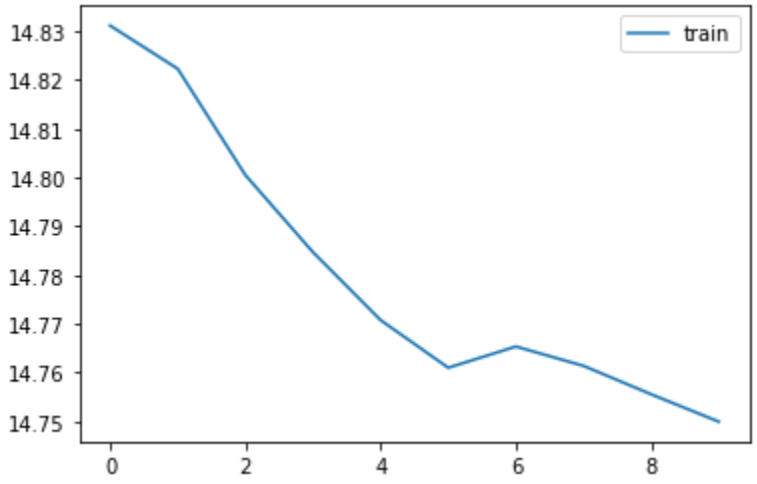
Loss of validation data:



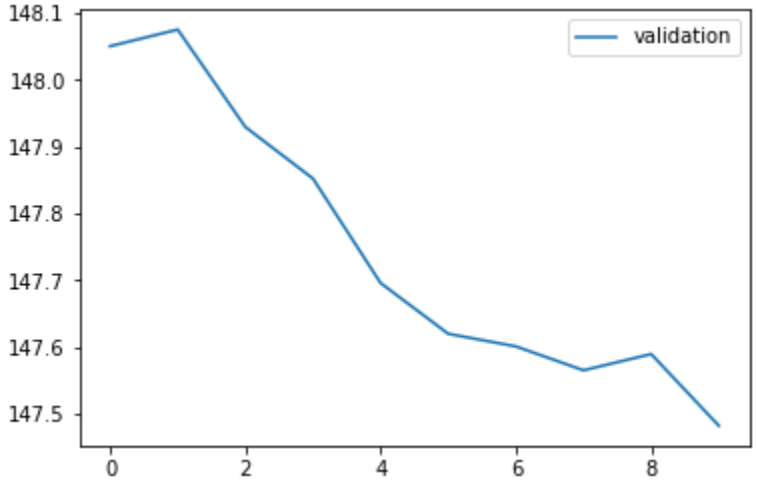
- When we used **Sigmoid** function as our activation function in our architecture(momentum = 0.9, lr = 0.001):
    - Accuracy on test data: 9.82%

As we can see that sigmoid is not a good activation function for this model.

Loss of training data:

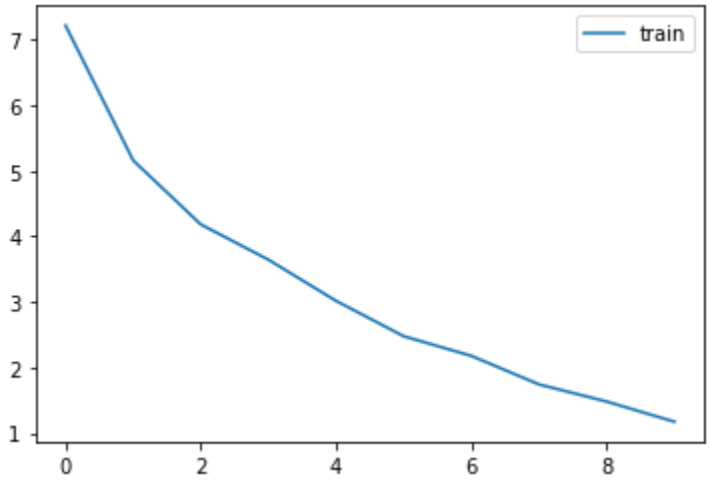


Loss of validation data:

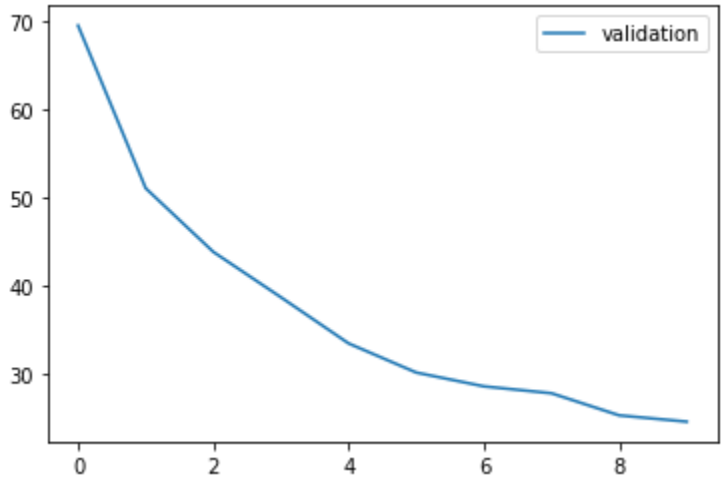


- When we used **Relu** function as our activation function in our architecture and also using adaptive learning rate:
    - Accuracy on test data: 82.95%

Loss of training data:

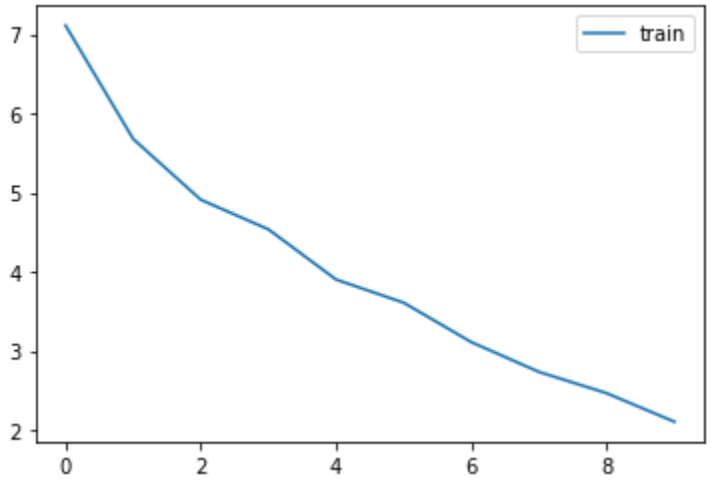


Loss of validation data:

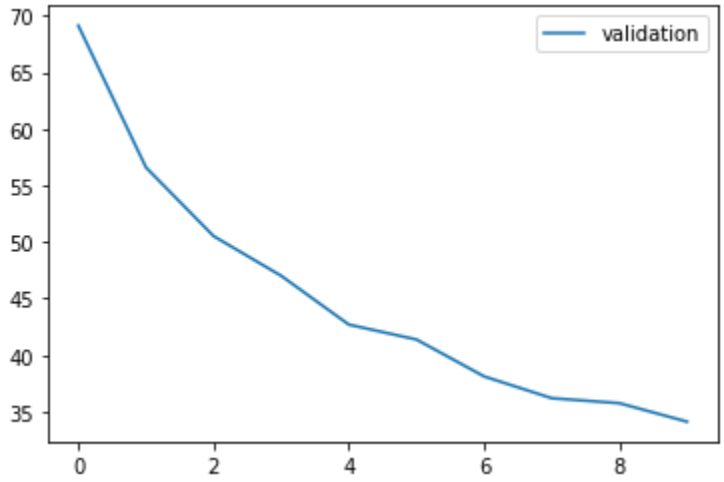


- When we used **Tanh** function as our activation function in our architecture and also using adaptive learning rate:
  - Accuracy on test data: 76.74%

Loss of training data:

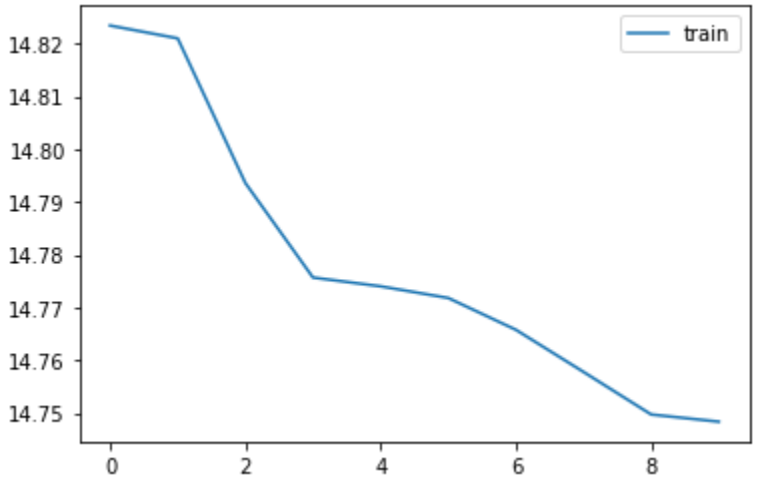


Loss of validation data:

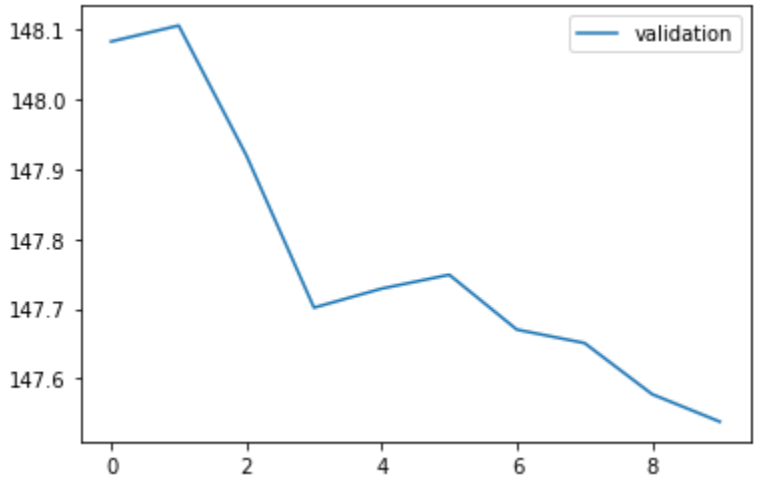


- When we used sigmoid function as our activation function in our architecture and also using adaptive learning rate:
    - Accuracy on test data: 10.1%

Loss of training data:



Loss of validation data:



Note:

- All data is trained by using Google Colab GPU.
- Classification time to train all models is around 5min.
- Classification time to train all models with adaptive learning is taking around 6min.
- Optimizer => Stochastic gradient descent; Learning rate = 0.001; Momentum = 0.9; batch size = 20; epochs = 10

Conclusion:

In the specified architecture when we use the RELU function as an activation function, the model is giving

better performance with 82.47% accuracy. When we train the data, we saw that on adding 1 convolution layer is adding more time in training the model than adding 1 fully connected layer in general. When we used the normal gradient descent it was taking a very long time in training, so we applied stochastic gradient descent which is must faster. In order to reach towards minima fastly we use momentum in stochastic gradient descent for same number of epochs which gave better accuracy every time.

So our recommended architecture would be the Alexnet with Relu function as an activation function which uses stochastic gradient descent as optimizer that also uses the concept of momentum for training the model to get better results.
We can also see that by using adaptive learning rate we are getting slightly better results, So this is also useful for our model.

Answer-3b

Image Classification on STL-10 dataset using Alexnet as a feature extractor ➖

General Information:-

STL-10 Dataset :- The 13000 96X96 colour images that make up the STL-10 dataset. The CIFAR-10 dataset served as its inspiration. There are 1300 photos in each of the 10 classes comprising these 13000 total photographs. A plane, a bird, a car, a cat, a deer, a dog, a horse, a monkey, a ship, and a truck are among the dataset's 10 classifications.
The train set contains 5000 images, while the test set contains 8000 images.

Alexnet:- 5 convolutional layers, 3 max-pooling layers, 2 normalization layers, 2 fully connected layers, and 1 softmax layer make up this architecture.

It was the first convolutional network to incorporate a graphics processing unit to improve speed.
Every convolutional and fully connected layer is followed by Relu.

The multinomial logistic regression goal is maximized by this network, which makes advantage of the pooling layers to execute max pooling.
Dropout is employed to address overfitting rather than regularization. Yet, a dropout rate of 0.5 results in a doubling of training time.
AlexNet includes 60 million parameters in total.

Model Used:- Pre-trained Alexnet model(available in torch.hub)

The AlexNet model utilized in this code comprises eight layers, five of which are convolutional and three of which are fully connected. Max-pooling layers and ReLU activation processes come after the convolutional layers. The final convolutional layer's output is flattened before being given to the fully connected layers. 4096 neurons make up the first fully connected layer, and 1024 neurons make up the second. Ten neurons make up the final fully connected layer, which is equal to the number of classes in the STL-10 dataset.

Also, we have normalized the given data using the parameters for normalize as (0.485, 0.456, 0.406), (0.229, 0.224, 0.225) for mean and std respectively.

The learning rate and momentum are both set by the code to 0.0009 and 0.85 respectively during training. Additionally, it shuffles the data after each epoch and employs mini-batch stochastic gradient descent with a batch size of 8.

Following training, the algorithm assesses the model's performance on the test set and calculates its accuracy, reporting it as a percentage. In order to build an SVM model to categorize the photos, the code retrieves the results from the penultimate layer of the trained model for both the train and test sets.

The SVM model utilizes a linear kernel and achieves a test set accuracy of 99.8 percent. The logistic regression model gives an accuracy of 99.9 percent. To achieve the following accuracy the output features and labels of both test and train and data loader which are obtained after training the AlexNet model have been merged to create a new dataset and then split into test and train (split ratio as 20 %).

Coming to part b of question 2 here we have to implement the code for the bike vs horses data set.
The given bike vs horse data set is very small when compared to the traditional datasets. The whole dataset consists of 161 images with 80 images of bikes and 81 images of horses. As the given data is not divided into test and train. So firstly we are reading the data and dividing it into train and test randomly having a split ratio of 0.7. After that, we are transforming the given data. Firstly we are resizing it to (224, 224), and after that, we are transforming the image by using mean and std as (0.485, 0.456, 0.406), (0.229, 0.224, 0.225) respectively. Finally, the data is passed through a pertained Alexnet model. We haven't added any layers to the Alexnet model because we observed that we weren't able to get good accuracy when we increase the model complexity also using the pre-trained Alexnet model gives results pretty fast as the data size is small so we stuck to pertained Alexnet model. After passing the data through the model we would get features for our image according to the no of neurons in the last layer. Now that feature vector would serve as our new train and test data.

After this, we applied SVM with linear kernel and logistic regression with max_iter = 1000,l1_ratio=0, which give us an accuracy of 100 % on test data.

Answer-3c

5 Additional features of YOLO V2

YOLO (You Only Look Once) is a popular object detection algorithm that uses a single neural network to predict the bounding boxes and class probabilities of objects in an image. YOLO has evolved over time, with YOLOv2 being an improvement over the original YOLOv1. Here are 5 additional features of YOLOv2 compared to YOLOv1:

- Better architecture: YOLOv2's architecture is more sophisticated than YOLOv1's, which enables improved performance. The architecture combines skip connections and extra convolutional layers to enhance feature representation.

- Batch normalisation is a technique used by YOLOv2 to aid in training and make the model more resilient to changes in input data. This aids in lowering overfitting and enhancing generalisation abilities.

- Anchor boxes: YOLOv2 introduces the idea of anchor boxes, which are already established bounding boxes

used to aid in the prediction of the proper bounding boxes by the model. YOLOv2 can enhance its localization performance and lessen false positives by employing anchor boxes.

- Various scales: YOLOv2 makes predictions at many scales thanks to a feature pyramid network (FPN). This enhances overall detection performance and aids in the detection of objects of various sizes in the image.

- Softmax loss: YOLOv2 makes use of a softmax loss function, allowing the model to forecast various classes for each item. As a result, object categorization is more accurate and the model can identify more item categories.

Overall, YOLOv2 significantly outperforms YOLOv1 in terms of speed and accuracy. It performs better thanks to its enhanced design, batch normalization, anchor boxes, numerous scales, and softmax loss function.

Answer-3d

DeepSORT (Deep Learning-Based SORT) and SORT (Simple Online and Real-Time Tracking) are both object tracking algorithms, but they differ in a number of important ways.

- ***Architecture:*** SORT is a traditional computer vision algorithm that uses a combination of optical flow and Kalman filtering to track objects. On the other hand, DeepSORT is a deep learning-based algorithm that integrates a deep sorting graph to improve tracking accuracy.
- ***Accuracy***: DeepSORT achieves higher tracking accuracy than SORT because it uses a deep neural network to create a more discriminative representation of objects that can handle variations in lighting, pose, and occlusions. SORT is based on hand-crafted functions that may not be as robust to these variations.
- ***Speed:*** SORT is faster than DeepSORT because it does not require the use of a neural network to generate placement functions. It uses only simple feature techniques such as histogram of oriented gradients (HOG) and color histograms. In contrast,

DeepSORT requires extra time to compute deep functions.

- ***Multipurpose Tracking:*** DeepSORT can track multiple objects at once, while SORT is limited to tracking one object at a time. DeepSORT uses an algorithm to match detected objects in frames using their appearance features, while SORT performs this task by applying a Kalman filter to each detected object independently.
- ***Hardware Requirements:*** DeepSORT requires more powerful hardware because it uses a deep neural network to extract features, which can be computationally expensive. SORT, on the other hand, can run on less efficient hardware because it is based only on traditional computer vision algorithms.

The steps we used to implement a car counter using YOLO for object detection and SORT for object tracking:

- **Install the required dependencies:** install Yolo and SORT.

- **Train the YOLO model:** Train the YOLO model using the gathered dataset. This will enable the

model to detect cars accurately in images or videos.

- **Implement SORT:** Implement the SORT algorithm to track the cars in the video. SORT is a simple online and real-time tracking algorithm that is based on the Kalman filter. It predicts the location of the object in each frame and assigns unique IDs to each object.

- **Detect cars using YOLO:** In each frame of the video, run the YOLO model to detect the cars. This will give you a list of bounding boxes that represent the location of each car in the frame.

- **Apply SORT algorithm:** Apply the SORT algorithm to track each car through the video frames. The algorithm will predict the location of each car in the next frame and assign a unique ID to it.

- **Count the cars:** Once you have the bounding boxes and IDs for each car in each frame, count the number of cars that enter and exit the video. Do this by comparing the bounding boxes and IDs of each car in consecutive frames and

determining if a car has entered or exited the video.

- **Visualize the results** Visualize the results of the car counting algorithm by drawing the bounding boxes around each car and displaying the count of cars that have entered and exited the video.

We used the similar steps to implement a car counter using (YOLO +Deepsort), (Faster-RCNN+SORT) and (Faster-RCNN + Deepsort).
For Deepsort, we replaced every step of SORT with the corresponding Deepsort thing. Similarly, replace every step of YOLO with the corresponding Faster-RCNN step.

We have taken 3 videos for test data.

Observations and results :

Ground Truth:-

As seen in video the total no of car observed by us :-

Video 1 = 6 cars

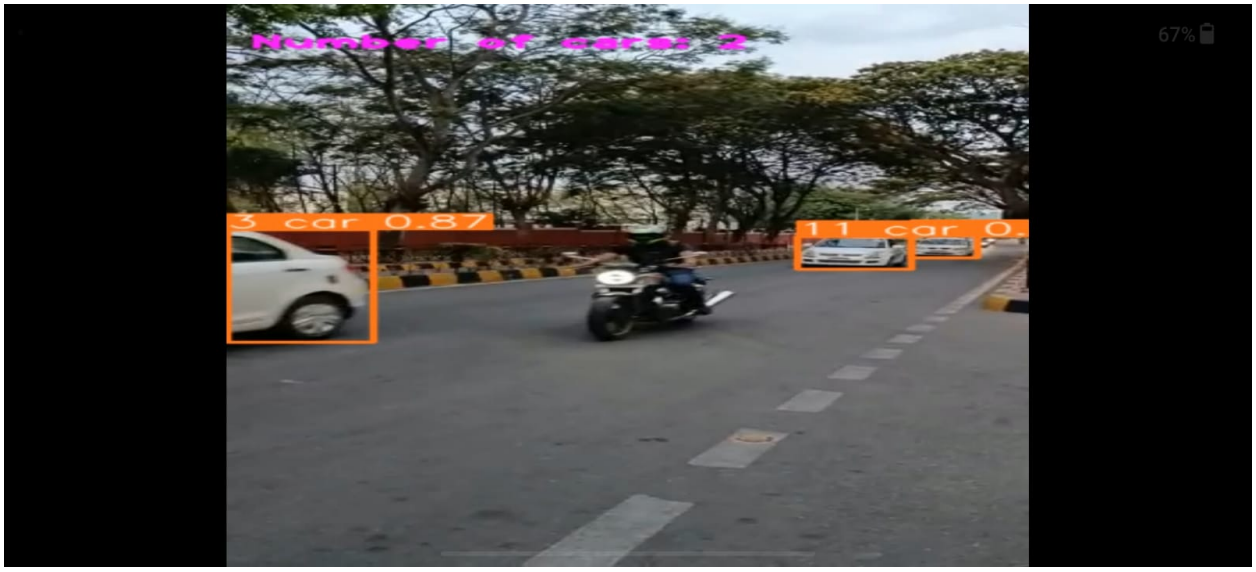
Video 2 = 12 cars

Video 3 = 14 cars

For YOLO with Sort

(Screenshots of the results obtained)

For Video 1



Total no of cars detected at the end: 10

For Video 2

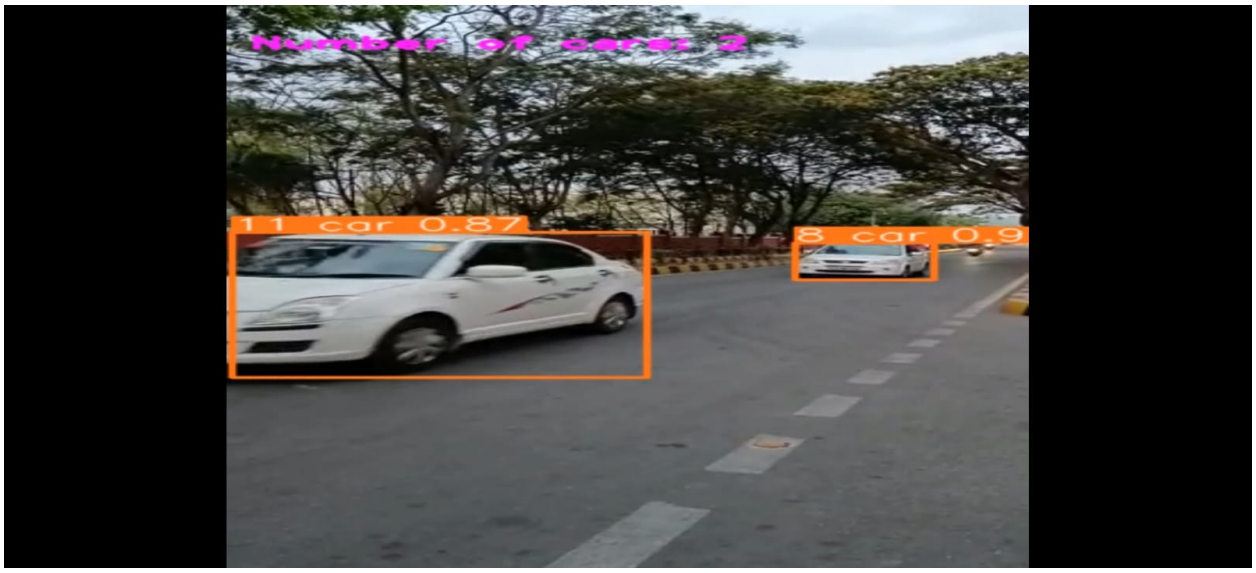
Total no of cars detected at the end: 17

For Video 3

Total no of cars detected at the end: 29

For YOLO with deepsort

For Video 1



Total no of cars detected at the end: 9

For Video 2

Total no of cars detected at the end: 13

For Video 3

Total no of cars detected at the end: 22

For Faster RCNN and sort

Video 1

Total no of cars detected at the end: 10

Video 2

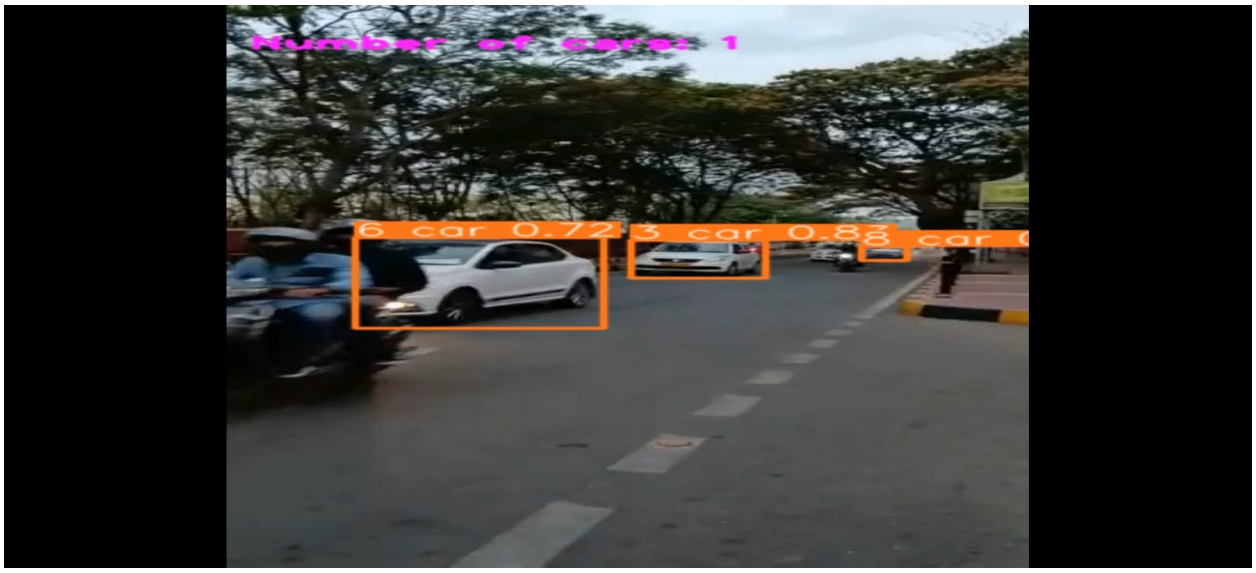
Total no of cars detected at the end: 15

Video 3

Total no of cars detected at the end: 27

For Faster RCNN with deepsort

For Video 1

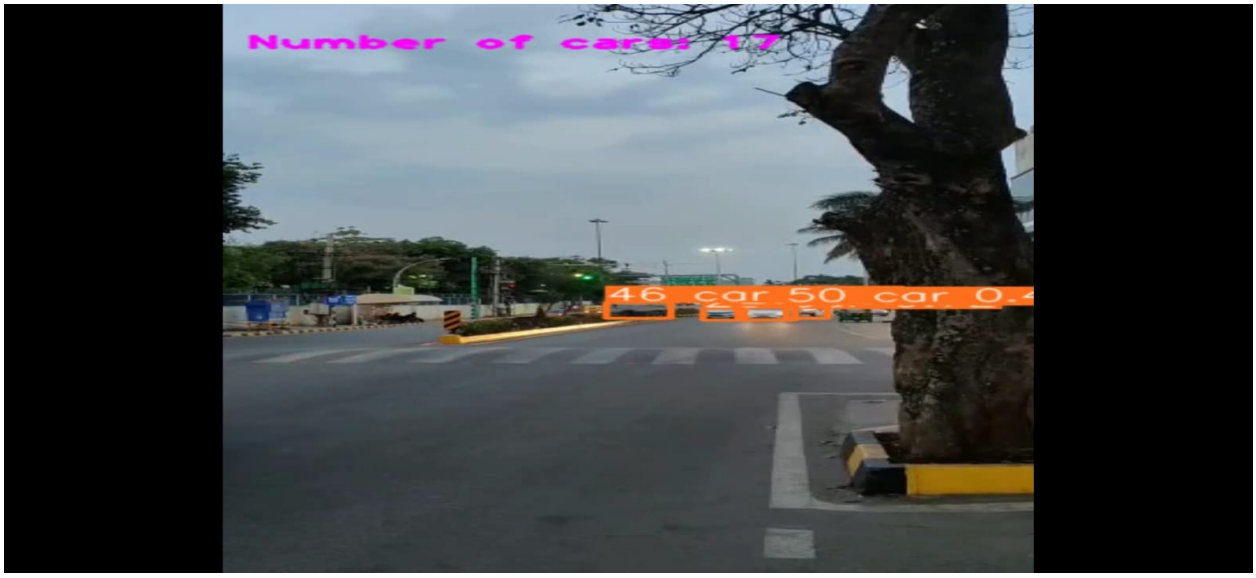


Total no of cars detected at the end: 7

For Video 2

Total no of cars detected at the end: 12

For Video 3



Total no of cars detected at the end: 20

General observations and results obtained:-

1). It has been seen that for all above the no of cars detected is generally a bit more than the total no of cars present in the actual video. It is due to the working of sort and deep sort also the same car position changes many times so it's possible to a detect a same car more times. Also, the deep sort in general gives better results when compares the sort algorithm .

2). For video 3 we observed that all the 4 different combinations are giving results (total no of cars) more than the actual no of cars .

Here are some of the deeper differences between the combinations you mentioned in terms of their accuracy and the results they get.

- **YOLO+SORT:** This combination is known for its speed and simplicity, as both YOLO and SORT are relatively fast and efficient algorithms. However, in terms of accuracy, this combination may not perform as well as some other combinations, especially when

dealing with crowded scenes or hidden objects. SORT relies on a simple motion model to predict the location of objects, which may not be as accurate as more sophisticated methods.

- **YOLO+DeepSORT:** DeepSORT is an advanced version of SORT that uses deep learning to extract and combine features. This can potentially improve tracking accuracy, especially in crowded or enclosed scenes. However, DeepSORT is also computationally more expensive than SORT, which can affect the system's overall speed. In terms of target detection accuracy, YOLO is generally considered a strong performer with high accuracy and relatively low computational requirements.

- **Faster-RCNN+SORT:** Faster-RCNN is a more complex object detection algorithm than YOLO, which can lead to better object detection. However, this comes at the cost of increased computer requirements, which can affect the overall speed of the system. SORT is a relatively simple algorithm that may not perform as well as DeepSORT in terms of tracking accuracy, especially in congested scenes.

- **Faster-RCNN+DeepSORT:** This combination combines the strengths of both Faster R-CNN and

DeepSORT, resulting in high target detection and tracking. However, this combination is also the most computationally expensive of the four, which can limit its real-time capabilities. Additionally, Faster R-CNN can be more complex and harder to train than YOLO, requiring more expertise and resources.

- In general, the best combination to use depends on the specific needs and requirements of the application. If speed is a priority, YOLO SORT may be the best choice. If tracking accuracy is more important, Faster-RCNN DeepSORT may be the way to go. It is also worth noting that there are other factors to consider, such as the size of the dataset and the amount of training data available.