Visual Recognition

Assignment 3

Team Members:

Venkat Suprabath Bitra (IMT2019091)

Vikram Madhavan (IMT2019093)

Prem Sai (IMT2019067)

Venkat Suprabath Bitra, Vikram Madhavan, Prem Sai
IMT2019091, IMT2019093, IMT2019067

Assignment 3a

For the given problem, we load the CIFAR-10 dataset from the PyTorch TorchVision library. The train dataset is split into 4:1 ratio for training set to validation set.

We have tried 4 Models whose architectures and basic description is given below:

1. LeNetReLu: We used 2 convolutional layers with 3 fully connected layers with ReLU activation function. Max-Pooling is applied after each application of convolution layer. The model architecture is as shown below:

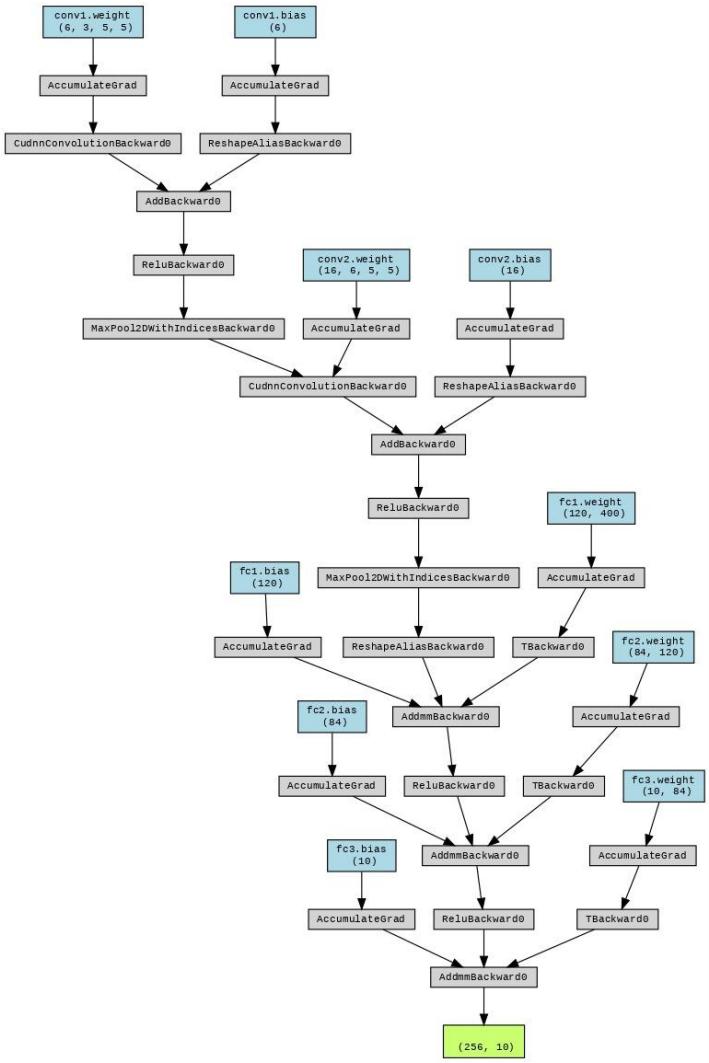


Figure 1: Network Architecture of LeNetReLu

2. LeNet: We used 2 convolutional layers with 3 fully connected layers with sigmoid activation function. Max-Pooling is applied after each application of convolution layer. The model architecture is as shown below:

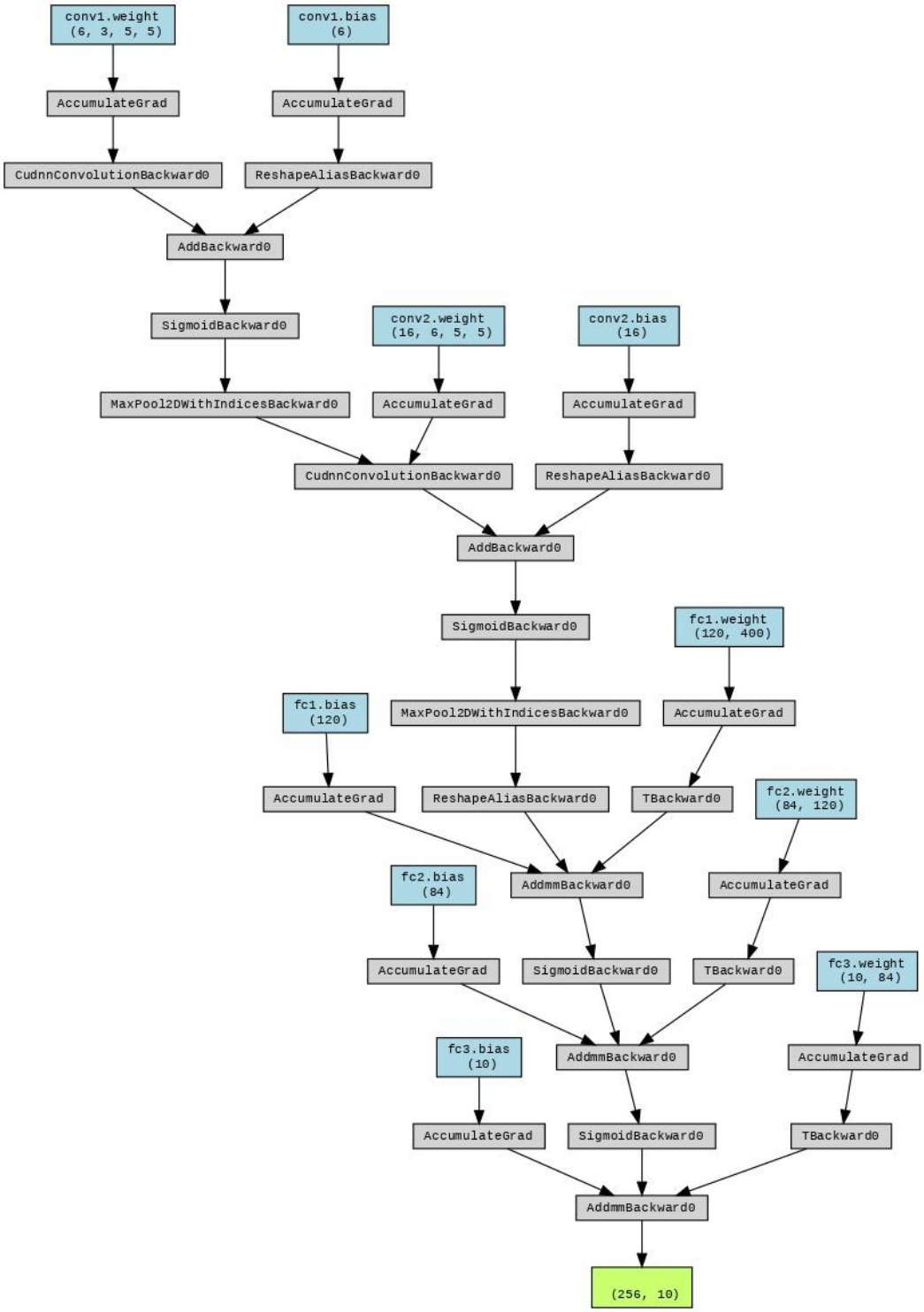


Figure 2: Network Architecture of LeNet

3. LeNet_tanh: We used 2 convolutional layers with 3 fully connected layers with tanh activation function. Max-Pooling is applied after each application of convolution layer. The model architecture is as shown below:

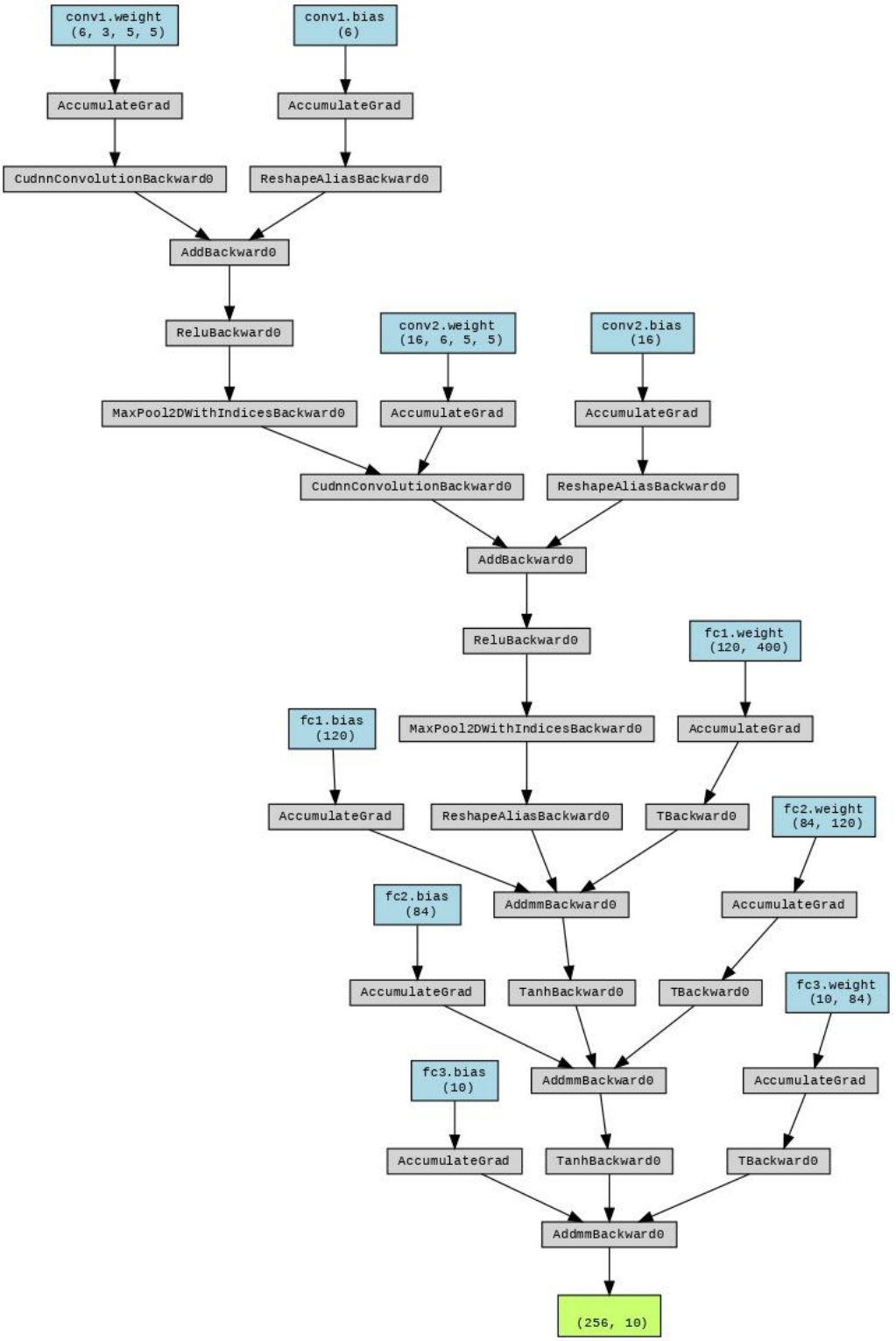


Figure 3: Network Architecture of LeNet_tanh

4. AlexNet: We used 5 convolutional layers with 3 fully connected layers with ReLU activation function. Max-Pooling is applied after every two applications of convolution layer. The model architecture is as shown below:

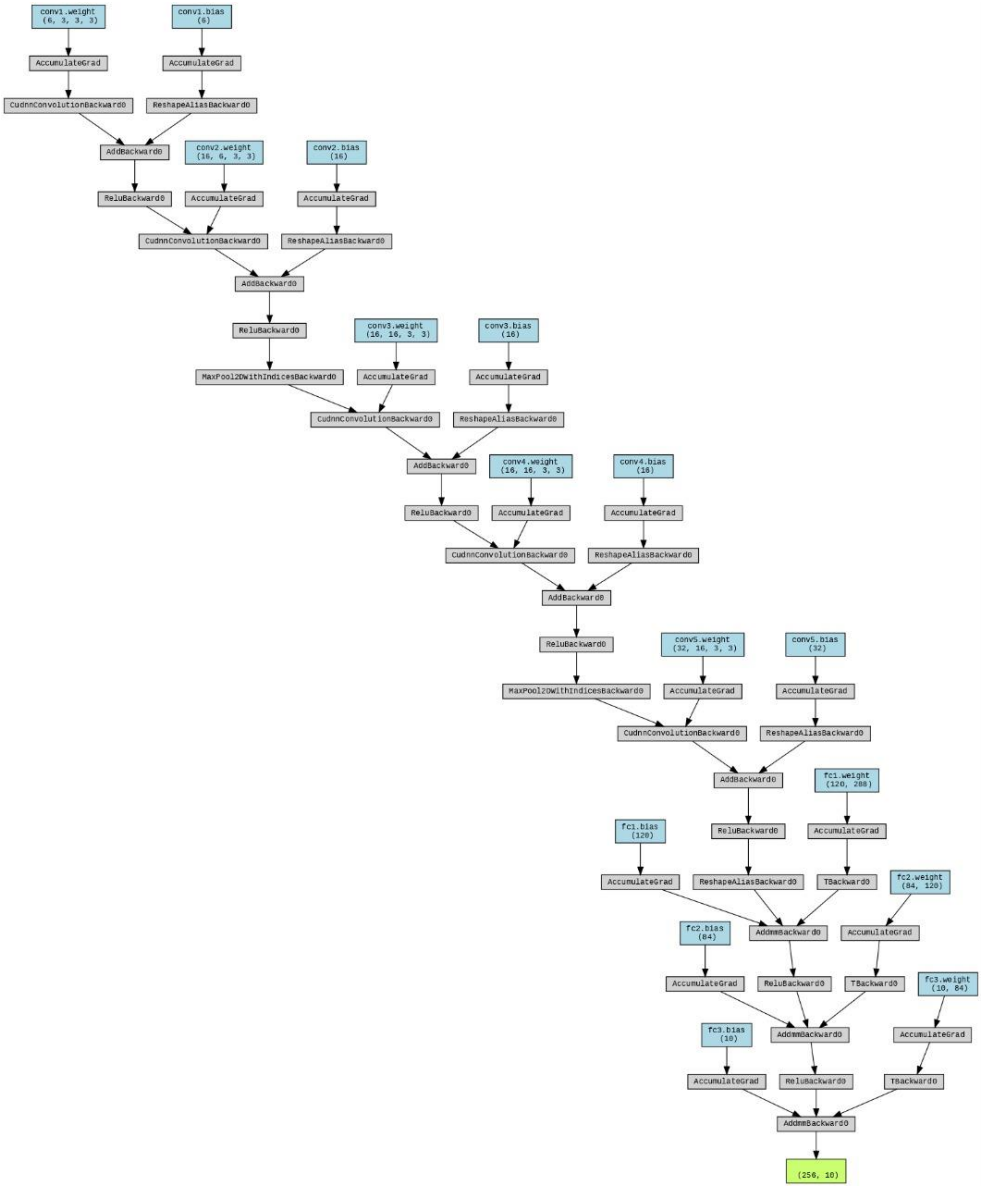


Figure 4: Network Architecture of AlexNet

The results of the training are shown in the table below (Time per Epoch is calculated on Google Colab):

Model	Momentum	Adaptive LR	LR	Test Data	Number of Epochs	Time per Epoch	Accuracy
LeNetReLu	TRUE	FALSE	0.05	Validation data	50	11s	63
LeNetReLu	TRUE	FALSE	0.05	Test Data	30	13.5s	63.56
LeNetReLu	FALSE	FALSE	0.05	Validation data	150	10	58

LeNetReLu	FALSE	FALSE	0.05	Test Data	60	13	47.97
LeNetReLu	TRUE	TRUE	0.05	Validation data	50	10.5	51.4
LeNetReLu	TRUE	TRUE	0.05	Test Data	30	13	52.3
LeNetReLu	FALSE	TRUE	0.05	Validation data	50	11	48.6
LeNetReLu	FALSE	TRUE	0.05	Test Data	30	11	48.3
LeNet	TRUE	FALSE	0.5	Validation data	50	10.5	62.4
LeNet	TRUE	FALSE	0.5	Test Data	30	13	63.3
LeNet	FALSE	FALSE	0.5	Validation data	50	10	56.9
LeNet	FALSE	FALSE	0.5	Test Data	30	13	48.04
LeNet	TRUE	TRUE	0.5	Validation data	50	13	55.8
LeNet	TRUE	TRUE	0.5	Test Data	30	12	49.2
LeNet	FALSE	TRUE	0.5	Validation data	50	13	10.1
LeNet	FALSE	TRUE	0.5	Test Data	25	13	10.1
LeNet_tanh	TRUE	FALSE	0.05	Validation data	50	13	60.08
LeNet_tanh	TRUE	FALSE	0.05	Test Data	30	13	61.54
LeNet_tanh	FALSE	FALSE	0.05	Validation data	125	11	55.4
LeNet_tanh	FALSE	FALSE	0.05	Test Data	30	13	48.41
LeNet_tanh	TRUE	TRUE	0.05	Validation data	50	11	55.2
LeNet_tanh	TRUE	TRUE	0.05	Test Data	30	11	46.34
LeNet_tanh	FALSE	TRUE	0.05	Validation data	50	11	54.4
LeNet_tanh	FALSE	TRUE	0.05	Test Data	25	14	5.27
AlexNet	TRUE	FALSE	0.05	Validation data	150	11	77.76
AlexNet	TRUE	FALSE	0.05	Test Data	25	13	76.35
AlexNet	FALSE	FALSE	0.05	Validation data	50	11	66.84
AlexNet	FALSE	FALSE	0.05	Test Data	25	14	64.86
AlexNet	TRUE	TRUE	0.05	Validation data	50	11	77.64
AlexNet	TRUE	TRUE	0.05	Test Data	25	13	12.6
AlexNet	FALSE	TRUE	0.05	Validation data	100	11	73.32
AlexNet	FALSE	TRUE	0.05	Test Data	35	14	12.8

From the above results we can see that AlexNet architecture with 5 convolutional and 3 fully connected layers had better performance as the time per epoch is similar to LeNet but has higher accuracy. AlexNet architecture with Momentum with or without Adaptive Learning Rate seems to be best in the models which have been tested (shown in the table above).

Assignment 3b

The object recognition dataset used is “Intel Image Classification” from Kaggle [1]. It has 25000 images of size 150 x 150, with 6 categories namely, Buildings, Forest, Glacier, Mountain, Sea and Street. The dataset is split into train, validation and test with 14000, 3000 and 7000 images respectively.

The CNN extractor tried first was AlexNet whose network architecture is given in Figure 5.

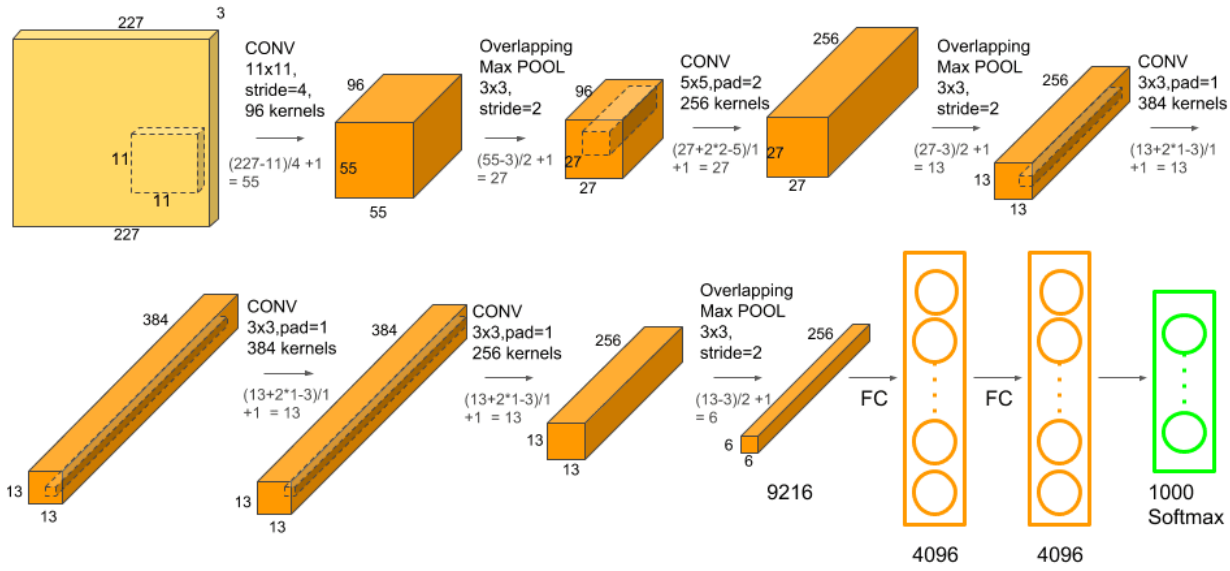


Figure 5: AlexNet Architecture

The idea of transfer learning is to take accurate models produced from large training datasets and apply it to smaller sets of images for faster and more accurate training. This includes transferring the more general aspects of the model, such as the process for identifying the edges of objects in images (done in the initial layers of the model). The more specific layer of the model which deals with identifying types of objects or shapes (the final layers) can then be trained. The model’s parameters will need to be refined and optimized, but the core functionality of the model will have been set through transfer learning.

In the current dataset, we used AlexNet as a base model whose weights are trained on ImageNet dataset. The last two layers of the AlexNet architecture (shown in Figure 5) has been modified with a Linear Layer having 4096 nodes (6^{th} classifier layer in PyTorch) and a Softmax Layer having 6 (number of classes) nodes (7^{th} classifier layer in PyTorch). For training AlexNet in PyTorch the weights of the feature layers and first 5 classifier layers (from the pretrained weights of ImageNet dataset) have been frozen and the 6^{th} and 7^{th} layer have been updated as mentioned above.

The results of training and validation accuracy is shown in Figure 6.

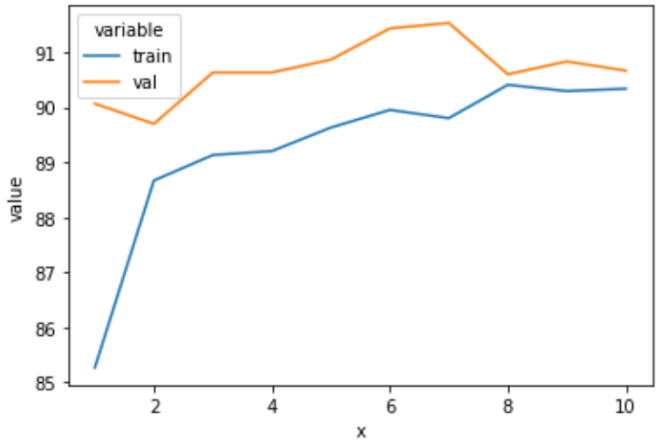


Figure 6: Train and Validation Accuracies for AlexNet

The testing accuracy obtained using AlexNet is 91%.

Another CNN exactor tried out is the ResNet50 model whose network architecture is given in Figure 7.

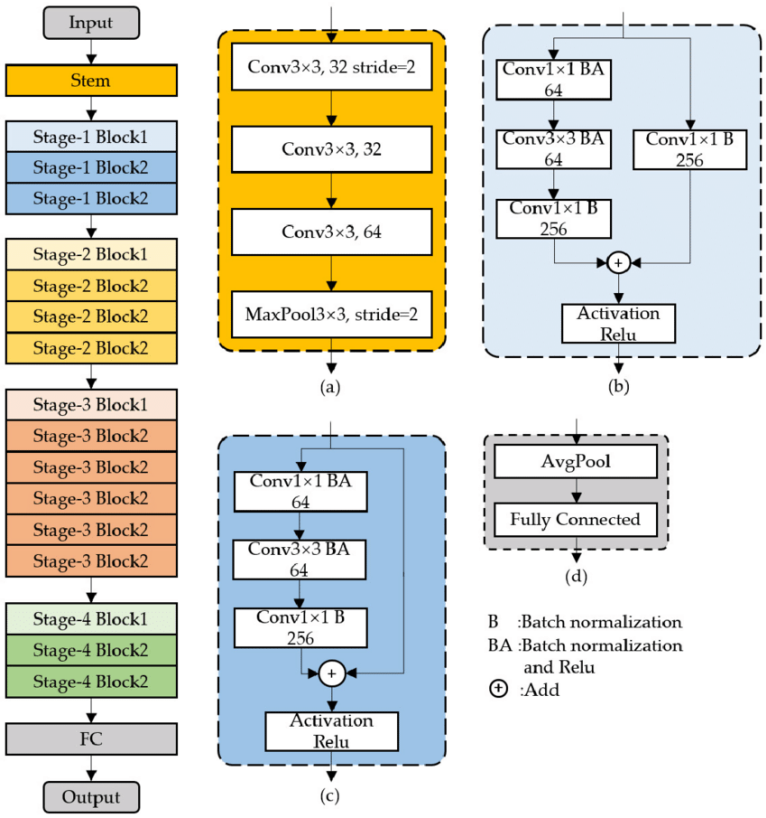


Figure 7: Network Architecture of ResNet

The results of training and validation accuracy of the ResNet model for 25 epochs is given in Figure 8.

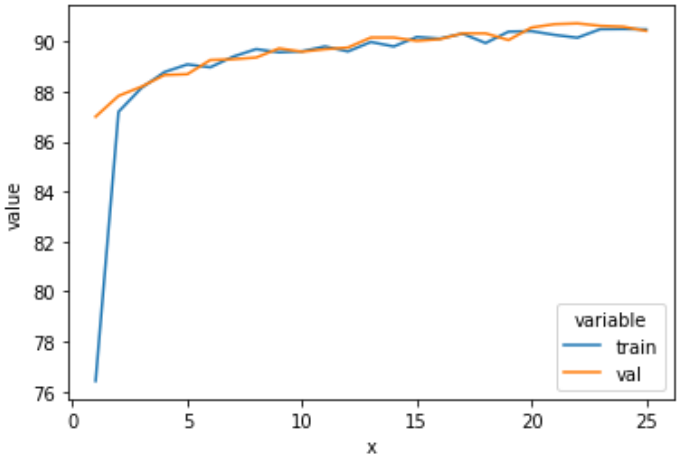


Figure 8: Train and Validation Accuracies for ResNet

The training was carried out for only 25 epochs because the time it took for each epoch was around 3 minutes on an Intel i9-9880H CPU and Nvidia GeForce GTX 1650 GPU.

The testing accuracy obtained for ResNet is similar to AlexNet at around 91%.

ResNet took much longer than AlexNet to train for comparable results, so we will use AlexNet to train for the Bikes vs Horses Classification.

Applying AlexNet on the dataset we obtain 100% accuracy, which is expected as we obtain the same for the BoW approach which was done in Assignment 2b.

Assignment 3c

An integral part of computer vision is object classification and object detection. The difference between object detection algorithms and classification algorithms is that in detection algorithms, we try to draw a bounding box around the object of interest to locate it within the image whereas in classification algorithms we determine which class the entire image belongs to given a predefined set of classes. Also, we might not necessarily draw just one bounding box in an object detection case, and there could be many bounding boxes representing different objects of interest within the image (exact number is unknown beforehand).

The major reason why we cannot proceed with this problem by building a standard convolutional network followed by a fully connected layer is that, the length of the output layer is variable as the number of occurrences of the objects of interest is unknown. A naive approach to solve this problem would be to take different regions of interest from the image, and use a CNN to classify the presence of the object within that region. However, the problem with this approach is that the objects of interest might have different spatial locations within the image and different aspect ratios. Hence, you would have to select a huge number of regions and this could computationally blow up. Therefore, algorithms like R-CNN, YOLO etc. have been developed to find these occurrences and find them fast.

RCNN involves generating candidate regions in an image and use greedy algorithms (selective search algorithm) to combine similar regions into larger ones. These regions (around 2000) are then used to make candidate region proposals. These regions are then passed to a CNN which acts a feature extractor and these extracted features are passed to a SVM to classify the presence of the object in the region. This approach is very expensive as we have to classify 2000 region proposals, where no learning is taking place and which could be bad. It is observed to take around 47 seconds for each test image, which is lot of time for real world applications.

Fast RCNN is an improvement on RCNN which rather than generating the candidate region proposals generates a convolutional feature map directly from the image by passing the image to the CNN used as the feature extractor. We use this to identify the region of proposals and warp them into squares and by using a Region of Interest (RoI) pooling layer and reshape them into a fixed size so that it can be fed into a fully connected layer. From the RoI feature vector, we use a softmax layer to predict the class of the proposed region and also the offset values for the bounding box. Fast R-CNN is significantly faster in training and testing sessions over R-CNN. During testing time, including region proposals slows down the algorithm significantly when compared to not using region proposals. Therefore, region proposals become bottlenecks in Fast R-CNN algorithm affecting its performance.

Faster RCNN improves the region proposal bottleneck by training a preliminary network which learns the region proposal so that the prediction time goes down significantly and unlike the selective search algorithm which is not good in all cases being a static algorithm, a NN can learn region proposal much better.

The testing time for a single image for the different RCNN networks is given in Figure 9.

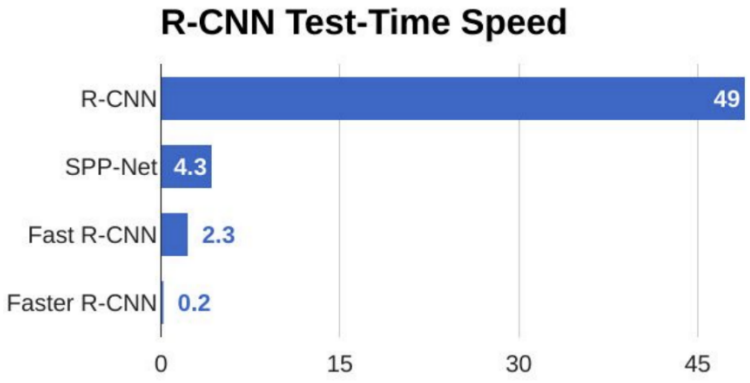


Figure 9: Speed Comparison of different RCNN models

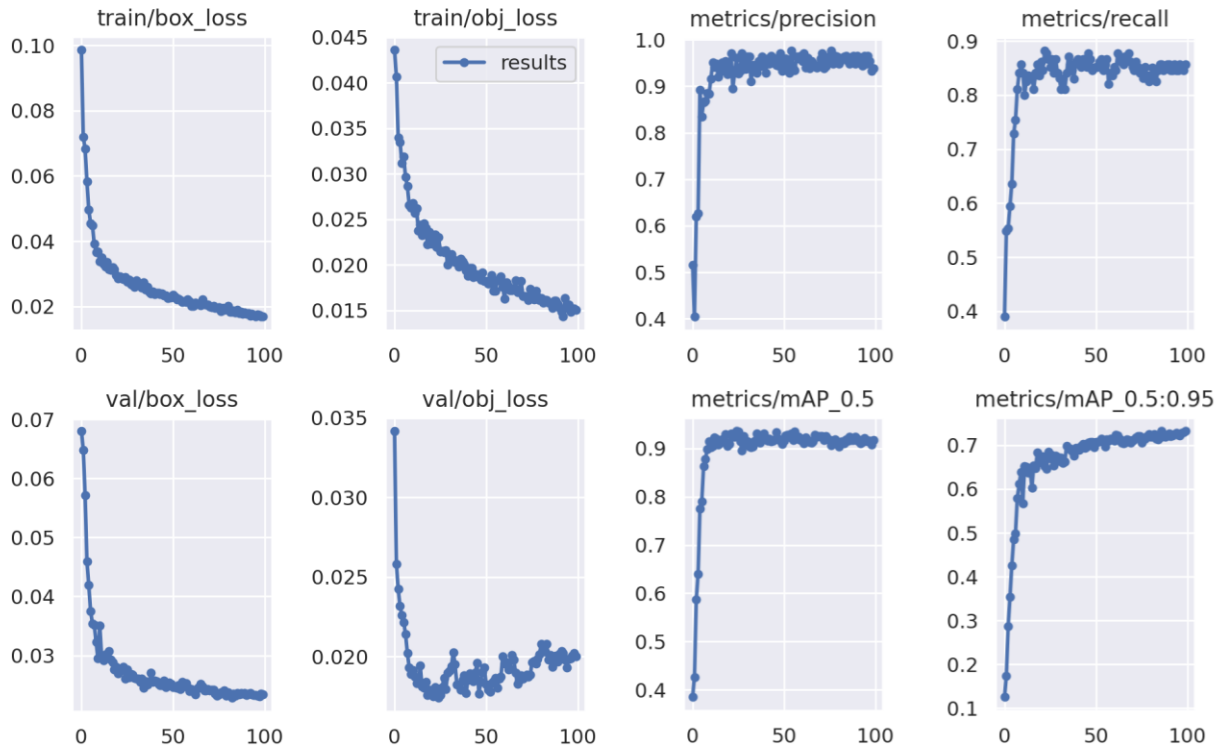
The previous RCNN object detection algorithms use regions to localize the object within the image. The network does not look at the complete image. Instead, parts of the image which have high probabilities of containing the object. YOLO or You Only Look Once is an object detection algorithm much different from the region-based algorithms seen above. In YOLO a single convolutional network predicts the bounding boxes and the class probabilities for these boxes. YOLO works by taking an image and splitting it into an SxS grid, within each of the grid we take m bounding boxes. For each of the bounding box, the network outputs a class probability and offset values for the bounding box. The bounding boxes having the class probability above a threshold value is selected and used to locate the object within the image. YOLO is orders of magnitude faster (45 frames per second) than other object detection algorithms. The

limitation of YOLO algorithm is that it struggles with small objects within the image, for example it might have difficulties in detecting a flock of birds. This is due to the spatial constraints (assumed grid size) of the algorithm.

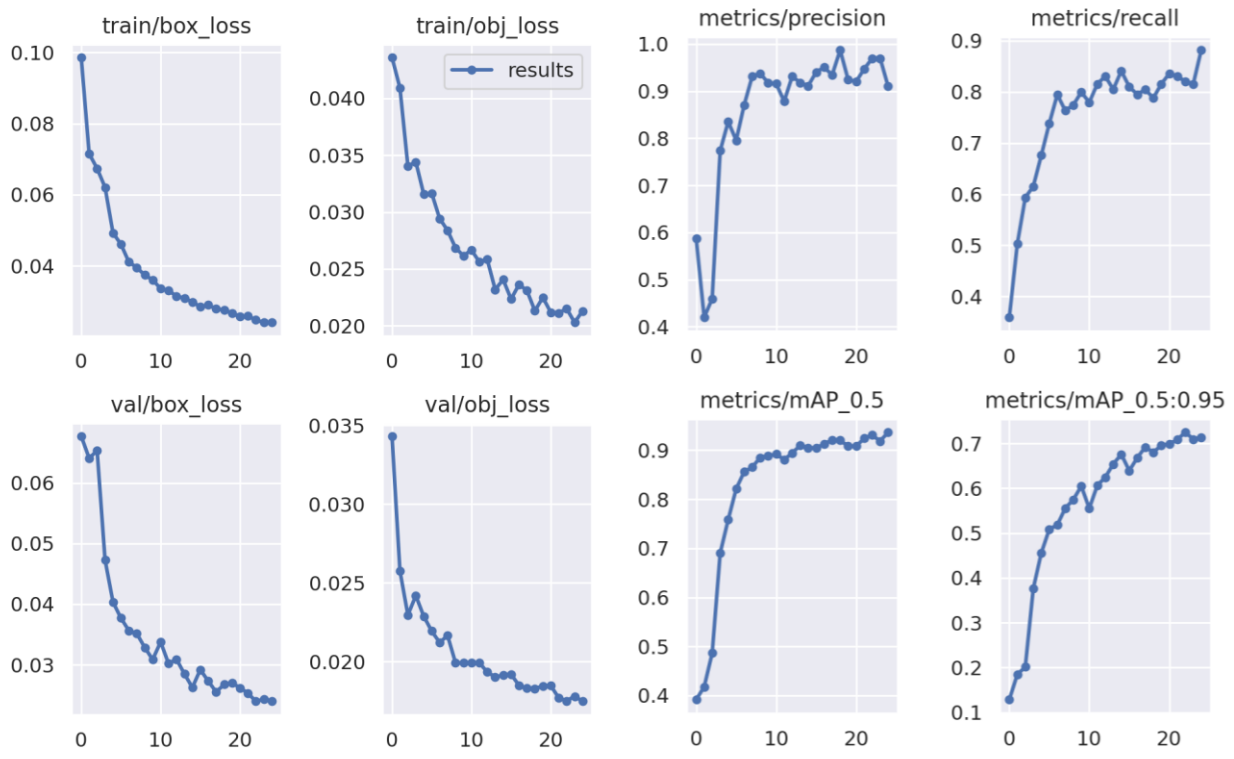
As YOLO is the fastest to train and test, we have chosen to use YOLO for the auto (object) detection problem. For this we have used open-source implementation of YOLO, Ultralytics YOLOv5, which has its pretrained weights derived from the COCO dataset. YOLOv5 is like a tool, allowing easy train, validation and testing with a single command. It requires creating the dataset for the custom problem by making a YAML file with the paths to train, validation and test images and labels. The labels are assigned to each image with each line representing a bounding box. Each line is a 5 tuple with <class> <x_center> <y_center> <width> <height>, where x_center, y_center, width and height are in relative coordinates, i.e., it ranges between 0 and 1 to be immune to resizing of the image. For the given problem, we have chosen the dataset from “Autorickshaw Detection Challenge” [4], which has annotated dataset of 800 autorickshaws. Some examples of the annotations are as shown below:

For training the model, we can choose a model from the set of models provided by YOLOv5 or we can make a custom network architecture. The model selected was YOLOv5s, which is fast and efficient in training being small in size. The dataset was split into 600 – 100 – 100 for train, validation and test datasets.

The training was initially run for 100 epochs to study the losses and select the optimal number of epochs to prevent underfitting and overfitting. The losses and precision of the training is given below. The training took 6 hours on Intel i9-9880H processor and Nvidia GeForce GTX 1650 GPU.



From the results, we retrained the model for 25 epochs as after that validation objective loss started increasing and the precision begins oscillating. The results after training for 25 epochs is shown below:



We use the YOLOv5's best model from epochs to select the best model. The predictions made from the test data are as given below:

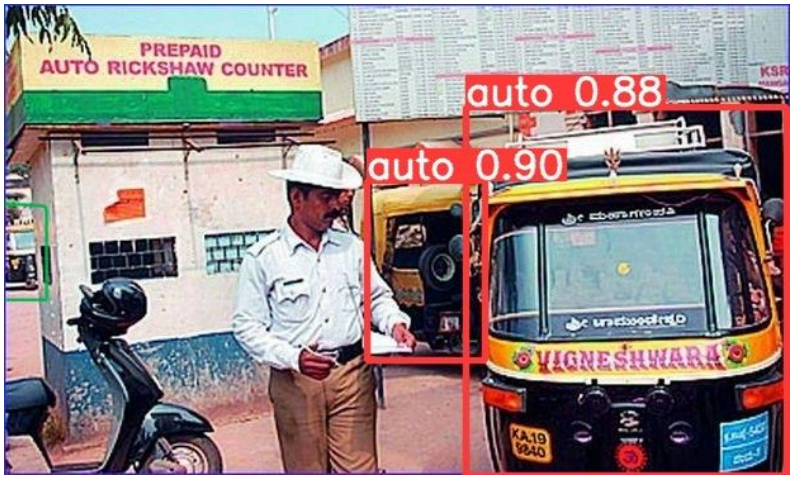
The above images show how YOLOv5 is able to detect multiple autorickshaws in the image with high accuracy.

The above images show how YOLOv5 is able to detect autorickshaws obstructed by other autos or people. Although the prediction accuracy is low but we can see it is able to detect them.

It can also be seen that YOLOv5 is able to detect autorickshaws when they are moving or when the image taken in blurred by a significant magnitude. This can be seen in the image below.

However, YOLOv5 has some problems in detection of autorickshaws. It is unable to detect auto's when they are rotated by 90 degrees in any direction. It also detected Tata Nano and part of a bike as autorickshaw. These are shown in the images below:

We also observe that it is unable to determine autos which are smaller than a certain threshold because of the nature of this algorithm. An example of the same is given below. The undetected auto is shown in green box.



So, we observe that YOLOv5 is good in classifying objects when:

1. There are multiple of them
2. When the classification object is obstructed
3. The size of the image is sufficiently large
4. The image is blurred

However, it can be seen that it performs poorly when the image is:

1. Rotated by a large angle
2. Other objects which have partial characteristics to the trained object
3. Both small and obstructed

References:

1. https://www.kaggle.com/datasets/puneet6060/intel-image-classification
2. https://github.com/ultralytics/yolov5
3. https://blog.paperspace.com/train-yolov5-custom-data/
4. http://cvit.iiit.ac.in/autorickshaw_detection/