**Visual Recognition** March 22, 2024

# Assignment 3 - Report

*Ketaki Tamhanakar (IMT2021017)*
*Munagala Kalyan Ram (IMT2021023)*
*Neha Tamhanakar (IMT2021025)* *Instructor:* Prof. Dinesh Babu Jaygopi

## Part 3a) Training a CNN

We worked with the CIFAR-10 dataset. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

A CNN was trained using the training CIFAR-10 dataset. Testing was done on the CIFAR-10 testing dataset. For the purpose of training, a batch size of 32 was used.

The following three sections cover the optimizations used (to improve back propagation) and the respective outcomes that were observed (for each architecture choice):

Architecture 1:

2 Convolutional layer, 3 fully connected layers:

```
torch.Size([2, 128, 8, 8])
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 32, 32]           1,792
       BatchNorm2d-2          [-1, 64, 32, 32]             128
              ReLU-3          [-1, 64, 32, 32]               0
         MaxPool2d-4          [-1, 64, 16, 16]               0
            Conv2d-5         [-1, 128, 16, 16]          73,856
       BatchNorm2d-6         [-1, 128, 16, 16]             256
              ReLU-7         [-1, 128, 16, 16]               0
         MaxPool2d-8           [-1, 128, 8, 8]               0
            Linear-9                 [-1, 1024]       8,389,632
             ReLU-10                 [-1, 1024]               0
           Dropout-11                [-1, 1024]               0
           Linear-12                 [-1, 512]         524,800
             ReLU-13                 [-1, 512]               0
           Dropout-14                [-1, 512]               0
           Linear-15                  [-1, 10]           5,130
================================================================
Total params: 8,995,594
Trainable params: 8,995,594
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 2.47
Params size (MB): 34.32
Estimated Total Size (MB): 36.80
----------------------------------------------------------------
```

Architecture 2:

4 Convolutional layer, 3 fully connected layers:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 34, 34]           1,792
       BatchNorm2d-2          [-1, 64, 34, 34]             128
              ReLU-3          [-1, 64, 34, 34]               0
            Conv2d-4         [-1, 128, 36, 36]          73,856
       BatchNorm2d-5         [-1, 128, 36, 36]             256
              ReLU-6         [-1, 128, 36, 36]               0
         MaxPool2d-7         [-1, 128, 18, 18]               0
            Conv2d-8         [-1, 256, 18, 18]         295,168
       BatchNorm2d-9         [-1, 256, 18, 18]             512
             ReLU-10         [-1, 256, 18, 18]               0
           Conv2d-11           [-1, 256, 9, 9]         590,080
      BatchNorm2d-12           [-1, 256, 9, 9]             512
             ReLU-13           [-1, 256, 9, 9]               0
        MaxPool2d-14           [-1, 256, 4, 4]               0
           Linear-15                [-1, 2048]       8,390,656
             ReLU-16                [-1, 2048]               0
          Dropout-17                [-1, 2048]               0
           Linear-18                 [-1, 512]       1,049,088
             ReLU-19                 [-1, 512]               0
          Dropout-20                 [-1, 512]               0
           Linear-21                  [-1, 10]           5,130
================================================================
Total params: 10,407,178
Trainable params: 10,407,178
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 8.27
Params size (MB): 39.70
Estimated Total Size (MB): 47.98
----------------------------------------------------------------
```

Architecture 3:

5 Convolutional layer, 3 fully connected layers:

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 32, 32]           1,792
       BatchNorm2d-2           [-1, 64, 32, 32]             128
              ReLU-3           [-1, 64, 32, 32]               0
            Conv2d-4          [-1, 128, 32, 32]          73,856
       BatchNorm2d-5          [-1, 128, 32, 32]             256
              ReLU-6          [-1, 128, 32, 32]               0
         MaxPool2d-7          [-1, 128, 16, 16]               0
            Conv2d-8          [-1, 256, 16, 16]         295,168
       BatchNorm2d-9          [-1, 256, 16, 16]             512
             ReLU-10          [-1, 256, 16, 16]               0
           Conv2d-11          [-1, 256, 16, 16]         590,080
      BatchNorm2d-12          [-1, 256, 16, 16]             512
             ReLU-13          [-1, 256, 16, 16]               0
        MaxPool2d-14            [-1, 256, 8, 8]               0
           Conv2d-15            [-1, 512, 8, 8]       1,180,160
      BatchNorm2d-16            [-1, 512, 8, 8]           1,024
             ReLU-17            [-1, 512, 8, 8]               0
        MaxPool2d-18            [-1, 512, 4, 4]               0
           Linear-19                 [-1, 2048]      16,779,264
             ReLU-20                 [-1, 2048]               0
          Dropout-21                 [-1, 2048]               0
           Linear-22                  [-1, 512]       1,049,088
             ReLU-23                  [-1, 512]               0
          Dropout-24                  [-1, 512]               0
           Linear-25                   [-1, 10]           5,130
================================================================
Total params: 19,976,970
Trainable params: 19,976,970
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 8.75
Params size (MB): 76.21
Estimated Total Size (MB): 84.96
----------------------------------------------------------------
```

## Without Momentum

Stochastic gradient was used for optimization, with the learning rate initially set to 0.1. The following results have been obtained using this optimization:

Architecture 1:

| | Number of epochs | Learning rate | Dropout rate | Activation function | Accuracy |
|---|---|---|---|---|---|
| 1 | 7 | 0.1 | 0.3 | ReLU | 71.48% |
| 2 | 10 | 0.1 | 0.3 | ReLU | 56.08% |
| 3 | 7 | 0.1 | 0.3 | Tanh | 65.00% |
| 4 | 10 | 0.1 | 0.3 | Tanh | 53.14% |
| 5 | 7 | 0.001 | 0.3 | Sigmoid | 42.56% |
| 6 | 10 | 0.001 | 0.3 | Sigmoid | 46.22% |

Architecture 2:

| | Number of epochs | Learning rate | Dropout rate | Activation function | Test Accuracy | Test f1 score |
|---|---|---|---|---|---|---|
| 1 | 30 | 0.1 | 0.3 | ReLU | 83.87% | 84.05% |
| 2 | 30 | 0.1 | 0.3 | Tanh | 78.46% | 78.26% |
| 3 | 30 | 0.1 | 0.3 | Sigmoid | 60.95% | 60.25% |

Architecture 3:

| | Number of epochs | Learning rate | Dropout rate | Activation function | Test Accuracy | Test f1 score |
|---|---|---|---|---|---|---|
| 1 | 30 | 0.1 | 0.3 | ReLU | 87.51% | 87.54% |
| 2 | 30 | 0.1 | 0.5 | ReLU | 87.07% | 86.97% |
| 3 | 20 | 0.1 | 0.3 | ReLU | 86.19% | 86.08% |
| 4 | 40 | 0.1 | 0.3 | ReLU | 88.11% | 88.08% |
| 5 | 30 | 0.001 | 0.3 | ReLU | 78.99% | 78.82% |
| 6 | 30 | 0.01 | 0.3 | ReLU | 85.51% | 85.47% |

Points to note:

- ReLU activation seems to be giving the best results among all three activations.
- More number of epochs led to better accuracy results.
- The third architecture tried out had 5 convolutional layers and 3 linear layers, which gave the best performance in all the trials.
- The number of fully connected layers were not increased too much as that would increase the number of parameters by a large amount. Adding convolutional layers seemed more useful in terms of accuracy (perhaps as more features are captured accurately).
- Dropout was only applied in the fully connected layer as the majority of parameters are there and are more likely to co-adapt themselves and cause over-fitting.

## With Momentum

Stochastic gradient was used for optimization and adding momentum. Results were observed using the following hyperparameters:
The learning rate was set as 0.1, 0.01 and 0.001 , and the best results were given using 0.001.

Architecture 2:

| | Number of epochs | Learning rate | Momentum value | Dropout rate | Activation function | Test Accuracy |
|---|---|---|---|---|---|---|
| 1 | 20 | 0.001 | 0.9 | 0.3 | ReLU | 0.819 |
| 2 | 30 | 0.001 | 0.9 | 0.3 | ReLU | 0.813 |
| 3 | 30 | 0.001 | 0.9 | 0.3 | Tanh | 0.756 |
| 4 | 30 | 0.001 | 0.9 | 0.3 | Sigmoid | 0.35 |
| 5 | 30 | 0.001 | 0.8 | 0.3 | ReLU | 0.788 |
| 6 | 40 | 0.001 | 0.9 | 0.3 | ReLU | 0.820 |

It was observed that the ReLU activation function was giving the best results. Hence the rest of the results were obtained using ReLU.

Architecture 3:

| | Number of epochs | Learning rate | Momentum value | Dropout rate | Activation function | Test Accuracy |
|---|---|---|---|---|---|---|
| 1 | 20 | 0.001 | 0.9 | 0.3 | ReLU | 0.821 |
| 2 | 30 | 0.001 | 0.9 | 0.3 | ReLU | 0.845 |
| 3 | 30 | 0.001 | 0.7 | 0.3 | ReLU | 0.820 |
| 4 | 40 | 0.1 | 0.9 | 0.3 | ReLU | 0.484 |
| 5 | 40 | 0.01 | 0.9 | 0.3 | ReLU | 0.873 |
| 6 | 40 | 0.001 | 0.9 | 0.3 | ReLU | 0.849 |
| 7 | 20 | 0.001 | 0.9 | 0.3 | ReLU | 0.858 |
| 8 | 20 | 0.001 | 0.7 | 0.3 | ReLU | 0.86 |
| 9 | 20 | 0.001 | 0.7 | 0.1 | ReLU | 0.820 |

Architecture 3 gave the best results. The best accuracy obtained is equal to 0.873.

## Adam Optimization

The Adam Optimizer was used for optimization and the results were observed using the following hyperparameters and architectures:

The learning rate was set to 0.001 as it gave the best results for this setting.

The architecture designed was inspired from the VGG-16 architecture where we made use of multiple 3*3 filters and max pooling layers. Batch Normalization is applied after each convolution layer followed by the activation function. A dropout layer is added after each fully connected layer.

Architecture Adam1 : It has 4 Convolutional Layers and 3 Fully Connected Layers.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 34, 34]           1,792
       BatchNorm2d-2          [-1, 64, 34, 34]             128
              ReLU-3          [-1, 64, 34, 34]               0
            Conv2d-4         [-1, 128, 36, 36]          73,856
       BatchNorm2d-5         [-1, 128, 36, 36]             256
              ReLU-6         [-1, 128, 36, 36]               0
            Conv2d-7         [-1, 128, 34, 34]         147,584
       BatchNorm2d-8         [-1, 128, 34, 34]             256
              ReLU-9         [-1, 128, 34, 34]               0
         MaxPool2d-10         [-1, 128, 17, 17]               0
           Conv2d-11         [-1, 256, 17, 17]         295,168
      BatchNorm2d-12         [-1, 256, 17, 17]             512
             ReLU-13         [-1, 256, 17, 17]               0
           Conv2d-14           [-1, 256, 9, 9]         590,080
      BatchNorm2d-15           [-1, 256, 9, 9]             512
             ReLU-16           [-1, 256, 9, 9]               0
         MaxPool2d-17           [-1, 256, 4, 4]               0
           Linear-18                 [-1, 2048]       8,390,656
             ReLU-19                 [-1, 2048]               0
          Dropout-20                 [-1, 2048]               0
           Linear-21                  [-1, 512]       1,049,088
             ReLU-22                  [-1, 512]               0
          Dropout-23                  [-1, 512]               0
           Linear-24                   [-1, 10]           5,130
================================================================
Total params: 10,555,018
Trainable params: 10,555,018
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 11.42
Params size (MB): 40.26
Estimated Total Size (MB): 51.69
----------------------------------------------------------------
```

Architecture Adam2 : It has 4 Convolutional Layers and 4 Fully Connected Layers

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 34, 34]           1,792
       BatchNorm2d-2          [-1, 64, 34, 34]             128
              ReLU-3          [-1, 64, 34, 34]               0
            Conv2d-4         [-1, 128, 36, 36]          73,856
       BatchNorm2d-5         [-1, 128, 36, 36]             256
              ReLU-6         [-1, 128, 36, 36]               0
         MaxPool2d-7         [-1, 128, 18, 18]               0
            Conv2d-8         [-1, 256, 18, 18]         295,168
       BatchNorm2d-9         [-1, 256, 18, 18]             512
             ReLU-10         [-1, 256, 18, 18]               0
           Conv2d-11           [-1, 512, 9, 9]       1,180,160
      BatchNorm2d-12           [-1, 512, 9, 9]           1,024
             ReLU-13           [-1, 512, 9, 9]               0
         MaxPool2d-14           [-1, 512, 4, 4]               0
           Linear-15                 [-1, 2048]      16,779,264
             ReLU-16                 [-1, 2048]               0
          Dropout-17                 [-1, 2048]               0
           Linear-18                  [-1, 512]       1,049,088
             ReLU-19                  [-1, 512]               0
          Dropout-20                  [-1, 512]               0
           Linear-21                   [-1, 60]          30,780
             ReLU-22                   [-1, 60]               0
          Dropout-23                   [-1, 60]               0
           Linear-24                   [-1, 10]             610
================================================================
Total params: 19,412,638
Trainable params: 19,412,638
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 8.78
Params size (MB): 74.05
Estimated Total Size (MB): 82.84
----------------------------------------------------------------
```

Architecture Adam3 : It has 5 Convolutional Layers and 3 Fully Connected Layers

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 34, 34]           1,792
       BatchNorm2d-2          [-1, 64, 34, 34]             128
              ReLU-3          [-1, 64, 34, 34]               0
            Conv2d-4         [-1, 128, 36, 36]          73,856
       BatchNorm2d-5         [-1, 128, 36, 36]             256
              ReLU-6         [-1, 128, 36, 36]               0
            Conv2d-7         [-1, 128, 34, 34]         147,584
       BatchNorm2d-8         [-1, 128, 34, 34]             256
              ReLU-9         [-1, 128, 34, 34]               0
         MaxPool2d-10         [-1, 128, 17, 17]               0
           Conv2d-11         [-1, 256, 17, 17]         295,168
      BatchNorm2d-12         [-1, 256, 17, 17]             512
             ReLU-13         [-1, 256, 17, 17]               0
           Conv2d-14           [-1, 256, 9, 9]         590,080
      BatchNorm2d-15           [-1, 256, 9, 9]             512
             ReLU-16           [-1, 256, 9, 9]               0
         MaxPool2d-17           [-1, 256, 4, 4]               0
           Linear-18                 [-1, 2048]       8,390,656
             ReLU-19                 [-1, 2048]               0
          Dropout-20                 [-1, 2048]               0
           Linear-21                  [-1, 512]       1,049,088
             ReLU-22                  [-1, 512]               0
          Dropout-23                  [-1, 512]               0
           Linear-24                   [-1, 10]           5,130
================================================================
Total params: 10,555,018
Trainable params: 10,555,018
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.01
Forward/backward pass size (MB): 11.42
Params size (MB): 40.26
Estimated Total Size (MB): 51.69
----------------------------------------------------------------
```

| Model | Number of epochs | Learning rate | Batch Norm | Dropout rate | Activation function | Time (min) | F1 Score |
|---|---|---|---|---|---|---|---|
| Adam1 | 25 | 0.001 | Yes | 0.3 | ReLU | 8 | 0.82042 |
| Adam1 | 35 | 0.001 | Yes | 0.3 | Sigmoid | 20 | 0.71014 |
| Adam1 | 25 | 0.001 | Yes | 0.3 | Tanh | 10 | 0.78367 |
| Adam1 | 30 | 0.01 | Yes | 0.4 | ReLU | 10 | 0.80315 |
| Adam2 | 25 | 0.001 | Yes | 0.3 | ReLU | 11 | 0.80861 |
| Adam2 | 25 | 0.001 | No | 0.3 | ReLU | 9.5 | 0.76263 |
| Adam1 | 25 | 0.001 | No | 0.3 | ReLU | 7 | 0.76504 |
| Adam3 | 30 | 0.001 | Yes | 0.3 | ReLU | 10 | 0.82297 |

From the table above we can see:

- The Batch Normalization layer helps in improving the test accuracy by 4% to 5% and without this layer , the training time reduces by approximately 1 min.
- On comparing the different activation functions, we can see that ReLU gives the best result and has the least training time , Tanh is slower and gives a lower accuracy and Sigmoid gives the worst accuracy (10 % drop) and takes twice the time to train and converge.
- By comparing the 3 architectures, we can see that adding additional fully connected layers after 3 layers doesn't improve the accuracy and causes a significant increase in parameters however increasing convolution layers improves the accuracy without a significant increase in parameters.

### Recommended Architecture

Following our experiments, we have concluded that architecture 3 (shown above), gave the best accuracy results, using ReLU for activation, a dropout rate of 0.3 and a learning rate of 0.1 for the stochastic gradient descent algorithm, with no momentum/adam optimization.

## Part 3b) Transfer learning

For this part, we chose to work with the Fruits and Vegetables Image Recognition Dataset.

This dataset contains images of the following food items:

Fruits- banana, apple, pear, grapes, orange, kiwi, watermelon, pomegranate, pineapple, mango.

Vegetables- cucumber, carrot, capsicum, onion, potato, lemon, tomato, raddish, beetroot, cabbage, lettuce, spinach, soy bean, cauliflower, bell pepper, chilli pepper, turnip, corn, sweetcorn, sweet potato, paprika, jalepeño, ginger, garlic, peas, eggplant.

We used the following networks as feature extractors:

### Alexnet

AlexNet is a convolutional neural network that is 8 layers deep.The network was trained on more than a million images from the ImageNet database. A pre-trained model was used for the purpose of this assignment.

Here are the results obtained by different classifiers, applied on the output of AlexNet, on the validation set of the Fruits and Vegetables Image Recognition dataset:

- Logistic regression

| **max iterations** | **Accuracy** |
|---|---|
| 100000 | 0.974 |

- SVC

| **kernel** | **Accuracy** |
|---|---|
| linear | 0.974 |

- Random forest

| **max depth** | **Accuracy** | **Number of estimators** |
|---|---|---|
| 10 | 0.968 | 100 |
| 20 | 0.974 | 100 |
| 20 | 0.977 | 500 |
| 50 | 0.971 | 100 |

- KNN

| Number of neighbours | Accuracy |
|---|---|
| 4 | 0.795 |

We also used this architecture (AlexNet) to make classifications in the Bike vs Horse dataset, in a similar manner. Here are the results of the classification, by the following classifiers:

- Logistic regression

| max iterations | Accuracy |
|---|---|
| 100000 | 0.988 |

- SVC

| kernel | Accuracy |
|---|---|
| linear | 0.988 |

- Random forest

| max depth | Accuracy | Number of estimators |
|---|---|---|
| 10 | 1 | 100 |
| 20 | 1 | 100 |

- KNN

| Number of neighbours | Accuracy |
|---|---|
| 4 | 0.9888 |

The pre-trained weights of AlexNet overall did a good job in performing classifiaction. Following the experimentation, the best accuracy achieved was 0.977 for the Fruits and Vegetables dataset, and an accuracy of 1 for the Bike vs Horse dataset.

## VGG-16

With VGG as the feature extractor, the following models were used for classification:

- Logistic regression

| max iterations | Accuracy |
|---|---|
| 100000 | 0.994 |

- SVC

| kernel | Accuracy |
|---|---|
| linear | 0.994 |
| rbf | 0.982 |

- Random forest

| max depth | Accuracy |
|---|---|
| 10 | 0.956 |

- KNN

| Number of neighbours | Accuracy |
|---|---|
| 4 | 0.994 |

For the Bikes and Horses dataset used in the previous assignment, the following results were obtained using VGG-16:

- Logistic regression

| max iterations | Accuracy |
|---|---|
| 100000 | 1.0 |

- SVC

| kernel | Accuracy |
|---|---|
| linear | 0.922 |
| rbf | 0.633 |

- Random forest

| max depth | Accuracy |
|---|---|
| 10 | 1.0 |

- KNN

| Number of neighbours | Accuracy |
|---|---|
| 4 | 1.0 |

## Resnet

With ResNet as the feature extractor, 3 fully connected layers were added where the number of outputs is the number of classes in the dataset. Pytorch's pretrained resNet weights were used as initial weights.

- **Fruit and vegetable Dataset**

  The new model was trained on the train data for 75 epochs with a learning rate of 0.001 and used the Adam optimizer with a cross entropy loss function.

  A batch size of 16 is used where the image dimensions are (3,720,720).

  The dropout rate is 0.3 and the ReLU activation function is used.

  The test accuracy obtained is **0.96379** and the f1 score is **0.96311**.

- **Bike and Horse Dataset**

  The dataset is split into train and validation datasets using a split of 0.8 .

  The model is trained for 15 epochs with a learning rate of 0.005 and used the Adam optimizer with a cross entropy loss function.

  The batch size is 4 and the image dimensions are reshaped to (3,280,480)

  The dropout rate if 0.3 the ReLU activation function is used.

  The accuracy obtained on the validation data is **1.00** and the f1 score is **1.00**.

# Part 3c) YOLO v1 vs YOLO v2

1. **High Resolution Classifier**: The original YOLO trains the classifier network at 224 $\times$ 224 and increases the resolution to 448 for detection. This means the network has to simultaneously switch to learning object detection and adjust to the new input resolution. For YOLOv2 we first fine tune the classification network at the full 448 $\times$ 448 resolution for 10 epochs on ImageNet. This gives the network time to adjust its filters to work better on higher resolution input. We then fine tune the resulting network on detection.

2. **Batch Normalization**: The incorporation of batch normalization in YOLO offers notable enhancements in convergence and obviates the necessity for alternative regularization techniques. Implementing batch normalization across all convolutional layers results in a considerable increase of over 2% in mean Average Precision (mAP). Additionally, batch normalization serves as a form of regularization for the model. Consequently, the inclusion of batch normalization permits the removal of dropout from the model without risking overfitting.

3. **Convolutional With Anchor Boxes**: In YOLOv2, convolutional layers with anchor boxes are used to predict bounding boxes, eliminating the need for fully connected layers. The region proposal network (RPN) in Faster R-CNN, which predicts offsets and confidences for anchor boxes, serves as inspiration for this approach. By predicting offsets instead of coordinates, the problem is simplified, facilitating network learning. Additionally, anchor boxes decouple class prediction from spatial location, allowing for prediction of class and objectness for every anchor box. Although the use of anchor boxes results in a slight decrease in accuracy, the increase in recall indicates improved potential for the model to refine its predictions.

4. **Dimension Clusters**: In YOLOv2, anchor box dimensions are traditionally hand-picked, leading to potential challenges in optimal box selection. To address this, k-means clustering is employed on training set bounding boxes to automatically identify suitable anchor box priors. By using a distance metric based on IOU, the clustering process aims to generate anchor boxes that are independent of box size. Through experimentation, a value of k=5 is determined to strike a balance between model complexity and recall performance. The resulting cluster centroids exhibit significant differences from hand-picked anchor boxes, with a notable emphasis on tall, thin boxes. Comparison with hand-picked anchor boxes demonstrates that clustering-based priors yield comparable or superior average IOU scores, indicating improved model representation and learning ease.

5. **Multi-Scale Training**: In YOLOv2, multi-scale training enables the model to adapt to varying input image sizes, promoting robustness across different resolutions. By randomly selecting new image dimensions every few iterations and resizing the network accordingly, the model learns to predict detections effectively across a range of resolutions. This approach facilitates a flexible trade-off between speed and accuracy, with YOLOv2 performing as a cost-effective and accurate detector at lower resolutions while maintaining state-of-the-art performance at higher resolutions. With impressive mAP scores and real-time processing capabilities, YOLOv2 offers versatility for applications spanning from smaller GPUs to high-resolution video processing.

# Part 3d) Object Tracking

## Data Collection

We collected 4 videos from the traffic junctions of Brigade Road and Vidhana Soudha which consists of multiple cars to perform car tracking.

**Link to Dataset:** Link

## Modules Used

1. For Faster RCNN, we used the pretrained weights present in pytorch which is trained on the MS-COCO dataset.
2. We used a pre-implemented version of SORT which was present on Github and used the "deepsort-realtime" python library for DEEPSORT .

## Procedure Used

1. We first split the video to its individual frames and executed the steps below.
2. Applied the detection model( YOLOv5 /Faster RCNN ).
3. Filtered out the detections corresponding to cars.
4. Applied a Multiple Object Tracking (MOT) instance(SORT / DeepSORT) to track the filtered detections.
5. Counted the number of cars using the tracking IDs. To do this, we maintained a set with the unique tracking IDs
6. Added necessary annotations to the frame.
7. Saved the annotated frame to a file.

For detection, FasterRCNN and YOLOv5 were used. For object tracking, SORT and DeepSORT were used.

## YOLOV5 Vs FasterRCNN

In terms of speed, YOLOv5 provided superior real-time performance in terms of frames per second and did well in tracking cars closer in the frame, but failed to track cars that were far away from the frame.

However, we found that Faster RCNN gave better overall detection of on-screen objects, particularly cars that were situated farther away in the frame.

## SORT vs DeepSORT

Both algorithms aim to achieve object tracking, with distinctions as follows:

1. SORT employs velocity and filters for detection, leveraging appearance and motion, whereas DeepSORT utilizes a CNN as a feature extractor alongside Kalman filters.
2. In real-time scenarios for online video processing, SORT yields superior results, particularly on devices with limited GPU resources, as it avoids the overhead of neural network forward passes.
3. DeepSORT excels in handling occlusions and crowded screens, as the CNN architecture aids in better tracking under such conditions.
4. Considering the trade-off between real-time performance and accuracy, the following conclusions emerge:
   (a) For real-time or online detection on standard hardware, SORT is preferable, as online data processing typically compromises accuracy due to frame loss and compression.
   (b) For scenarios prioritizing accuracy over frame rates, DeepSORT, powered by deep learning, is the preferred choice.

## Observations

We compared the count obtained from running the algorithms with the ground truth value by manually counting the number of cars.

**Link to the outputs:** Link

- Video 1 (Brigade Road) : Has 42 cars
  1. FasterRCNN + DeepSort : 25
  2. FasterRCNN + Sort : 50
  3. YoloV5 + DeepSort : 8
  4. YoloV5 + Sort : 41
- Video 2 (Vidhana Soudha 1) : Has 14 cars
  1. FasterRCNN + DeepSort : 25
  2. FasterRCNN + Sort : 22
  3. YoloV5 + DeepSort : 6
  4. YoloV5 + Sort : 18
- Video 3 (Vidhana Soudha 2) : Has 5 cars
  1. FasterRCNN + DeepSort : 5
  2. FasterRCNN + Sort : 8
  3. YoloV5 + DeepSort : 0

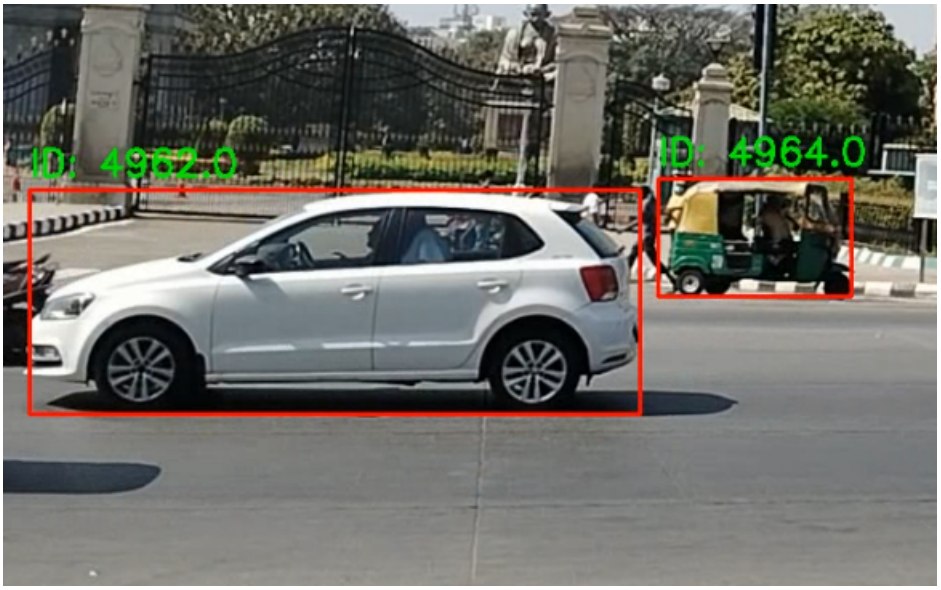
4. YoloV5 + Sort : 6

- Video 4 (Vidhana Soudha 3) : Has 20 cars

1. FasterRCNN + DeepSort : 19
2. FasterRCNN + Sort : 23
3. YoloV5 + DeepSort : 5
4. YoloV5 + Sort : 30

## Conclusions

- Autos are falsely being tracked as the dataset on which the model is pretrained on doesn't have an exclusive class for an auto.




- In video 2,3,4 , the FasterRCNN + DeepSort algorithm gives a result which is closer to the ground truth value. However, it takes longer to perform the tracker.
- However, in video 1, FasterRCNN gives a better result with the sort algorithm whereas deepsort's answer is not close to the ground truth value. The result obtained with YOLOv5 + SORT was very close to the ground truth.
- DeepSORT did not seem to pick up on far away objects in the videos used for testing.
- The results obtained are reasonable given that the traffic junctions are crowded and the videos are taken with the device moving along with the vechicles, which has affected the final accuracy of car counting.

## References

1. Fruit and Vegetable Dataset
2. Faster RCNN Weights

3. SORT

4. DeepSORT

5. YoloV5 Weights

6. YoloV1 Paper

7. YoloV2 Paper

8. YOLO v5 + DeepSort

9. Torchvision documentation for pre-trained models, sklearn and Pytorch documentation.