Vibhor Agarwal

2020349

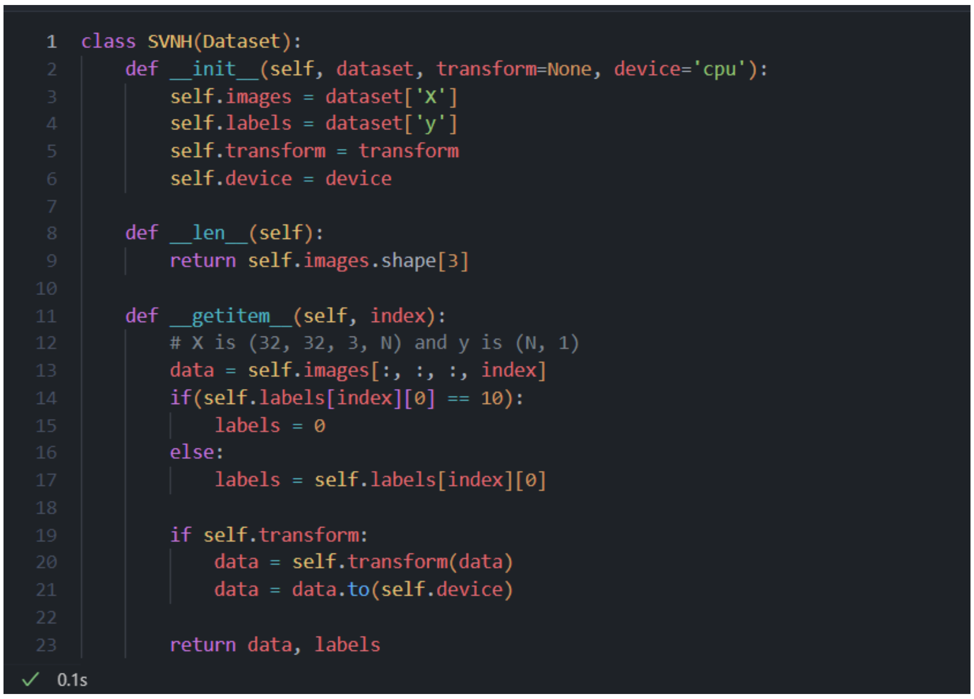
CV Assignment 1

Q1

Part 1

(A)

- Here we create custom dataset class and then make the custom dataset object containing all the data.

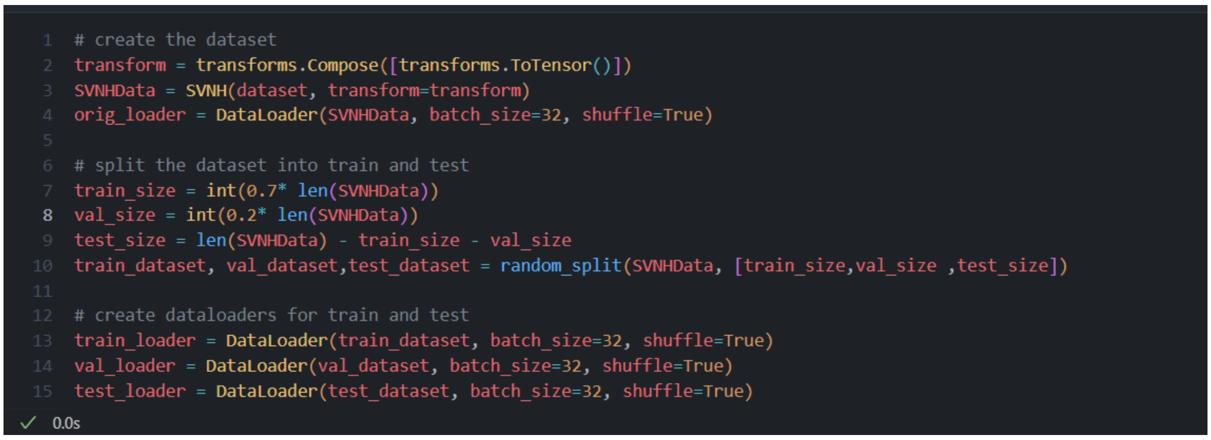


```python
class SVNH(Dataset):
    def __init__(self, dataset, transform=None, device='cpu'):
        self.images = dataset['X']
        self.labels = dataset['y']
        self.transform = transform
        self.device = device

    def __len__(self):
        return self.images.shape[3]

    def __getitem__(self, index):
        # X is (32, 32, 3, N) and y is (N, 1)
        data = self.images[:, :, :, index]
        if(self.labels[index][0] == 10):
            labels = 0
        else:
            labels = self.labels[index][0]

        if self.transform:
            data = self.transform(data)
            data = data.to(self.device)

        return data, labels
```

```
✓ 0.1s
```

(B)

- Now we make custom dataloaders and make train,val and test split.

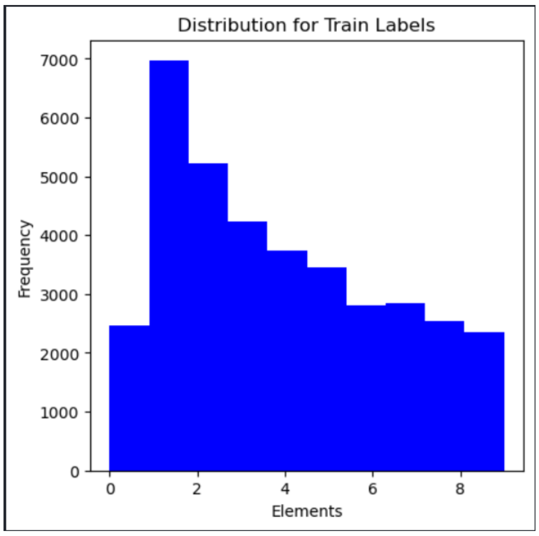


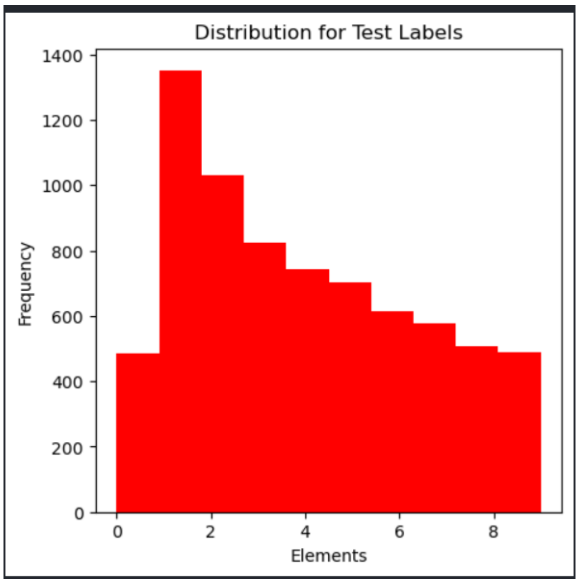
```python
# create the dataset
transform = transforms.Compose([transforms.ToTensor()])
SVNHData = SVNH(dataset, transform=transform)
orig_loader = DataLoader(SVNHData, batch_size=32, shuffle=True)

# split the dataset into train and test
train_size = int(0.7* len(SVNHData))
val_size = int(0.2* len(SVNHData))
test_size = len(SVNHData) - train_size - val_size
train_dataset, val_dataset,test_dataset = random_split(SVNHData, [train_size,val_size ,test_size])

# create dataloaders for train and test
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=True)
```

```
✓ 0.0s
```

(C)

- Now we visualize the class distribution for train and test split.

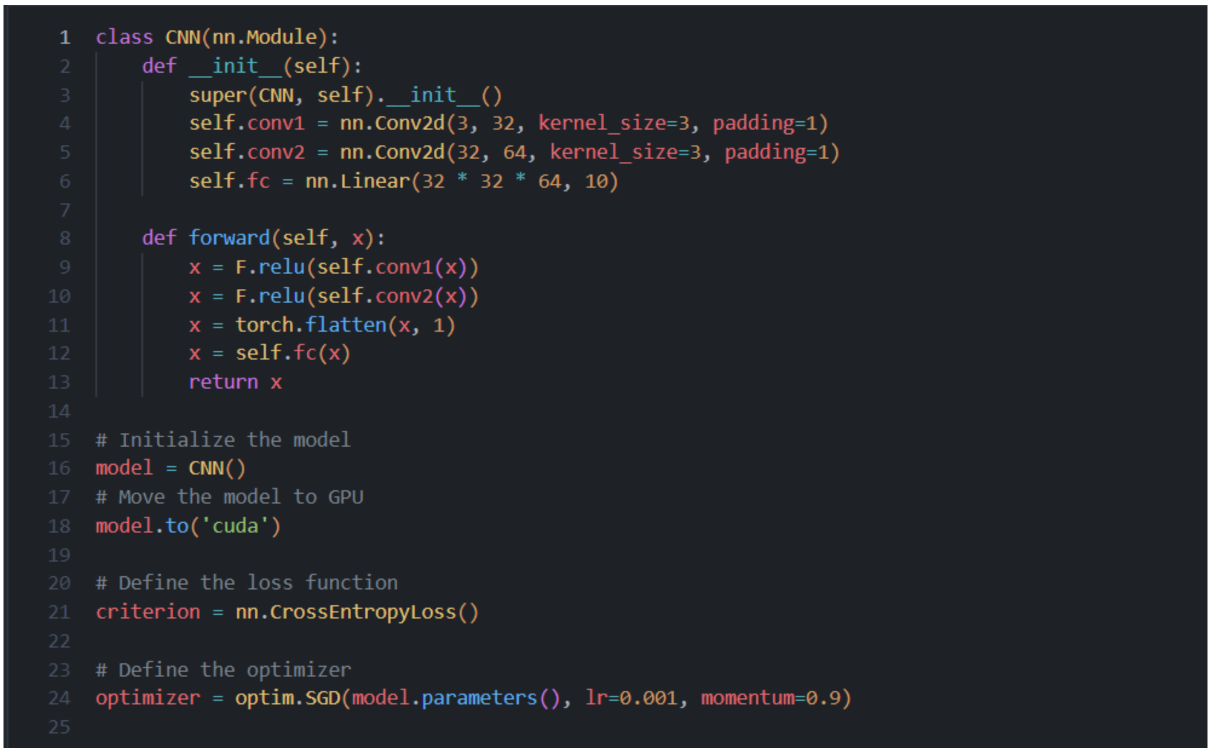




Part 2

(A)

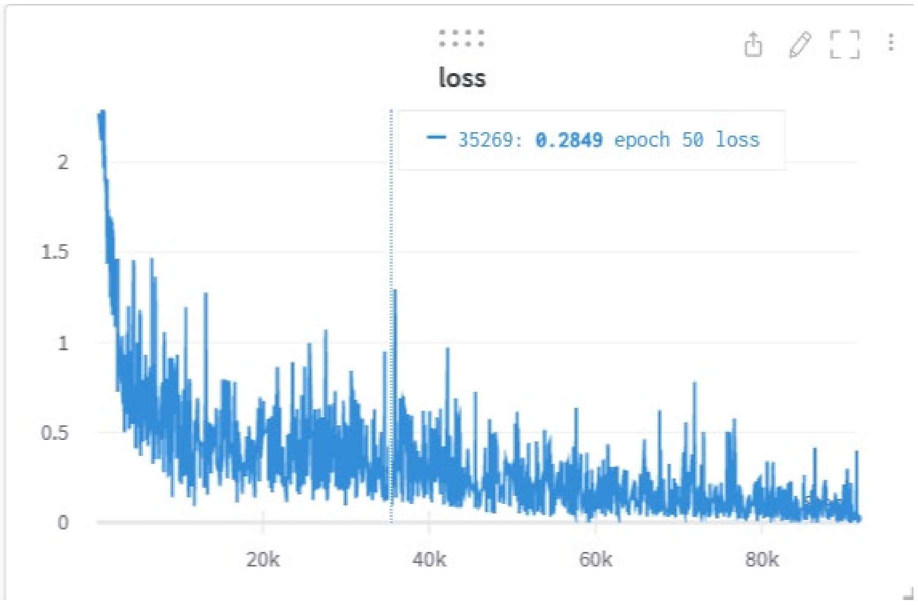
- Here we create a custom model class with given classification and last layer being a fully connected layer with output classes = 10.

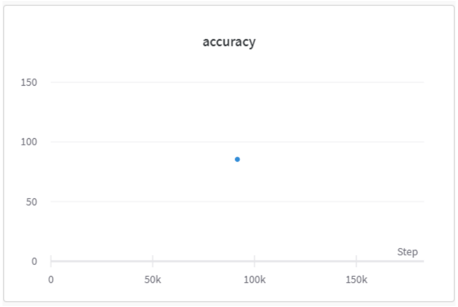


```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.fc = nn.Linear(32 * 32 * 64, 10)

    def forward(self, x):
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = torch.flatten(x, 1)
        x = self.fc(x)
        return x

# Initialize the model
model = CNN()
# Move the model to GPU
model.to('cuda')

# Define the loss function
criterion = nn.CrossEntropyLoss()

# Define the optimizer
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

```

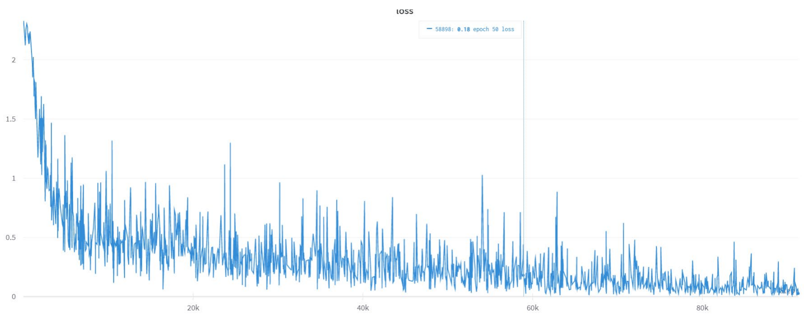
(B)

We train the model using cross entropy loss and the loss is logged using the Wandb and following is the loss of the model on training set

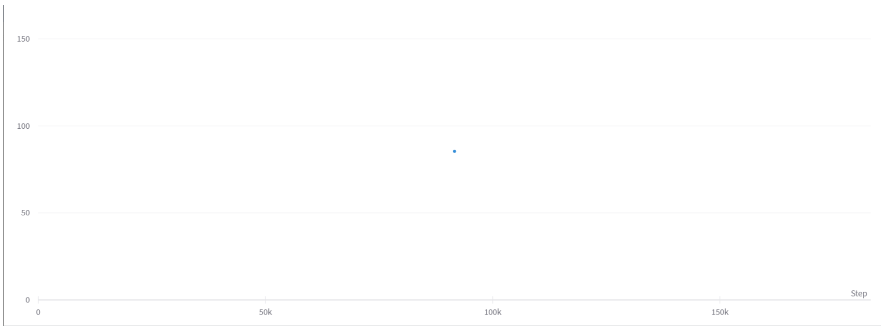




Here is the loss on the validation set.

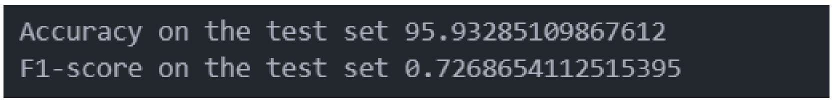


Here is the accuracy on the validation set.

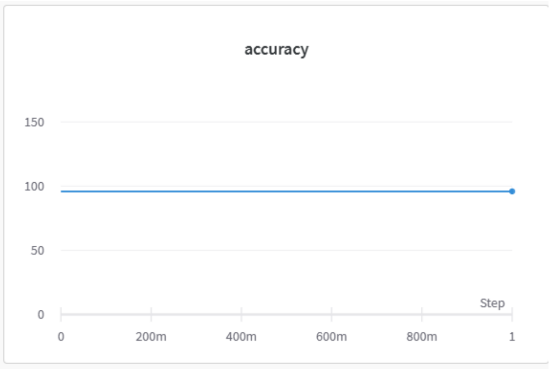


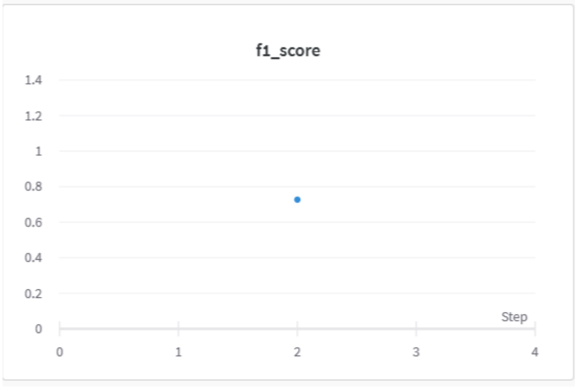
(C)

The accuracy and F1 Score on the test set is as follows.

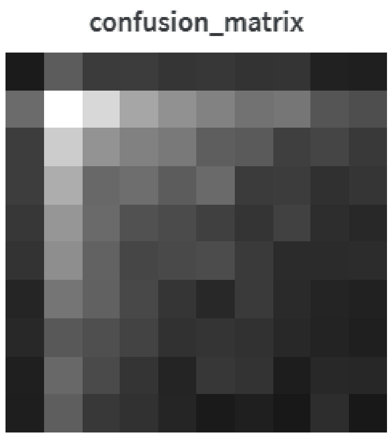


```
Accuracy on the test set 95.93285109867612
F1-score on the test set 0.7268654112515395
```





The Confusion Matrix on the test set is as follows.

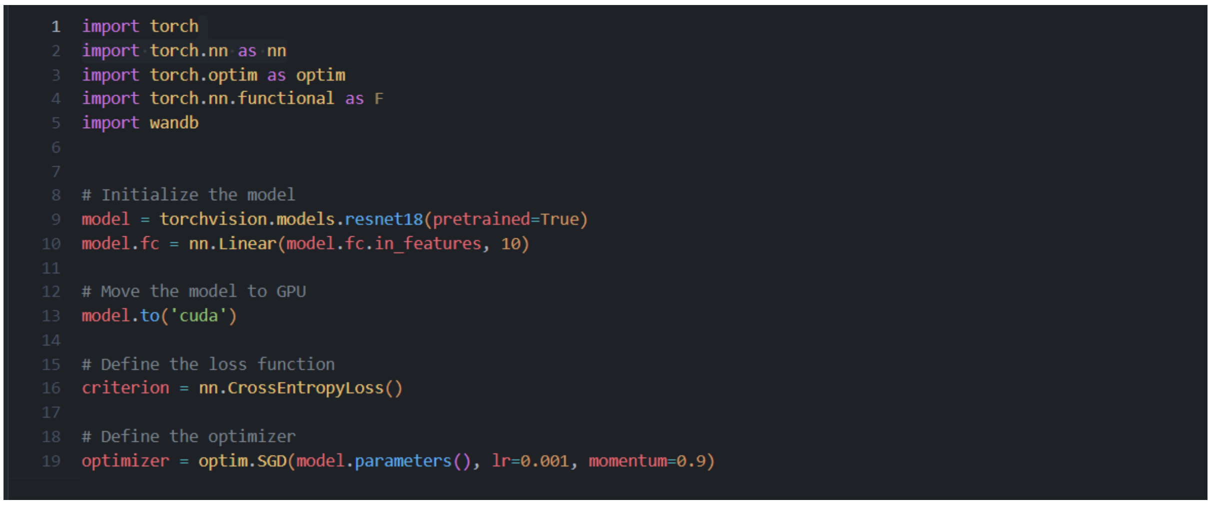


Confusion Matrix

Part 3

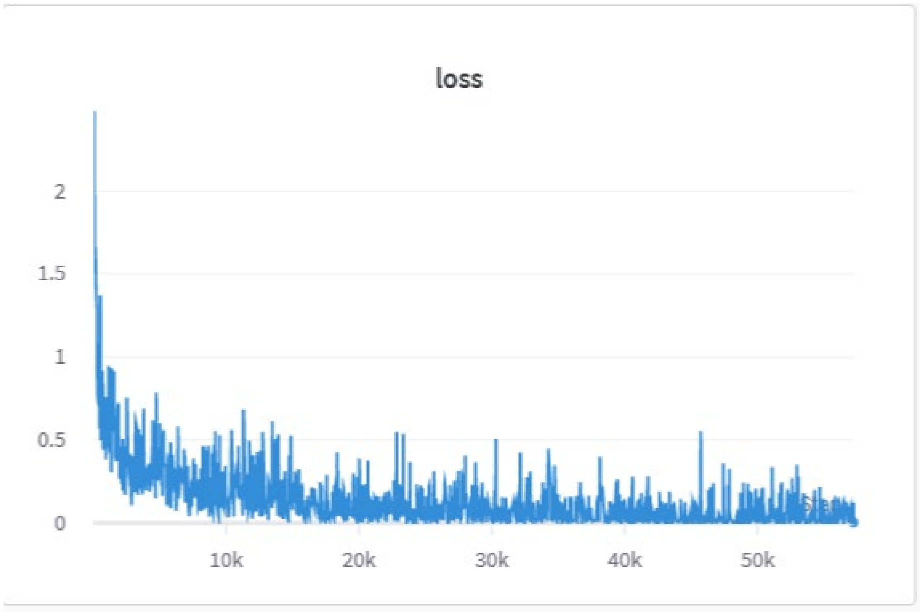
(A)

Here we use the resnet18 model with pretrained weights and train it again



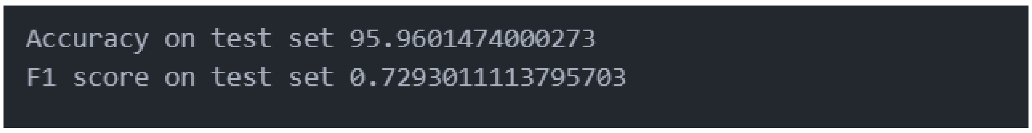
```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import wandb


# Initialize the model
model = torchvision.models.resnet18(pretrained=True)
model.fc = nn.Linear(model.fc.in_features, 10)

# Move the model to GPU
model.to('cuda')

# Define the loss function
criterion = nn.CrossEntropyLoss()

# Define the optimizer
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

The training loss is as follows

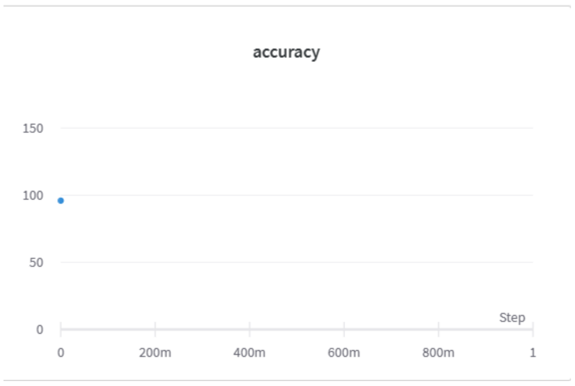


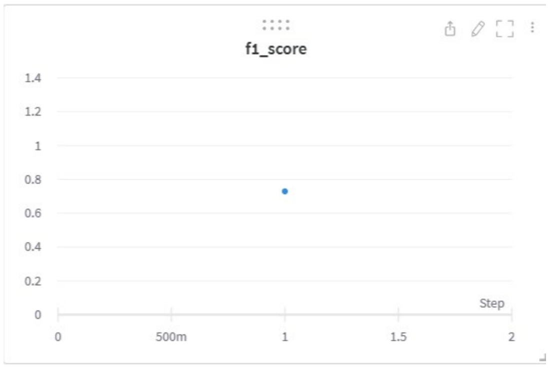
(B)

The accuracy on test set is as follows

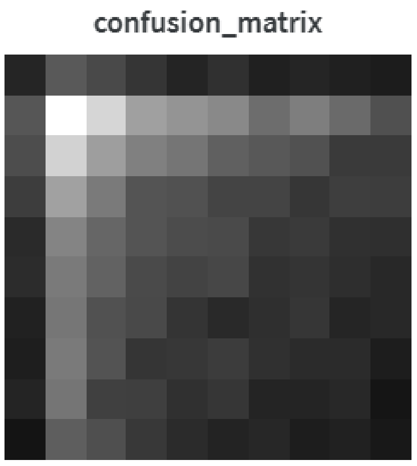


```
Accuracy on test set 95.9601474000273
F1 score on test set 0.7293011113795703
```





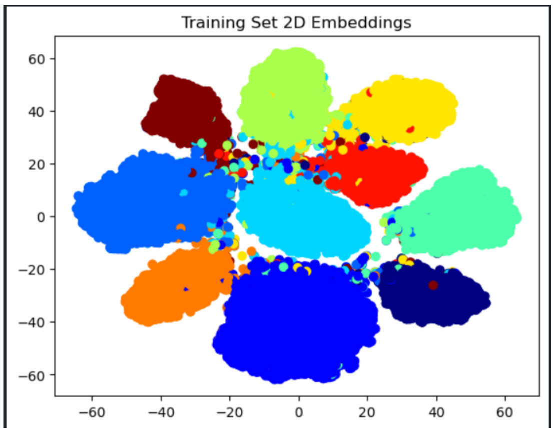
The confusion matrix is as follows

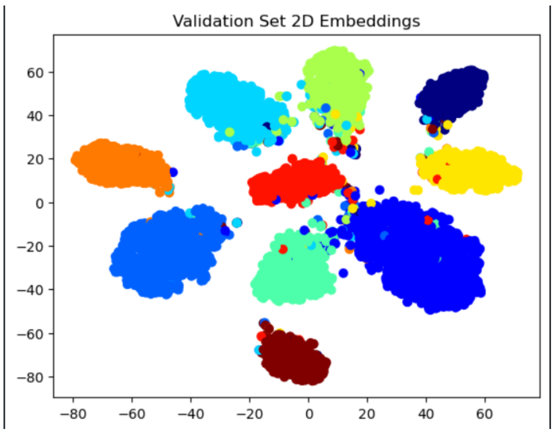


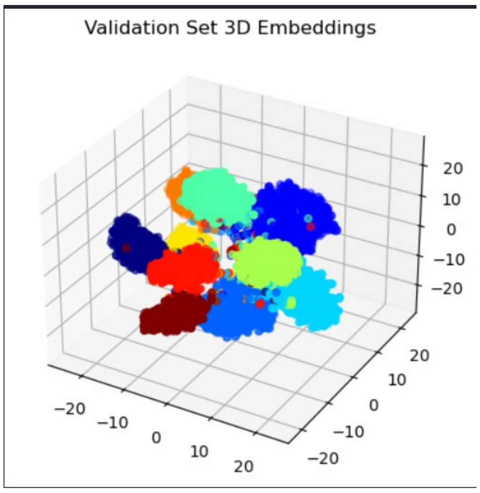
Confusion Matrix

(C)

The TSNE Maps of feature vectors are as follows



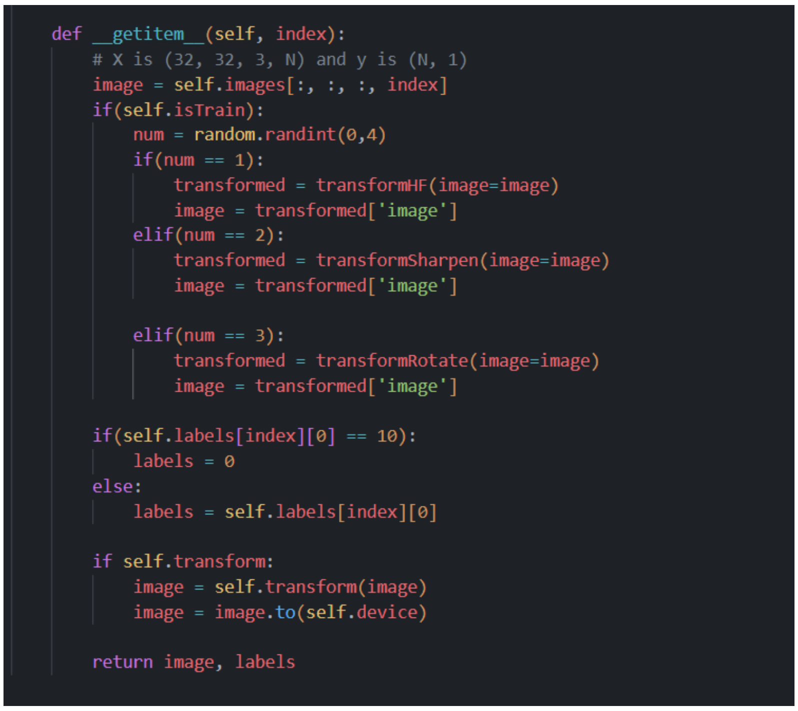




Part 4

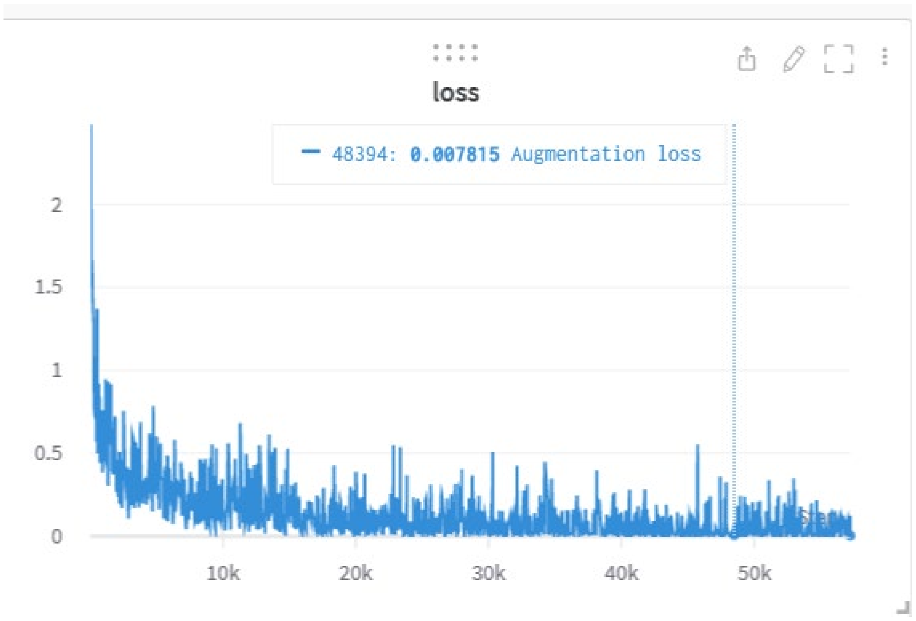
The following code does the Image augmentation randomly on the training data

- We do horizontal flip, sharpen and rotation randomly if any

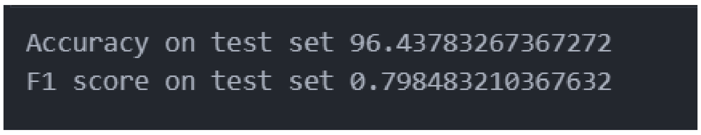


```
def __getitem__(self, index):
    # X is (32, 32, 3, N) and y is (N, 1)
    image = self.images[:, :, :, index]
    if(self.isTrain):
        num = random.randint(0,4)
        if(num == 1):
            transformed = transformHF(image=image)
            image = transformed['image']
        elif(num == 2):
            transformed = transformSharpen(image=image)
            image = transformed['image']

        elif(num == 3):
            transformed = transformRotate(image=image)
            image = transformed['image']

    if(self.labels[index][0] == 10):
        labels = 0
    else:
        labels = self.labels[index][0]

    if self.transform:
        image = self.transform(image)
        image = image.to(self.device)

    return image, labels
```

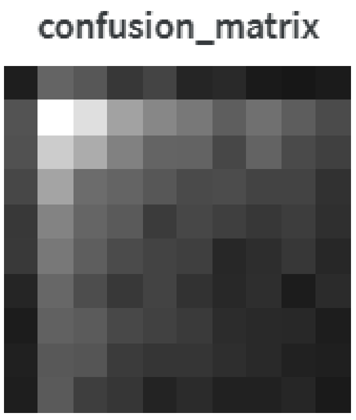
The loss by augmenting the data is as follows



The accuracy and the precision score on the test set are as follows


```
Accuracy on test set 96.43783267367272
F1 score on test set 0.798483210367632
```

The confusion matrix on the test set after augmentation is as follows



Confusion Matrix

Part 5

- The accuracy and F1 Score on custom model is 95.93% and 0.7268
- The accuracy and F1 Score on resnet18 is 95.96% and 0.7293 respectively
- The accuracy and F1 Score on Augmented Resnet 18 is 96.43% and 0.798 respectively

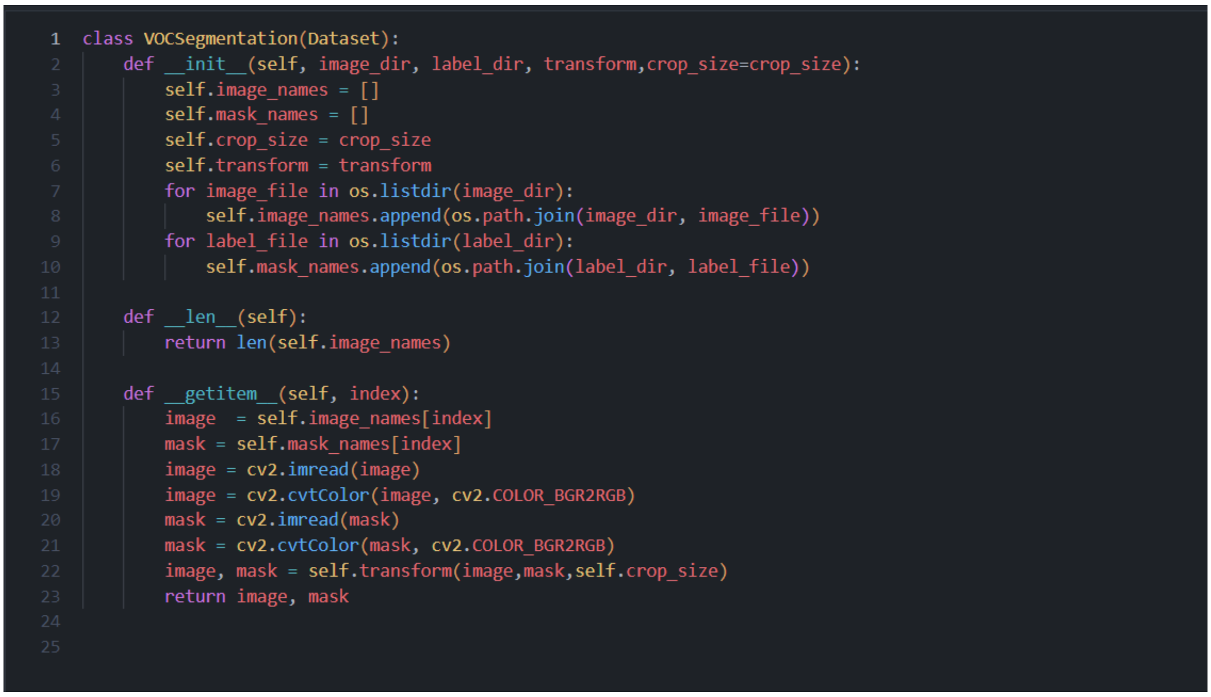
We can conclude that the Augmentation does help in increasing the accuracy and it performs the best out of all the three models. Also both the models perform very well in classifying the images as indicated by high accuracy and F1 Score.

Q2

Part 1

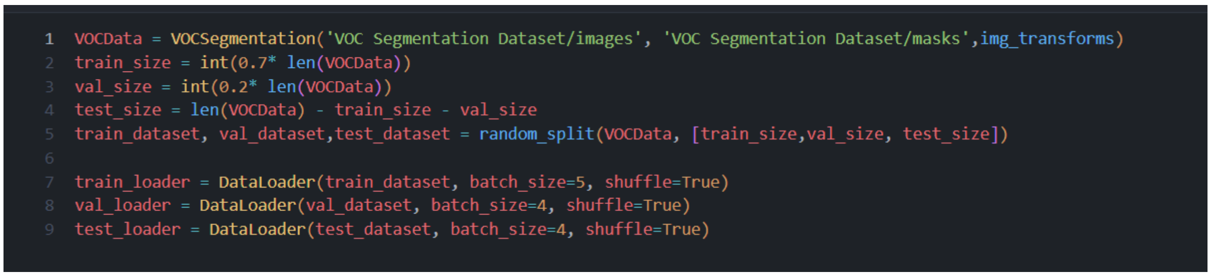
(A) and (B)

We create a custom data class for the dataset



```python
class VOCSegmentation(Dataset):
    def __init__(self, image_dir, label_dir, transform,crop_size=crop_size):
        self.image_names = []
        self.mask_names = []
        self.crop_size = crop_size
        self.transform = transform
        for image_file in os.listdir(image_dir):
            self.image_names.append(os.path.join(image_dir, image_file))
        for label_file in os.listdir(label_dir):
            self.mask_names.append(os.path.join(label_dir, label_file))

    def __len__(self):
        return len(self.image_names)

    def __getitem__(self, index):
        image  = self.image_names[index]
        mask = self.mask_names[index]
        image = cv2.imread(image)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        mask = cv2.imread(mask)
        mask = cv2.cvtColor(mask, cv2.COLOR_BGR2RGB)
        image, mask = self.transform(image,mask,self.crop_size)
        return image, mask
```

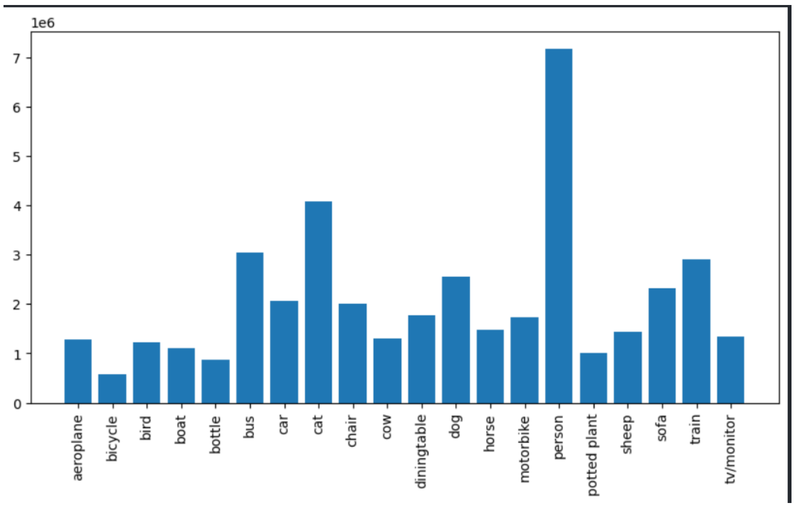
Now we do the train test and val split and create custom dataloaders for them.



```python
VOCData = VOCSegmentation('VOC Segmentation Dataset/images', 'VOC Segmentation Dataset/masks',img_transforms)
train_size = int(0.7* len(VOCData))
val_size = int(0.2* len(VOCData))
test_size = len(VOCData) - train_size - val_size
train_dataset, val_dataset,test_dataset = random_split(VOCData, [train_size,val_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=5, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=4, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=True)
```

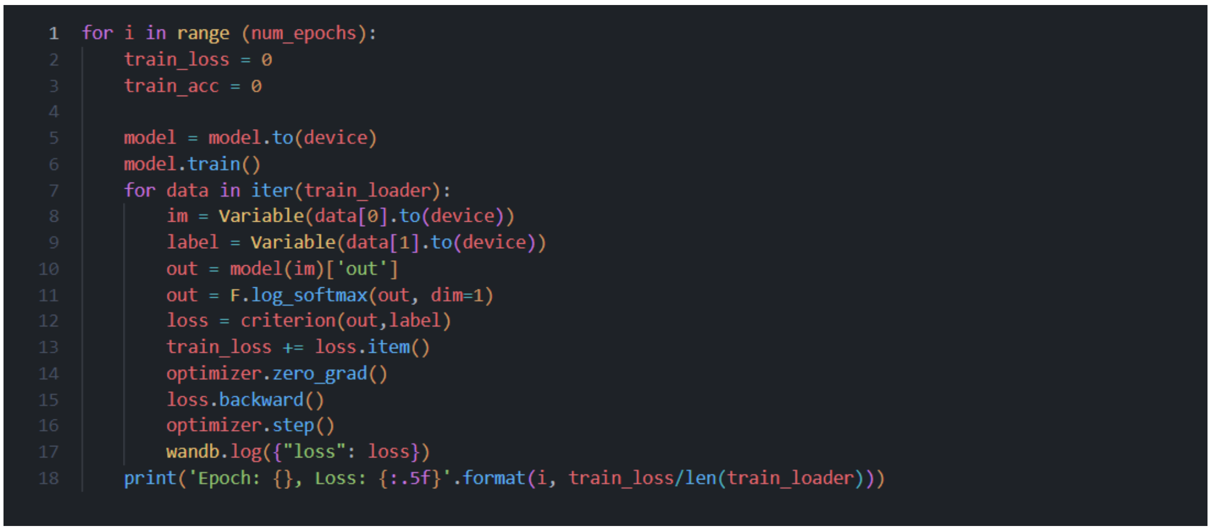
(C)

The pixelwise distribution of the classes are as follows. We observe that the person class is most widely present.



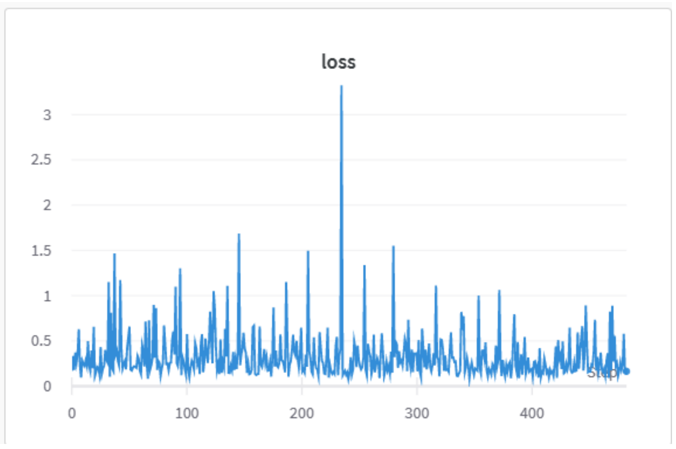
Part 2

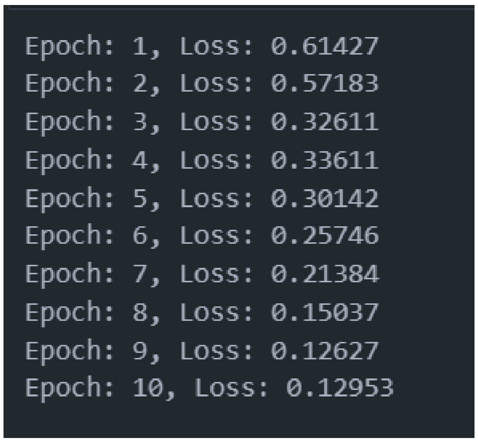
(A)



```
for i in range (num_epochs):
    train_loss = 0
    train_acc = 0

    model = model.to(device)
    model.train()
    for data in iter(train_loader):
        im = Variable(data[0].to(device))
        label = Variable(data[1].to(device))
        out = model(im)['out']
        out = F.log_softmax(out, dim=1)
        loss = criterion(out,label)
        train_loss += loss.item()
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        wandb.log({"loss": loss})
    print('Epoch: {}, Loss: {:.5f}'.format(i, train_loss/len(train_loader)))
```

The losses on training set are as follows

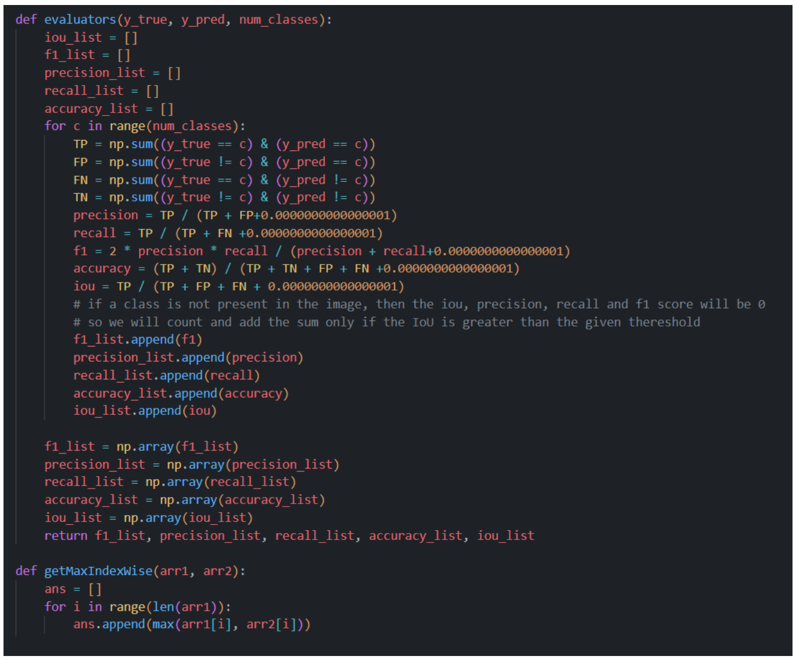




```
Epoch: 1, Loss: 0.61427
Epoch: 2, Loss: 0.57183
Epoch: 3, Loss: 0.32611
Epoch: 4, Loss: 0.33611
Epoch: 5, Loss: 0.30142
Epoch: 6, Loss: 0.25746
Epoch: 7, Loss: 0.21384
Epoch: 8, Loss: 0.15037
Epoch: 9, Loss: 0.12627
Epoch: 10, Loss: 0.12953
```

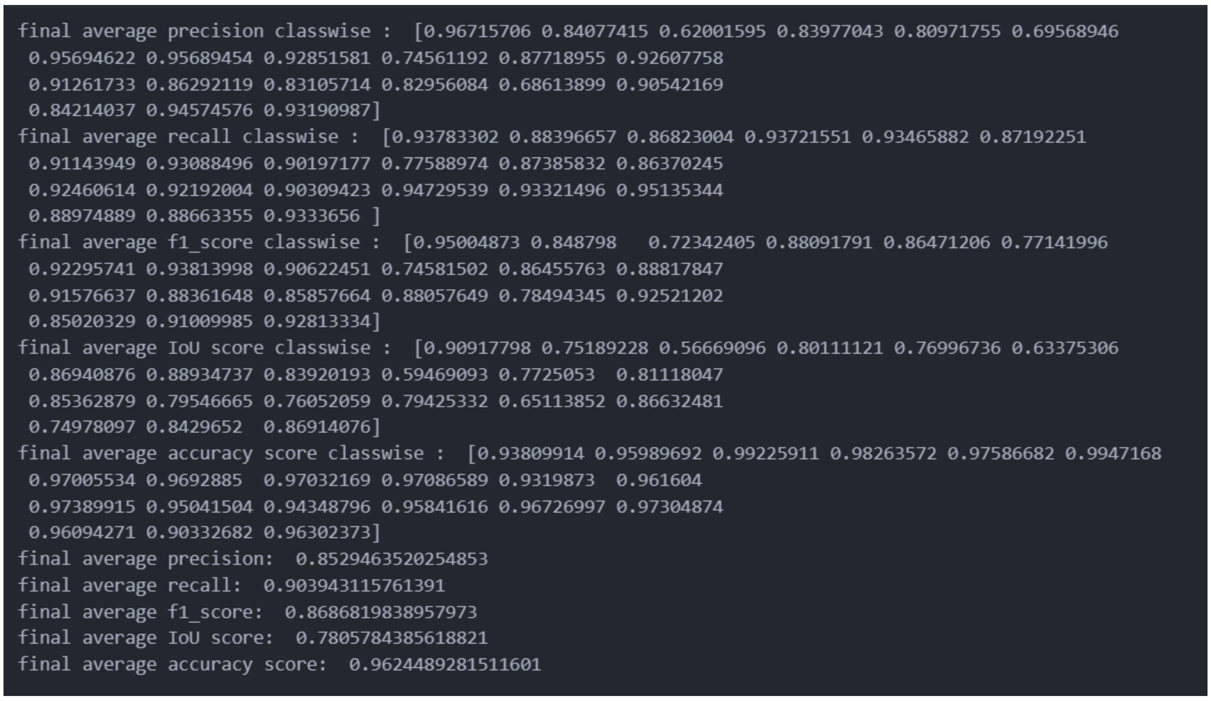
(B)

The code to get the pixel wise accuracy, IoU, F1 Score and Precision is as follows



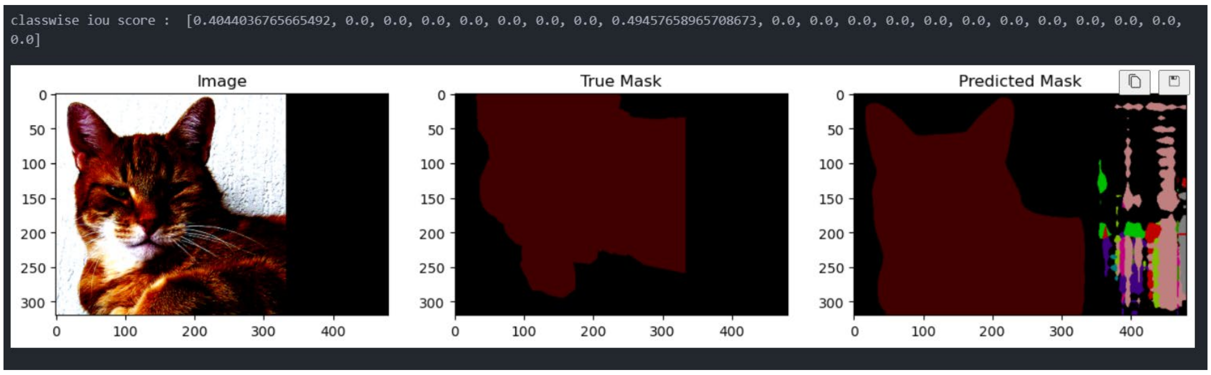
```python
def evaluators(y_true, y_pred, num_classes):
    iou_list = []
    f1_list = []
    precision_list = []
    recall_list = []
    accuracy_list = []
    for c in range(num_classes):
        TP = np.sum((y_true == c) & (y_pred == c))
        FP = np.sum((y_true != c) & (y_pred == c))
        FN = np.sum((y_true == c) & (y_pred != c))
        TN = np.sum((y_true != c) & (y_pred != c))
        precision = TP / (TP + FP+0.0000000000000001)
        recall = TP / (TP + FN +0.0000000000000001)
        f1 = 2 * precision * recall / (precision + recall+0.0000000000000001)
        accuracy = (TP + TN) / (TP + TN + FP + FN +0.0000000000000001)
        iou = TP / (TP + FP + FN + 0.0000000000000001)
        # if a class is not present in the image, then the iou, precision, recall and f1 score will be 0
        # so we will count and add the sum only if the IoU is greater than the given thereshold
        f1_list.append(f1)
        precision_list.append(precision)
        recall_list.append(recall)
        accuracy_list.append(accuracy)
        iou_list.append(iou)

    f1_list = np.array(f1_list)
    precision_list = np.array(precision_list)
    recall_list = np.array(recall_list)
    accuracy_list = np.array(accuracy_list)
    iou_list = np.array(iou_list)
    return f1_list, precision_list, recall_list, accuracy_list, iou_list

def getMaxIndexWise(arr1, arr2):
    ans = []
    for i in range(len(arr1)):
        ans.append(max(arr1[i], arr2[i]))
```

The final performance metrics are as follows.

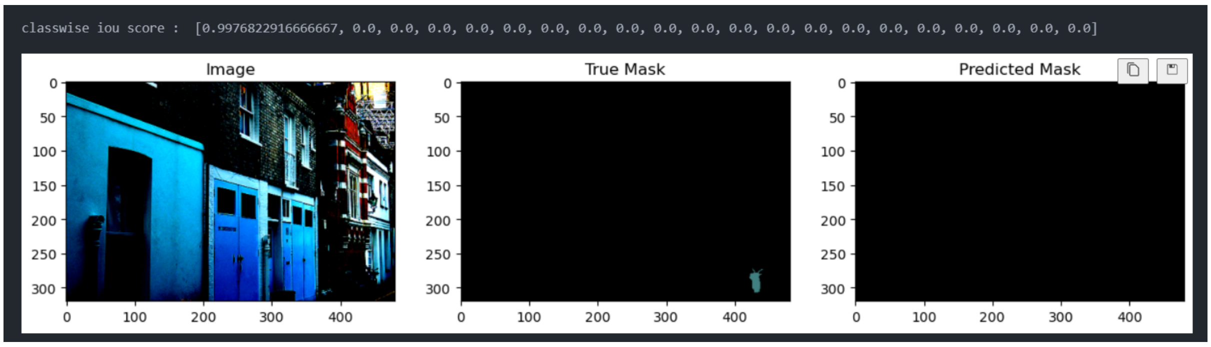


```
final average precision classwise :  [0.96715706 0.84077415 0.62001595 0.83977043 0.80971755 0.69568946
 0.95694622 0.95689454 0.92851581 0.74561192 0.87718955 0.92607758
 0.91261733 0.86292119 0.83105714 0.82956084 0.68613899 0.90542169
 0.84214037 0.94574576 0.93190987]
final average recall classwise :  [0.93783302 0.88396657 0.86823004 0.93721551 0.93465882 0.87192251
 0.91143949 0.93088496 0.90197177 0.77588974 0.87385832 0.86370245
 0.92460614 0.92192004 0.90309423 0.94729539 0.93321496 0.95135344
 0.88974889 0.88663355 0.9333656 ]
final average f1_score classwise :  [0.95004873 0.848798   0.72342405 0.88091791 0.86471206 0.77141996
 0.92295741 0.93813998 0.90622451 0.74581502 0.86455763 0.88817847
 0.91576637 0.88361648 0.85857664 0.88057649 0.78494345 0.92521202
 0.85020329 0.91009985 0.92813334]
final average IoU score classwise :  [0.90917798 0.75189228 0.56669096 0.80111121 0.76996736 0.63375306
 0.86940876 0.88934737 0.83920193 0.59469093 0.7725053  0.81118047
 0.85362879 0.79546665 0.76052059 0.79425332 0.65113852 0.86632481
 0.74978097 0.8429652  0.86914076]
final average accuracy score classwise :  [0.93809914 0.95989692 0.99225911 0.98263572 0.97586682 0.9947168
 0.97005534 0.9692885  0.97032169 0.97086589 0.9319873  0.961604
 0.97389915 0.95041504 0.94348796 0.95841616 0.96726997 0.97304874
 0.96094271 0.90332682 0.96302373]
final average precision:  0.8529463520254853
final average recall:  0.903943115761391
final average f1_score:  0.8686819838957973
final average IoU score:  0.7805784385618821
final average accuracy score:  0.9624489281511601
```

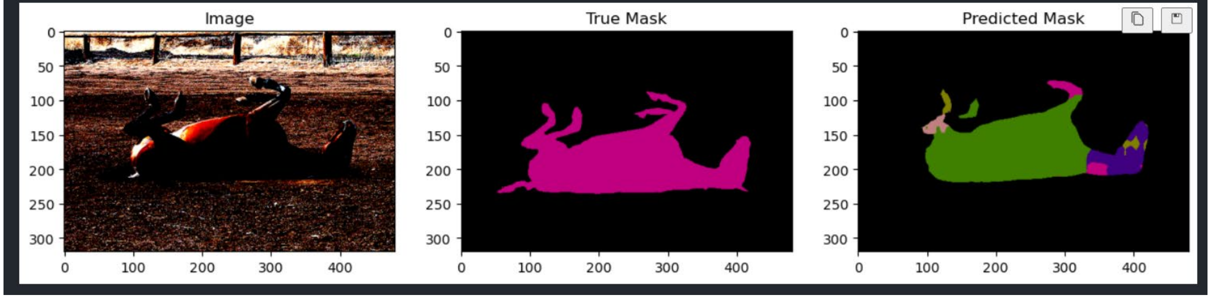
(C)



```
classwise iou score :  [0.4044036765665492, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.49457658965708673, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0]
```



```
classwise iou score :  [0.9976822916666667, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
classwise iou score :  [0.8857911812417048, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.015289838937133527, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0]
```



Here are the three images having the IoU of each class < 0.5 except for the background class.

In the second image the blue gates have been missed completely. It might be because of the surrounding which appears mostly is black. So the actual classes are missed due to class imbalance in the surrounding.

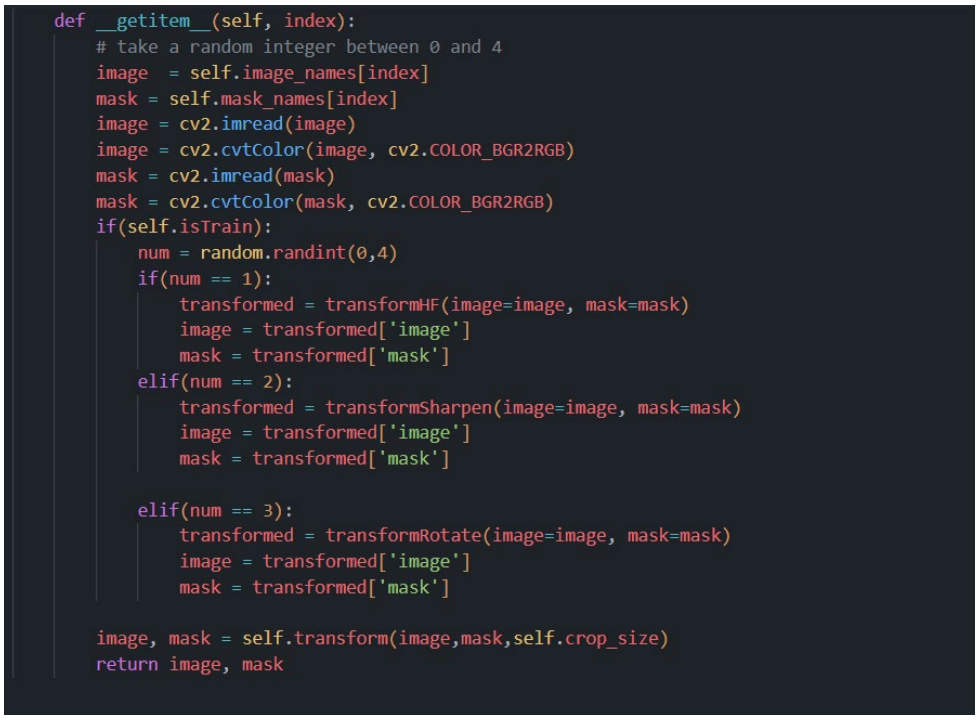
In the third image the the the object has been misclassified completely. It might be because the region of the horse is mostly dark and so it is difficult to segment the image properly and pink region is classified as green.

Part 3

(A)

The following code does the Image augmentation randomly on the training data

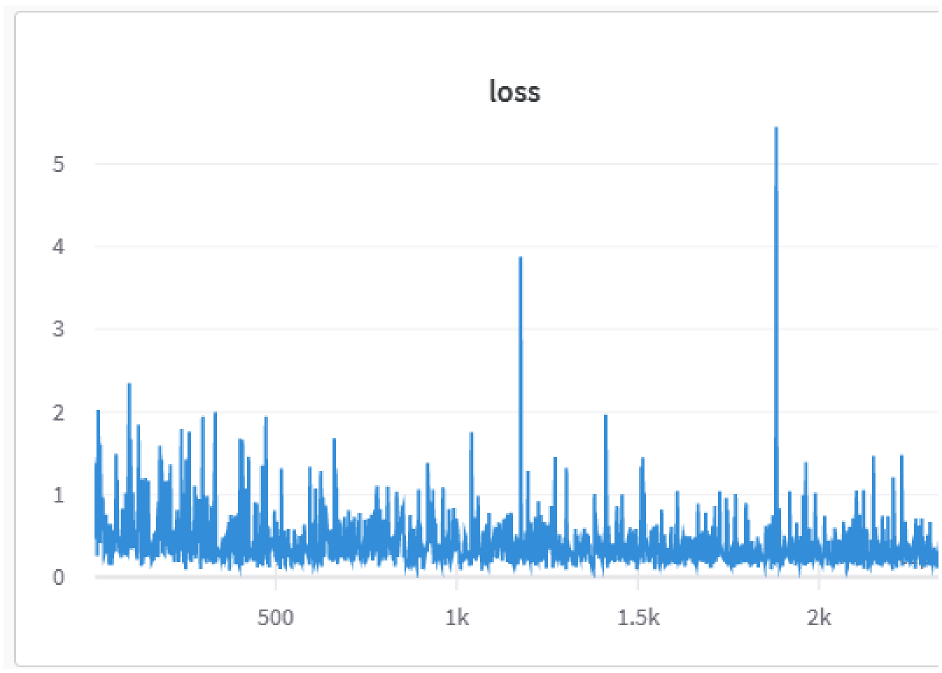
- We do horizontal flip, sharpen and rotation randomly if any.

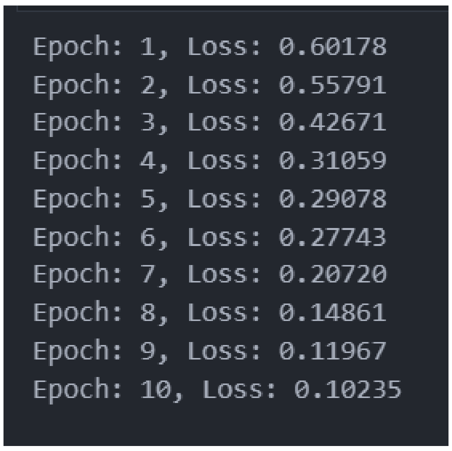


```
def __getitem__(self, index):
    # take a random integer between 0 and 4
    image  = self.image_names[index]
    mask = self.mask_names[index]
    image = cv2.imread(image)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    mask = cv2.imread(mask)
    mask = cv2.cvtColor(mask, cv2.COLOR_BGR2RGB)
    if(self.isTrain):
        num = random.randint(0,4)
        if(num == 1):
            transformed = transformHF(image=image, mask=mask)
            image = transformed['image']
            mask = transformed['mask']
        elif(num == 2):
            transformed = transformSharpen(image=image, mask=mask)
            image = transformed['image']
            mask = transformed['mask']

        elif(num == 3):
            transformed = transformRotate(image=image, mask=mask)
            image = transformed['image']
            mask = transformed['mask']

    image, mask = self.transform(image,mask,self.crop_size)
    return image, mask
```

(B)

The loss on the augmented model is as follows.

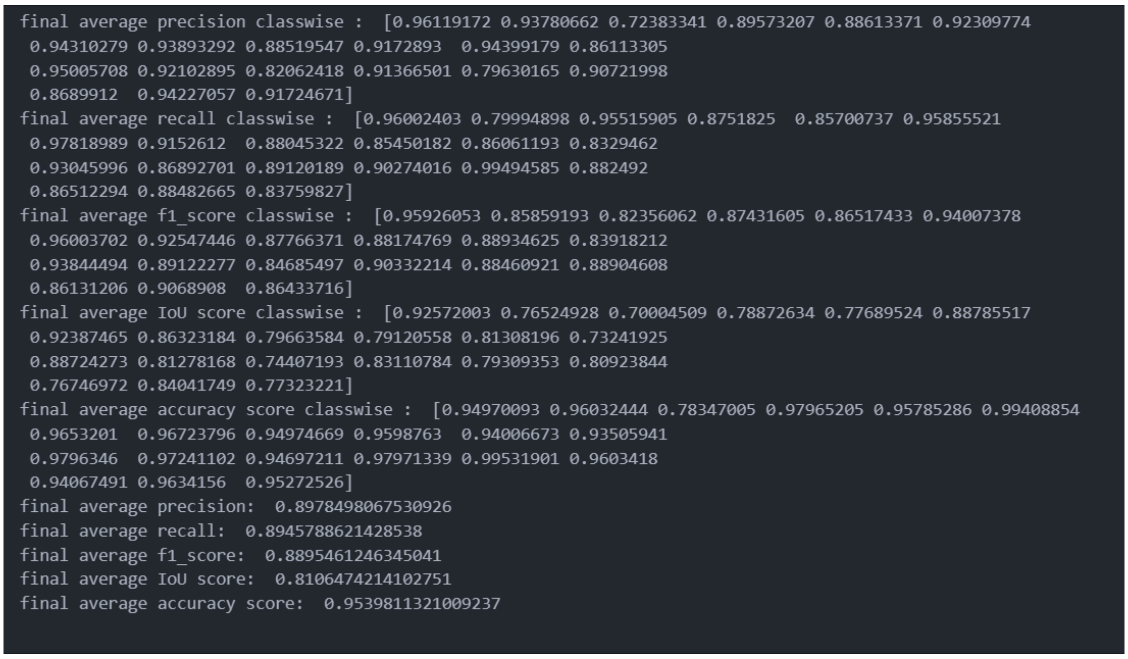




```
Epoch: 1, Loss: 0.60178
Epoch: 2, Loss: 0.55791
Epoch: 3, Loss: 0.42671
Epoch: 4, Loss: 0.31059
Epoch: 5, Loss: 0.29078
Epoch: 6, Loss: 0.27743
Epoch: 7, Loss: 0.20720
Epoch: 8, Loss: 0.14861
Epoch: 9, Loss: 0.11967
Epoch: 10, Loss: 0.10235
```

(C)

The performance metrics classwise on the augmented data is as follows.



```
final average precision classwise :  [0.96119172 0.93780662 0.72383341 0.89573207 0.88613371 0.92309774
 0.94310279 0.93893292 0.88519547 0.9172893  0.94399179 0.86113305
 0.95005708 0.92102895 0.82062418 0.91366501 0.79630165 0.90721998
 0.8689912  0.94227057 0.91724671]
final average recall classwise :  [0.96002403 0.79994898 0.95515905 0.8751825  0.85700737 0.95855521
 0.97818989 0.9152612  0.88045322 0.85450182 0.86061193 0.8329462
 0.93045996 0.86892701 0.89120189 0.90274016 0.99494585 0.882492
 0.86512294 0.88482665 0.83759827]
final average f1_score classwise :  [0.95926053 0.85859193 0.82356062 0.87431605 0.86517433 0.94007378
 0.96003702 0.92547446 0.87766371 0.88174769 0.88934625 0.83918212
 0.93844494 0.89122277 0.84685497 0.90332214 0.88460921 0.88904608
 0.86131206 0.9068908  0.86433716]
final average IoU score classwise :  [0.92572003 0.76524928 0.70004509 0.78872634 0.77689524 0.88785517
 0.92387465 0.86323184 0.79663584 0.79120558 0.81308196 0.73241925
 0.88724273 0.81278168 0.74407193 0.83110784 0.79309353 0.80923844
 0.76746972 0.84041749 0.77323221]
final average accuracy score classwise :  [0.94970093 0.96032444 0.78347005 0.97965205 0.95785286 0.99408854
 0.9653201  0.96723796 0.94974669 0.9598763  0.94006673 0.93505941
 0.9796346  0.97241102 0.94697211 0.97971339 0.99531901 0.9603418
 0.94067491 0.9634156  0.95272526]
final average precision:  0.8978498067530926
final average recall:  0.8945788621428538
final average f1_score:  0.8895461246345041
final average IoU score:  0.8106474214102751
final average accuracy score:  0.9539811321009237
```

Part 4

- The accuracy and F1 Score on FCN_resnet50 is 96.24% and 0.8686 respectively
- The accuracy and F1 Score on Augmented FCN_resnet50 is 95.39% and 0.88 respectively

We can conclude that both the models perform very similar on the test data. The former had a slightly better accuracy while the latter had a slightly better F1 Score. From the accuracy and F1 score of both the models we can infer that they both work really well on image segmentation tasks.

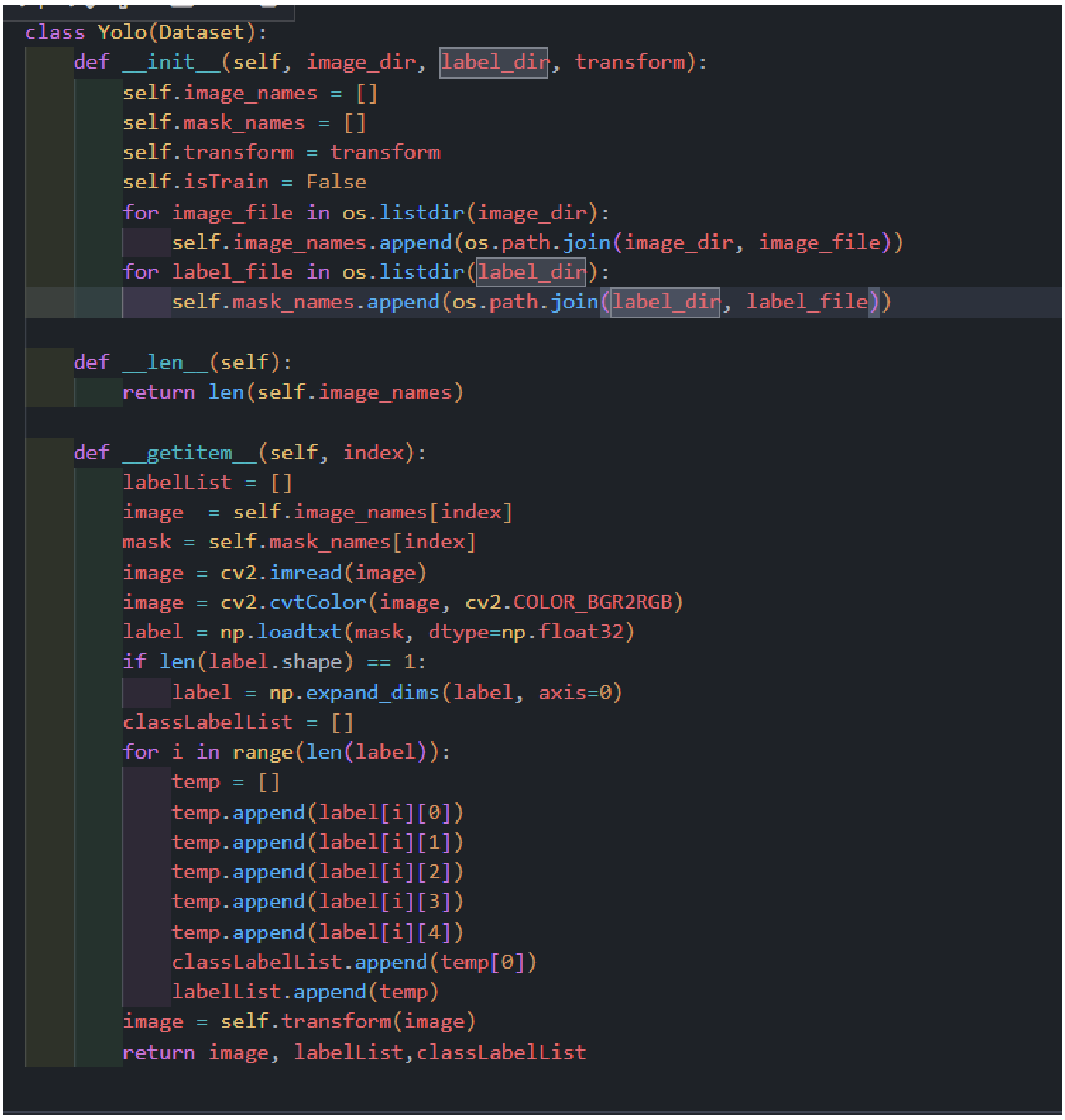
The mean IoU of both models is >0.8 which shows that the model's performance on each class is really good.

Q3

Part 1

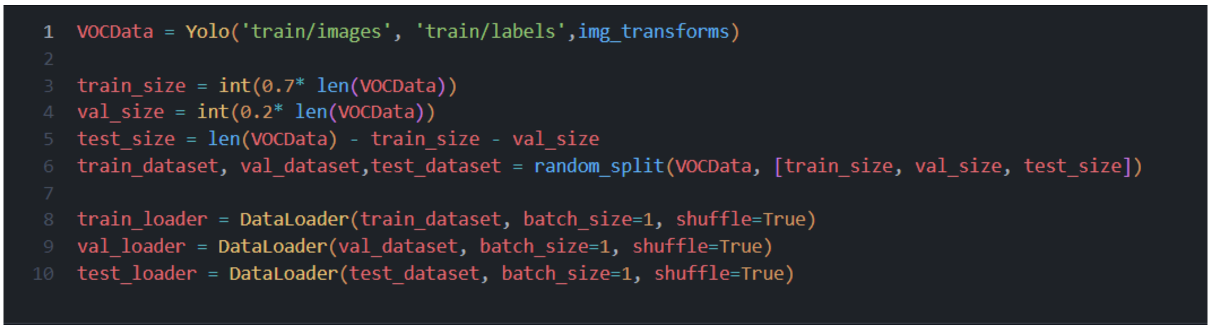
(A) and (B)

- Here we make a custom data class for loading the data



```
class Yolo(Dataset):
    def __init__(self, image_dir, label_dir, transform):
        self.image_names = []
        self.mask_names = []
        self.transform = transform
        self.isTrain = False
        for image_file in os.listdir(image_dir):
            self.image_names.append(os.path.join(image_dir, image_file))
        for label_file in os.listdir(label_dir):
            self.mask_names.append(os.path.join(label_dir, label_file))

    def __len__(self):
        return len(self.image_names)

    def __getitem__(self, index):
        labelList = []
        image  = self.image_names[index]
        mask = self.mask_names[index]
        image = cv2.imread(image)
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        label = np.loadtxt(mask, dtype=np.float32)
        if len(label.shape) == 1:
            label = np.expand_dims(label, axis=0)
        classLabelList = []
        for i in range(len(label)):
            temp = []
            temp.append(label[i][0])
            temp.append(label[i][1])
            temp.append(label[i][2])
            temp.append(label[i][3])
            temp.append(label[i][4])
            classLabelList.append(temp[0])
            labelList.append(temp)
        image = self.transform(image)
        return image, labelList,classLabelList
```

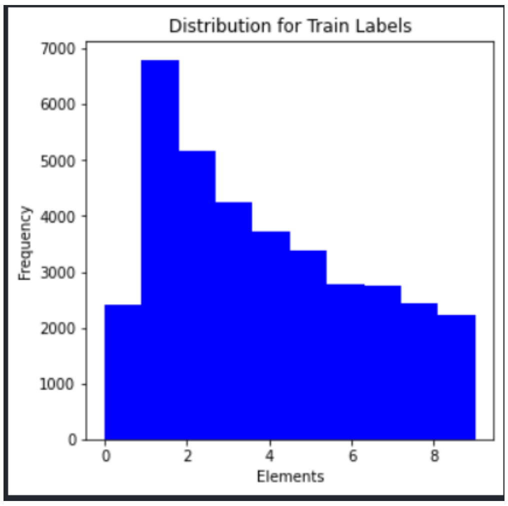
Now we do the train, test and validation split and make the custom dataloader for the data

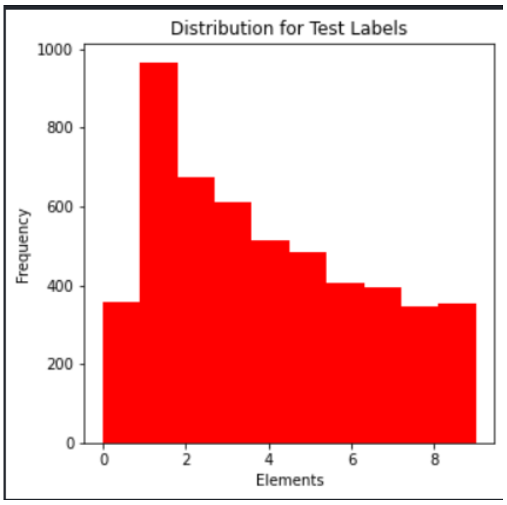


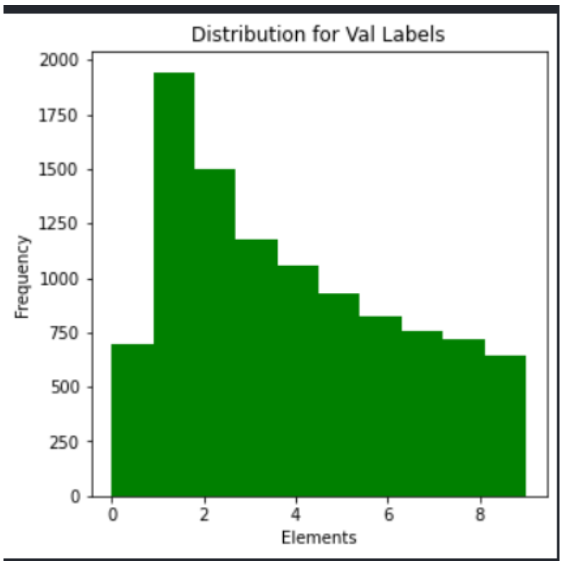
```
VOCData = Yolo('train/images', 'train/labels',img_transforms)

train_size = int(0.7* len(VOCData))
val_size = int(0.2* len(VOCData))
test_size = len(VOCData) - train_size - val_size
train_dataset, val_dataset,test_dataset = random_split(VOCData, [train_size, val_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=1, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=1, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=True)
```

(C)

The distribution of each class from 0-9 are as follows for train test and validation labels



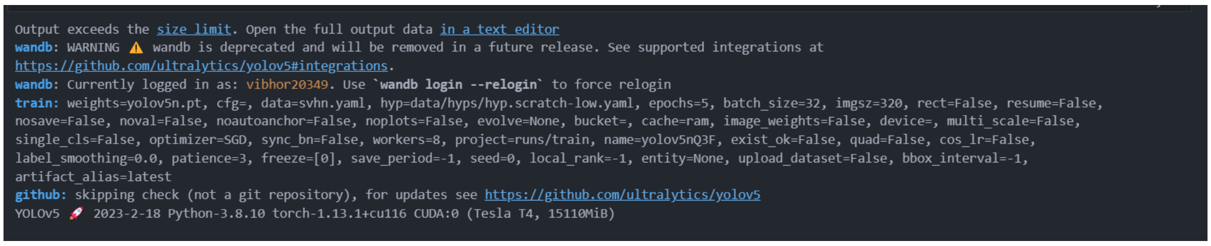




Part 2

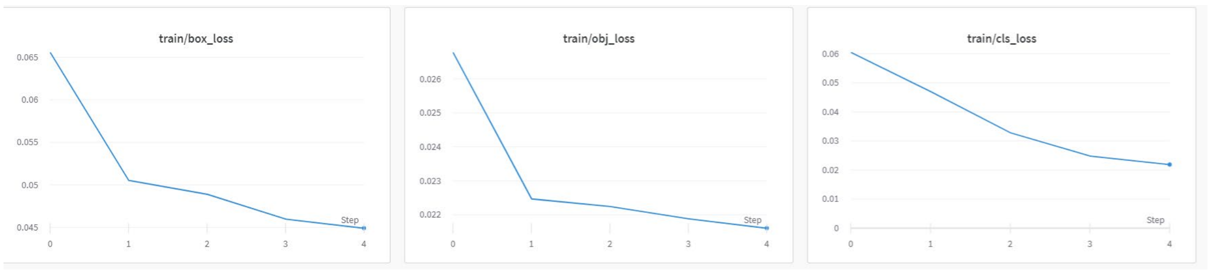
(A)

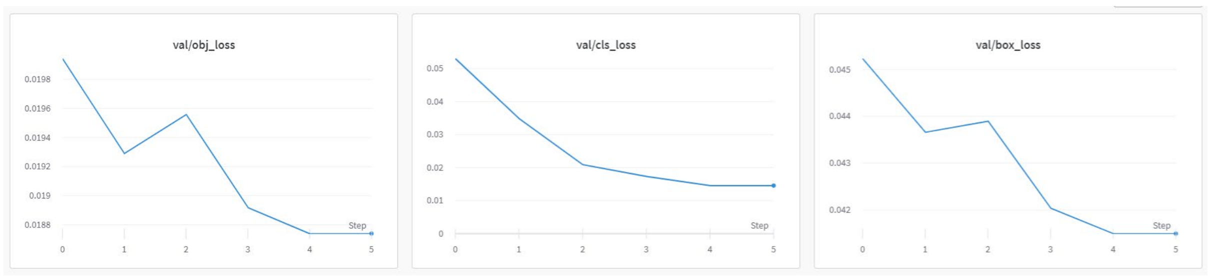
We import the model in the PyTorch and train the model using the following command. We use yolov5n.pt initially for the training.



```
Output exceeds the size limit. Open the full output data in a text editor
wandb: WARNING ⚠ wandb is deprecated and will be removed in a future release. See supported integrations at
https://github.com/ultralytics/yolov5#integrations.
wandb: Currently logged in as: vibhor20349. Use `wandb login --relogin` to force relogin
train: weights=yolov5n.pt, cfg=, data=svhn.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=5, batch_size=32, imgsz=320, rect=False, resume=False,
nosave=False, noval=False, noautoanchor=False, noplots=False, evolve=None, bucket=, cache=ram, image_weights=False, device=, multi_scale=False,
single_cls=False, optimizer=SGD, sync_bn=False, workers=8, project=runs/train, name=yolov5nQ3F, exist_ok=False, quad=False, cos_lr=False,
label_smoothing=0.0, patience=3, freeze=[0], save_period=-1, seed=0, local_rank=-1, entity=None, upload_dataset=False, bbox_interval=-1,
artifact_alias=latest
github: skipping check (not a git repository), for updates see https://github.com/ultralytics/yolov5
YOLOv5 🚀 2023-2-18 Python-3.8.10 torch-1.13.1+cu116 CUDA:0 (Tesla T4, 15110MiB)
```

The training and validation losses are as follows.

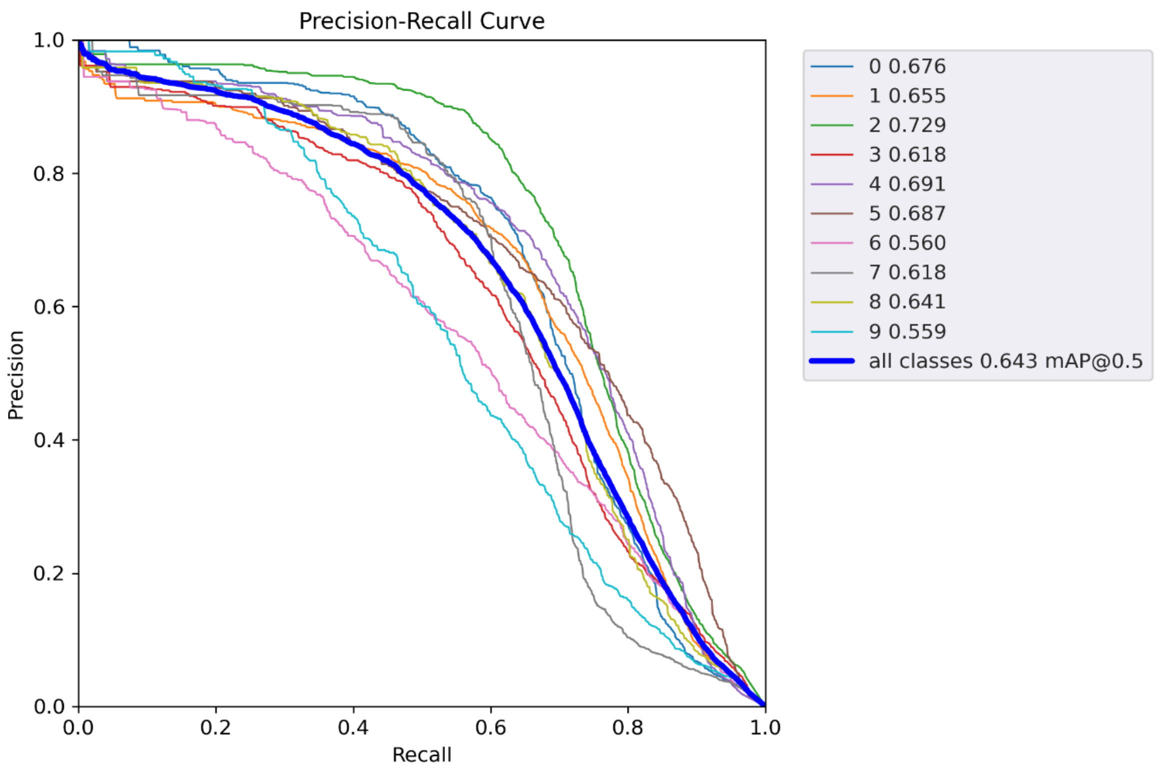




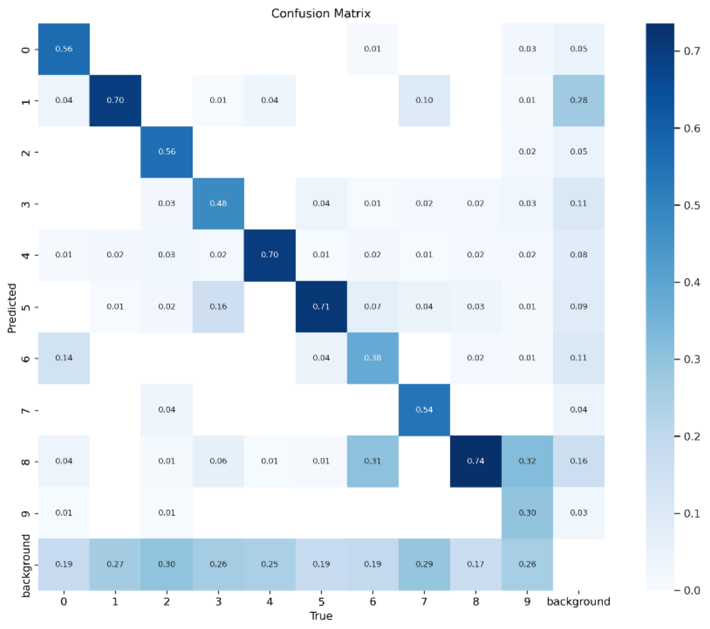
(B)

The performance metrics are as follows

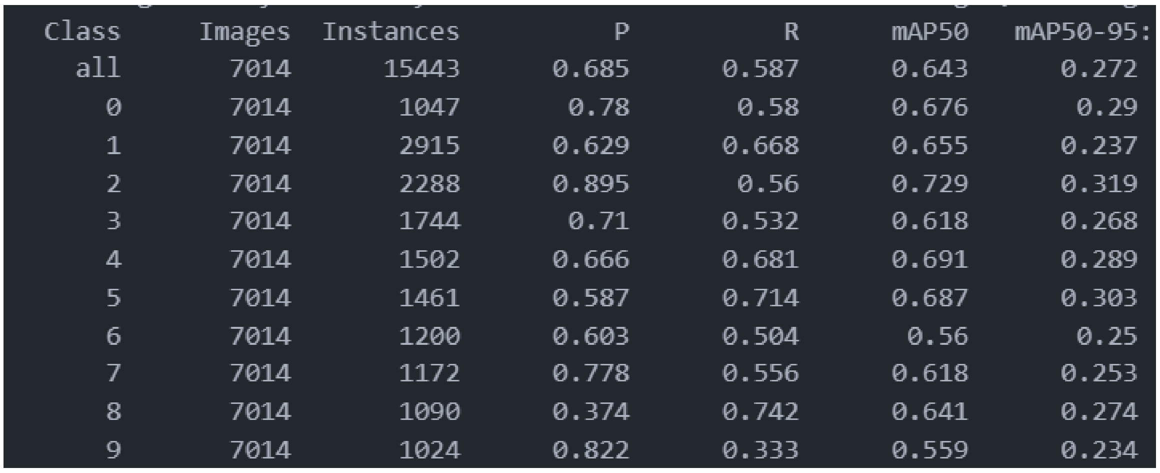
- Precision Recall Curve is as follows



- Confusion Matix is as follows

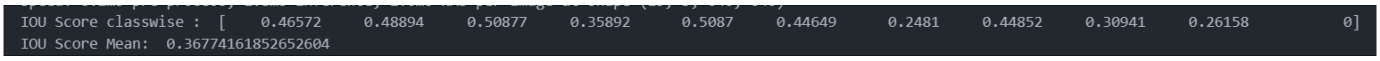


The classwise Average Precision, Recall and mAP on the test set are as follows.



```
 Class     Images  Instances          P          R      mAP50  mAP50-95:
   all       7014      15443      0.685      0.587      0.643      0.272
     0       7014       1047       0.78       0.58      0.676       0.29
     1       7014       2915      0.629      0.668      0.655      0.237
     2       7014       2288      0.895       0.56      0.729      0.319
     3       7014       1744       0.71      0.532      0.618      0.268
     4       7014       1502      0.666      0.681      0.691      0.289
     5       7014       1461      0.587      0.714      0.687      0.303
     6       7014       1200      0.603      0.504       0.56       0.25
     7       7014       1172      0.778      0.556      0.618      0.253
     8       7014       1090      0.374      0.742      0.641      0.274
     9       7014       1024      0.822      0.333      0.559      0.234
```

The classwise IoU and mean IoU scores are as follows.



```
IOU Score classwise :  [    0.46572     0.48894     0.50877     0.35892      0.5087     0.44649      0.2481     0.44852     0.30941     0.26158           0]
IOU Score Mean:  0.36774161852652604
```

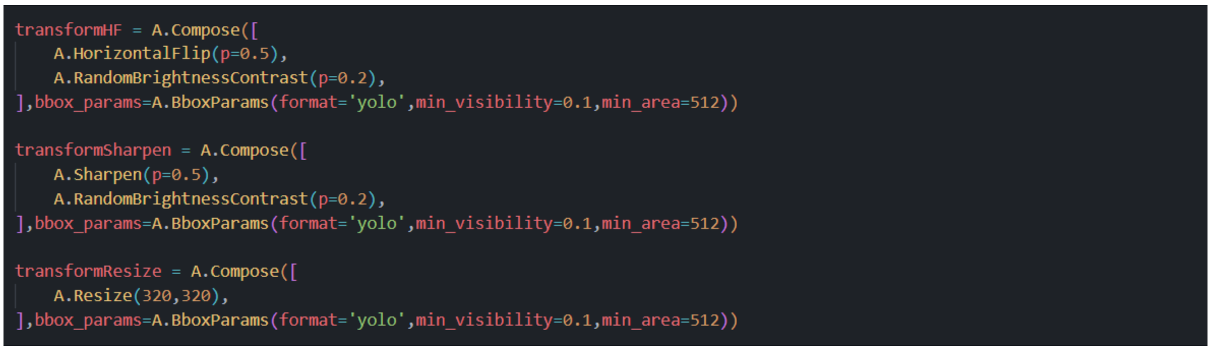
Part 3

(A)

Yolo do the image augmentations by default if the Albumentations library is installed on the system.

The following code does the Image augmentation randomly on the training data.

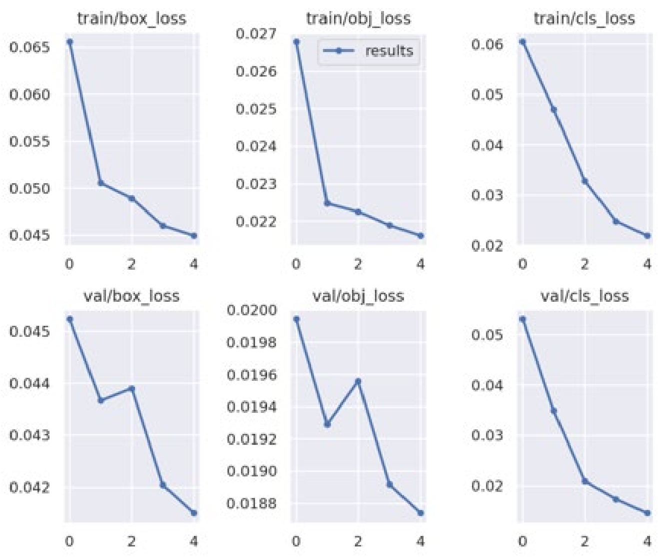
- We do horizontal flip, sharpen and rotation randomly if any.



```python
transformHF = A.Compose([
    A.HorizontalFlip(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
],bbox_params=A.BboxParams(format='yolo',min_visibility=0.1,min_area=512))

transformSharpen = A.Compose([
    A.Sharpen(p=0.5),
    A.RandomBrightnessContrast(p=0.2),
],bbox_params=A.BboxParams(format='yolo',min_visibility=0.1,min_area=512))

transformResize = A.Compose([
    A.Resize(320,320),
],bbox_params=A.BboxParams(format='yolo',min_visibility=0.1,min_area=512))
```

(B)

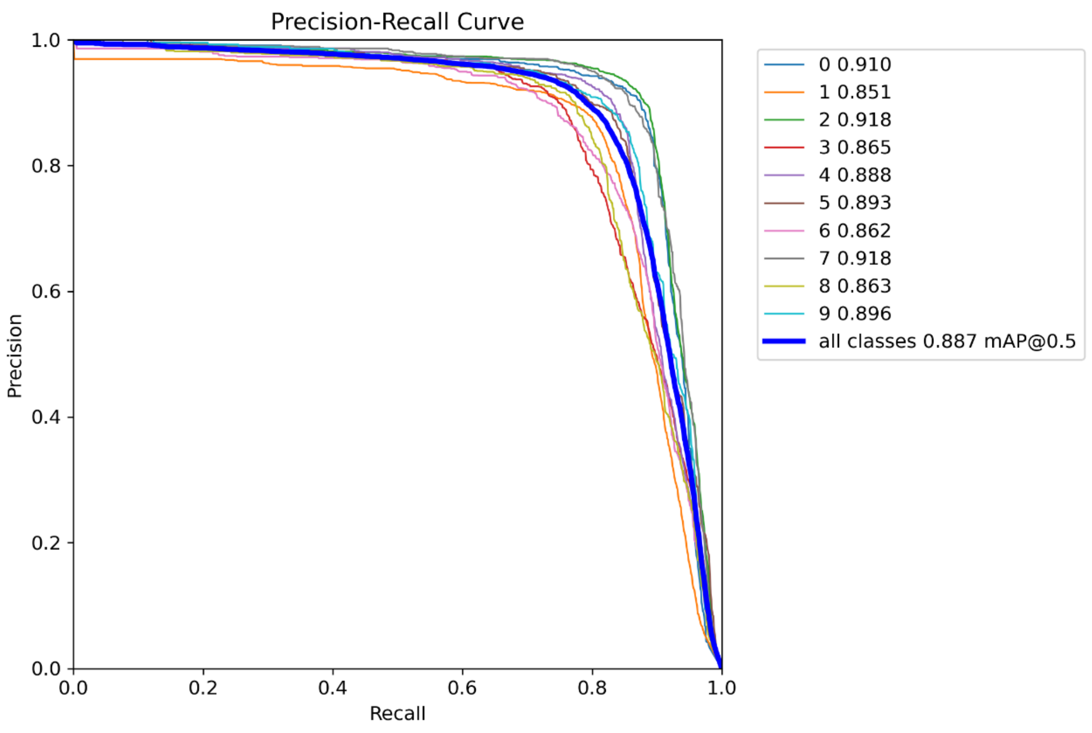
The training and validation losses are as follows applying the augmentation.



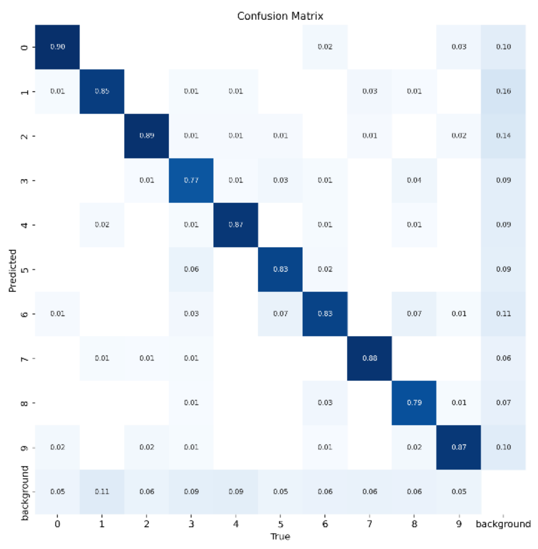
(C)

The performance metrics are as follows

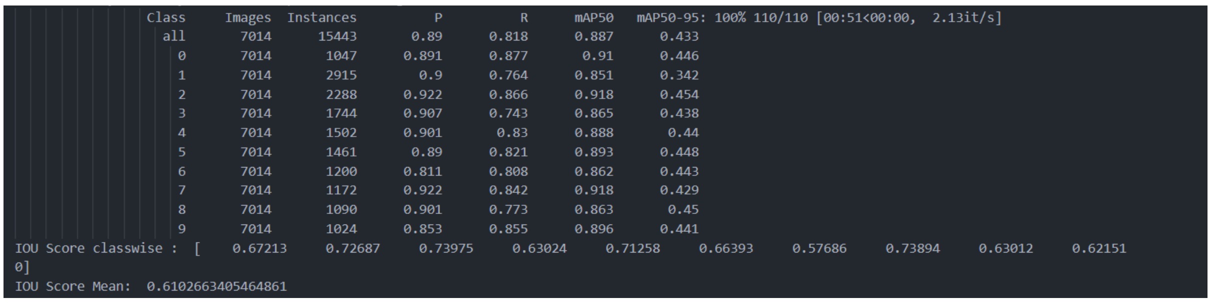
- Precision Recall Curve is as follows



The confusion matrix is as follows



The classwise Average Precision, Recall and mAP, classwise IoU and mean IoU on the test set are as follows.



```
                 Class     Images  Instances          P          R      mAP50   mAP50-95: 100% 110/110 [00:51<00:00,  2.13it/s]
                   all       7014      15443       0.89      0.818      0.887      0.433
                     0       7014       1047      0.891      0.877       0.91      0.446
                     1       7014       2915        0.9      0.764      0.851      0.342
                     2       7014       2288      0.922      0.866      0.918      0.454
                     3       7014       1744      0.907      0.743      0.865      0.438
                     4       7014       1502      0.901       0.83      0.888       0.44
                     5       7014       1461       0.89      0.821      0.893      0.448
                     6       7014       1200      0.811      0.808      0.862      0.443
                     7       7014       1172      0.922      0.842      0.918      0.429
                     8       7014       1090      0.901      0.773      0.863       0.45
                     9       7014       1024      0.853      0.855      0.896      0.441
IOU Score classwise :  [    0.67213     0.72687     0.73975     0.63024     0.71258     0.66393     0.57686     0.73894     0.63012     0.62151
0]
IOU Score Mean:  0.6102663405464861
```

Part 4

- The mAP50 of F1 Score and mean IoU on YoloV5 without augmentation is 0.643 and 0.36 respectively.
- The mAP50 of F1 Score on YoloV5 with augmentation is 0.889 and 0.61 respectively.

We can conclude that the Augmentation does help in increasing the performance of the YoloV5 as shown above. The model also does perform well on the object detection tasks. However, the model can perform even better and the performance of YoloV5 shown here is bottlenecked by the training hardware.