Assignment_1___Q2___Solution

February 23, 2023

1 Khushdev Pandit

2 Roll no: 2020211

3 *Assignment Question-2*

[ ]:
```
import os
from PIL import Image
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
import torchvision.datasets as datasets
import torchvision.models as models
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
from torch.utils.data import TensorDataset
import torch.optim as optim
import matplotlib.pyplot as plt
import matplotlib as mpl
mpl.rcParams['figure.facecolor'] = 'white'
import torch.nn.functional as F
import cv2
import numpy as np
from torch.utils.data.sampler import SubsetRandomSampler
import pickle
from sklearn import manifold
import pandas as pd
import seaborn as sns
from sklearn.metrics import classification_report
from tqdm import tqdm
```

4 Q2 Part-1

4.1 1(a) *Download the training files. Use 20% of the training dataset for validation and 10% for testing. Initialize Weights & Biases (WandB).*

[ ]:
```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

[ ]:
```
!pip install wandb
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting wandb
  Downloading wandb-0.13.10-py3-none-any.whl (2.0 MB)
                           2.0/2.0 MB
26.9 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.8/dist-packages (from wandb) (4.5.0)
Collecting setproctitle
  Downloading setproctitle-1.3.2-cp38-cp38-
manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_6
4.whl (31 kB)
Collecting GitPython>=1.0.0
  Downloading GitPython-3.1.31-py3-none-any.whl (184 kB)
                          184.3/184.3 KB
25.2 MB/s eta 0:00:00
Requirement already satisfied: protobuf!=4.21.0,<5,>=3.12.0 in
/usr/local/lib/python3.8/dist-packages (from wandb) (3.19.6)
Collecting sentry-sdk>=1.0.0
  Downloading sentry_sdk-1.15.0-py2.py3-none-any.whl (181 kB)
                          181.3/181.3 KB
24.0 MB/s eta 0:00:00
Collecting pathtools
  Downloading pathtools-0.1.2.tar.gz (11 kB)
  Preparing metadata (setup.py) … done
Requirement already satisfied: setuptools in /usr/local/lib/python3.8/dist-
packages (from wandb) (57.4.0)
Requirement already satisfied: Click!=8.0.0,>=7.0 in
/usr/local/lib/python3.8/dist-packages (from wandb) (7.1.2)
Requirement already satisfied: appdirs>=1.4.3 in /usr/local/lib/python3.8/dist-
packages (from wandb) (1.4.4)
Requirement already satisfied: requests<3,>=2.0.0 in
/usr/local/lib/python3.8/dist-packages (from wandb) (2.25.1)
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.8/dist-
packages (from wandb) (5.4.8)
Collecting docker-pycreds>=0.4.0
  Downloading docker_pycreds-0.4.0-py2.py3-none-any.whl (9.0 kB)
```

```
Requirement already satisfied: PyYAML in /usr/local/lib/python3.8/dist-packages
(from wandb) (6.0)
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.8/dist-
packages (from docker-pycreds>=0.4.0->wandb) (1.15.0)
Collecting gitdb<5,>=4.0.1
  Downloading gitdb-4.0.10-py3-none-any.whl (62 kB)
                           62.7/62.7 KB
8.7 MB/s eta 0:00:00
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.8/dist-packages (from requests<3,>=2.0.0->wandb)
(2022.12.7)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-
packages (from requests<3,>=2.0.0->wandb) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in
/usr/local/lib/python3.8/dist-packages (from requests<3,>=2.0.0->wandb) (1.24.3)
Requirement already satisfied: chardet<5,>=3.0.2 in
/usr/local/lib/python3.8/dist-packages (from requests<3,>=2.0.0->wandb) (4.0.0)
Collecting urllib3<1.27,>=1.21.1
  Downloading urllib3-1.26.14-py2.py3-none-any.whl (140 kB)
                          140.6/140.6 KB
20.0 MB/s eta 0:00:00
Collecting smmap<6,>=3.0.1
  Downloading smmap-5.0.0-py3-none-any.whl (24 kB)
Building wheels for collected packages: pathtools
  Building wheel for pathtools (setup.py) … done
  Created wheel for pathtools: filename=pathtools-0.1.2-py3-none-any.whl
size=8806
sha256=7f10fec119c3a843e48d41cde216defc4ecc7792d5888efbe7ca5cd3332b49e0
  Stored in directory: /root/.cache/pip/wheels/4c/8e/7e/72fbc243e1aeecae64a96875
432e70d4e92f3d2d18123be004
Successfully built pathtools
Installing collected packages: pathtools, urllib3, smmap, setproctitle, docker-
pycreds, sentry-sdk, gitdb, GitPython, wandb
  Attempting uninstall: urllib3
    Found existing installation: urllib3 1.24.3
    Uninstalling urllib3-1.24.3:
      Successfully uninstalled urllib3-1.24.3
Successfully installed GitPython-3.1.31 docker-pycreds-0.4.0 gitdb-4.0.10
pathtools-0.1.2 sentry-sdk-1.15.0 setproctitle-1.3.2 smmap-5.0.0 urllib3-1.26.14
wandb-0.13.10
```

[ ]:
```python
import wandb
wandb.login()
```

```
<IPython.core.display.Javascript object>
```

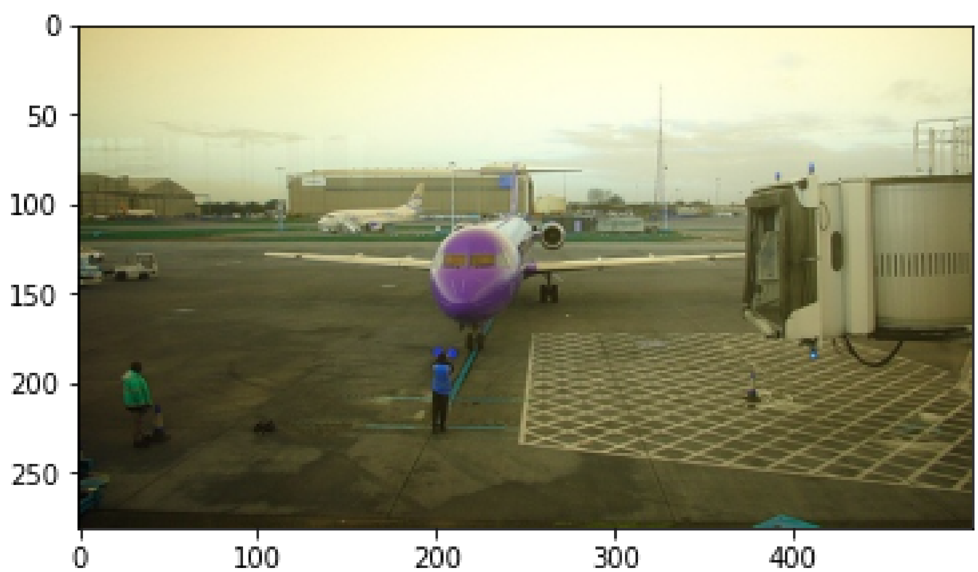
```
wandb: Appending key for api.wandb.ai to your netrc file:
/root/.netrc
```

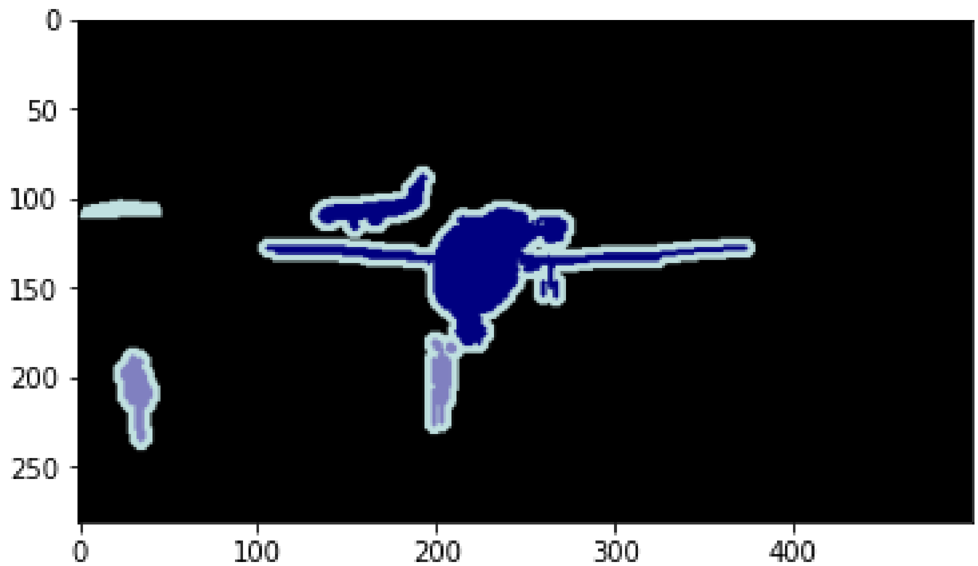
[ ]: True

[ ]:
```
path_data_img = "/content/drive/MyDrive/ECE344: CV (Computer Vision)/
 ↪Assignments/Assignment-1/Q2/VOC Segmentation Dataset/images"
path_data_mask = "/content/drive/MyDrive/ECE344: CV (Computer Vision)/
 ↪Assignments/Assignment-1/Q2/VOC Segmentation Dataset/masks"

img_path_list = os.listdir(path_data_img)
mask_path_list = os.listdir(path_data_mask)
img_path_list.sort()
mask_path_list.sort()
print("Number of images: ", len(img_path_list))
print("Number of masks: ", len(mask_path_list))
```

```
Number of images:  1464
Number of masks:  1464
```

[ ]:
```
img = cv2.imread(os.path.join(path_data_img, img_path_list[0])).astype(np.int64)
mask = cv2.imread(os.path.join(path_data_mask, mask_path_list[0])).astype(np.
 ↪int64)
plt.imshow(img);
plt.show()
plt.imshow(mask);
plt.show()
```





[ ]:
```
print('Shape of Image and Mask')
print(img.shape)
print(mask.shape)
print(mask[150,220])
np.unique(mask, return_counts=True)
```

```
Shape of Image and Mask
(281, 500, 3)
(281, 500, 3)
[  0   0 128]
```

[ ]:
```
(array([  0, 128, 192, 224]), array([398103,   6466,   6221,  10710]))
```

[ ]:
```
# calculate the indices for the splits
total_images = len(img_path_list)
indices = np.arange(total_images)
val_split = int(0.2 * total_images)
test_split = int(0.1 * total_images)

# create the splits
train_images_index = indices[val_split+test_split:]
val_images_index = indices[:val_split]
test_images_index = indices[val_split:val_split+test_split]

print(len(train_images_index), len(val_images_index), len(test_images_index))
```

```
1026 292 146
```

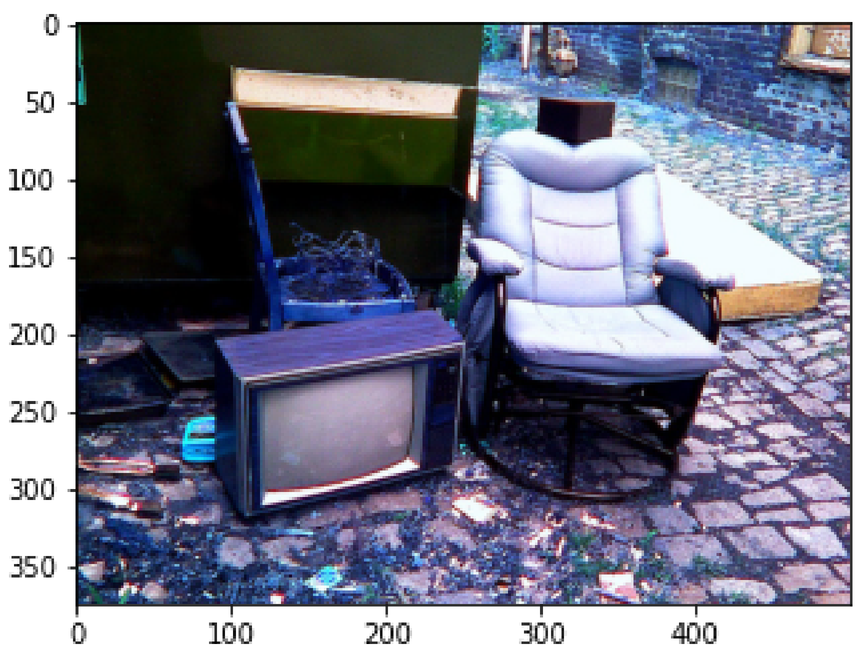
4.2 1(b) *Create dataloaders for all the splits (train, val and test) using PyTorch to load the images and their corresponding segmentation masks.*

[ ]:
```
class VOC2012Dataset(Dataset):
    def __init__(self, images, masks, img_indices, transform=None):
        self.img_indices = img_indices
        self.images = images
        self.masks = masks
        self.transform = transform

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

    def __getitem__(self, idx, dontApplyTransform=False):
        img_name = self.images[self.img_indices[idx]]
        mask_name = self.masks[self.img_indices[idx]]
        img = cv2.imread(os.path.join(path_data_img, img_name)).astype(np.int32)
        mask = cv2.imread(os.path.join(path_data_mask, mask_name)).astype(np.
 ↪int32)
        if self.transform and not dontApplyTransform:
            img = cv2.imread(os.path.join(path_data_img, img_name)).astype(np.
 ↪float64) / 255
            mask = cv2.imread(os.path.join(path_data_mask, mask_name)).
 ↪astype(np.float64) / 255
            img = self.transform(img)
            mask = self.transform(mask)
        return img, mask


# define the transforms
transform1 = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
    # transforms.Resize((356, 356))
])
transform2 = transforms.Compose([
    transforms.ToTensor()]
)

# Datset for training, validation and testing
train_dataset = VOC2012Dataset(img_path_list, mask_path_list,␣
 ↪train_images_index, transform=transform1)
val_dataset = VOC2012Dataset(img_path_list, mask_path_list, val_images_index,␣
 ↪transform=transform1)
test_dataset = VOC2012Dataset(img_path_list, mask_path_list, test_images_index,␣
 ↪transform=transform2)
```

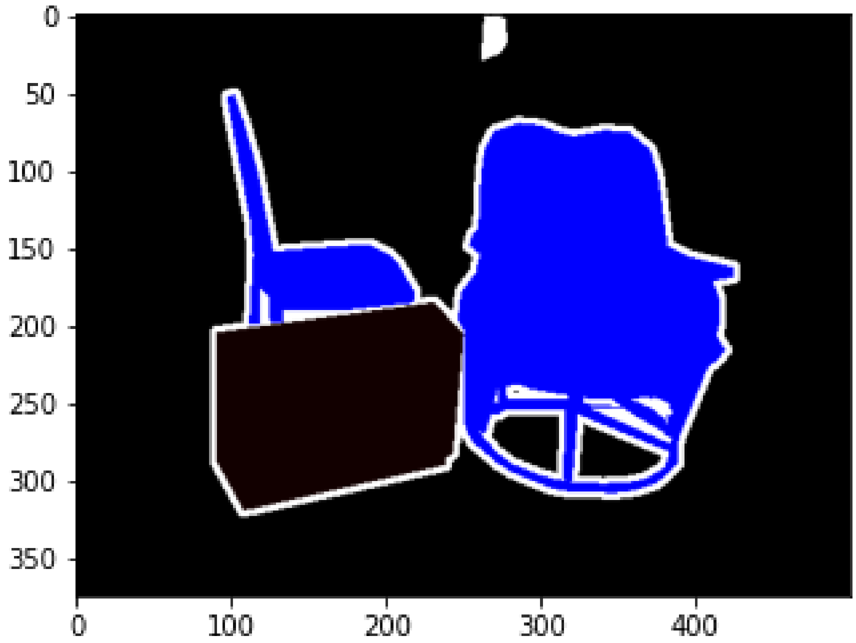
```
print('Dataset lengths:', len(train_dataset), len(val_dataset),␣
 ↪len(test_dataset))

# create the dataloaders
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)
print('DataLoader lengths:', len(train_loader), len(val_loader),␣
 ↪len(test_loader))
```

```
Dataset lengths: 1026 292 146
DataLoader lengths: 33 10 5
```

[ ]:
```
plt.imshow(train_dataset.__getitem__(1, dontApplyTransform=True)[0]); plt.show()
plt.imshow(train_dataset.__getitem__(1)[1].permute(1,2,0)); plt.show()

print('\nLabels/Masks size:')
print(train_dataset.__getitem__(1, dontApplyTransform=True)[1].shape)
print(train_dataset.__getitem__(1)[0].shape)
print(train_dataset.__getitem__(1)[0].permute(1,2,0).shape)
print(train_dataset.__getitem__(1)[1].shape)

print('\nCount of pixel values:')
print(np.unique(train_dataset.__getitem__(1, dontApplyTransform=True)[1],␣
 ↪return_counts=True))
print(np.unique(train_dataset.__getitem__(1)[1], return_counts=True))
```



```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
```




```
Labels/Masks size:
(375, 500, 3)
torch.Size([3, 375, 500])
torch.Size([375, 500, 3])
torch.Size([3, 375, 500])

Count of pixel values:
(array([  0,  64, 128, 192, 224], dtype=int32), array([469227,  16973,  16973,
41019,  18308]))
(array([-2.11790393, -2.03571429, -1.80444444, -0.91526611,  0.07406456,
        1.17004881,  1.54196078,  1.88585434,  2.09969499]), array([161373,
161373, 146481,  16973,  16973,   9154,  31865,   9154,
         9154]))
```

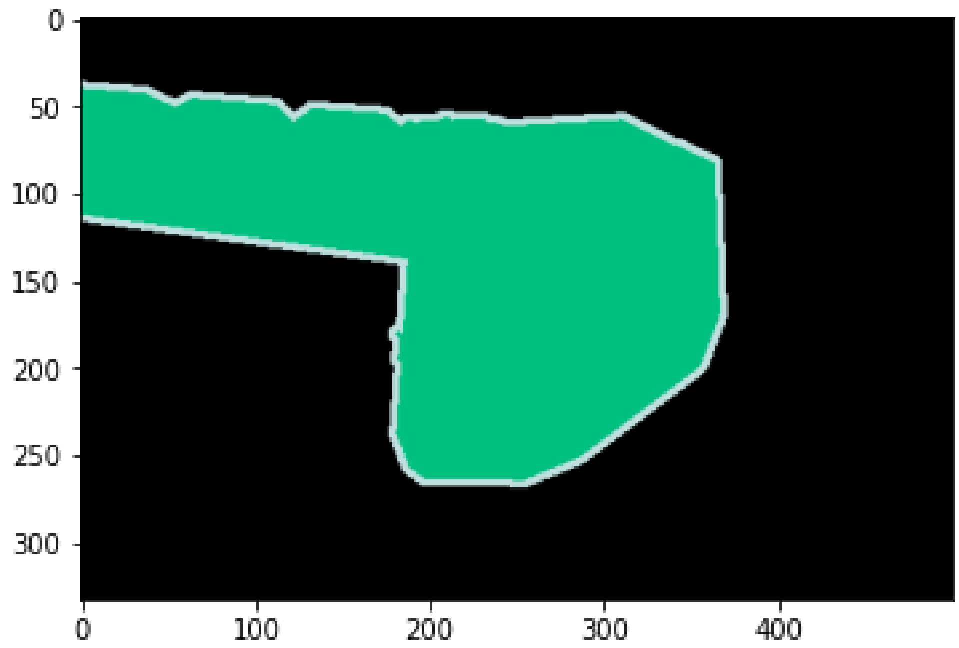
4.3 1(c) *Visualize the data distribution across class labels for training and validation sets.*

[ ]:
```
class_pixels = np.array([[0, 0, 0], [128, 0, 0], [0, 128, 0], [128, 128, 0],
                 [0, 0, 128], [128, 0, 128], [0, 128, 128], [128, 128, 128],
                 [64, 0, 0], [192, 0, 0], [64, 128, 0], [192, 128, 0],
                 [64, 0, 128], [192, 0, 128], [64, 128, 128], [192, 128, 128],
                 [0, 64, 0], [128, 64, 0], [0, 192, 0], [128, 192, 0],
```

```
                  [0, 64, 128]])
class_pixels = np.array([pixel[::-1] for pixel in class_pixels])

class_labels = ['background', 'aeroplane', 'bicycle', 'bird', 'boat', 'bottle',
                'bus', 'car', 'cat', 'chair', 'cow', 'diningtable', 'dog',
                'horse', 'motorbike', 'person', 'pottedplant', 'sheep', 'sofa',
                'train', 'tvmonitor']
```

[ ]:
```
img, mask = val_dataset.__getitem__(10, dontApplyTransform=True)
print(img.shape, mask.shape)
plt.imshow(img); plt.show()
plt.imshow(mask); plt.show()
```

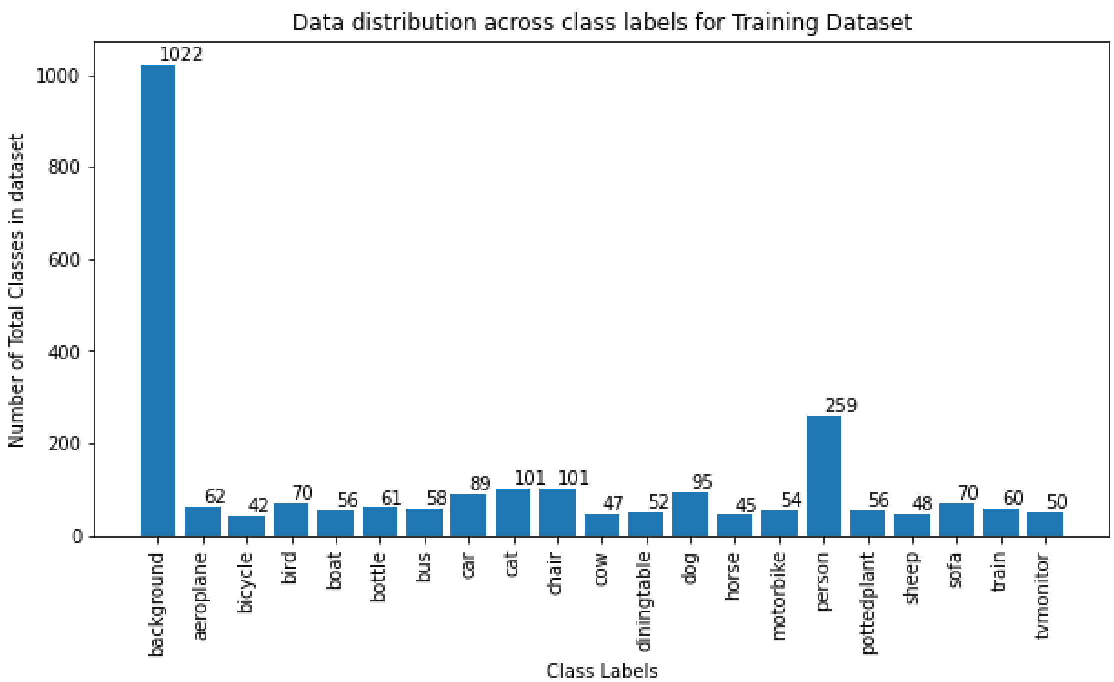
```
(333, 500, 3) (333, 500, 3)
```



[ ]:
```python
import seaborn as sns

# plot the class labels for Traininig and Validation dataset
def plot_class_labels(dataset:VOC2012Dataset, class_labels:list, class_pixels:
 ↪list, set_name:str):
    num_classes = len(class_labels)
    dataset_labels_count = np.zeros((num_classes,), dtype=np.int64)

    for i in range(len(dataset)):
        img, mask = dataset.__getitem__(i, dontApplyTransform=True)

        # Count the number of pixels for each class in the current image
        label_pixel_cnt_img = np.zeros((num_classes,), dtype=np.int64)
        for j in range(num_classes):
            label_pixel_cnt_img[j] += np.sum(np.all(mask==class_pixels[j],␣
 ↪axis=-1))

        # If the current image has pixels for a class, increment the count for␣
 ↪that class in the dataset
        for j in range(num_classes):
            dataset_labels_count[j] += 1 if label_pixel_cnt_img[j] > 0 else 0

    plt.figure(figsize=(10, 5))
    plt.bar(class_labels, dataset_labels_count)
    plt.xticks(rotation=90)
```

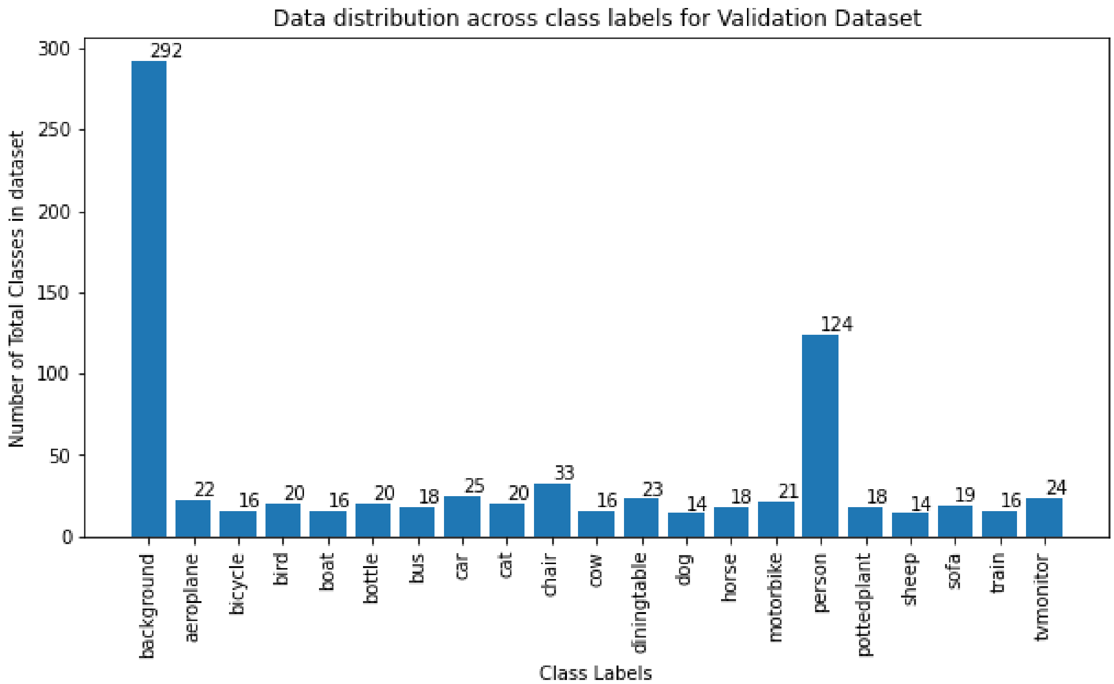
```
    for index, value in enumerate(dataset_labels_count):
        plt.text(index, value, str(value), va='bottom', )
    plt.xlabel('Class Labels')
    plt.ylabel('Number of Total Classes in dataset')
    plt.title(f'Data distribution across class labels for {set_name} Dataset')
    plt.show()

    return dataset_labels_count
```

[ ]:
```
train_dataset_labels_count = plot_class_labels(train_dataset, class_labels,␣
 ↪class_pixels, 'Training')
print(train_dataset_labels_count)
```




```
[1022   62   42   70   56   61   58   89  101  101   47   52   95   45
   54  259   56   48   70   60   50]
```

[ ]:
```
val_dataset_labels_count = plot_class_labels(val_dataset, class_labels,␣
 ↪class_pixels, 'Validation')
print(val_dataset_labels_count)
```




```
[292  22  16  20  16  20  18  25  20  33  16  23  14  18  21 124  18  14
  19  16  24]
```

5 Q2 Part-2 *Fine-tune a segmentation model*

5.1 2(a) *Train fcn resnet50 model using pre-defined network weights using an appropriate loss function. Use wandb for logging the loss and accuracy.*

[ ]:
```
wandb.init(entity="cv_assignment", project="Assignment-1",name="Q2-Part2")
wandb.config = {"learning_rate": 0.001, "epochs": 10, "batch_size": 16}
```

```
wandb: Currently logged in as: khushdev20211
(cv_assignment). Use `wandb login --relogin` to force relogin

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

[ ]:
```
transform  = transforms.Compose([
        transforms.ToTensor(),
```

```
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
 ↪225]),
        transforms.Resize((128, 128))
])

# Datset for training, validation and testing
train_dataset = VOC2012Dataset(img_path_list, mask_path_list,␣
 ↪train_images_index, transform=transform)
val_dataset = VOC2012Dataset(img_path_list, mask_path_list, val_images_index,␣
 ↪transform=transform)
test_dataset = VOC2012Dataset(img_path_list, mask_path_list, test_images_index,␣
 ↪transform=transform)
print('Dataset lengths:', len(train_dataset), len(val_dataset),␣
 ↪len(test_dataset))

# create the dataloaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=True)
print('DataLoader lengths:', len(train_loader), len(val_loader),␣
 ↪len(test_loader))
```

```
Dataset lengths: 1026 292 146
DataLoader lengths: 65 19 10
```

[ ]:
```
import torch.optim as optim
from torchvision.models.segmentation import FCN_ResNet50_Weights

# Define the FCN resnet50 model
model = models.segmentation.fcn_resnet50(weights=FCN_ResNet50_Weights.DEFAULT)

# Define the loss function (cross-entropy loss) and the optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-3)
```

```
Downloading:
"https://download.pytorch.org/models/fcn_resnet50_coco-1167a1af.pth" to
/root/.cache/torch/hub/checkpoints/fcn_resnet50_coco-1167a1af.pth

  0%|          | 0.00/135M [00:00<?, ?B/s]
```

[ ]:
```
epochs = 3
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = model.to(device)
print('Training on:', device)

for epoch in range(epochs):
    print('\nEpoch:', epoch)
```

```
    # =========================================== Training Phase =============================================
    model.train()
    train_loss = 0          # Training loss
    correct_pixels = 0      # Number of pixels correctly predicted
    total_pixels = 0        # Total number of pixels in the training set
    for image, mask in tqdm(train_loader):
        image = image.to(device)
        mask = mask.to(device)
        # mask = torch.mean(mask, dim=1, keepdim=True)
        image = image.type(torch.cuda.FloatTensor)
        mask = mask.type(torch.cuda.FloatTensor)
        optimizer.zero_grad()
        outputs = model(image)['out']
        loss = criterion(outputs, mask.argmax(dim=1))
        loss.backward()
        optimizer.step()

        # calculate training loss
        train_loss += loss.item()
        # calculate training accuracy
        predicted_mask = torch.argmax(outputs, dim=1)
        correct_pixels += torch.sum(predicted_mask == mask.argmax(dim=1)).item()
        total_pixels += torch.numel(predicted_mask)
        batch_accuracy = correct_pixels / total_pixels
    training_accuracy = 100 * correct_pixels / total_pixels

    # =========================================== Validation Phase ========================================
    model.eval()
    with torch.no_grad():
      # Validation loss
      valid_loss = 0          # Training loss
      correct_pixels = 0      # Number of pixels correctly predicted
      total_pixels = 0        # Total number of pixels in the validation set
      for image, mask in tqdm(val_loader):
          image = image.to(device)
          mask = mask.to(device)
          # mask = torch.mean(mask, dim=1, keepdim=True)
          image = image.type(torch.cuda.FloatTensor)
          mask = mask.type(torch.cuda.FloatTensor)
          outputs = model(image)['out']
          loss = criterion(outputs, mask.argmax(dim=1))

          # calculate training loss
          valid_loss += loss.item()
          # calculate training accuracy
```

```
            predicted_mask = torch.argmax(outputs, dim=1)
            correct_pixels += torch.sum(predicted_mask == mask.argmax(dim=1)).
↪item()
            total_pixels += torch.numel(predicted_mask)
            batch_accuracy = correct_pixels / total_pixels
        valid_accuracy = 100 * correct_pixels / total_pixels

    print("\nTraining Loss: {:.4f},  Training Accuracy: {:.4f}".
↪format(train_loss / len(train_loader), training_accuracy))
    print("Validation Loss: {:.4f},  Validation Accuracy: {:.4f}".
↪format(valid_loss / len(val_loader), valid_accuracy))
    # Log the loss and accuracy to W&B
    wandb.log({'training_loss': train_loss / len(train_loader),␣
↪'validation_loss': valid_loss / len(val_loader),
               'training_accuracy': training_accuracy, 'validation_accuracy':␣
↪valid_accuracy})

print('Finished Training')
```

```
Training on: cuda:0

Epoch: 0

100%|          | 65/65 [03:42<00:00,  3.43s/it]
100%|          | 19/19 [00:47<00:00,  2.51s/it]


Training Loss: 1.7867,  Training Accuracy: 56.6210
Validation Loss: 0.4708,  Validation Accuracy: 90.3016

Epoch: 1

100%|          | 65/65 [00:42<00:00,  1.52it/s]
100%|          | 19/19 [00:08<00:00,  2.28it/s]


Training Loss: 0.4131,  Training Accuracy: 88.5652
Validation Loss: 0.3489,  Validation Accuracy: 90.6994

Epoch: 2

100%|          | 65/65 [00:42<00:00,  1.53it/s]
100%|          | 19/19 [00:08<00:00,  2.15it/s]


Training Loss: 0.3913,  Training Accuracy: 88.6495
Validation Loss: 0.3251,  Validation Accuracy: 90.9837
Finished Training
```

[ ]:
```
model.eval()
with torch.no_grad():
    # Testing loss
    correct_pixels = 0    # Number of pixels correctly predicted
    total_pixels = 0      # Total number of pixels in the Testing set
    for i, (image, mask) in enumerate(test_loader, 0):
          image = image.to(device)
          mask = mask.to(device)
          image = image.type(torch.cuda.FloatTensor)
          mask = mask.type(torch.cuda.FloatTensor)
          outputs = model(image)['out']
          loss = criterion(outputs, mask.argmax(dim=1))
          # calculate training accuracy
          predicted_mask = torch.argmax(outputs, dim=1)
          correct_pixels += torch.sum(predicted_mask == mask.argmax(dim=1)).
 ↪item()
          total_pixels += torch.numel(predicted_mask)
          batch_accuracy = correct_pixels / total_pixels
    test_accuracy = 100 * correct_pixels / total_pixels
print("Testing Accuracy: {:.4f}".format(test_accuracy))
```

```
Testing Accuracy: 91.8881
```

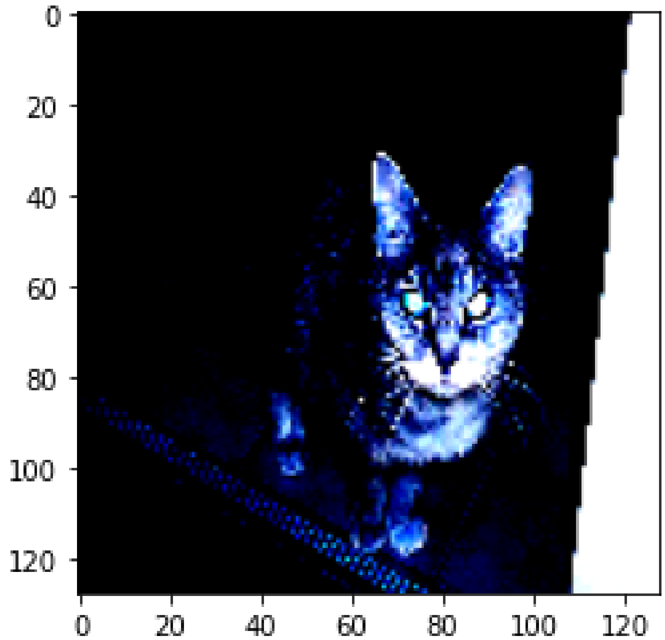
[ ]:
```
#   W&B: Save Model
torch.save(model, 'fcs_resnet50_q2_b.pt')
torch.save(model.state_dict(),"fcs_resnet50_q2_b.pth")
artifact = wandb.Artifact('model', type='model')
artifact.add_file('fcs_resnet50_q2_b.pt')
artifact.add_file('fcs_resnet50_q2_b.pth')
wandb.log_artifact(artifact)
wandb.finish()
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

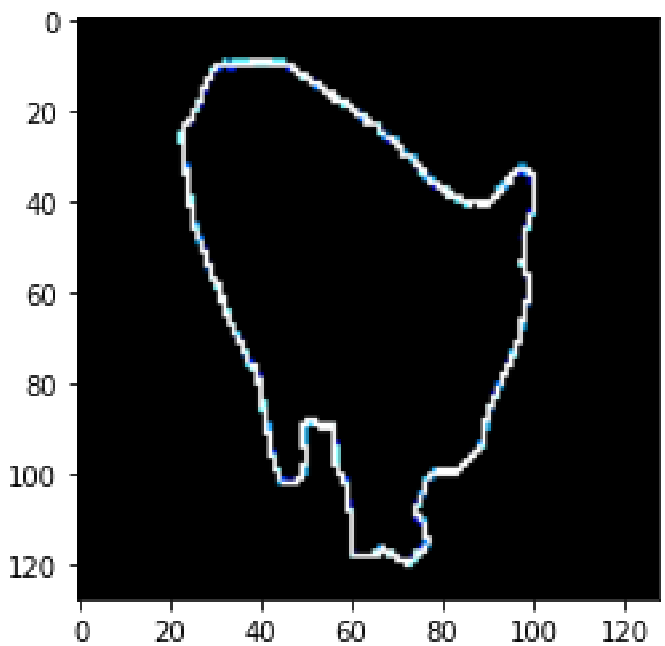
[ ]:
```
import pickle
with open('fcs_resnet50_model.pickle', 'wb') as f:
  pickle.dump(model, f)

with open('fcs_resnet50_model.pickle', 'rb') as pickle_in:
    fcs_resnet50 = pickle.load(pickle_in)
```

[ ]:
```
img, mask = train_loader.dataset.__getitem__(0)
plt.imshow(img.permute(1,2,0)); plt.show()
plt.imshow(mask.permute(1,2,0)); plt.show()
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
```



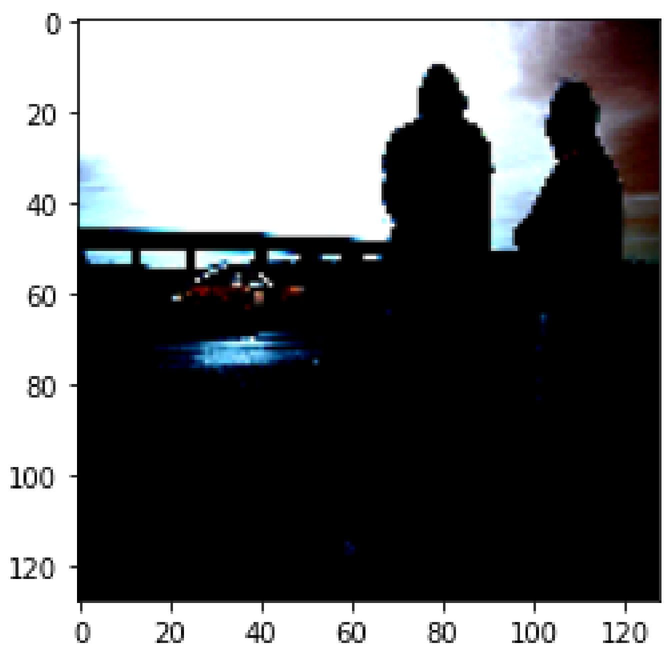

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
```

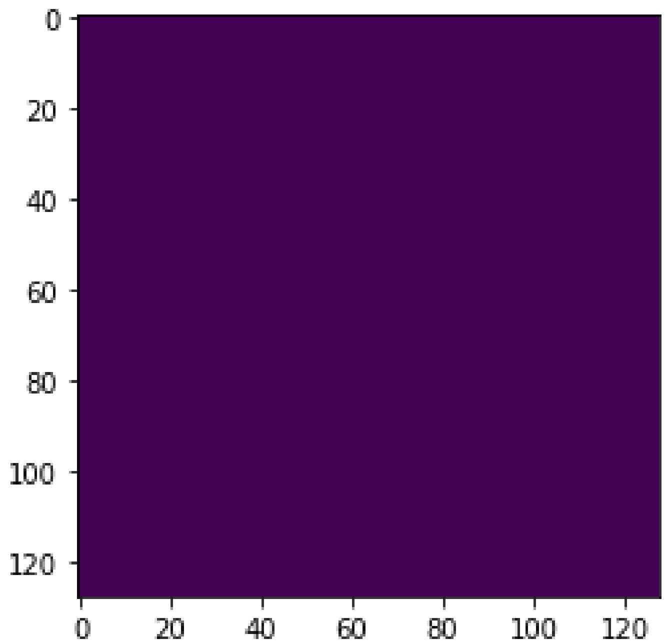


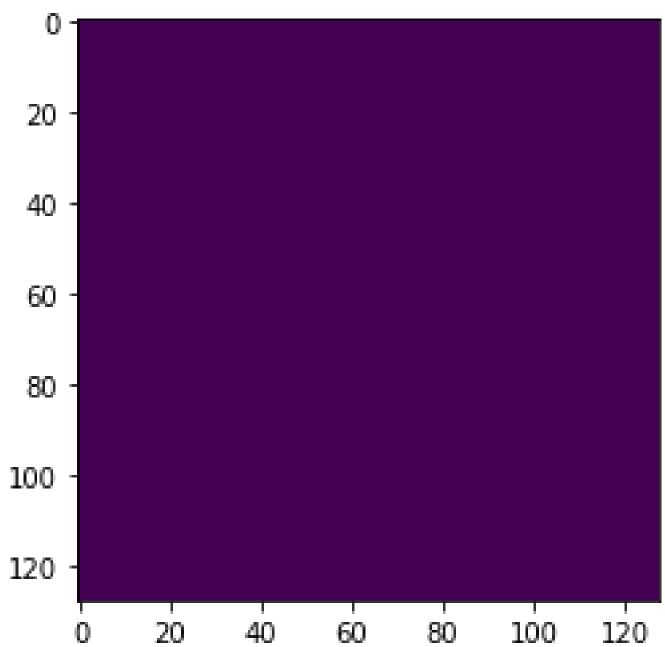
[ ]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model.to(device)

for i, batch in enumerate(train_loader, 0):
        if i == 1: break
        image, mask = batch
        image = image.to(device)
        mask = mask.to(device)
        image = image.type(torch.cuda.FloatTensor)
        mask = mask.type(torch.cuda.FloatTensor)
        mask = torch.argmax(mask, dim=1)
        outputs = model(image.to(device))['out']
        predicted_mask = torch.argmax(outputs, dim=1)
        print(image.shape, mask.shape, outputs.shape, predicted_mask.shape)
        print()

index = 12
plt.imshow(image.cpu()[index].permute(1,2,0)); plt.show()
plt.imshow(mask.cpu()[index]); plt.show();
plt.imshow(predicted_mask.cpu()[index]); plt.show();
```

```
torch.Size([16, 3, 128, 128]) torch.Size([16, 128, 128]) torch.Size([16, 21,
128, 128]) torch.Size([16, 128, 128])
```

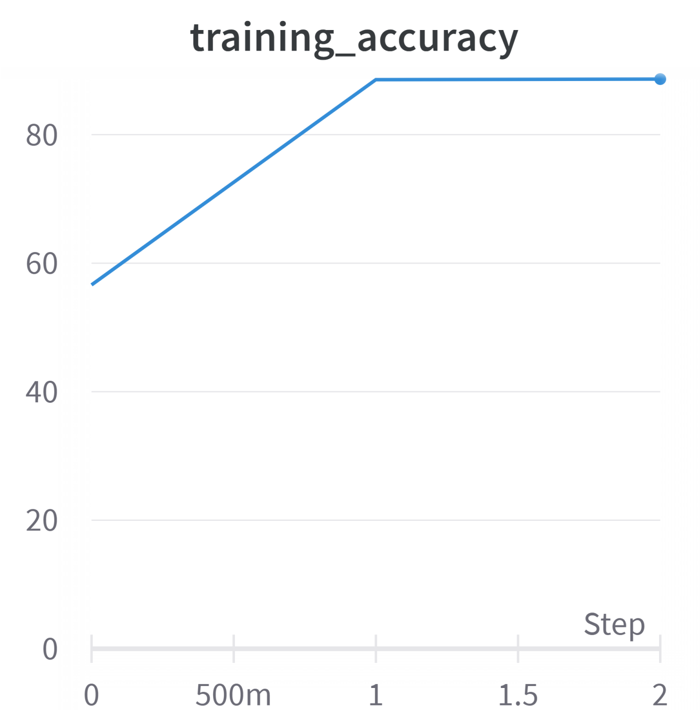
```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers).
```

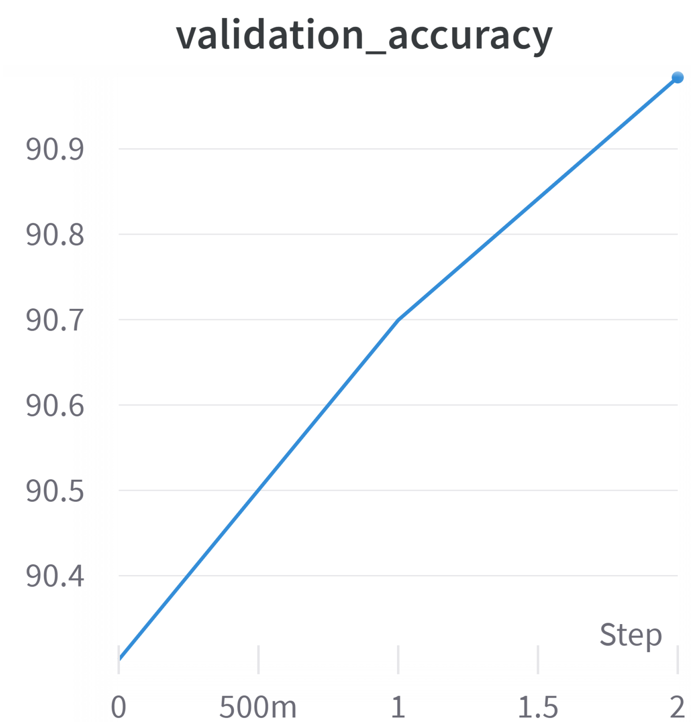


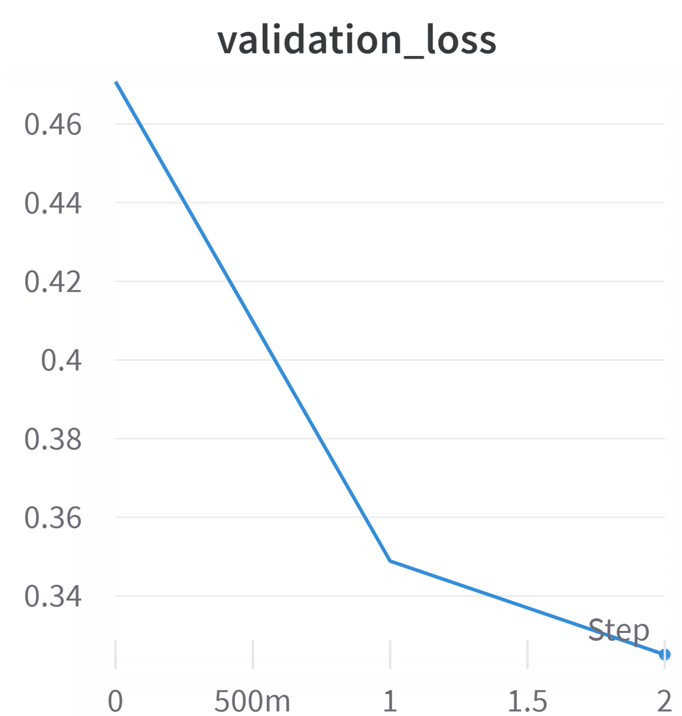

0
20
40
60
80
100
120
0
20
40
60
80
100
120


0
20
40
60
80
100
120
0
20
40
60
80
100
120

WandB Training and Validation Plots







5.2 2(b) *(6 points) Report the classwise performance of the test set in terms of pixel-wise accuracy, F1-Score and IoU (Intersection Over Union). Also report precision, recall and average precision (AP). Use the IoUs within range [0, 1] with 0.1 interval size for computation of the above metrics. You may refer to this article to learn more about the evaluation of segmentation models. Include all your findings in the submitted report.*

[ ]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score,␣
↪recall_score, average_precision_score, jaccard_score,␣
↪precision_recall_fscore_support
import numpy as np
criterion = nn.CrossEntropyLoss()
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = model.to(device)

def calculate_metrics_for_class(class_idx):
    # class_idx  ->  index of the class to evaluate
    threshold = 0.5  # probability threshold for binarizing the output mask
    model.eval()
    y_true = []  # true segmentation masks
    y_pred = []  # predicted segmentation masks

    with torch.no_grad():
      valid_loss = 0          # Training loss
      correct_pixels = 0      # Number of pixels correctly predicted
      total_pixels = 0        # Total number of pixels in the validation set
      for image, mask in tqdm(val_loader):
          image = image.to(device)
          mask = mask.to(device)
          # mask = torch.mean(mask, dim=1, keepdim=True)
          image = image.type(torch.cuda.FloatTensor)
          mask = mask.type(torch.cuda.FloatTensor)
          outputs = model(image)['out']
          loss = criterion(outputs, mask.argmax(dim=1))

          # calculate training loss
          valid_loss += loss.item()
          # calculate training accuracy
          predicted_mask = torch.argmax(outputs, dim=1)
          correct_pixels += torch.sum(predicted_mask == mask.argmax(dim=1)).
↪item()
          total_pixels += torch.numel(predicted_mask)
          batch_accuracy = correct_pixels / total_pixels
      valid_accuracy = 100 * correct_pixels / total_pixels

    # Flatten the arrays
    y_true = np.concatenate(y_true)
```

```
    y_pred = np.concatenate(y_pred)

    # Calculate the metrics
    accuracy = accuracy_score(y_true.ravel(), y_pred.ravel())
    f1 = f1_score(y_true.ravel(), y_pred.ravel())
    precision = precision_score(y_true.ravel(), y_pred.ravel())
    recall = recall_score(y_true.ravel(), y_pred.ravel())
    average_precision = average_precision_score(y_true.ravel(), y_pred.ravel())
    iou = jaccard_score(y_true.ravel(), y_pred.ravel())
    return accuracy, f1, precision, recall, average_precision, iou


classwise_metrics = {}
num_classes = 21
for class_idx in range(num_classes):
    # Calculate the metrics for the current class
    accuracy, f1, precision, recall, average_precision, iou =␣
 ↪calculate_metrics_for_class(class_idx)

    # Store the metrics in a dictionary
    classwise_metrics[class_labels[class_idx]] = {
        'accuracy': accuracy,
        'f1': f1,
        'precision': precision,
        'recall': recall,
        'average_precision': average_precision,
        'iou': iou
    }

for class_idx, metric in classwise_metrics.items():
    print('Metric for class: ', class_labels[class_idx])
    print(classwise_metrics[class_labels[class_idx]])
    print()
```

6 Q2 Part-3 *Data augmentation techniques*

6.1 3(a) *Use any 2 (or more) Data Augmentation techniques that are suitable for this problem. Remember that data augmentation techniques are used for synthetically adding more training data so that the model can train on more variety of data samples.*

[ ]:
```
class VOC2012Dataset(Dataset):
    def __init__(self, images, masks, img_indices, transform1=None,␣
 ↪transform2=None):
        self.img_indices = img_indices
        self.images = images
```

```
        self.masks = masks
        self.transform1 = transform1
        self.transform2 = transform2

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

    def __getitem__(self, idx, dontApplyTransform=False):
        img_name = self.images[self.img_indices[idx]]
        mask_name = self.masks[self.img_indices[idx]]
        img = cv2.imread(os.path.join(path_data_img, img_name)).astype(np.
 ↪float32) / 255
        mask = cv2.imread(os.path.join(path_data_mask, mask_name)).astype(np.
 ↪float32) / 255
        if self.transform1 and np.random.choice(2, 1) == 1:
            img = self.transform1(img)
            mask = self.transform1(mask)
        else:
            img = self.transform2(img)
            mask = self.transform2(mask)
        return img, mask

transform1 = transforms.Compose([
    transforms.ToTensor(),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(degrees=10),
    transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4, hue=0.
 ↪1),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
    transforms.Resize((128, 128))]
)
transform2 = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),
    transforms.Resize((128, 128))]
)

# Datset for training, validation and testing
train_dataset = VOC2012Dataset(img_path_list, mask_path_list,␣
 ↪train_images_index, transform1=transform1, transform2=transform2)
val_dataset = VOC2012Dataset(img_path_list, mask_path_list, val_images_index,␣
 ↪transform1=transform1, transform2=transform2)
test_dataset = VOC2012Dataset(img_path_list, mask_path_list, test_images_index,␣
 ↪transform1=None, transform2=transform2)
print('Dataset lengths:', len(train_dataset), len(val_dataset),␣
 ↪len(test_dataset))
```

```
# create the dataloaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=16, shuffle=True)
print('DataLoader lengths:', len(train_loader), len(val_loader),␣
↪len(test_loader))
```

```
Dataset lengths: 1026 292 146
DataLoader lengths: 65 19 10
```

6.2 3(b) *(4 points) Follow the same steps as in Question 2.2.(a) to train the model.*

[ ]:
```
wandb.init(entity="cv_assignment", project="Assignment-1",name="Q2-Part3")
wandb.config = {"learning_rate": 0.001, "epochs": 10, "batch_size": 16}
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

[ ]:
```
import torch.optim as optim
from torchvision.models.segmentation import FCN_ResNet50_Weights

# Define the FCN resnet50 model
model = models.segmentation.fcn_resnet50(weights=FCN_ResNet50_Weights.DEFAULT)

# Define the loss function (cross-entropy loss) and the optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001, weight_decay=1e-3)
```

[ ]:
```
epochs = 3
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = model.to(device)
print('Training on:', device)

for epoch in range(epochs):
    print('\nEpoch:', epoch)
    # =========================================== Training Phase␣
↪============================================
    model.train()
    train_loss = 0         # Training loss
    correct_pixels = 0     # Number of pixels correctly predicted
    total_pixels = 0       # Total number of pixels in the training set
```

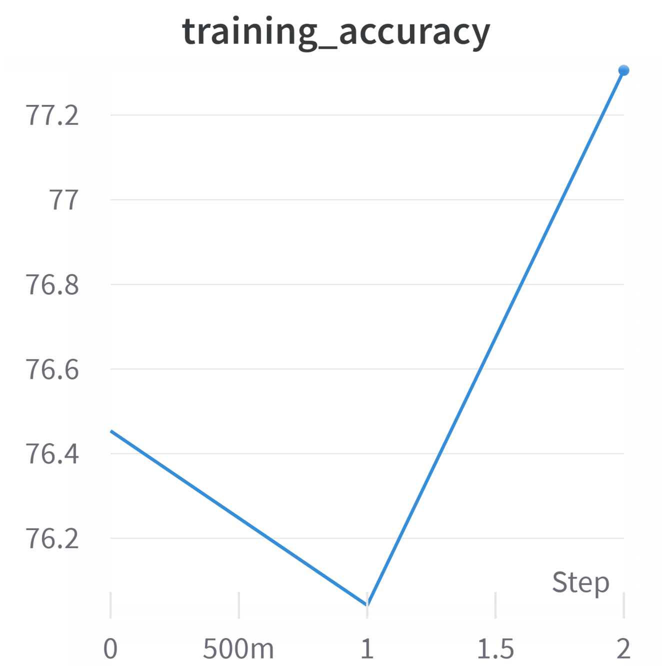
```
    for image, mask in tqdm(train_loader):
        image = image.to(device)
        mask = mask.to(device)
        image = image.type(torch.cuda.FloatTensor)
        mask = mask.type(torch.cuda.FloatTensor)
        optimizer.zero_grad()
        outputs = model(image)['out']
        loss = criterion(outputs, mask.argmax(dim=1))
        loss.backward()
        optimizer.step()

        # calculate training loss
        train_loss += loss.item()
        # calculate training accuracy
        predicted_mask = torch.argmax(outputs, dim=1)
        correct_pixels += torch.sum(predicted_mask == mask.argmax(dim=1)).item()
        total_pixels += torch.numel(predicted_mask)
        batch_accuracy = correct_pixels / total_pixels
    training_accuracy = 100 * correct_pixels / total_pixels

    # =========================================== Validation Phase␣
↪========================================
    model.eval()
    with torch.no_grad():
      # Validation loss
      valid_loss = 0         # Training loss
      correct_pixels = 0     # Number of pixels correctly predicted
      total_pixels = 0       # Total number of pixels in the validation set
      for image, mask in tqdm(val_loader):
          image = image.to(device)
          mask = mask.to(device)
          image = image.type(torch.cuda.FloatTensor)
          mask = mask.type(torch.cuda.FloatTensor)
          outputs = model(image)['out']
          loss = criterion(outputs, mask.argmax(dim=1))

          # calculate training loss
          valid_loss += loss.item()
          # calculate training accuracy
          predicted_mask = torch.argmax(outputs, dim=1)
          correct_pixels += torch.sum(predicted_mask == mask.argmax(dim=1)).
↪item()
          total_pixels += torch.numel(predicted_mask)
          batch_accuracy = correct_pixels / total_pixels
      valid_accuracy = 100 * correct_pixels / total_pixels
```

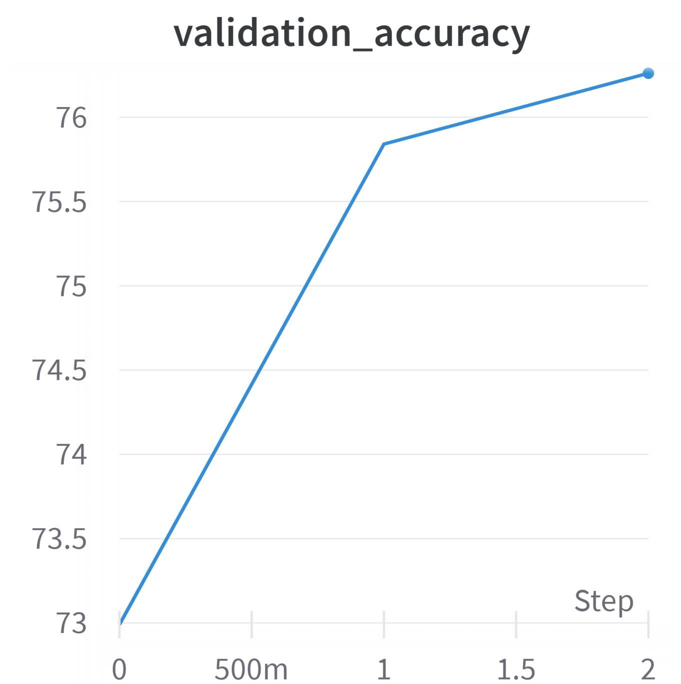
```
    print("\nTraining Loss: {:.4f},  Training Accuracy: {:.4f}".
↪format(train_loss / len(train_loader), training_accuracy))
    print("Validation Loss: {:.4f},  Validation Accuracy: {:.4f}".
↪format(valid_loss / len(val_loader), valid_accuracy))
    # Log the loss and accuracy to W&B
    wandb.log({'training_loss': train_loss / len(train_loader),␣
↪'validation_loss': valid_loss / len(val_loader),
               'training_accuracy': training_accuracy, 'validation_accuracy':␣
↪valid_accuracy})

print('Finished Training')
```

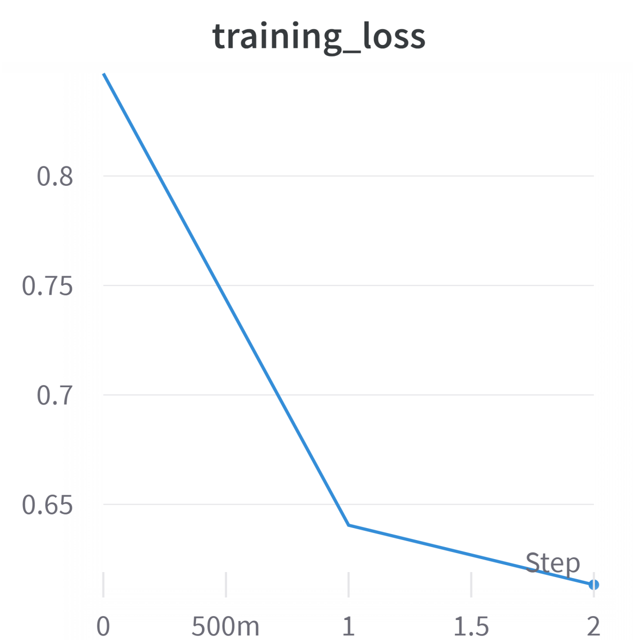
```
Training on: cuda:0

Epoch: 0

100%|      | 65/65 [01:14<00:00,  1.14s/it]
100%|      | 19/19 [00:17<00:00,  1.09it/s]


Training Loss: 0.8468,  Training Accuracy: 76.4536
Validation Loss: 0.6797,  Validation Accuracy: 72.9897

Epoch: 1

100%|      | 65/65 [01:10<00:00,  1.09s/it]
100%|      | 19/19 [00:17<00:00,  1.09it/s]


Training Loss: 0.6405,  Training Accuracy: 76.0417
Validation Loss: 0.6757,  Validation Accuracy: 75.8406

Epoch: 2

100%|      | 65/65 [01:11<00:00,  1.10s/it]
100%|      | 19/19 [00:16<00:00,  1.17it/s]


Training Loss: 0.6133,  Training Accuracy: 77.3058
Validation Loss: 0.6143,  Validation Accuracy: 76.2606
Finished Training
```

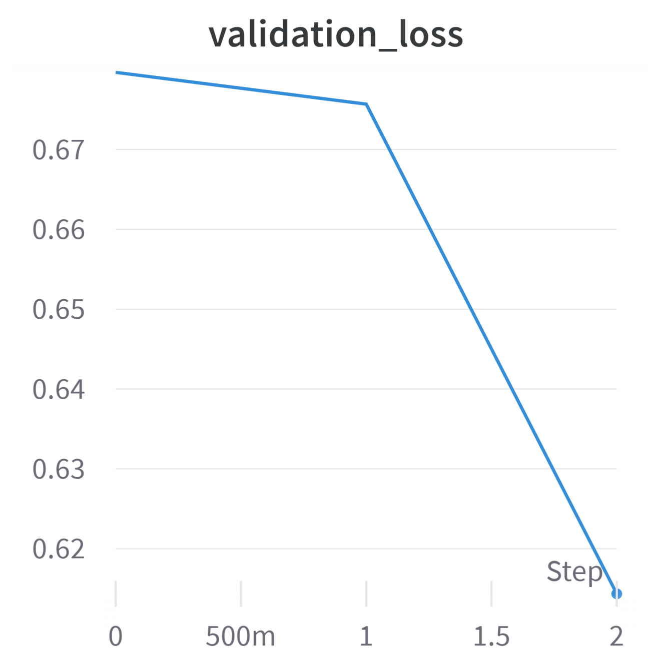
[ ]:
```
#  W&B: Save Model
torch.save(model, 'fcs_resnet50_q2_c.pt')
torch.save(model.state_dict(),"fcs_resnet50_q2_c.pth")
artifact = wandb.Artifact('model', type='model')
artifact.add_file('fcs_resnet50_q2_c.pt')
artifact.add_file('fcs_resnet50_q2_c.pth')
wandb.log_artifact(artifact)
```

WandB Training and Validation Plots









```
wandb.finish()
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

[ ]:
```
import pickle
with open('fcs_resnet50_aug_model.pickle', 'wb') as f:
  pickle.dump(model, f)

with open('fcs_resnet50_aug_model.pickle', 'rb') as pickle_in:
    fcs_resnet50 = pickle.load(pickle_in)
```

7 Q4 *Compare and comment on the performance of both the trained architectures.*

FCN Resnet-50 Model

Training Accuracy: 88.6495 Validation Accuracy: 90.9837

FCN Resnet-50 Model with Data augmentation

Training Accuracy: 77.3058 Validation Accuracy: 76.2606

FCS Resnet-50 trained without augmented data performed better than the Resnet50 trained with augmented data. Resnet 50 with data augmentation modifies the input images, so it makes the input images somewhat harder to classify and hence the performance with data augmentation is somewhat lesser