IMPLEMENTATION

- Google colab is used to write all the codes.
- Pytorch is used for all deep learning applications
- PascalVOC dataset was uploaded to colab from drive.
- Question 2 has 3 parts
- Part 1 and 3 are coded together, part 2 is coded separately.

1st Part:

- The pre trained ResNet50 model was loaded to my notebook.
- Model is brought to evaluation mode.
- A look up table for mapping colours to image classes is created.
- Dataset and data loaders are created (following steps are followed):
    1. From image folder in drive, image files and annotation files are stored in a list called 'image_files' and 'annot_files'. Only available annotations files are read which can be found using 'test.txt' file.
    2. A custom dataset function is created that takes in input image and annotation image and converts annotation image to class map (ground truth) using the look up table and returns them back
    3. Then finally dataloader is created using the above custom dataset.
- The dataloader is iterated and for each image the output of model is converted to colour map and its accuracy and IOU calculated.
- A separate function for calculating IOU is created that takes predicted image map and ground truth label and returns back mean IOU value.
- Mean of accuracy and IOU for all images are then calculated.

2nd Part:

- At first dataset and data loaders are created (following steps are followed):
    1. From image folder in drive image files and annotation files are stored in a list called 'train_image_files' and 'train_annot_files' for training purpose and 'val_image_files' and 'val_annot_files' for validation purpose. Only available annotations files are read which can be found using 'train.txt' file for training images and 'val.txt' for validation images.

2. A custom dataset function is created that takes in input image and annotation image and converts annotation image to class map (ground truth) using the look up table and returns them back
3. Then finally train_dataloader and test_dataloader is created using the above custom dataset.

- In next step, model is built by removing 'classifier' layer of mobilenet_v2 and adding a de-convolution layer to it. The de-convolution layer weights are initialised to bilinear kernel.
- The code for only bilinear kernel part is referred from the following blog: https://d2l.ai/chapter_computer-vision/fcn.html.
- Then loss function and optimiser are defined and then model is trained on train dataset for 40 epochs. Model is then brought to 'eval' mode.
- Next model then tested on few images and then on validation dataset and an accuracy of 76 percent is noted.

3rd Part:

- The test4.jpg and test5.jpg image are loaded on to colab (in the code of part_1) and they are segmented using the FCN model (ResNet50).
- The images are rotated and then segmented using the FCN model.
- Then Gaussian noises are added to images and then further segmented.

RESULTS AND ANALYSIS

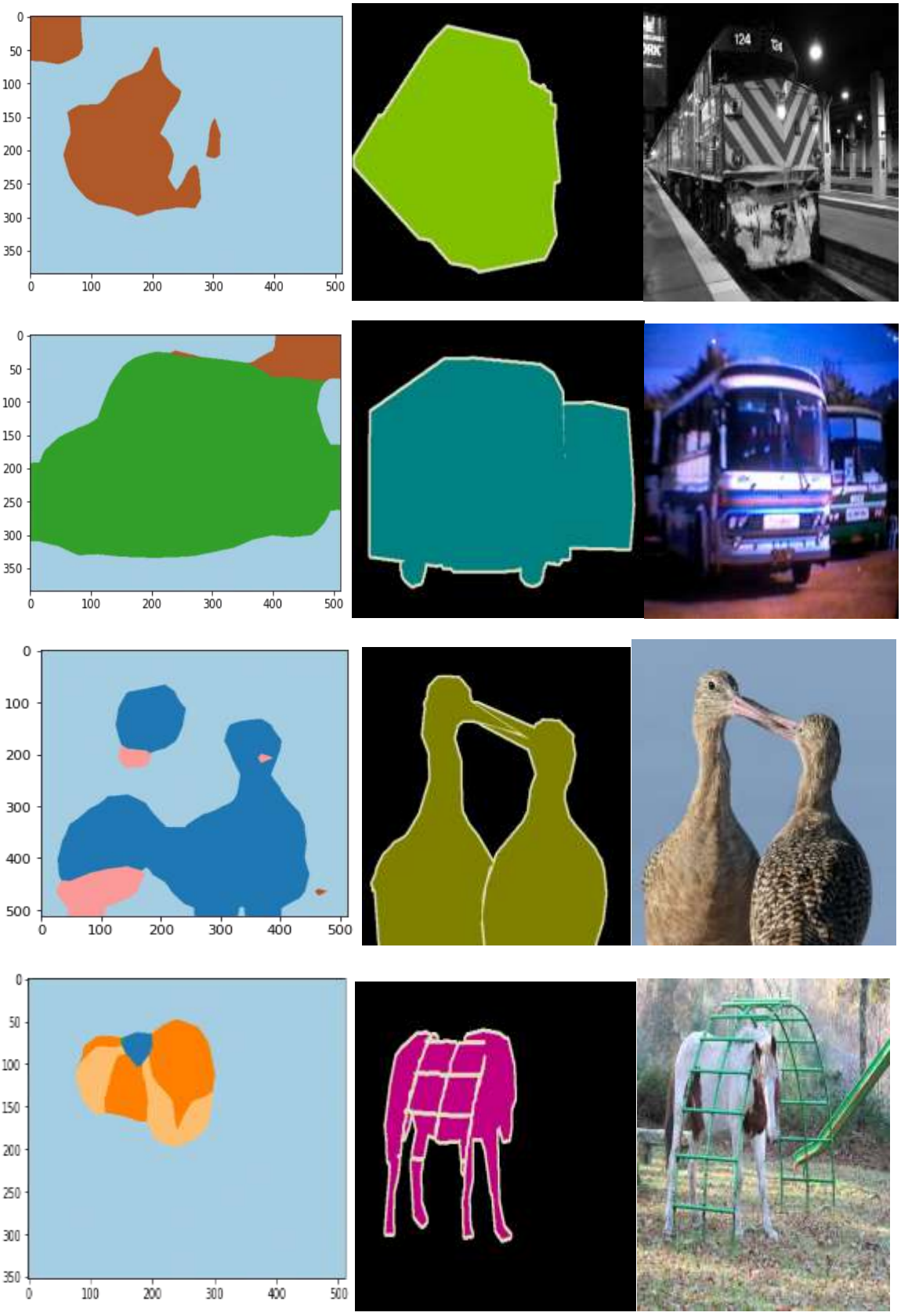
1st Part:

- The segmentation is done for a test image and the result segmentation is good with 98 percent accuracy and 84.3 percent IOU.
- Now for the whole test folder images are segmented and the overall mean accuracy was 91.6 percent and mean IOU was 64 percent.

2nd Part:

- The segmented image is not that good as that of Resnet50 but still segments are visible and can be seen.
- Pixel accuracy on validation set was found to be 76 percent and mean IOU was found around 40 percent.
- Some images segmented using changed mobilenet_v2 and their corresponding annotations and images are shown below:

Code output Annotation image Image

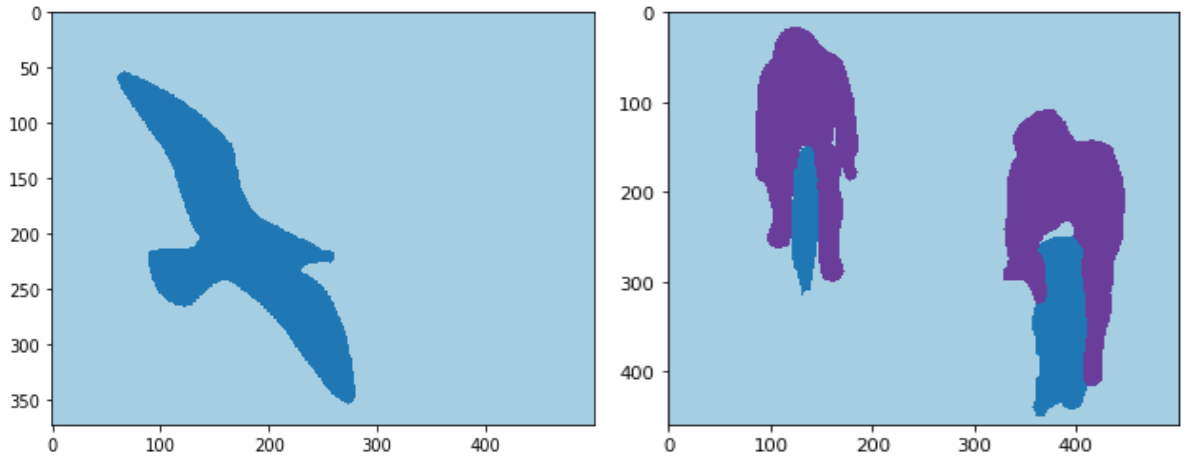


- The pixel colour of annotations and my code output are different because of matplotlib library being used to show the images.

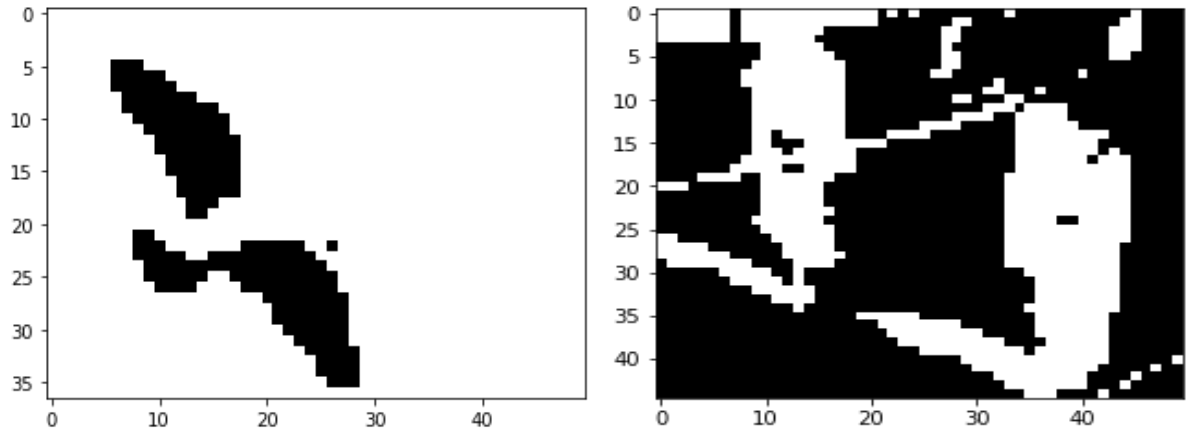
3rd Part:

- Both the n-cut and FCN (Resnet50 based) algorithms segmented the picture correctly.
- Difference is n-cut algorithm separated the image into broadly two segments, while FCN has segmented into multi-class.
- FCN is relatively faster than n-cut, for n-cut use similarity matrix and eigen value calculations that are computationally heavy.
- Below are the results for both the algorithms.

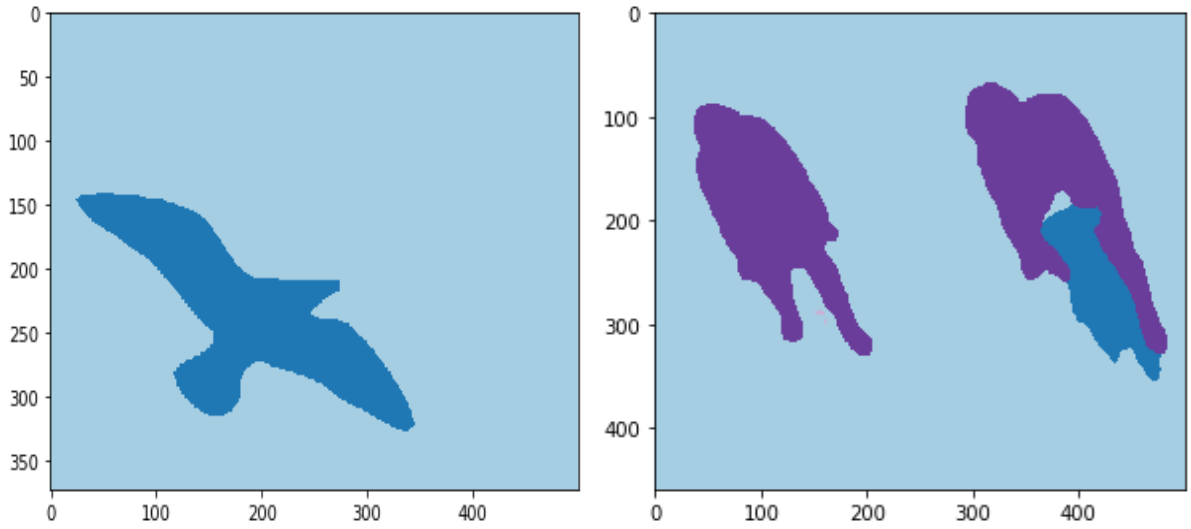
FCN normal image output:



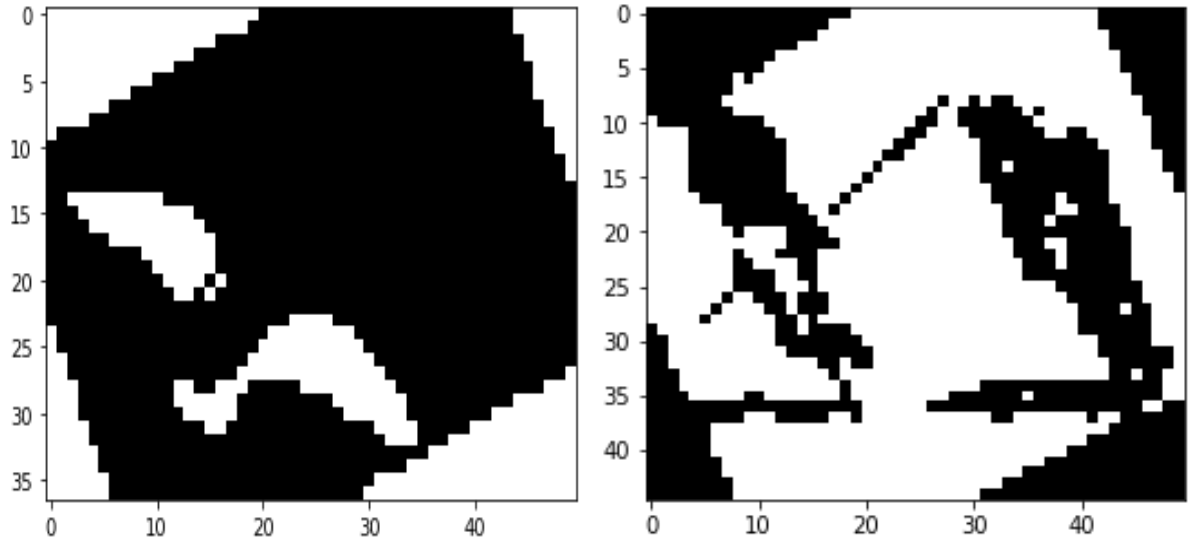
N-cut normal image output:



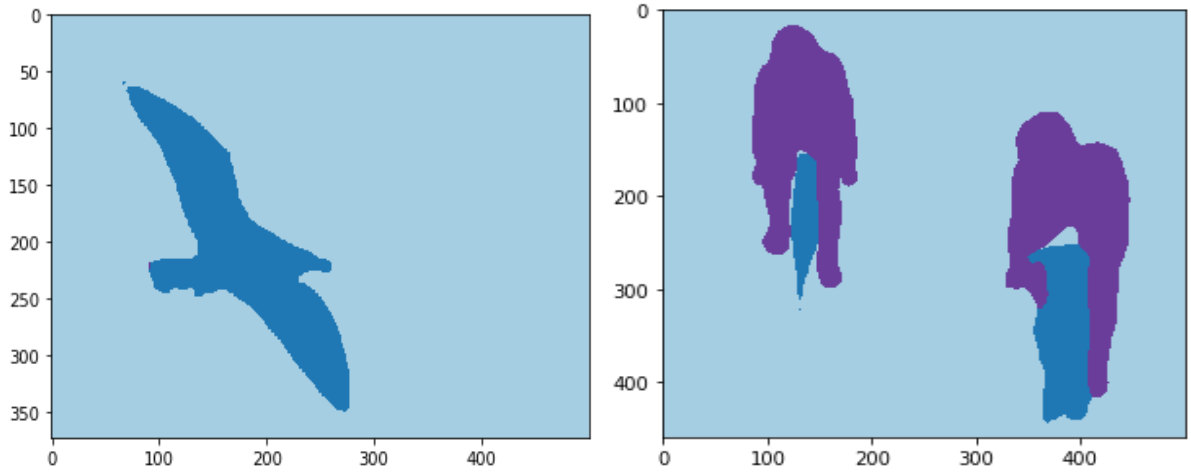
FCN rotated image output:



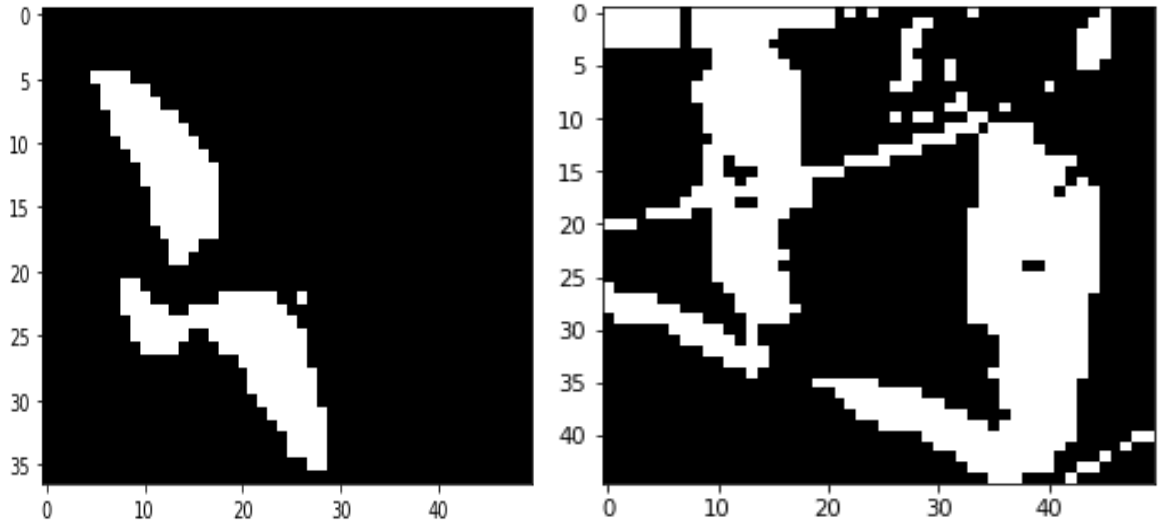
N-cut rotated image output:



FCN Gaussian noise added image output:



N-cut Gaussian noise added image output:



PROBLEMS FACED

1st Part:

- Not all files that are present in the test have annotation files. So only those files were loaded using the OS module that have name in the 'test.txt' file in the test folder of pascalVOC folder. The paths have to be carefully stored in the list after sorting else the image files won't match with corresponding annotation files.

2nd Part:

- A lot of issues were found in this part of problem.
- Various model were tried and tested but in most of them even after training for 50 epochs the results were random coloured pixels and not some real segments.
- These are some examples of models that were tried and failed:
    1. Model containing 5 de-conv layers but without any initialised weights and with no weights given to any class in the cross entropy loss function with adam optimiser.
    2. Model containing 5 de-conv layers, without any weights and with zero weight given to background class in the cross entropy loss function with adam optimiser. (Without the zero weight for background class all the output images were coming as background, probably as background class was majority the model was learning to annotate all pixel as background).

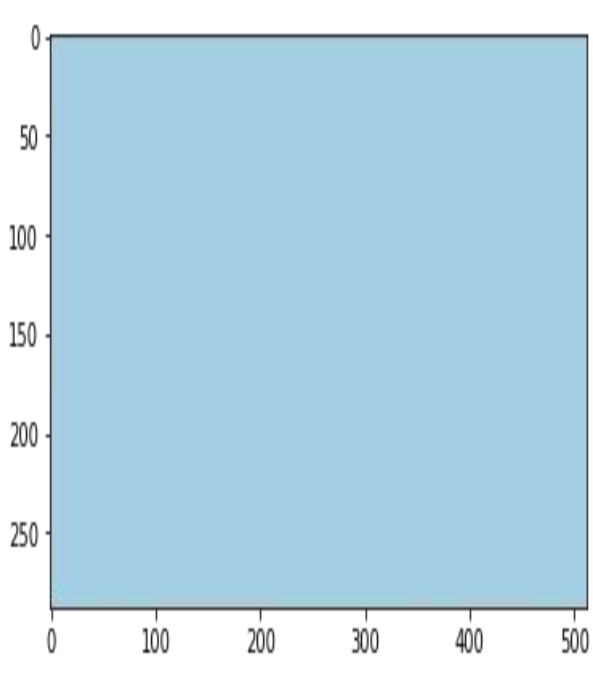
3. The above two models were also tested for SGD optimiser but with no improvements.
4. The above two models with a different version of cross entropy loss also did not give segments.

- Only after bilinear kernel was included and the weights of de-convolution layers initialised to it the model started giving some visible results. Probably because the weights were earlier getting to some local minima which is not optimum.
- An example of output of failed models and their code:

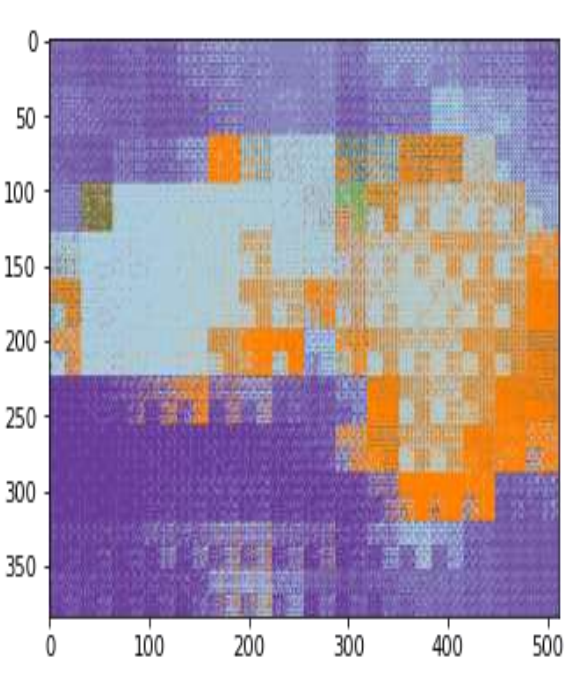
Failed code:

```
class FCN(nn.Module):
    def __init__(self, num_classes=21):
      super(FCN, self).__init__()

      ## Load the pretrained MobileNetV2 model
      self.backbone = torchvision.models.mobilenet_v2(pretrained=True)

      # Replace the last fully connected layer
      #self.backbone.features[18]=nn.Conv2d(320, num_classes, kernel_size=(1,1), stride=(1,1))
      in_features = self.backbone.classifier[1].in_features
      self.backbone.classifier = nn.Conv2d(in_features, 320, kernel_size=1, stride=1)

      # Initialize the upsampling layers
      self.deconv1=nn.ConvTranspose2d(320,160,kernel_size = 2,stride=2)
      self.deconv2=nn.ConvTranspose2d(160,96,kernel_size = 2,stride=2)
      self.deconv3=nn.ConvTranspose2d(96,64,kernel_size = 2,stride=2)
      self.deconv4=nn.ConvTranspose2d(64,num_classes,kernel_size = 2,stride=2)
      self.deconv5=nn.ConvTranspose2d(num_classes,num_classes,kernel_size = 2,stride=2)

    def forward(self, x):
      #print(x.shape)
      x = self.backbone.features(x)
      x = self.backbone.classifier(x)
      #print(x.shape)
      x = self.deconv1(x)
      x = self.deconv2(x)
      x = self.deconv3(x)
      x = self.deconv4(x)
      x = self.deconv5(x)
      #print(x.shape)

      return x
```

Failed outputs:



With weight to background class



With 0 weight to background class

- The segments after adding bilinear filter is also not that great in comparison to ResNet50 model, probably because it needs more training image (it was only trained on 209 images).

3rd Part:

- After doing transformations like rotation and adding Gaussian noise the image becomes of type float-64 format and need to be converted back to float-32 format for the FCN (ResNet50) model to process else it throws an error.

QUALITATIVE COMPARISION

- N-Cut is a slower than FCN based method because it calculates the eigen values and computes the similarity matrix.
- For N-cut a great deal of down-sampling is required to speed up the process whereas no such requirements is necessary for FCN based methods.
- As we start increasing number of segments in n-cut the algorithm becomes more complex. There is no such restriction in fcn based methods

- FCN based method not only segments the image but also give a label to each segment, this is not possible in n-cut, it does not tell how many different classes are there or if 2 different objects belong to same class or not.
- No training is required for N-cut, it's a classical method but training is required in case of FCN based approach.

Code for Part 1 and 3 -
https://colab.research.google.com/drive/1srd9_5DQWKOksfs0gbcoiKb5S1XSbdgq?usp=sharing

Code for part 2 -
https://drive.google.com/file/d/1oNWC2GJk9eTrRbsHJxI5GZcoCI3LbEqa/view?usp=sharing