Deep Learning Individual Project Report

TEAM 5

Name: Ujjawal Dwivedi

Topic: Deep Learning Based Multi-Compartment Brain Glioma Segmentation.

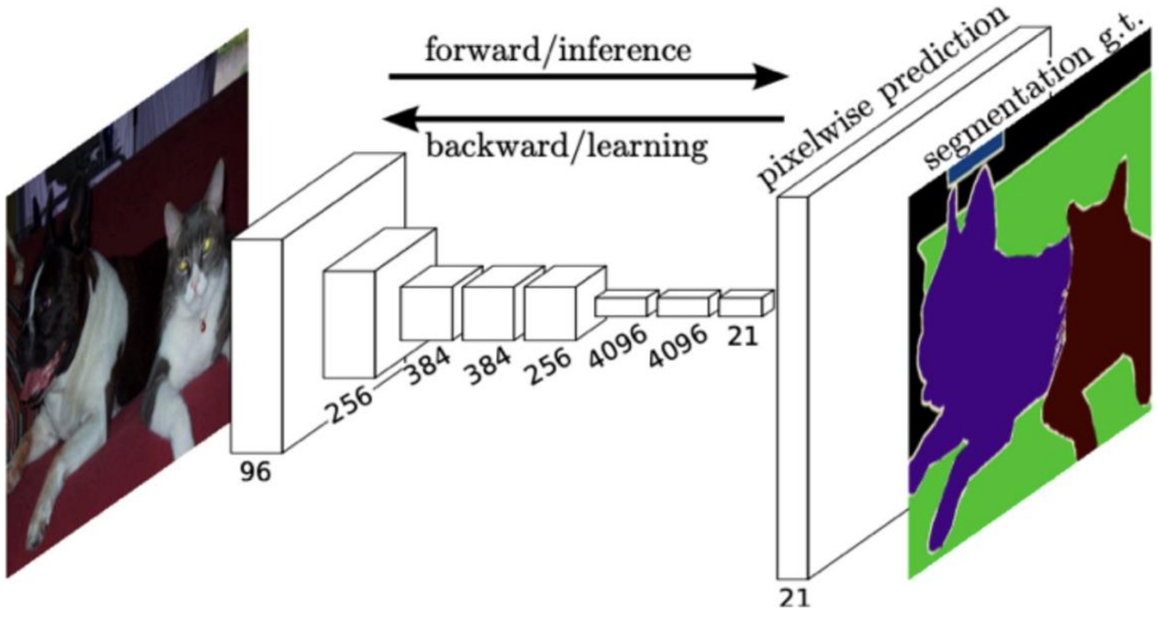
Contents

Background:

- **Problem:** Medical images, such as MRI, CT scans, ultrasound images, and histopathological images, often contain complex structures and anatomical details that need to be accurately analyzed and interpreted for diagnostic and treatment purposes.
- **Significance**: Biomedical image segmentation aids in the accurate localization and quantification of abnormalities, tumors, lesions, organs, and anatomical structures within medical images, which is crucial for clinical diagnosis and treatment planning. Segmentation enables researchers to analyze large-scale medical image datasets for studying disease progression, treatment outcomes, and developing new diagnostic and therapeutic techniques. Automated segmentation serves as a fundamental component in CAD systems, assisting healthcare professionals in early disease detection, risk assessment, and decision-making.
- **Solution:** Utilize deep learning architectures, such as convolutional neural networks (CNNs), U-Nets, FCNs, and attention mechanisms, for end-to-end learning-based segmentation tasks, which have demonstrated superior performance in various medical imaging applications. While employ appropriate evaluation metrics, such as Dice similarity coefficient (DSC), Intersection over Union (IoU), Hausdorff distance, and precision-recall curves, to assess the performance of segmentation algorithms and ensure clinically meaningful results.

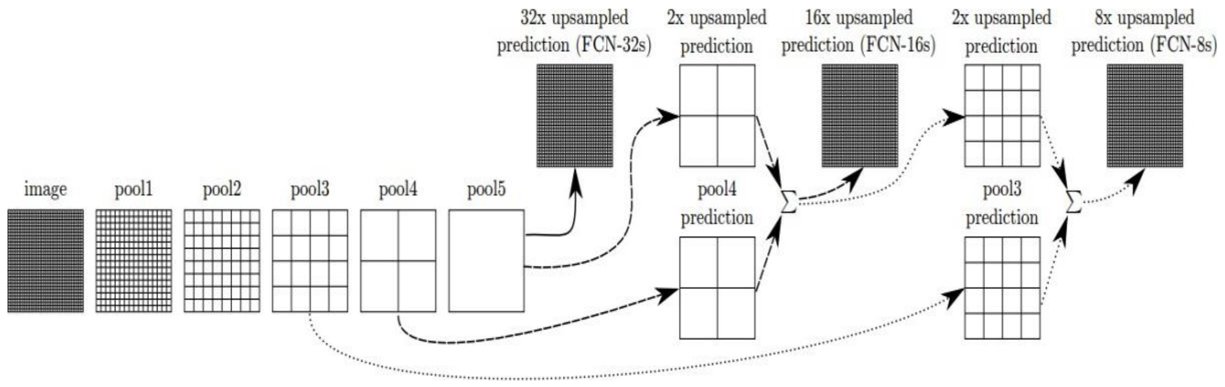
Description of individual work:

FCN:



The model architecture consists of an encoder network and a decoder network. The encoder network is based on a pre-trained or a custom convolutional neural network (CNN) modified to output multiple-scale feature maps. The decoder network is used for upsampling these feature maps using transposed convolutions and combining them using skip connections, which helps preserve spatial resolution and capture fine-grained details in the segmentation map.

Convolution is a process of getting the output size smaller. Thus, the name deconvolution comes from when we want to have upsampling to get the output size larger. It is also called up convolution and transposed convolution.



After going through pool5, then 32× upsampling is done to make the output have the same size as input image. But it also makes the output label map rough. And it is called FCN-32s.

This is because deep features can be obtained when going deeper; spatial location information is also lost. That means output from shallower layers has more location information. If we combine both, we can enhance the result. This fusing operation is just like the boosting/ensemble technique used in AlexNet, VGGNet, and GoogLeNet, where they add the results by multiple models to make the prediction more accurate. But in this case, it is done for each pixel, and they are added from the results of different layers within a model.

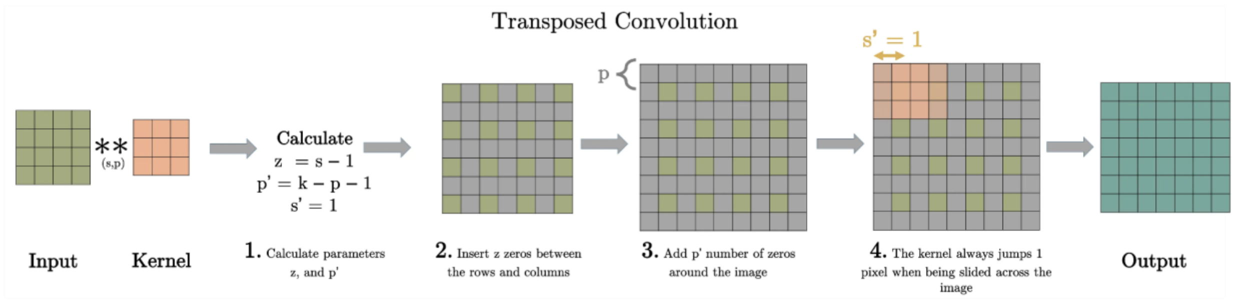
Transpose Convulation: A transpose convolution is not the same as deconvolution. A deconvulation layer reverses the standard convolution layer so it sounds similar to transpose. They are alike in one sense both generate the same spatial information. However, transpose only the reverse dimension rather than the values of standard convolution.

To explain a standard convolution, for a given size of the input (i), kernel (k), padding (p), and stride (s), the size of the output feature map (o) generated is given by

$$o = \frac{i + 2p - k}{s} + 1$$

A transposed convolutional layer, on the other hand, is usually carried out for upsampling i.e. to generate an output feature map that has a spatial dimension similar to that of the input feature map. For a given size of the input (i), kernel (k), padding (p), and stride (s), the size of the output feature map (o) generated is given by:

$$o = (i - 1) \times s + k - 2p$$


Transposed Convolution
**
(s,p)
Input
Kernel
Calculate
z = s − 1
p' = k − p − 1
s' = 1
1. Calculate parameters z, and p'
2. Insert z zeros between the rows and columns
p
3. Add p' number of zeros around the image
s' = 1
4. The kernel always jumps 1 pixel when being slided across the image
Output

Model Architecture: I first used an FCN8s model taken from GitHub to learn more about the architecture and how it works. The model used has 4 feature sequential layers consisting of convolution and pooling layers. However, an FCN can be implemented using a pretrained CNN combined with upsampling ie transpose layers for image segmentation.

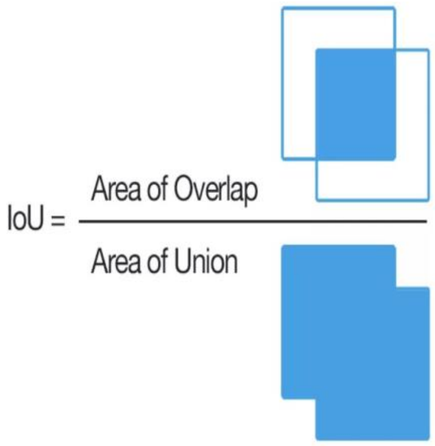
Custom Model: I built a custom model using a pretrained VGG16 coupled with upsampling layers. This enabled me to use transfer learning. This models starts off with a VGG16 layer, then a convolution layer followed by 4 transpose convolution layers ending with convolution layer to make size same as input.

```
class FCNVGG(nn.Module):
    def___init__(self):
        super(FCNVGG, self).__init__()

        vgg16 = models.vgg16(pretrained=True)
        self.vgg_features = vgg16.features
        # Freeze the VGG16 layers
        for param in self.vgg_features.parameters():
            param.requires_grad = False

        # Additional layers
        input_size = 7
        target_size = 224

        # Calculate the kernel_size and padding
        kernel_size = 2 * (target_size - input_size) + 1
        padding = kernel_size // 2

        # Define conv5 with adjusted parameters
          self.conv5 = nn.Conv2d(in_channels=512, out_channels=3,
kernel_size=kernel_size, padding=padding)
        nn.init.kaiming_normal_(self.conv5.weight)

        # Transpose convolution layers
            self.trans_conv1  = nn.ConvTranspose2d(in_channels=3,
out_channels=64, kernel_size=4, stride=2, padding=1)
        nn.init.kaiming_normal_(self.trans_conv1.weight)

            self.trans_conv2  = nn.ConvTranspose2d(in_channels=64,
out_channels=32, kernel_size=4, stride=2, padding=1)
        nn.init.kaiming_normal_(self.trans_conv2.weight)

            self.trans_conv3  = nn.ConvTranspose2d(in_channels=32,
out_channels=16, kernel_size=4, stride=2, padding=1)
        nn.init.kaiming_normal_(self.trans_conv3.weight)
```

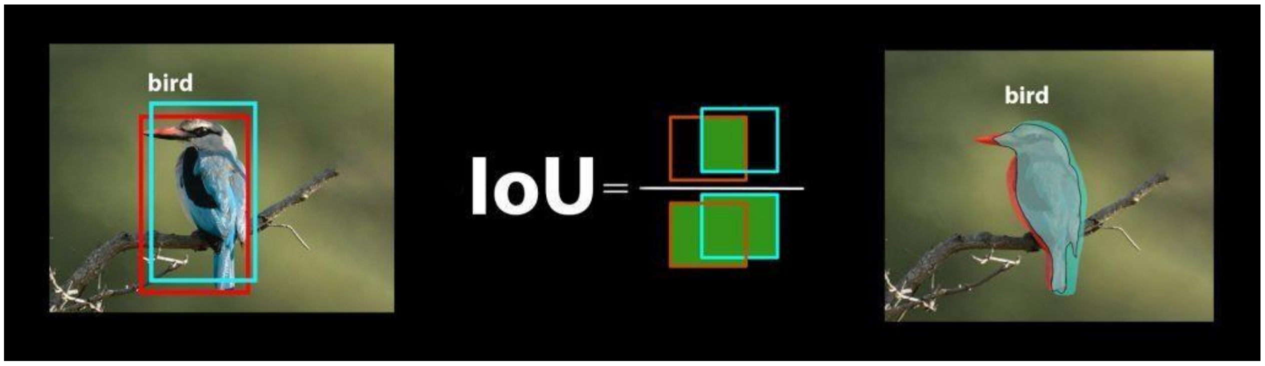
```
           self.trans_conv4  =  nn.ConvTranspose2d(in_channels=16,
out_channels=8, kernel_size=4, stride=2, padding=1)
       nn.init.kaiming_normal_(self.trans_conv4.weight)

           self.conv6  =  nn.Conv2d(in_channels=8,  out_channels=3,
kernel_size=3, padding=1)
       nn.init.kaiming_normal_(self.conv6.weight)

           self.conv7  =  nn.Conv2d(in_channels=3,  out_channels=3,
kernel_size=3, padding=1)
       nn.init.kaiming_normal_(self.conv7.weight)

   def forward(self, x):
       x = self.vgg_features(x)
       x = self.conv5(x)
       x = self.trans_conv1(x)
       x = self.trans_conv2(x)
       x = self.trans_conv3(x)
       x = self.trans_conv4(x)
       x = self.conv6(x)
       x = self.conv7(x)
       return x
```

Criterion: To calculate the metric performance I used a combination of two losses namely DICE loss and IOU loss. Both the losses were equally weighed to achieve the desired results.

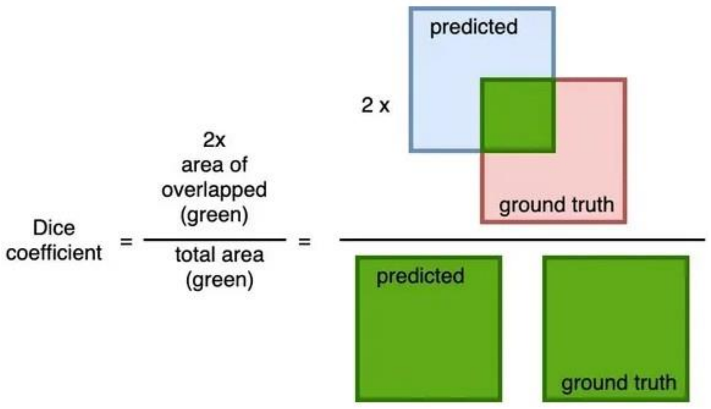
IoU Loss: The mathematical explanation for IoU loss looks like this.



The area is calculated as follows. The red box is the target mask given in dataset, whereas the green mask is the predicted region mask by the model.

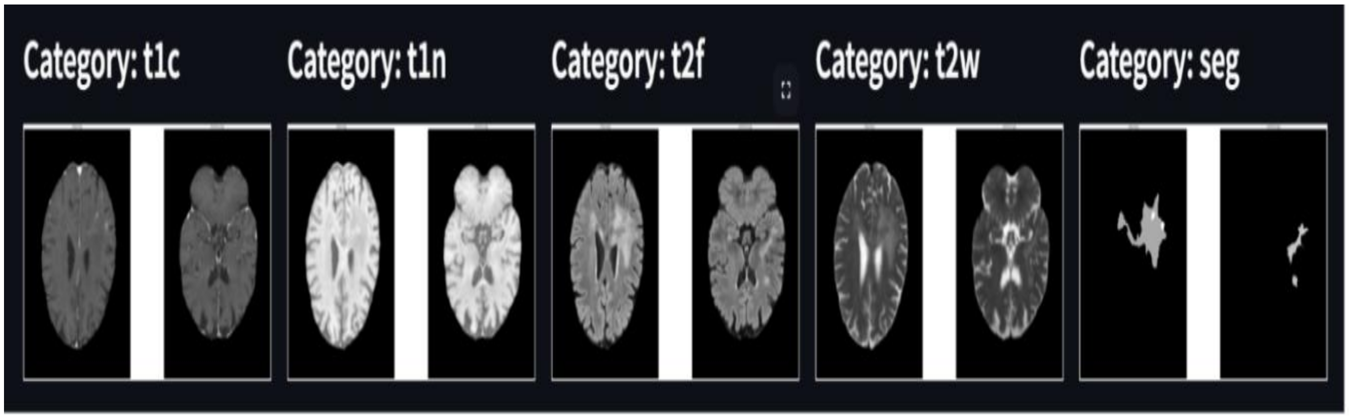


DICE Loss: The area is calculated the same way as in IoU. The mathematical equation varies a little.



Results:

The initial results without applying any masking or augmentation were quite promising. However, once augmentation was introduced, the performance deteriorated. Consequently, we had to discontinue using this model.



Hyperparameters:

- Batch size: 128
- Epochs: 10
- Learning Rate: 0.001
- Optimizer: torch.optim.Adam
- Scheduler: CosineAnnealingLR
- Image Size: 128*128

Summary and Conclusion:

In this individual project, the focus was on employing deep learning techniques, particularly Fully Convolutional Networks (FCNs), for biomedical image segmentation, with a specific emphasis on the segmentation of complex anatomical structures within medical images. The significance of accurate segmentation in medical imaging cannot be overstated, as it plays a critical role in clinical diagnosis, treatment planning, and research endeavors.
The FCN with pretrained model VGG model leveraging transfer learning performed better than the FCN8s model. I get better Loss score and predictions.

References:

- https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11
- https://github.com/SJTUzhanglj/FCN/blob/master/model.py
- https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11
- https://cvinvolution.medium.com/dice-loss-in-medical-image-segmentation-d0e476eb486
- https://learnopencv.com/iou-loss-functions-object-detection/
- https://medium.com/@datasciencemeetscybersecurity/image-segmentation-a-comprehensive-guide-c5ccded8af70
- https://medium.com/@jonas.cleveland/best-image-segmentation-models-a8620935b5b8#:~:text=Image%20segmentation%20types%20include%20semantic,is%20a%20combination%20of%20both.
- https://medium.com/geekculture/what-is-a-fcn-3135608d4903
- https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1