# COMP534-Assignment 3

SEMANTIC SEGMENTATION

**Introduction:**

This project is aimed to solve semantic segmentation using deep learning models and for this project we are using the Cambridge labeled objects in video dataset. Which is an implementation of computer vision in autonomous vehicles for object detection. It has video clips of objects in motion that are labeled. The dataset contains 101 images. The video clips in it have in total of 32 object classes and the classes are assigned to the objects that are relevant in it named such as tree, fence, traffic light and so on. All the 32 object classes mentioned are assigned to a specific color to it. Like void is denoted in black color, Sidewalk is denoted in dark blue color, Sky is denoted in grey color and so on. The dataset contains the original and ground truth images all the images are in 24-bit color PNG format. The original images are images that are unprocessed, and they might contain noise or distortion. We first load the data in zip format using given path and we extract the zip file and load it. Then We are splitting the data into train and test in the ratio of 90:10 so that the data can train better and produce better results. The ground truth images are images which we can reference to when we process the original image because it is the resembles the result that we want to obtain What we are doing is we are supposed to implement three deep learning models from various options given like Fully Convolutional Networks, UNet, Pyramid scene parsing network etc. In this project we are not using any traditional methods like clustering or watershed algorithm instead we are implementing three deep neural network model structures. There are a lot of models available but the three models we chose to implement are i) FCN16, ii) SDS, iii) UNet. We will see more in detail about these models later in this report. For training models, we have been given two options first: to start a model from the scratch or second: download a pretrained model. We chose to go with the second option and download a pretrained model and we will explain more about these pretrained models when we are explaining the individual models.

**First model (FCN16):**

The first model we implemented is FCN16(Fully Convolutional Network) which is a widely used model architecture, it is used in various applications like object detection, medical image analysis etc. They mainly work based on layers like convolution, pooling, and up sampling. For our model we have defined the FCN16 as a class and all the necessary functions inside a class. First, we define the init method which defines the structure of the model in which we defined the model architecture as FCN16. Next, we used the pretrained model ResNet 101 which enables the network to learn deep representations of the input data. which makes the data flow directly from the output of the one layer to the input of another layer, this actually removes the vanishing gradient problem which is a problem in updating weights in deep learning networks that restricts them from learning complex patterns. We are then removing the fully connected layer and the pooling layer because they are not required for the image segmentation. The layer of the resnet has a total of 2048 channels in total. Then we use conv to reduce it to 512. We also use batch normalization to normalize the data, we used ReLu as the activation function which is also another widely used activation function. The reason for using ReLu as the activation function is to have nonlinearity to the output of the neuron. We used dropout layer to avoid overfitting of the data. Then we use the forward function to define how the input information is passed. Which also initializes the layers like dropout, batch and so on. Then we use the weights function to initialize the weights and update the weights. These are the functions we define inside the class. Then we define another function for the train function where we declare the epochs, optimizer function and the losses, after that we define a validation function which we use to find the best evaluation metrics such as best accuracy and best intersection over union. We can use both of these functions to call. We also define functions for calculating IOU, calculating accuracy and function to plot loss and evaluation metrics.

**Second Model (SDS):**

The SDS model is a fully convolutional neural network utilized for semantic segmentation tasks, which receives an input picture and returns a pixel-wise classification map for each of the pixels in the input image.

The framework of the SDS model is identical to that of the FCN16 model. It begins by importing a pre-trained ResNet101 model and locking its parameters. Then, the fully linked layer and average pooling layer are eliminated to make the network completely convolutional. After this, the remaining layers in the ResNet model are retained and function as the feature extractor of the model. A convolutional layer with a kernel size of 2, stride of 1, and padding of 1 is inserted at the conclusion of the feature extractor. The output of the convolutional layer is then sent via a batch normalization layer, ReLU activation layer, dropout layer, and a 1x1 convolutional layer that correlates the output to the appropriate number of classes. Last, a transpose convolutional layer with a kernel size of 32, stride of 16, and no bias is included to upsample the output to the original picture size. In the beginning phase of the model, the weights of the transpose convolutional layer are initialized to conduct bilinear upsampling. During training, the Final two residual blocks of the feature extractor should be unfrozen during training to facilitate fine-tuning. Then we define the layers to normalize the data, to avoid overfitting and ReLu as the activation function. We define functions inside the class to initialize weights and update the weights.

**Third model (U-Net):**

The third model is a U-Net architecture that carries out picture segmentation using a series of convolutional and transposed convolutional layers. The encoder downsamples the input image to retain its characteristics, while the decoder upsamples the encoded features to the actual resolution of the image in the model. The major component of the model is defined using the DoubleConv class. Two convolutional layers are included, along with batch normalization and ReLU activation. By combining two convolutional layers, the DoubleConv seeks to learn more difficult information. We create an encoder-decoder structure in the UNet class using the encoder\_blocks and decoder\_blocks. The DoubleConv class is used to generate the encoder\_blocks, and each block has a max-pooling layer that downsamples the input mapping of features. Convolutional layers that have been transposed and DoubleConv blocks are used to create the decoder\_blocks. A transposed convolutional layer upsamples the input feature maps in each decoder block, which is followed by a DoubleConv block that combines the upsampled feature maps with the corresponding feature maps from the encoder\_blocks. Concatenation's goal is to provide a skip connection while maintaining the high-resolution characteristics. The model also includes a bottleneck layer made using the DoubleConv class, as well as a final\_conv layer that reduces the number of channels by utilizing a convolutional layer to reduce the output channels. The U-Net model typically has a symmetric structure with the same number of layers for the encoder and decoder, and the skip connections provide a fast cut for the gradients to propagate down to the lower levels, increasing the model's accuracy. All the above-mentioned functions are defined into a class.

**Hyperparameters used for Semantic Segmentation: -**

After defining all the above functions, we then call the functions where we call using the number of epochs, optimizer etc. We got the best results using these following set of hyperparameters. We are using optimizer as Adam which is another widely used optimizer in neural networks. We are using the learning rate of 0.001 Adam is an extension of the Stochastic gradient descent algorithm. The Adam optimizer is a combination of Adagrad and RMSprop. It updates the weight parameters depending upon the adaptive learning rate given. We use cross entropy loss as the Loss function where we get the loss values by subtracting the predicted and the actual values. Then we call the function to calculate the evaluation metrics and plotting them. We didn’t get any overfitting or underfitting problems for this model because we used batch normalization to normalize the data and dropout layers to avoid underfitting. Other hyperparameters we tried were using optimizer as RMSprop, learning rate as 0.01 and loss function as Pixel-wise Cross Entropy. But we did not receive the best results for that.

After defining the class, in the SDS model we define the hyperparameters and we got the best results by using the following set of parameters such as number of epochs as 10, The learning rate as 0.001, batch size as 16 and cross entropy loss as function for the SDS model and we call the train and validation functions that we defined early to calculate the evaluation metrics for the validation and the training dataset. After getting the best values of the evaluation metrics we then plot the best values. Other hyperparameters we tried with this model were setting the learning rate 0.1, optimizer as SGD and loss function as Dice Coefficient, but we did not get the best evaluation metrics by using these set of parameters. We didn’t get any overfitting or underfitting problems for this model because we used batch normalization to normalize the data and dropout layers to avoid underfitting.

After defining the class, in the UNET model we initialize the hyperparameters and we get the best results by using the following set of hyperparameters. Learning rate to 0.001, number of epochs to 10, batch size as 16 and cross entropy loss as loss function for the U-Net model. We used Adam as the optimizer for this model and then we called the functions for calculating the best IOU and accuracy and then we plot the values we got from that. We didn’t get any overfitting or underfitting problems for this model because we used batch normalization to normalize the data and dropout layers to avoid underfitting. Some other set of hyperparameters we tried were using the optimizer as Adagrad, learning rate as 1 and loss function as Weighted Pixel Cross Entropy. But we did not get the best results using these set of hyperparameters.

**Loss function and Evaluation Metrics: -**

The training loss is a measurement of the objective function, or cross-entropy loss in our model, that is computed on the training set during the training process. In our model, the FCN16 model was trained across 10 epochs, and each epoch saw a decrease in the training loss, which is a sign that the model is growing and learning. The objective function value determined on a validation set during the training phase is known as the validation loss. To evaluate the model's performance on data that was not used for training and to spot overfitting, the validation set was created. The model's performance on a test set, which is a set of data it hasn't seen during training or validation, is known as the model's testing accuracy. The FCN16 model in our instance has a testing accuracy of 0.8904, which is a strong result and indicates that the model can accurately predict the class labels for the test data.

A picture containing line, plot, diagram, slope

Description automatically generated

In the SDS model, the training loss decreases throughout the course of the 10 epochs, and it is ultimately 0.249. The model's ability to predict the class of each pixel in the testing data is measured by something called testing accuracy or pixel accuracy. It is calculated by dividing the total number of pixels by the number of pixels that were correctly anticipated. The average pixel accuracy in our SDS model for the testing data was 0.8981, meaning that the model correctly predicted the class of each pixel 89.81% of the time. The overlap between the anticipated and actual masks is measured by the intersection over union, or IoU. The answer is obtained by dividing the union of the predicted and actual masks by the intersection of those two masks. The model performs better the higher the IoU. The best IoU for your SDS model on the testing data is 0.8341, which indicates that the model can accurately predict where the items in the image will be.

A picture containing line, plot, diagram, slope

Description automatically generated

In the UNET model the training loss is quite significant in the first epoch but drops dramatically in the succeeding epochs, demonstrating that the model is learning from the training data. The best IoU and pixel accuracy on the validation data are 0.683 and 0.7717, respectively. The testing accuracy of the UNet model is stated to have a highest IoU of 0.654 and a mean pixel accuracy of 0.7476. These values imply that the model is not overfitting to the training data and is generalizing pretty well to unseen test data.

A picture containing line, plot, diagram, slope

Description automatically generated

Intersection over Union (IoU), commonly known as the Jaccard Index, and Pixel Accuracy are the assessment metrics used to evaluate the segmentation outcomes of the models.

* FCN16 Model - Based on the test data, the FCN16 model generated the best IoU of 0.8146 and pixel accuracy of 0.8904. This shows that the model performed better than the other models in properly classifying the pixels in the test data into the right categories.
* SDS Model - Based on the test data, the SDS model gave the best IoU of 0.8341 and a mean pixel accuracy of 0.8981. This shows that the SDS model is more accurate than the UNet model at properly predicting the segmentation of the test data.
* UNet Model - On the test data, the UNet model had the lowest IoU (0.6543) and mean pixel accuracy (0.7476). This suggests that the model did not anticipate the segmentation of the test data as precisely as the other models did.

Overall, the IoU and pixel accuracy of the SDS and FCN16 models were better than those of the UNet model, indicating that they are more appropriate for this job.

A picture containing line, plot, diagram, slope

Description automatically generatedA picture containing line, diagram, plot, text

Description automatically generated

A picture containing line, plot, diagram, slope

Description automatically generatedA picture containing text, diagram, line, plot

Description automatically generated

A picture containing line, plot, diagram, slope

Description automatically generatedA graph with blue lines

Description automatically generated with low confidence

CONCLUSION: -

The best results we got out of the three models we implemented was the SDS (Semantic Deep Segmentation) model, it was the one that gave us the best results for the hyperparameters we used. This image segmentation assignment was really informative and challenging for us. We all learned a lot from this assignment. This is the first time we worked with image data, and it was interesting to learn new things and implement new things. We had quite a few challenges while building up the U-Net model and choosing the hyperparameters for all the models. But it was a trial and error and we experimented with lot of hyperparameters to get the best results out of these models. There was a lot of model architectures and pretrained models to choose from and we think If we were given more time, we could have done more research and could have experimented with more models.