

University of New Haven Tagliatelle college of Engineering



Masters in Data Science

Course: Deep Learning

**Title: Semantic Image Segmentation with Deep Convolutional Nets, Atrous
Convolution, and Fully Connected CRFs**

By

Likith Sai Chaitanya Vadlapatla

Ajay Kumar Ganipineni

Under the guidance of,

Prof. Muhammad Aminul Islam, Ph.D.

mislam@newhaven.edu

Contents

Introduction.....	01
Dataset.....	01
Transfer learning.....	06
Mini network.....	11
Conclusion.....	13
Code and Dataset.....	13

Introduction:

Semantic segmentation is a fundamental computer vision job that requires classifying images at the pixel level, where each pixel is given a certain class name. It is essential to many applications, including autonomous driving, medical imaging, and scene comprehension. By permitting precise and effective predictions, advances in deep learning approaches have transformed semantic segmentation.

Transfer learning has become a potent strategy in the field of computer vision in recent years. It makes use of pre-trained models, which are developed using extensive datasets, and applies the information they have acquired to new tasks requiring little labeled data. Transfer learning eases the computational strain of building complicated models from the start and aids in overcoming the challenges of data scarcity. The U-Net is a well-liked design for semantic segmentation.

Due to its efficiency and simplicity, the U-Net architecture, which Ronneberger et al. introduced in 2015, has been frequently used for semantic segmentation tasks. It is an encoder-decoder structure with skip connections that concatenate encoder feature layers with their matching decoder layer counterparts. With the use of these skip connections, the model may more accurately segment data by capturing contextual information from the decoder as well as high-resolution features from the encoder.

This project's main goal is to investigate the idea of transfer learning in semantic segmentation and contrast it with the process of creating a mini-network from scratch using the U-Net architecture. We use a pre-trained U-Net model to accomplish this and assess its performance on a fresh dataset. A mini-network modeled after the U-Net architecture is also created and trained from scratch using the same dataset.

This research will help us better grasp the advantages and constraints of transfer learning in semantic segmentation tasks. Additionally, we seek to assess the viability of creating a mini network from scratch, which can shed light on the model's performance in the presence of scant labeled data.

Dataset:

- **Data collection**

For this project, we used a mobile device to take pictures all across the University of New Haven campus. The pictures were taken to reflect different situations and things found on campus. Our semantic segmentation challenge is built on top of this dataset.

- **Data Pre-Processing:**

The initially obtained photos ranged in size and format from 5MB to 12MB. We pre-processed the photos to maintain uniformity and lessen the computational load during training. The photographs had to be resized to a standard size and converted to the correct format as part of the pre-processing stages.

- **Annotation:**

Five identified classes were added to the photos to aid with supervised learning and evaluation: building, car, traffic sign, chair, and human. We manually labeled each object or area of interest in the photos using MATLAB to manually annotate the images. For our semantic segmentation models' training and evaluation, this annotation procedure produced ground truth labels.

- **Partitioning:**

We divided the annotated dataset into three subsets: the training dataset, the validation dataset, and the testing dataset in order to efficiently train and test our models. 80% of the annotated data were in the training dataset, while 10% each were in the validation and testing datasets. We were able to train the models on a significant chunk of the data thanks to this segmentation, while keeping distinct subsets for hyperparameter adjustment and final evaluation.

▼ **Train Valid Test split**

```
[ ] # lengths of training, validation, and test sets
data_length = len(df)
print(data_length)
train_length = int(0.8 * data_length)
validation_length = int(0.1 * data_length)
test_length = data_length - train_length - validation_length
```

```
130
```

```
[ ] train_length, validation_length, test_length
```

```
(104, 13, 13)
```

```
▶ from sklearn.model_selection import train_test_split

train_df, valid_df = train_test_split(df, test_size=0.1, shuffle=True)
train_df, test_df = train_test_split(train_df, test_size=0.11, shuffle=True)

train_df.shape, valid_df.shape, test_df.shape

((104, 2), (13, 2), (13, 2))
```

FIG:1 Data Partitioning Code Snippet

- **Augmentation:**

Because there was a lack of labeled data, we used data augmentation techniques to fictitiously expand our dataset. We generated additional samples that closely mirror the original data by applying various adjustments to the already existing photos. We used horizontal and vertical flipping of the photographs in this research, which effectively doubled the amount of the dataset. The addition of data aids in avoiding overfitting and improves the trained models' capacity for generalization.

```
import albumentations as A
from albumentations.pytorch import ToTensorV2

# Define a list of augmentations to apply
transforms_train = A.Compose([
    A.Resize(width=256, height=256),
    A.HorizontalFlip(p=0.5),
    A.Rotate(limit=30, p=0.5),
    A.RandomBrightnessContrast(p=0.2),
    A.Blur(p=0.1),
    A.Normalize(total_mean, total_std),
    ToTensorV2(),
])

transforms_val = A.Compose([
    A.Resize(width=256, height=256),
    A.Normalize(total_mean, total_std),
    ToTensorV2(),
])
```

FIG 2: Data Augmentation Code Snippet

- **Normalization:**

The preprocessing step of normalization is crucial for enhancing the convergence and stability of deep learning models. Using mean and standard deviation figures calculated from the complete dataset, we normalized the photos. By iterating through the dataset and adding up pixel-wise sums and squared sums, these values were determined. These totaled values were then used to calculate the mean and standard deviation. In order to help the models train, normalization makes sure that the input data have a zero mean and unit standard deviation.

Transfer learning:

- **Model architecture and objective function:**

We used a **MobileNetV2** encoder and the **U-Net architecture for transfer learning**. The ImageNet dataset's pre-trained weights were used to initialize the model. An encoder-decoder structure with skip links makes up the U-Net. The MobileNetV2 network, which is part of the encoder, collects hierarchical information from the input images. To capture both high-resolution details and contextual information, the decoder up samples the feature maps and employs skip connections to concatenate them with the associated encoder features. The **Cross-Entropy Loss objective function**, which is appropriate for multi-class semantic segmentation tasks, is what is employed for training.

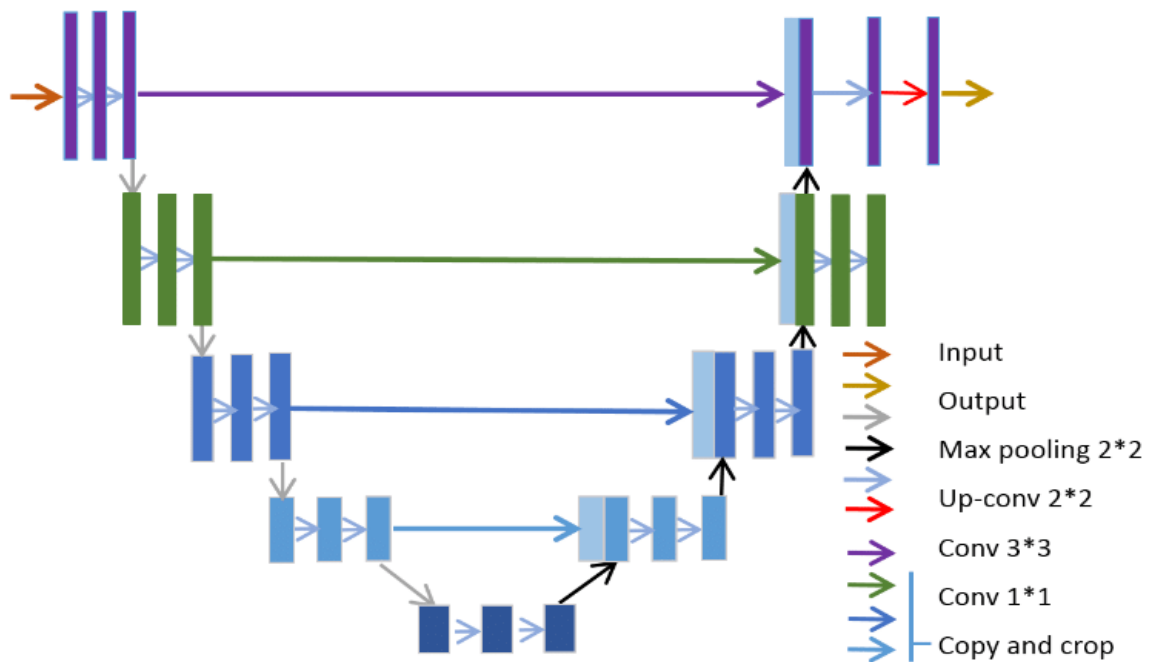


FIG: 3 UNet Architecture

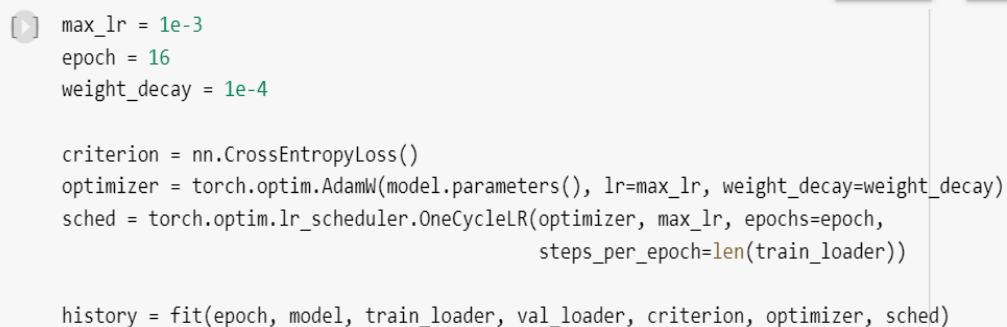
$$\text{Loss} = - \sum_{i=1}^{\text{output size}} y_i \cdot \log \hat{y}_i$$

FIG: 4 Objective Function

- **Experiments and results:**

The pre-trained U-Net model was further trained on our dataset as part of the fine-tuning procedure. With a learning rate of $1e-3$ and a weight decay of $1e-4$, we used the AdamW optimizer. These hyperparameters were selected in accordance with empirical findings and earlier work on semantic segmentation challenges.

We used a learning rate decay policy to stop overfitting and to keep the learning process stable. We employed the One Cycle Learning Rate Scheduler, which gradually raised the learning rate to its maximum value before lowering it as training came to a close. It has been demonstrated that using this strategy will increase convergence and broaden the application.



```
max_lr = 1e-3
epoch = 16
weight_decay = 1e-4

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(model.parameters(), lr=max_lr, weight_decay=weight_decay)
sched = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr, epochs=epoch,
                                             steps_per_epoch=len(train_loader))

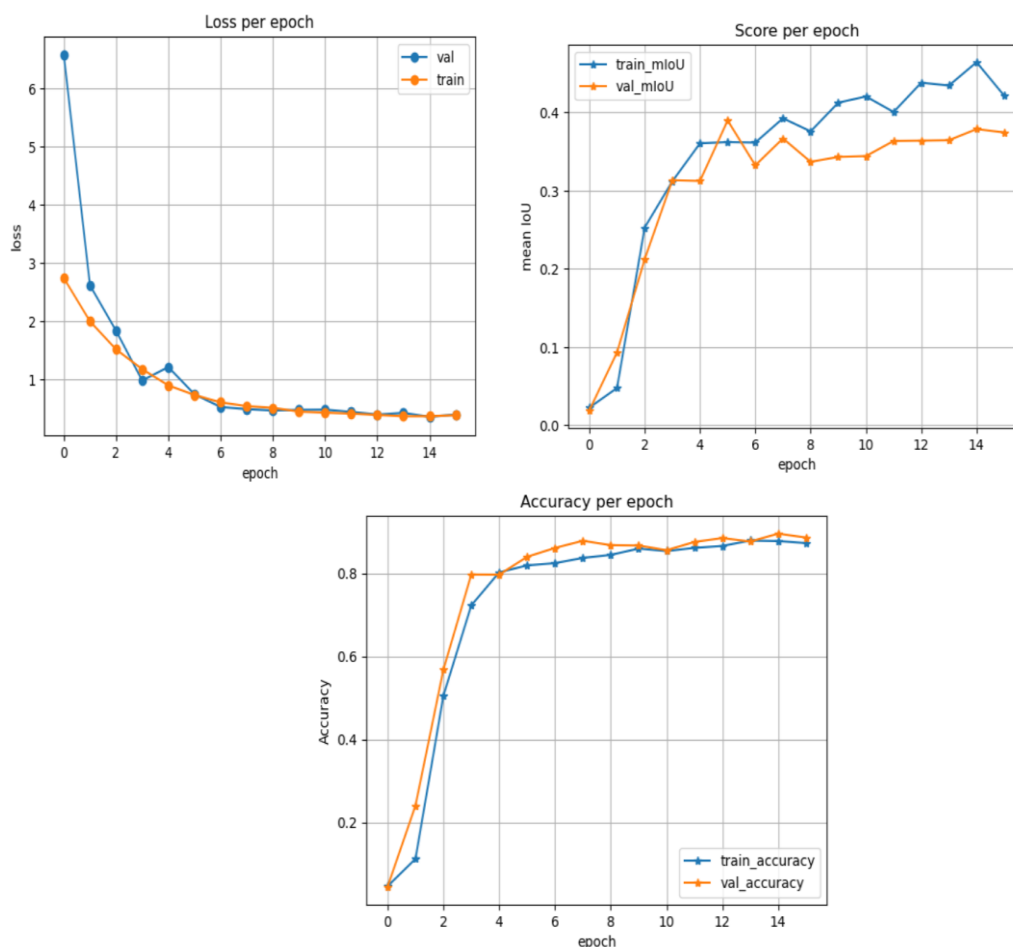
history = fit(epoch, model, train_loader, val_loader, criterion, optimizer, sched)
```

FIG 5: Hyperparameters and training

We kept an eye on the training and validation losses throughout the training phase to assess the effectiveness of the transfer learning approach. The best model was determined to have the lowest validation loss. The mean Intersection over Union (mIoU) and pixel accuracy measures were also used to evaluate the models.

The predetermined number of epochs, which was set to 16, was iterated over by the training and validation loops. Training loss, validation loss, training mIoU, validation mIoU, training accuracy, and validation accuracy were all recorded at each epoch. On the basis of the declining validation loss, the best model was kept. To avoid overfitting, training was halted if the validation loss did not reduce seven times in a row.

The transfer learning training procedure with the pre-trained U-Net model and the fine-tuning strategy produced encouraging results. Throughout the epochs, the model showed a decreasing training loss and validation loss, demonstrating effective learning. To achieve the best generalization performance, the best model was chosen based on the lowest validation loss.



FIG's : 6 Evaluation Metrics Plots

On a different test dataset, the evaluation metrics, such as the average test loss, average test mIoU, and average test correctness, were computed. The test loss revealed the model's overall error, whereas the test mIoU revealed information about the segmentation accuracy across various classes. When compared to the ground truth masks, the projected segmentation masks' pixel-level accuracy was measured as test accuracy.

Predictions:

```
print('Test Data mIoU', np.mean(Mean_IoU))
```

```
Test Data mIoU 0.6181245640293062
```

```
print('Test data pixel Accuracy', np.mean(pixel_accuracy))
```

```
Test data pixel Accuracy 0.8790450136885684
```

FIG: 7 Test data Metrics

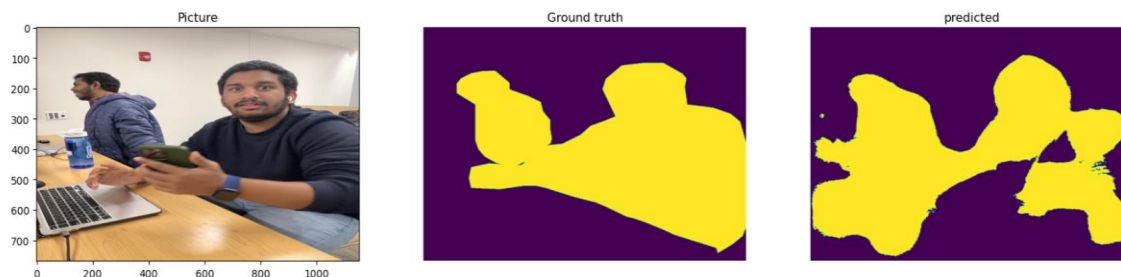


FIG : 8 Predicted Outputs Plots

- **Deployment**

We used a Python Flask API to implement the trained model for deployment. Users are able to make predictions by sending picture data through API endpoints thanks to the Flask API, which enables the model to be provided as a web service. This deployment strategy offers a practical and accessible way to use the trained model in practical applications.

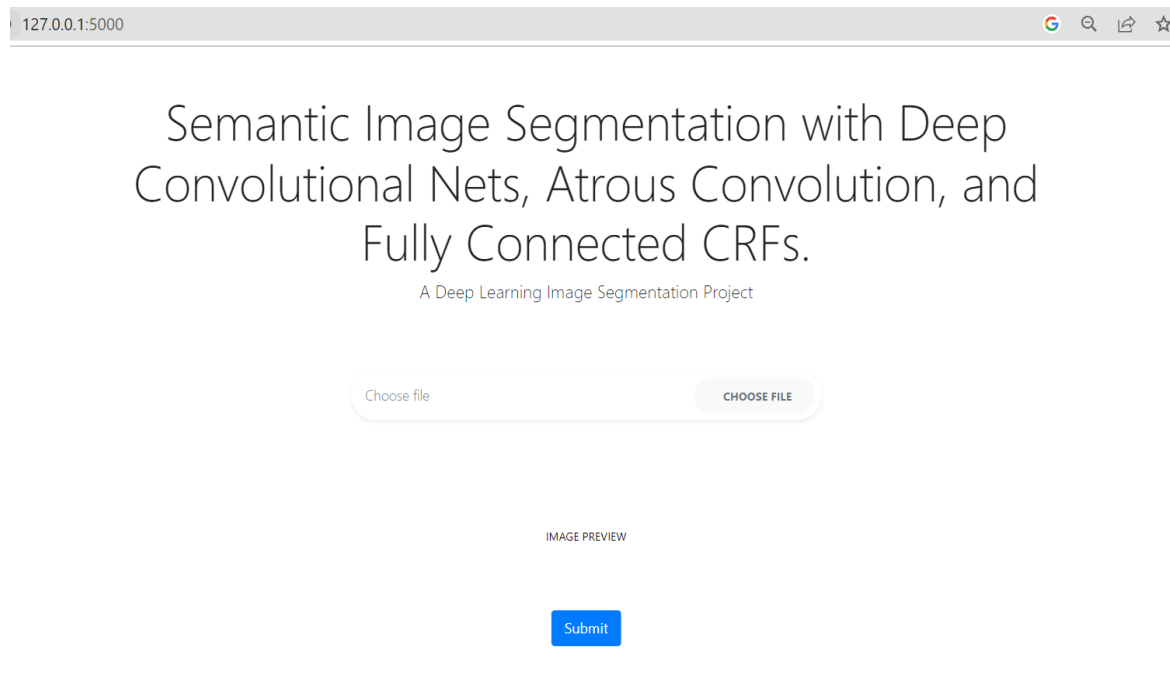


FIG : 9 Flask Web application

Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs.

A Deep Learning Image Segmentation Project

File name: img_00.jpeg

CHOOSE FILE



Submit

FIG 10: Uploading Image

127.0.0.1:5000/prediction

Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs.

A Deep Learning Image Segmentation Project



FIG 11 : Prediction

Mini network:

- **Model architecture and objective function:**

The UNet architecture is used to implement the small network. Encoding blocks, decoding blocks, skip connections, and a categorization layer at the bottom make up this system. The segmentation of images is the micro network's main goal.

Following is the model architecture:

Encoding Block:

An image input is subjected to a series of procedures in the encoding block. It has two convolutional layers and a ReLU activation function as well as batch normalization. Each convolutional layer has a padding of 1 and a kernel size of 3x3. The in channels and out channels parameters control the number of input and output channels.

Encoding and decoding blocks are used to create the UNet model. It consists of a bottleneck block, four decoding blocks, and four encoding blocks. Each encoding block increases the number of channels while decreasing the input image's spatial dimensions. Up sampling and concatenation are carried out by each decoding block with the output of the corresponding encoding block.

Bottleneck:

An encoding block with more output channels is the bottleneck block. It acts as the hub of the UNet and picks out the most ethereal aspects of the input image.

last Layer:

The segmentation mask is produced by the last layer, a 1x1 convolutional layer. In order to match the intended number of output classes, the number of channels is decreased.

The following hyperparameters were used to train the micro network:

1e-3 learning rate

5e-4 Weight Decay,

4 Batches

Number of Workers: 4

Number of Epochs: 30

The AdamW optimizer was employed during training with the predetermined learning rate and weight decay. The objective function was the cross-entropy loss function.

Training was done by switching between training and validation epochs. The model was trained on the training dataset and assessed on the validation dataset throughout each epoch. For both training and validation, the loss and dice score—a marker for segmentation performance—were recorded.

• Experiments and results:

The test's findings are as follows:

- Train Loss: Each epoch's average loss on the training dataset.
- The average dice score on the training dataset for each epoch is known as the train dice score.
- Valid Loss: Each epoch's average loss on the validation dataset.
- Valid Dice Score: The mean dice score after each epoch on the validation dataset.

Based on the validation dice score, the top-performing model was preserved.

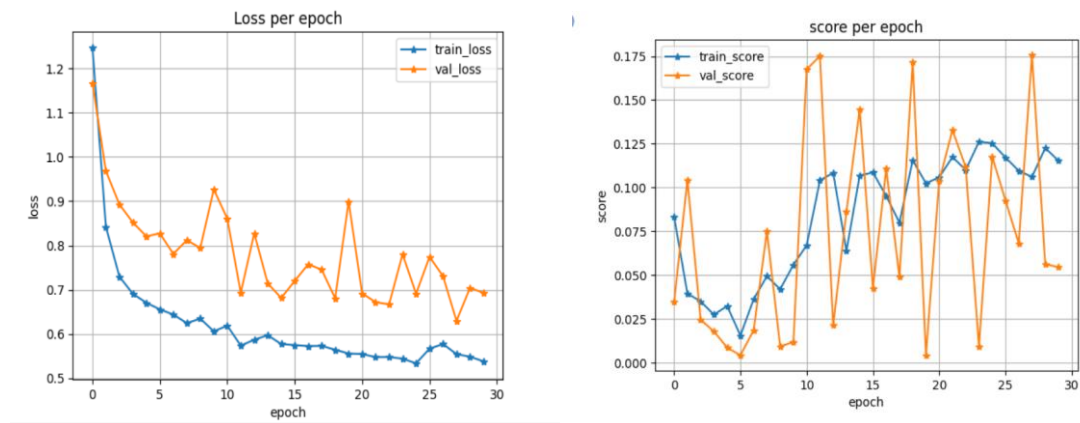


FIG 12 : Loss and Accuracy plots

Prediction:

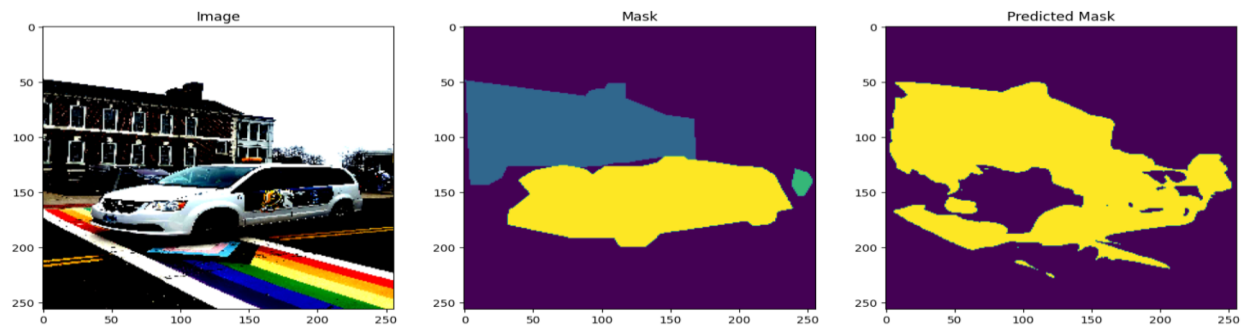


FIG 13 : Model Prediction Ouput plots

Conclusion:

In Conclusion, This study provided a thorough examination of the picture segmentation challenge utilizing a small network architecture. An summary of the dataset used for training and evaluation was provided in the dataset section. By modifying a pre-trained backbone network for better segmentation performance, the fine-tuning procedure illustrated the value of transfer learning. In addition, the micro network architecture was explained. It consists of blocks for feature extraction and up sampling, respectively, that are encoded and decoded. The trials demonstrated the value of hyperparameter selection, learning rate decay strategy, and model evaluation in producing the best segmentation outcomes. Overall, this study demonstrates the small network's capability for precisely segmenting images and opens the door for further development in image analysis and computer vision tasks.

Code and Dataset:

Our Project GitHub Repo:

https://github.com/agani3-UNH-DSCI/DL_FinalProject-Training-From-Scrath-based-on-UNET-architecture-.git

Dataset Link:

<https://drive.google.com/drive/folders/1tWDKV8xl0mWRbaTLReuJImEXfviCzrs?usp=sharing>