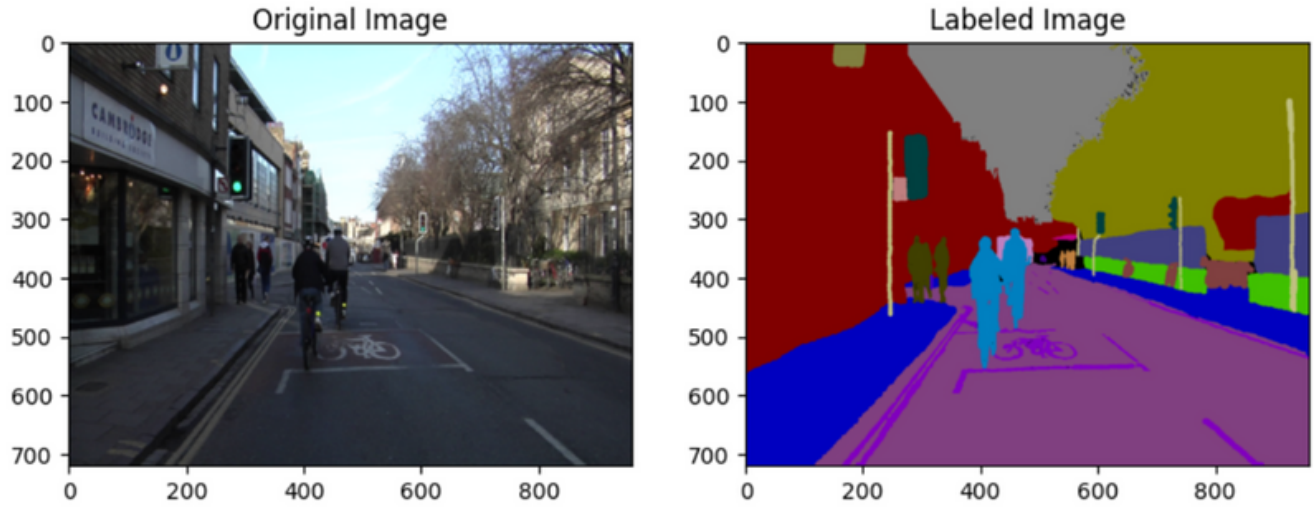
CA Assignment 3

IMPLEMENTING SEMANTIC SEGMENTATION

Introduction of the dataset

This project uses the Cambridge Labeled Objects in Video dataset, comprising 101 images (960x720 pixels). Each pixel is manually classified into one of 32 object classes relevant to driving, with an extra "void" label for ambiguous areas. The dataset includes a color-class association file (label_colors.txt) and the images are in 24-bit color PNG format. For every original frame, there's a labeled counterpart with an "_L" added to the filename.



Describe the structure of U-Net Architecture

The U-Net extends the architecture of the FCN for semantic segmentation introduced in [1]. Figure 1 illustrates the symmetric 'U' shaped architecture of the U-Net composed of a contracting path (left side) and an expansive path (right side). The former implements sequential double convolution with max pooling whereas a string of double convolution followed by transposed 2D convolution (up-sampling) occurs in the latter. Each block on the left of the 'U' shape is called an encoder and these are mainly purposed for feature extraction. The blocks on the right are called decoders and they are tasked with counteracting the dimensionality reduction performed by the encoders in order to reconstruct the image to its original size. This is necessary for segmentation as the network's output must be of the same dimension as the input image [2]. The base of the model is referred to as the bottleneck – an encoder block without the max pooling layer – and the final module of the architecture is the skip connections represented by the grey arrows. Skip connections exists between adjacent encoder-decoder pairs, and they act as information transmitters. Specifically, they pass on the location of the features extracted by encoder's convolutional layers to the decoder. The output of the encoder's final convolutional layer is concatenated with the input to the first layer of the decoder block thereby preventing information loss and enhancing the network's performance [2].
In this project, the implementation of the U-Net is nearly identical to the architecture presented in [3], with only two modifications. The contracting path consists of four double padded convolutions (padding) with batch normalization, each followed by a rectified linear unit (ReLU) activation function and a max pooling operation with stride . The original architecture in [3] uses unpadded convolutions (padding) without batch normalization – these are the only modifications made. The image was padded to preserve its dimensions during the convolution process, ensuring that output has the same height and width as the input. This simplifies the network as it eliminates the need to crop the activation map from the skip connection before concatenating in the decoder block. The number of input channels for the first encoder is (RGB images) and the number of output channels is . From the second to the fourth encoder, the number of input and output channels are defined according to (1) and (2) where is the encoder number.

$$In\_channels_i = Out\_channels_{i-1} \quad (1)$$

$$Out\_channels_i = In\_channels_i * 2 \quad (2)$$

$$Out\_channels_i = In\_channels_i \div 2 \quad (3)$$

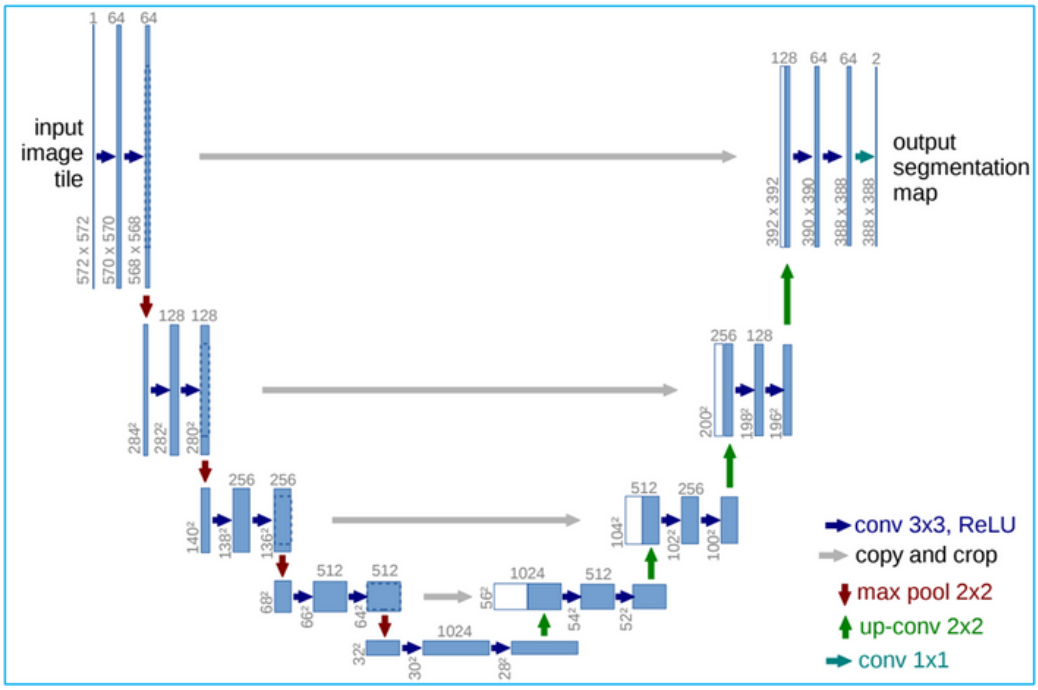


Figure 1 [3] : U-Net architecture

The expansive path comprises four decoder blocks, each performing a deconvolution step using a kernel with stride and no padding. The output is concatenated with the activation map from the skip connection and the result is passed through two convolution layers. The number of input and output channels from the second to the fourth decoder are defined by (l) and (3) is the decoder number. For the first decoder, the number of input channels is equal to the number of output channels of the bottleneck block. The final decoder block has an additional convolutional layer at the end for mapping each - component feature vector to the classes.

Describe the structure of DeepLab Architecture

DeepLabv3+ has an encoder-decoder structure, similar to the U-Net, but only has one of each. It inherits the architecture of its predecessor, DeebLabv3, to form the encoder block as shown in Figure 2. It has two essential components, namely Atrous (dilated) Convolution and Atrous Spatial Pyramid Pooling (ASPP). The former controls the DCNN's feature computation resolution and adjusts the filter's field of view, enabling it to capture long range information. Atrous convolution can increase the filter's field of view by dilating the kernel (inserting holes between filter weights) without facing a computation penalty (no increase in the number of parameters), making it more efficient than using regular convolutions with large filters [4].

The purpose of ASPP is to resample a feature layer at multiple rates. In reference to Figure 2, this is done by probing the original image with four filters that have complementary effective fields of view to capture information at multiple scales. One and three convolutions are used to achieve this. For the kernels, the dilation rates are , , and respectively and each convolutional layer has output channels. The resulting activation maps are concatenated along with the output from the parallel image pooling layer, and this is passed through another convolutional layer to complete the encoder block [4, 5].

DeepLabv3+ was implemented in this project by loading an ImageNet pretrained model with a ResNet- DCNN backbone. The architecture of this model is identical to the one in Figure 2. The model was imported using the Segmentation Models PyTorch module with input channels (RGB), classes, and a SoftMax activation function at the final layer to convert the logits to probability scores. The decoder was then trained on the 'Cambridge Labelled Objects in Video' dataset.

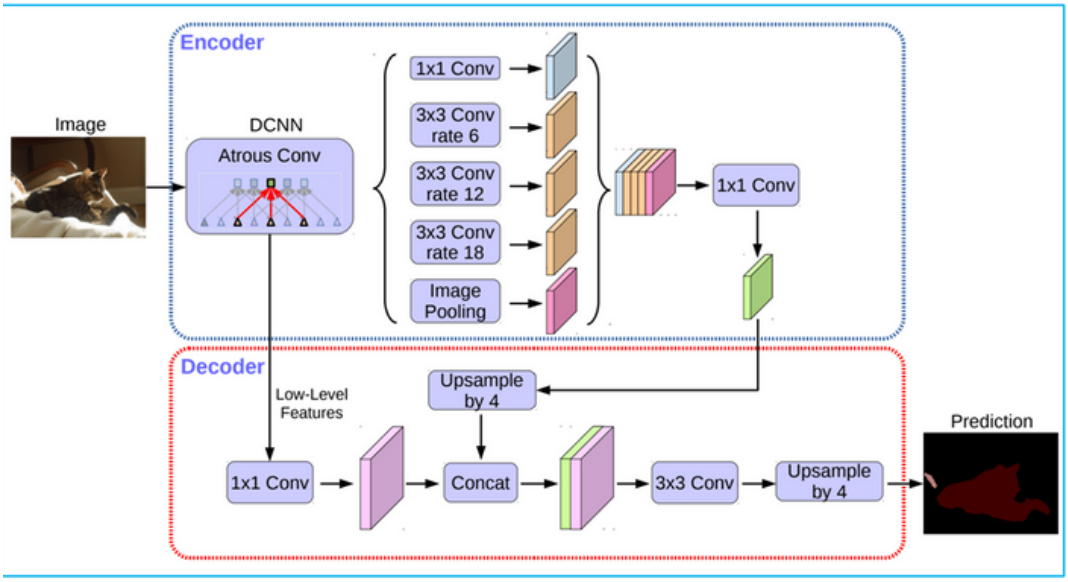


Figure 2 [4] : DeepLabv3+ architecture

Describe the structure of Fully Convolutional Networks (FCN) Architecture

The Fully Convolutional Network (FCN) is designed for semantic image segmentation tasks and comprises two main components:
Base Layers: The base layers serve as feature extractors. They consist of a series of convolutional layers, each followed by a ReLU activation and Max Pooling. They transform the input image while preserving spatial information.
Classifier Layers: The classifier is a sequence of convolutional layers, ReLU activation, and Dropout layers. The final convolutional layer maps the extracted features to the number of desired classes.
Interpolation: The output of the classifier is resized to the input image's original dimensions using bilinear interpolation to provide a pixel-wise class prediction.
The FCN model thus allows end-to-end learning for semantic segmentation, directly mapping input images to pixel-wise class predictions.

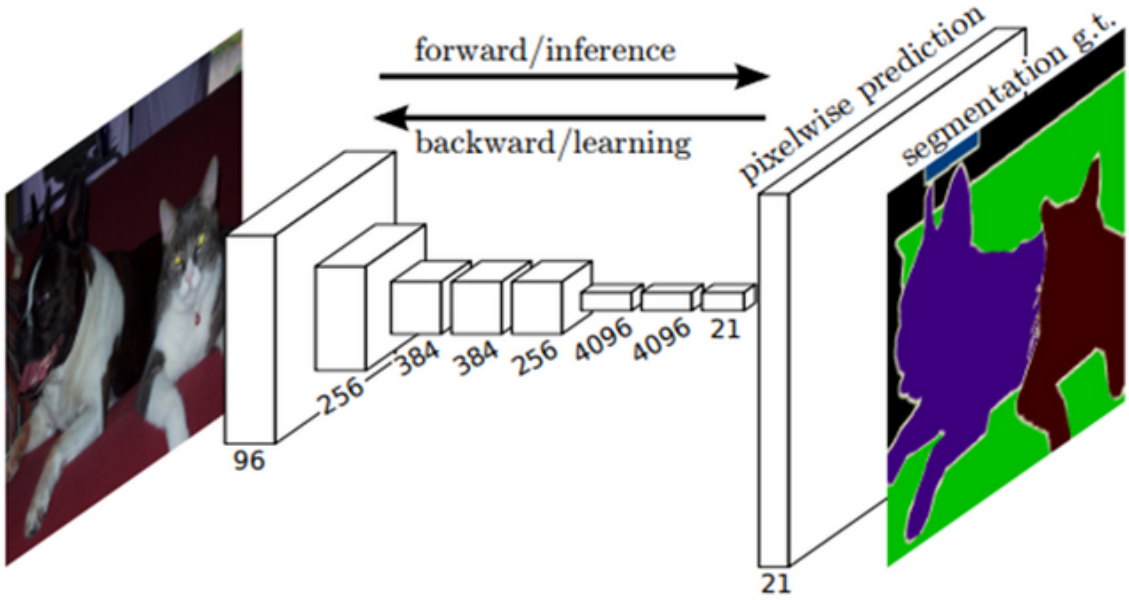


Figure 3 [1] : FCN architecture

Model Implementation and Evaluation Strategy

Our image segmentation model is implemented using a systematic approach that encompasses model training, testing, and hyperparameter optimization:

1. **Model Training:** The training process is conducted using the train function. This function facilitates the forward and backward propagation stages. It uses the provided model, dataloader, optimizer, and loss function to iteratively train the model, compute the loss, and update the model parameters accordingly. Additionally, performance metrics, such as Intersection over Union (IoU) and pixel accuracy, are calculated for each batch to track the model's learning progress.
2. **Model Testing:** The test function assesses the trained model's performance on unseen data. This function loads the pre-trained model and computes average loss, mean IoU, and average accuracy metrics over all batches in the testing phase, providing an unbiased estimation of the model's performance.
3. **Model Training Procedure:** The train_model function orchestrates the entire training process. It establishes the optimizer and loss function, sets the training duration (in epochs), and commences the training. Throughout the process, it records the training and validation metrics. If the model demonstrates improved performance on the validation data (in terms of IoU and accuracy), the model's current state is saved for future use. The training progress is visualized using various plots, enabling easy identification of learning trends and potential overfitting or underfitting scenarios.
4. **Hyperparameter Optimization:** Seven different hyperparameter combinations were explored for each network. Due to computational limitations, only 5 epochs and 30 images (27 – training, 3 – validation) were used to run training cycles with each combination of hyperparameters to identify the best-performing configuration on the validation data.
5. **Performance Visualization:** The plot_training_validation_metrics function is employed to graphically represent the model's training and validation performance over epochs. The visualized metrics include loss, IoU, and accuracy. This graphical representation provides a more intuitive understanding of the model's learning progress and performance.

Model Design Rationale

Kinds of hyperparameters or techniques do you choose to tune and explore: The model implements several strategies for tuning and exploration:

- **Learning Rate:** Different rates (1e-1, 1e-2, 1e-3) are tested to optimize the model's learning speed.
- **Optimizers:** SGD, Adam, and RMSprop are used to adjust the model parameters based on the computed error.
- **Loss Functions:** Dice Loss, MSELoss, and Cross Entropy Loss are examined to determine how to measure the model's prediction error.
- **Epochs:** The total number of iterations over the entire dataset.
- **Combinatorial Search:** Employed for hyperparameter tuning. A base model is defined and each hyperparameter/technique is tuned sequentially, preserving the most effective settings from previous explorations. The combinations are listed in Table 1 and results of this step are shown in Table 2.
- **Early Stopping:** The model's best state is saved when an improvement in validation IoU and accuracy is observed, thereby preserving the best performing model.
- **Metrics:** Loss, IoU, and pixel accuracy are used to measure the model's learning progress and segmentation effectiveness.

Combination	Optimizer	Loss Function	Learning Rate
1	Adam	Jaccard Loss	1e-2
2	SGD	Jaccard Loss	1e-2
3	RMSprop	Jaccard Loss	1e-2
4	Best Optimizer	Dice Loss	1e-2
5	Best Optimizer	Lovasz Loss	1e-2
6	Best Optimizer	Best Loss Function	1e-3
7	Best Optimizer	Best Loss Function	1e-4

Table 1: Combinatorial hyperparameter/technique search space

Impact of Techniques on Model Performance The chosen techniques significantly influence the model's performance. For instance, different learning rates can speed up training or provide more precise convergence at the expense of computational time. The optimizer selection can either hasten convergence or escape poor local minima; for example, Adam, with its adaptive learning rate, often results in faster and more stable convergence. The choice of loss function directly affects how the model perceives its performance and makes adjustments. A poorly chosen loss function can result in slower learning or even misleading training. Combinatorial search allows for an efficient exploration of the hyperparameter space and, although it doesn't guarantee the absolute best solution, it often finds a reasonably good set of hyperparameters in a shorter time than exhaustive search methods. The use of early stopping prevents overfitting by halting training when validation performance stops improving, hence ensuring the model generalizes well. Therefore, these techniques play a crucial role in boosting the model's performance and ensuring robust and effective learning.

Combination	U-Net		DeepLabv3+		FCN	
	Loss	IoU	Loss	IoU	Loss	IoU
1	0.387	0.240	0.387	0.062	0.381	0.059
2	0.403	0.000	0.404	0.000	0.405	0.002
3	0.370	0.251	0.358	0.335	0.381	0.155
4	0.379	0.046	0.345	0.338	0.377	0.232
5	1.032	0.000	1.712	0.000	1.173	0.000
6	0.392	0.295	0.322	0.745	0.351	0.182
7	0.397	0.195	0.339	0.793	0.278	0.296
Optimal	RMSProp, Jaccard Loss, 1e-3		RMSProp, Jaccard Loss, 1e-3		RMSProp, Dice Loss, 1e-4	

Table 2: Combinatorial search results for each network

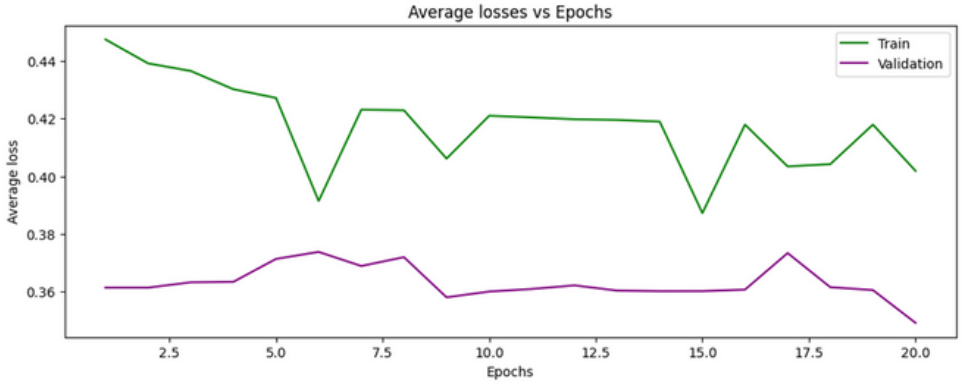
Addressing Overfitting and Underfitting Issues Addressing overfitting and underfitting, two common challenges in model training, involved several strategies. Overfitting, where the model excels on training data but poorly generalizes to new data, was mitigated using early stopping to prevent learning from noise in training data and by ensuring an appropriate model complexity. Underfitting, where the model fails to capture data patterns, was addressed by adjusting the model's architecture for increased complexity and experimenting with various activation functions and optimizers to improve learning capability. Lastly, a combinatorial search of hyperparameters helped in fine-tuning the learning process, balancing overfitting and underfitting to achieve robust model performance across both training and validation sets.

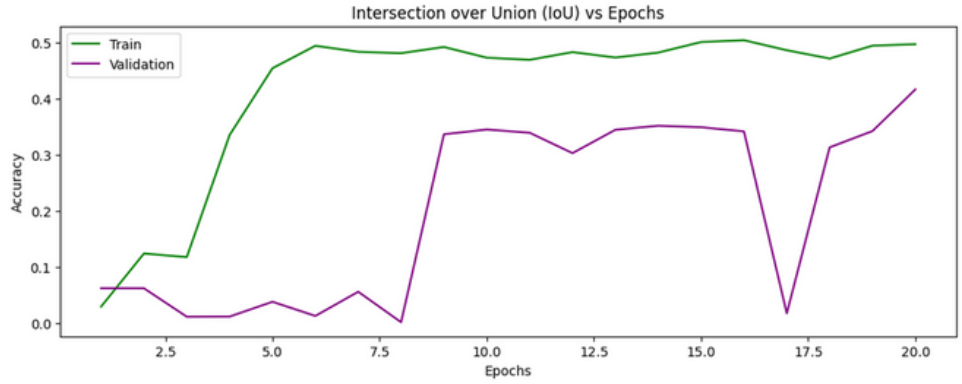
Model Performance Analysis UNet

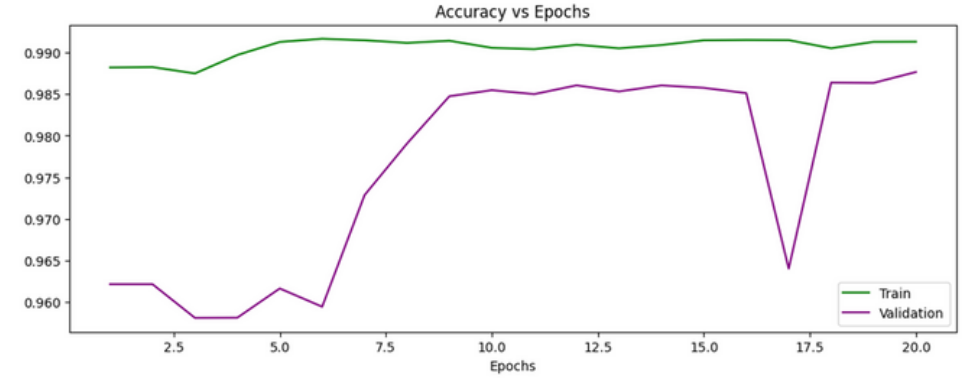
After finding the optimal configuration, the model was then trained and validated for 20 epochs. It demonstrated a consistent decrease in training loss from 0.447 to 0.402, and the validation loss dropped from 0.361 to 0.349 which indicates that the model was learning and improving its predictions on both data sets.

Similarly, the Intersection over Union (IoU), a measure of the overlap between the predicted and actual areas, increased from 0.030 to 0.497 on the training data, and from 0.063 to 0.417 on the validation data, which illustrates the model's improving ability to correctly identify areas of interest. The pixel accuracy was very high to start with and remained so throughout the training process. Peak accuracies of 0.991 and 0.988 for training and validation respectively were recorded.

The occasional divergence between training and validation metrics hints at potential overfitting, where the model learns the training data too well and performs less optimally on new data. The spikes in the validation IoU and accuracy indicate some level of instability in the model weight updates. To counter both issues, the model weights that achieved the best performance metrics on the validation data were saved and loaded for the testing phase





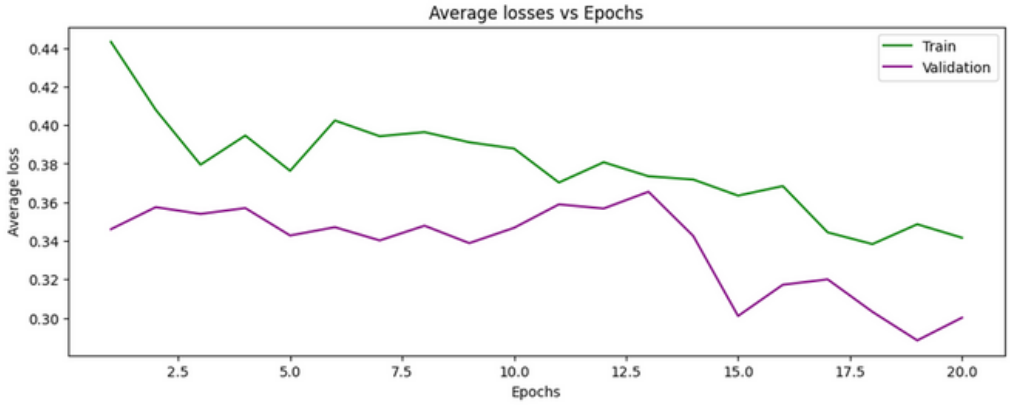


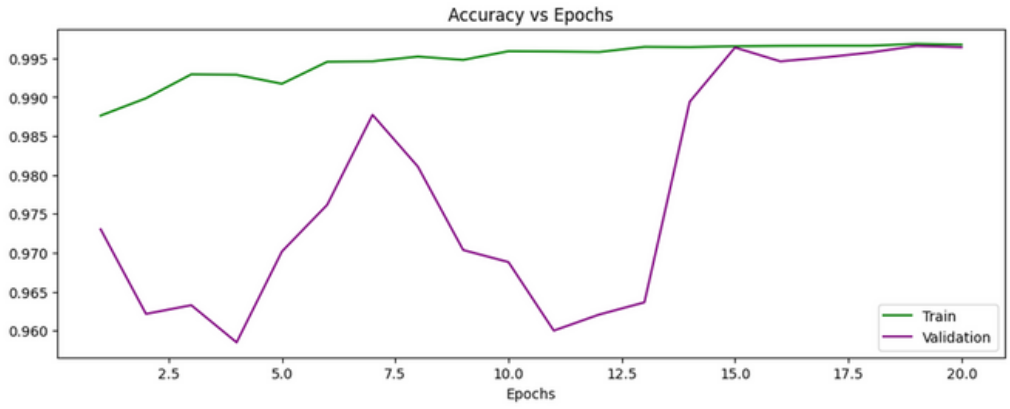
The training metrics demonstrate a clear improvement trend, indicating the model's ability to learn from the training data. However, the validation metrics show more variability and potential signs of overfitting. One learning outcome is to perform an early stopping, thus reducing number of epoch to 10.

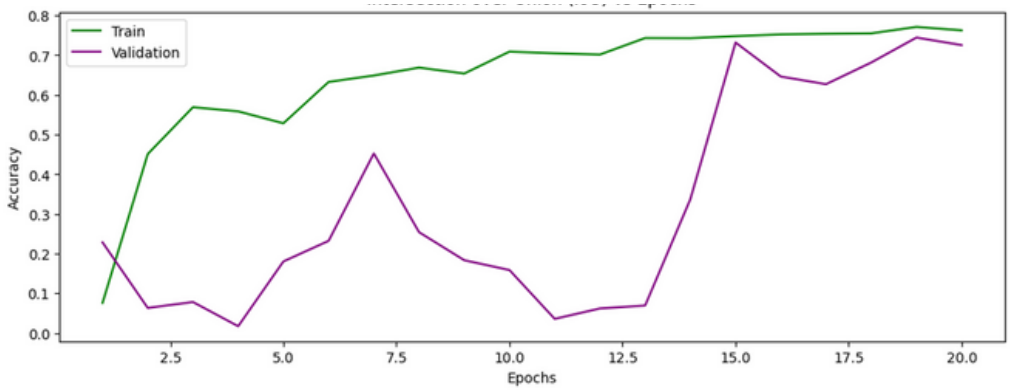
Model Performance Analysis DeepLab

DeepLabv3+ demonstrated a better performance than the U-Net but with increased instability as depicted by the oscillatory graphs. The training loss started off at 0.443 and dropped to 0.342 whilst the validation loss decreased from 0.346 to 0.300. The IoU and accuracy graphs were quite similar and showed major inconsistencies regarding the model's performance on the validation data. The model is potentially suffering from a high learning rate and a scheduler could be used to regulate this hyperparameter appropriately.

The IoU of the model improved from 0.076 to 0.763 for the training data, and from 0.228 to 0.725 on the validation data. After 20 epochs the training and validation IoU scores converged towards each other, and the same happened for the accuracy values which remained high throughout, and both peaked at 0.996 at epoch 20.



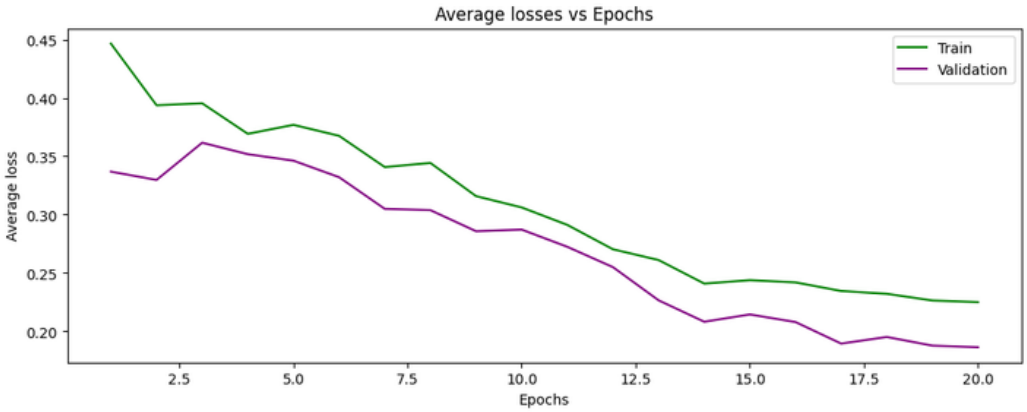


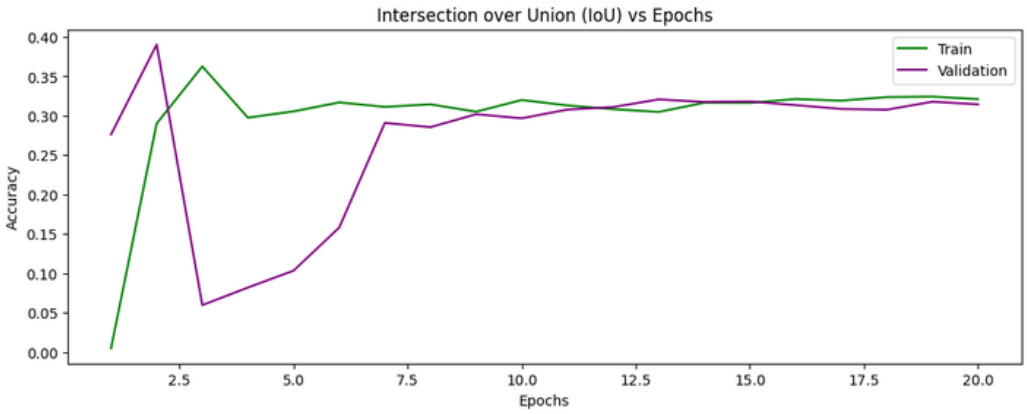


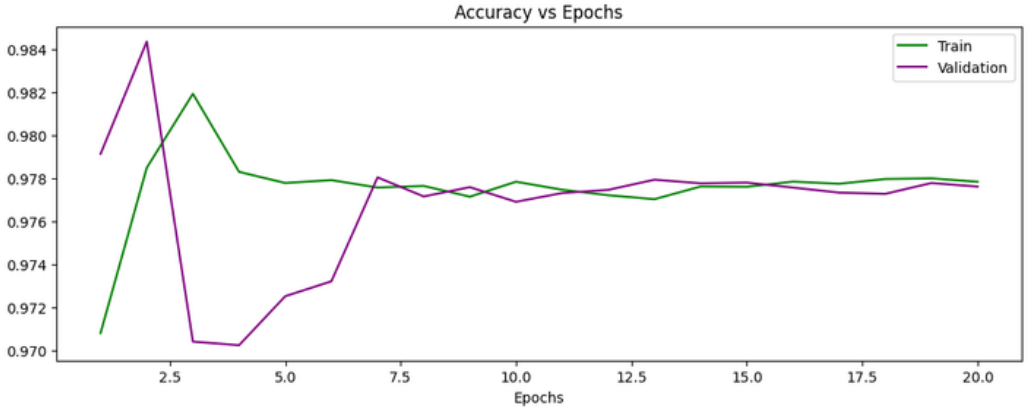
Model Performance Analysis FCN

The FCN was the most stable model as it demonstrated smoother and more gradual improvements for all three metrics over 20 epochs, as opposed to the U-Net and DeepLabv3+. The main reason for this is the learning rate which was less by a factor of 10 for the FCN (1e-4) compared to the other two models (1e-3). The training loss decreased from 0.447 to 0.225, and the validation loss from 0.337 to 0.186.

The FCN recorded lower loss values than the other two models, but its IoU values were third best – the best IoU on the training and validation sets were around 0.380 and 0.360 respectively. After 20 epochs, the training IoU was 0.321 and the validation IoU was 0.314. Like the U-Net and DeepLabv3+, the accuracies were very high and plateaued at 0.978 which was also third best.







Comparing the Performance of the Perfomance of the Three Models on Test Data

***Analysis of the performance of UNet, DeepLab and FCN based on my own AI knowledge*:** For the training phase, we utilized GPU. The UNet algorithm took the longest time to train with a total of 5 minutes and 2 seconds. DeepLab, on the other hand, required only 3 minutes and 15 seconds, making it faster than UNet. FCN was the fastest among the three, taking just 2 minutes and 49 seconds for training. This shows that FCN is the most efficient algorithm in terms of training time, followed by DeepLab and then UNet.
In the testing phase, which was performed on a CPU, UNet again took the longest time, requiring 1 minute and 10 seconds to complete. DeepLab and FCN, on the other hand, were significantly faster, taking 18 seconds and 14 seconds, respectively. This suggests that FCN and DeepLab are also more efficient than UNet in the testing phase, with FCN being the quickest.

Training Time	UNet	DeepLab	FCN
Model training with best Hyperparameter (with GPU)	5 mins 2 secs	3 mins 15 secs	2 mins 49 secs
Testing (with CPU)	1 min 10 secs	0 mins 18 secs	0 mins 14 secs

Comparing the training and testing times for these three algorithms, it can be observed that FCN performs the best in terms of speed, both for training and testing. DeepLab comes in second, while UNet is the slowest of the three. However, it's important to remember that speed isn't the only factor to consider in algorithm choice; accuracy and applicability to the specific task at hand are also crucial considerations.

Conclusion

***Reflection and Project Next Step*:** This project implemented the U-Net, DeepLabv3+, and FCN for semantic segmentation of images related to driving environments. The dataset, which consisted on 90 training images and 11 testing images, was imported from a google drive folder into a python environment. The training set was augmented to increase its volume before being further split into a train set (90%) and a validation set (10%). Due to limited computational resources, only 18 training images were used (test set - 16, validation set - 2). All three datasets were passed into a custom dataset class for image transformation prior to being loaded into PyTorch dataloaders. After pre-processing the data, the three networks were created (the U-Net was built from scratch whilst the DeepLabv3+ and FCN were loaded from the Segmentation Models library, each with a pretrained backbone on the ImageNet dataset). A combinatorial search space of hyperparameters/techniques was defined to find the best configuration for each model. Furthermore, the models were trained and validated, and the results were plotted to draw comparisons between them. The FCN demonstrated the most stability but had the least optimal performance whereas the DeepLabv3+ was the least stable network but had the best performance on both the training and testing data.

This project has introduced us to a new field of computer vision which was not familiar to us. We have realized the importance of semantic segmentation in real world applications and also how computationally intensive and challenging it is to pre-process images and train segmentation networks. The project was a steep learning curve for us, particularly given that it deals with multiclass segmentation rather than binary segmentation, which made the mask transformations more complex. Upon reflection, the fundamentals that we have learnt by completing this project will prove invaluable in the near future when we start to take on real-world projects.

Reference

1. Jonathan Long, Evan Shelhamer, Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation [Internet]. Boston: Massachusetts; CVPR; Mar 2015; [cited 10th May 2023]. Available from: [1411.4038] Fully Convolutional Networks for Semantic Segmentation (arxiv.org)
2. Conor O'Sullivan. U-Net Explained: Understanding its Image Segmentation Architecture [Internet]. In: Towards Data Science. Medium; Mar 2023; [cited 10th May 2023]. Available from: U-Net Explained: Understanding its Image Segmentation Architecture | by Conor O'Sullivan | Towards Data Science
3. Olaf Ronneberger, Philipp Fischer, Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation [Internet]. Munich: Germany; MICCAI; May 2015; [cited 10th May 2023]. Available from: [1505.04597] U-Net: Convolutional Networks for Biomedical Image Segmentation (arxiv.org)
4. Vaibhav Singh. The Ultimate Guide to DeepLabv3 – With PyTorch Inference [Internet]. LearnOpenCV; Dec 2022; [cited 12th May 2023]. Available from: The Ultimate Guide to DeepLabv3 - With PyTorch Inference (learnopencv.com)
5. Liang-Chieh Chen, et al. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs [Internet]. TPAMI; May 2017; [cited 12th May 2023]. Available from: 1606.00915v2.pdf (arxiv.org)