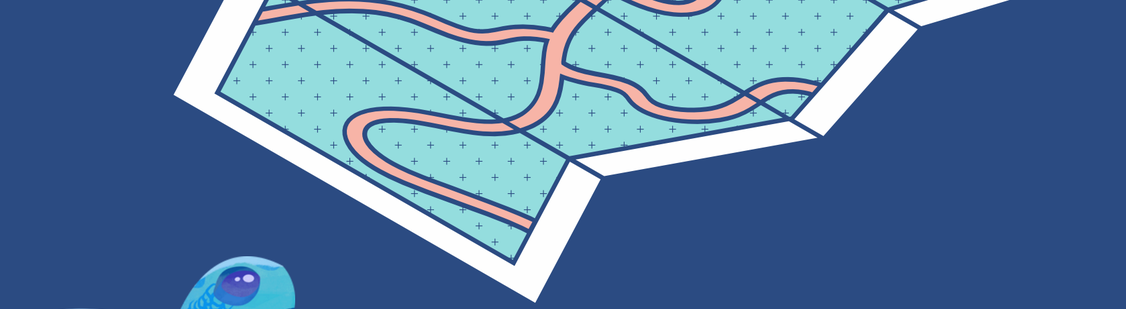




A PRESENTATION OF

Semantic Segmentation of Underwater Imagery

Michele PULVIRENTI, Marco RIVA,
Alessia SARRITZU, Alberto MARTINELLI

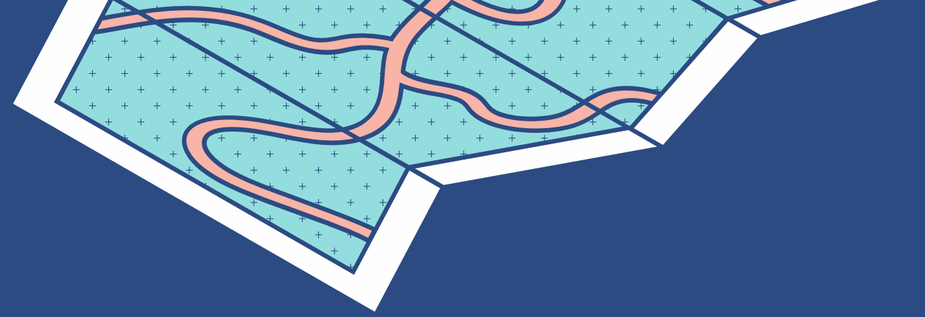
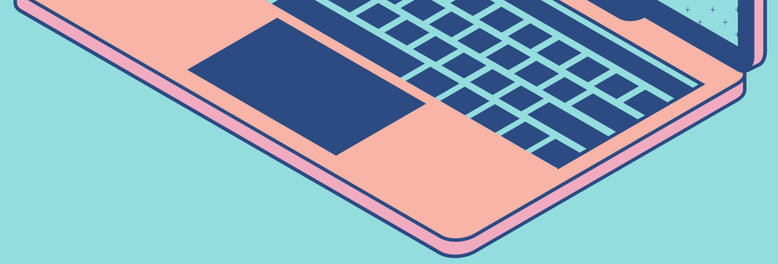
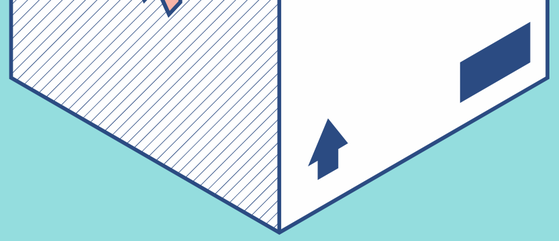


Illustration of the Problem

What is Semantic Segmentation?

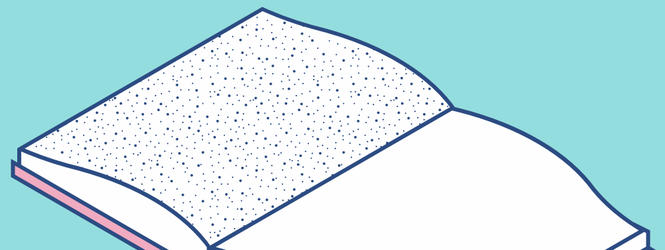
- Semantic segmentation is a computer vision task where each pixel of an image is labeled with a class or category, essentially dividing the image into meaningful sections
- In our project, the goal is to create a model that can segment underwater images into different objects like fish, reefs, plants, and more
- The model will be trained on a set of underwater images and will be able to automatically label each pixel according to the object it belongs to



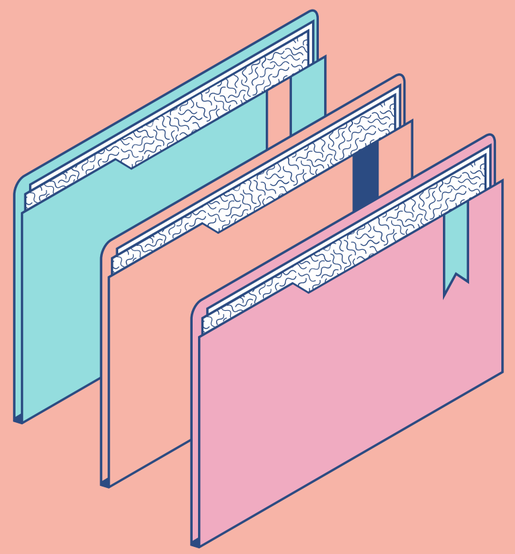


What We Aim to Achieve

- Train a **segmentation model** capable of accurately identifying and labeling underwater objects
- Produce **segmentation masks** where each pixel color corresponds to a specific class like fish, wrecks, or aquatic plants



Dataset and folder structure



THE DATASET CONSISTS OF IMAGES AND THEIR CORRESPONDING SEGMENTATION MASKS, ORGANIZED INTO TWO MAIN DIRECTORIES: TRAIN_VAL/ AND TEST/

train_val/ contains 1525 paired samples for training and validation

- **images/**: RGB images of underwater scenes
- **masks/**: Ground truth segmentation labels, where each RGB color represents a different object category

TEST/ contains 110 paired samples for benchmarking the model's performance

- **images/**: RGB test images
- **masks/**: Ground truth segmentation labels
- Combined RGB masks and separate binary masks for each object class are provided

Object Categories and RGB Codes

- THE DATASET INCLUDES EIGHT OBJECT CATEGORIES THAT REPRESENT DIFFERENT UNDERWATER ENTITIES
- EACH OBJECT CATEGORY IS LABELED WITH A UNIQUE RGB CODE, MAKING IT EASIER FOR THE MODEL TO RECOGNIZE AND SEGMENT THEM

Categories	RGB Codes
Fish	Yellow: 110
Reefs	Pink: 101
Aquatic Plants	Green: 010
Wrecks/Ruins	Sky: 011
Human Divers	Blue: 001
Robots	Red: 100
Sea-floor	White: 111
Background	Black: 000

Example

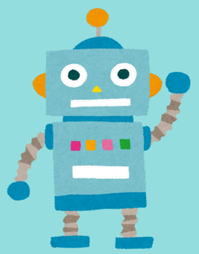
Applications of the Trained Model

1. **Marine Ecology Research**: Tracking and classifying underwater species such as fish and coral reefs.

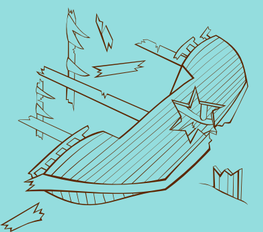
2. **Pollution Detection**: Identifying and mapping underwater debris, pollution, and its environmental impact

3. **Environmental Monitoring**: Assessing underwater vegetation health, sea-floor conditions, and environmental changes

4. **Autonomous Underwater Vehicles (AUVs)**: Enhancing object detection for underwater navigation and exploration

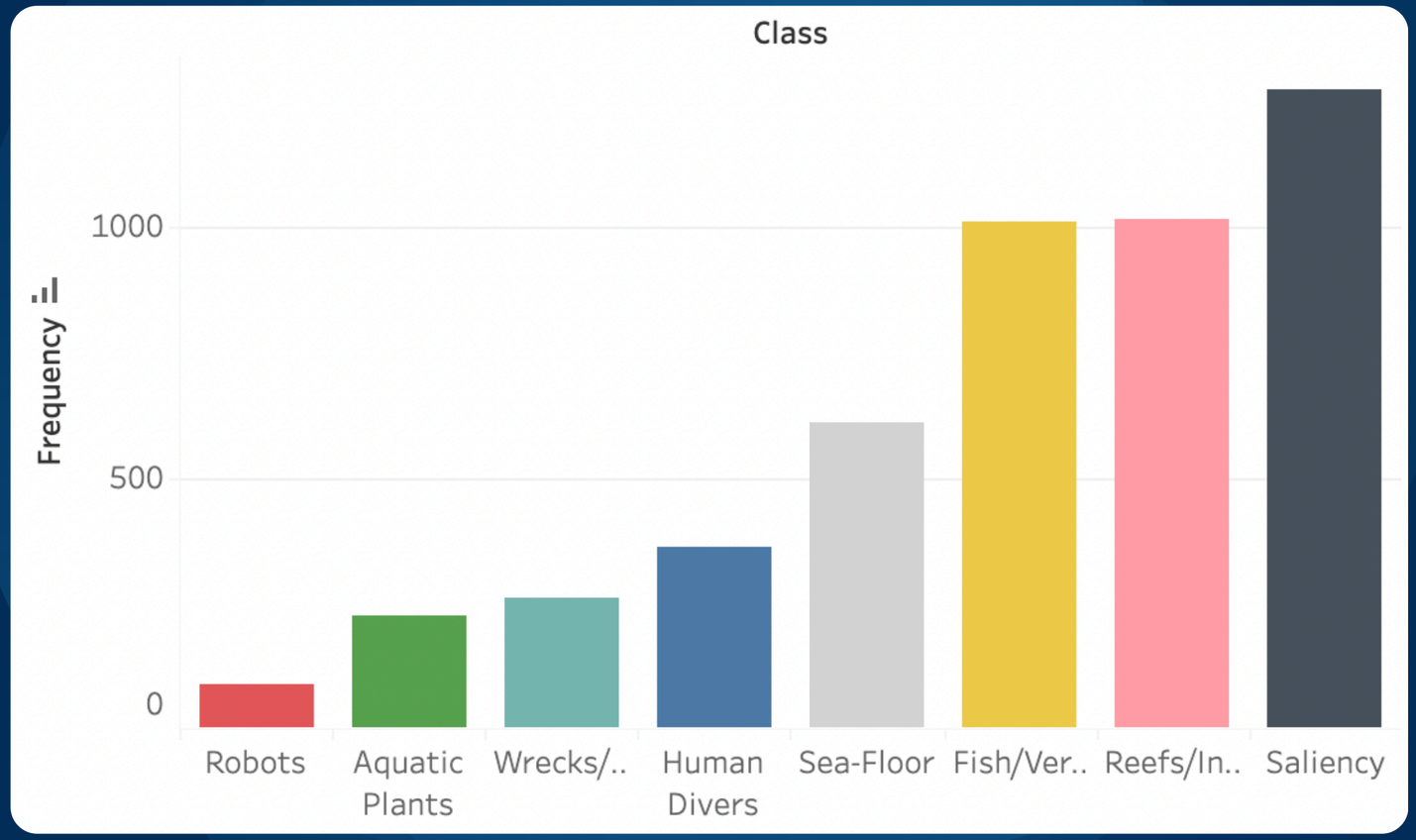


5. **Marine Archaeology**: Identifying and cataloging shipwrecks, ruins, and other submerged historical sites



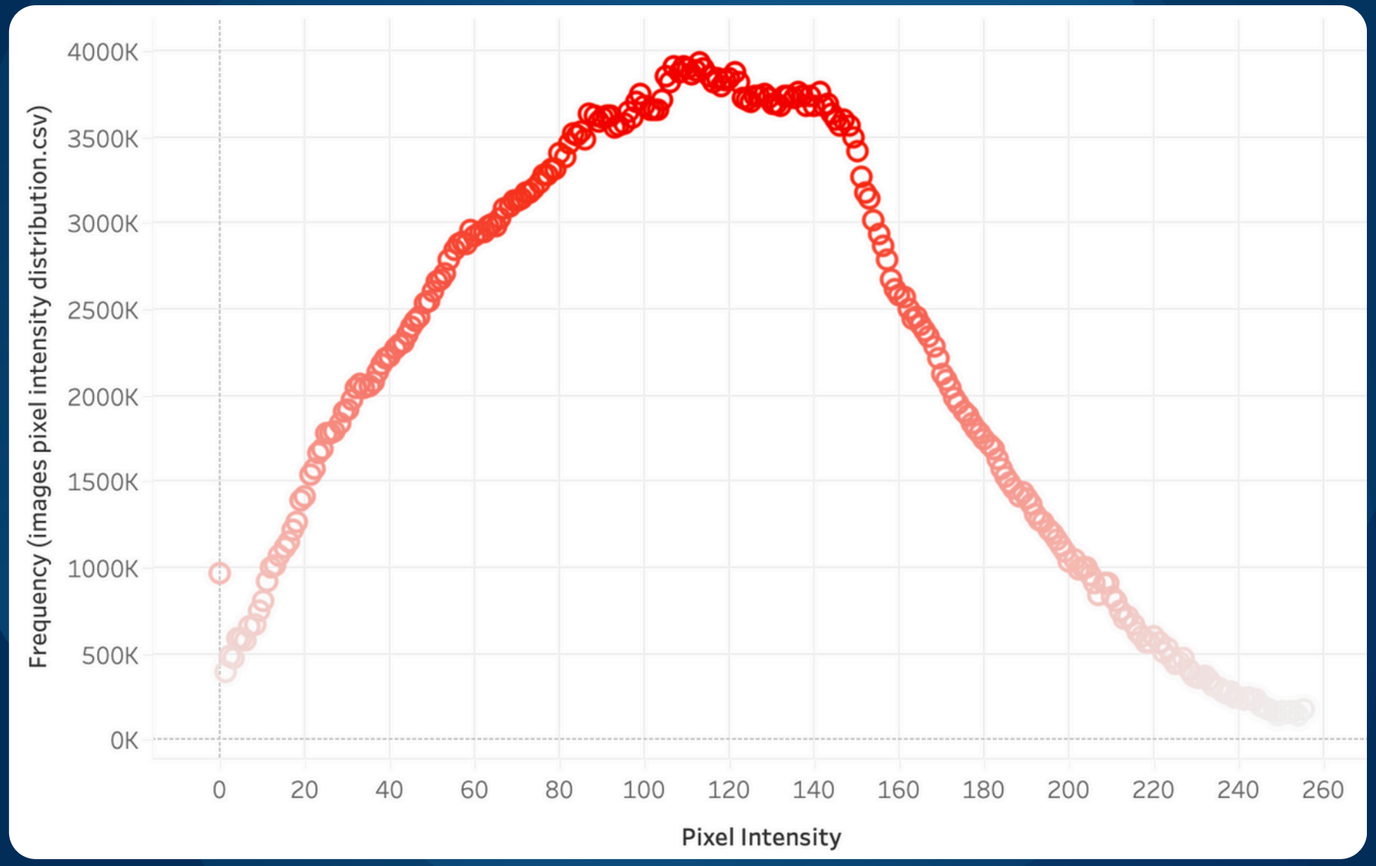
Data Visualization

Class frequency



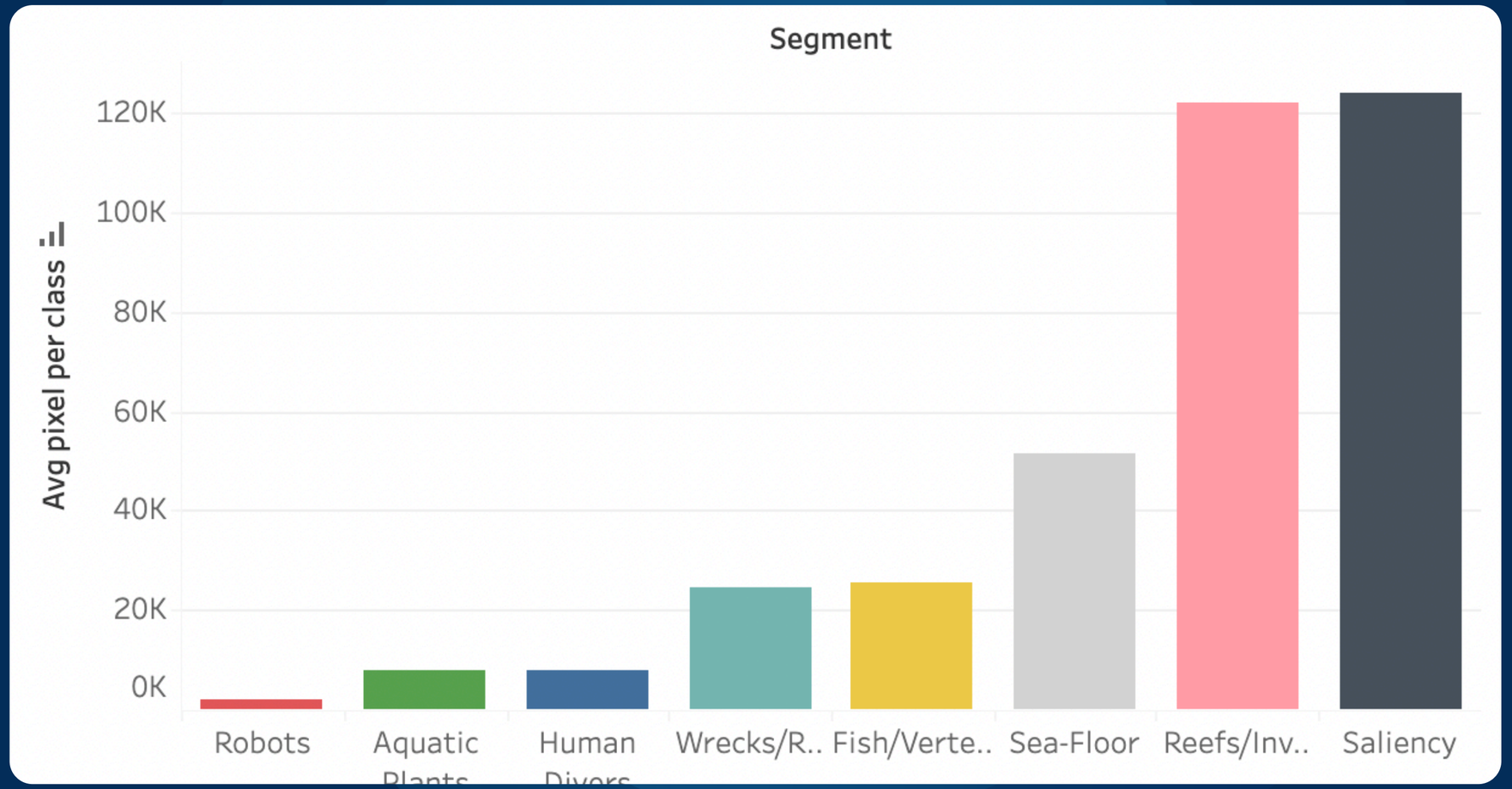
Fish/vertebrates, reefs and background (saliency) seem to be the most common categories across the images

Distribution of Pixel Intensity



This distribution tells us about the lighting conditions in the underwater environment and how pixel values are spread across different objects

Distribution of average pixels by segment (across all images)

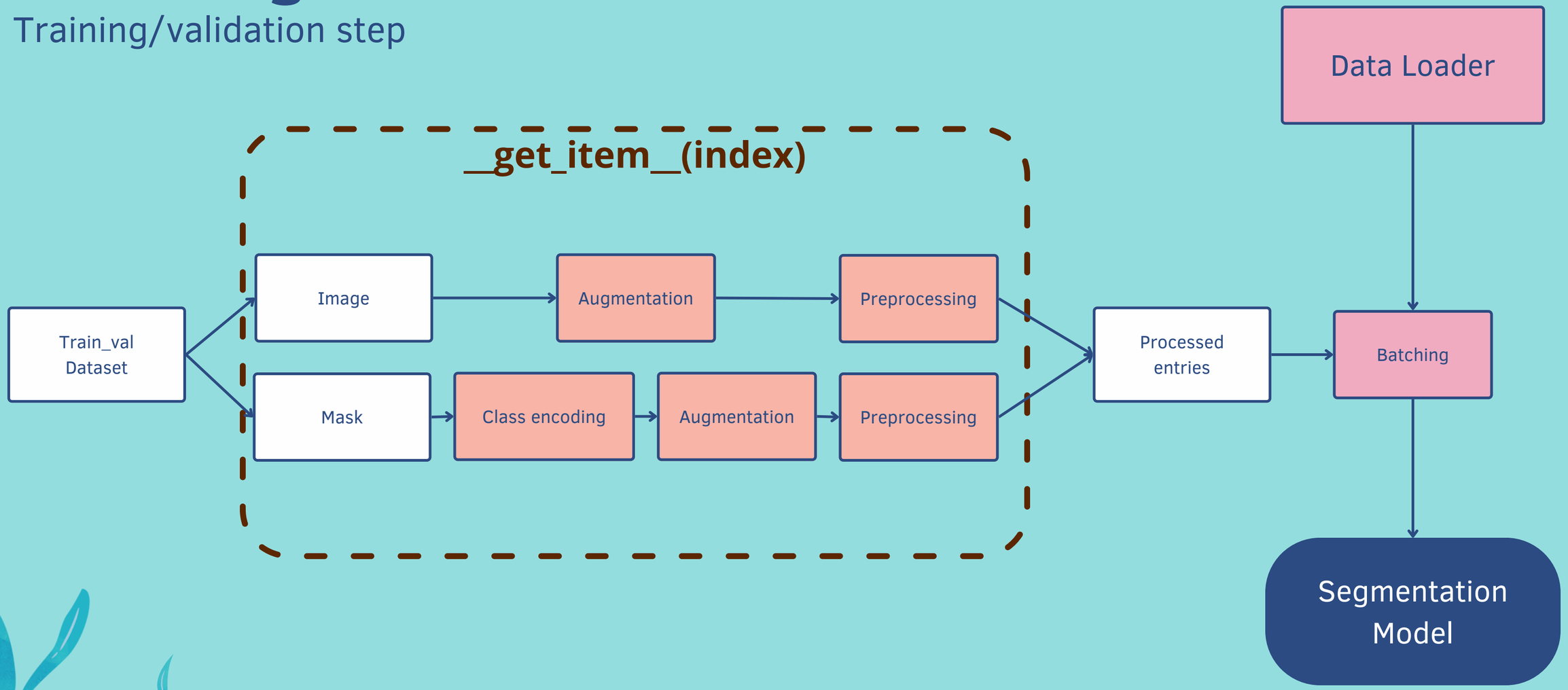


Link to the tableau dashboard

Methodology

How images are fed into the model

Training/validation step



Techniques applied

AUGMENTATION

Augmentation techniques are applied to provide diversity to the training data, helping the model generalize better.

This includes horizontal flipping, shifting, rotation, Gaussian noise, perspective changes, and color/brightness adjustments.

PREPROCESSING

Images are passed through a series of transformations that first apply data normalization and then convert the images and masks to PyTorch-friendly formats.

This ensures the model works with standard input shapes.

CLASS ENCODING

Only masks are transformed into a class-index format based on the color of each pixel.

Then, it is converted into a one-hot encoded array. This step is crucial for transforming the mask in an optimal format for the model

Augmentation examples

Original Image

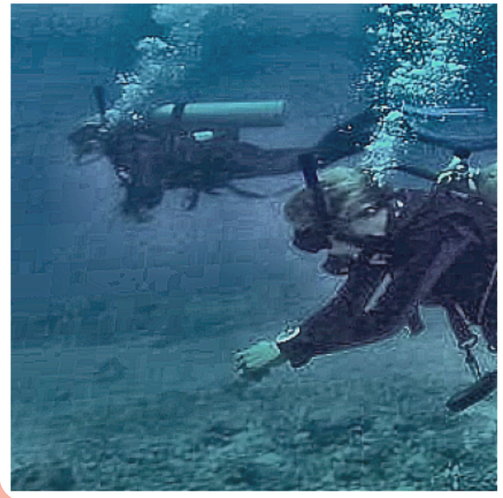
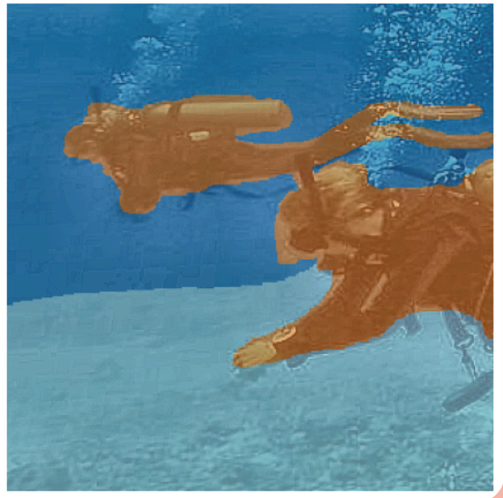


Image with Segmentation Overlay



Original Image

Image with Segmentation Overlay

Original Image

Image with Segmentation Overlay

Original Image

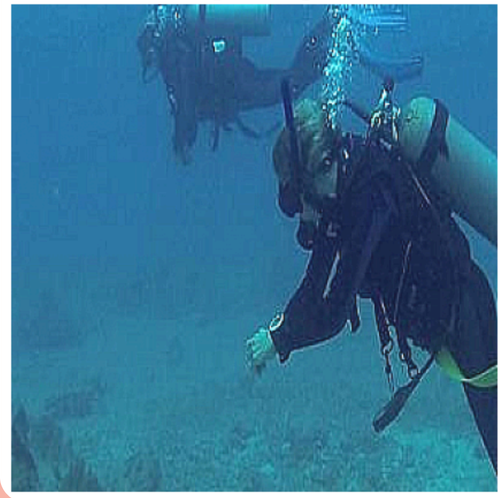
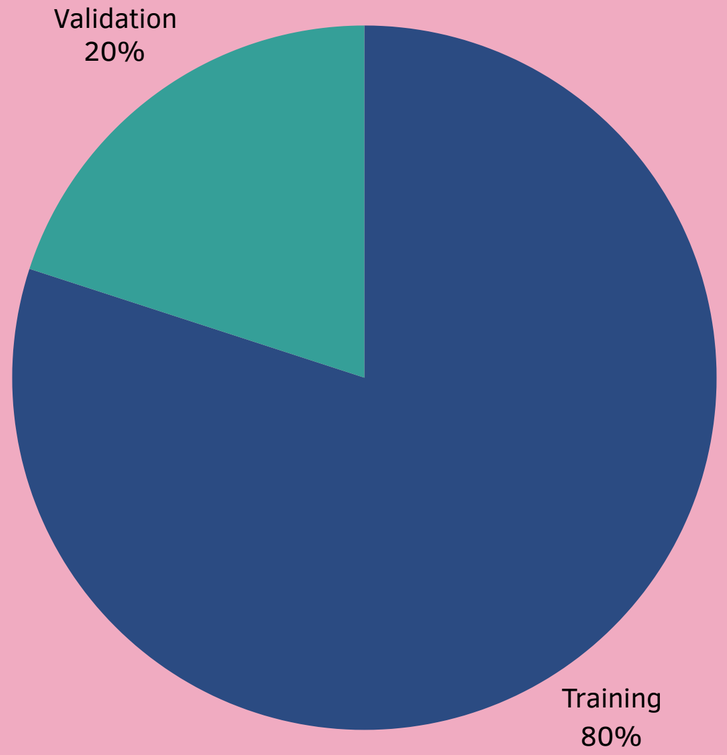


Image with Segmentation Overlay

Data splitting and loading

The train_val dataset is split into train and validation before being used.
This is useful to check at every epoch how the model performs with unseen data

Furthermore, data is given to the model using a DataLoader.
It loads a defined number of entries (batch size = 8) at once to feed into the model during training or evaluation.
It also randomly shuffles the training data after each epoch to avoid patterns and overfitting.

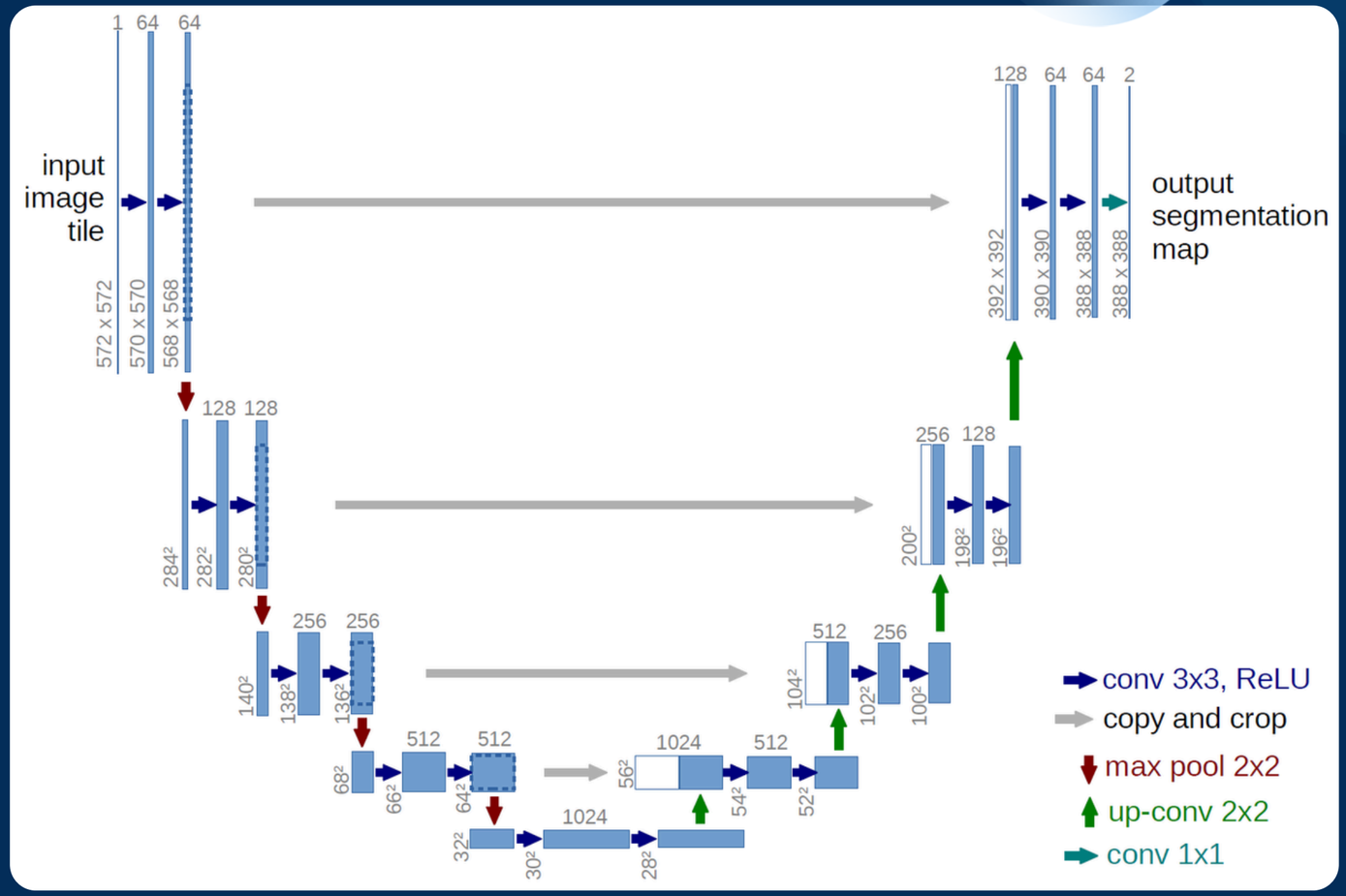


Unet model

U-Net is a fully convolutional neural network architecture designed for semantic image segmentation.

It consists of two main parts:
1. An encoder (downsampling path) that extracts increasingly abstract features
2. A decoder (upsampling path) that gradually recovers spatial details

A key step is the use of skip connections between the corresponding encoder and decoder layers. These connections allow the encoder to transfer spatial information that may be lost during the downsampling process back to the upsampling process, ensuring better reconstruction of fine details in the output segmentation.



Why these steps?

Downsampling (Contracting Path)

- To capture contextual information and identify high-level features such as edges, shapes, and textures in the image.
- Each convolution layer extracts increasingly abstract features as the image is downsampled (spatial dimensions decrease, feature channels increase).
- Max-pooling reduces spatial resolution, focusing on the most important features while discarding redundant information.

Upsampling (Expanding Path)

- To restore spatial resolution and generate a segmentation map matching the size of the input image.
- Up-convolutions (upsampling) increase spatial dimensions.
- Skip connections combine high-resolution features from downsampling to help localize features better.

Our implementation

We implemented the UNet model, introducing a dropout mechanism to enhance regularization and mitigate overfitting.

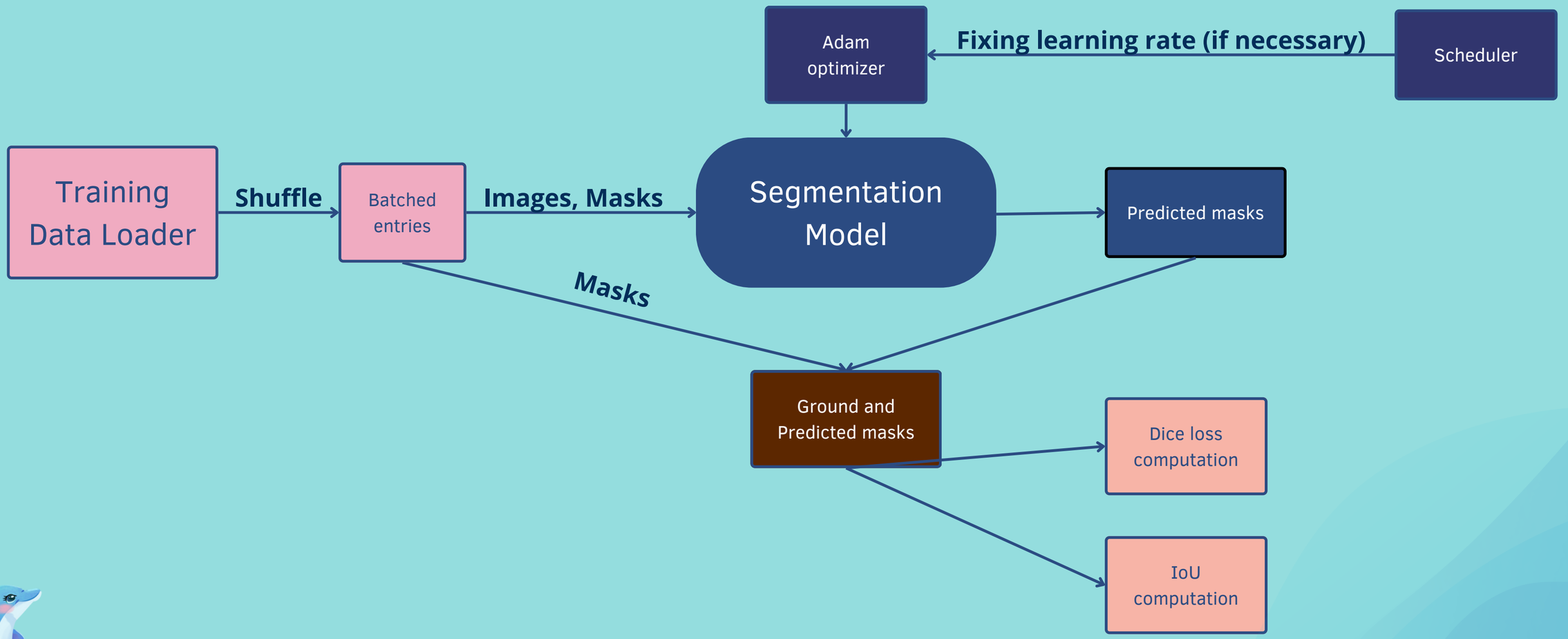
Specifically, a Dropout layer is applied to the output of each encoder layer to randomly zero out activations during training.

This ensures that the model does not overly rely on specific features, improving generalization, especially in cases with limited data.

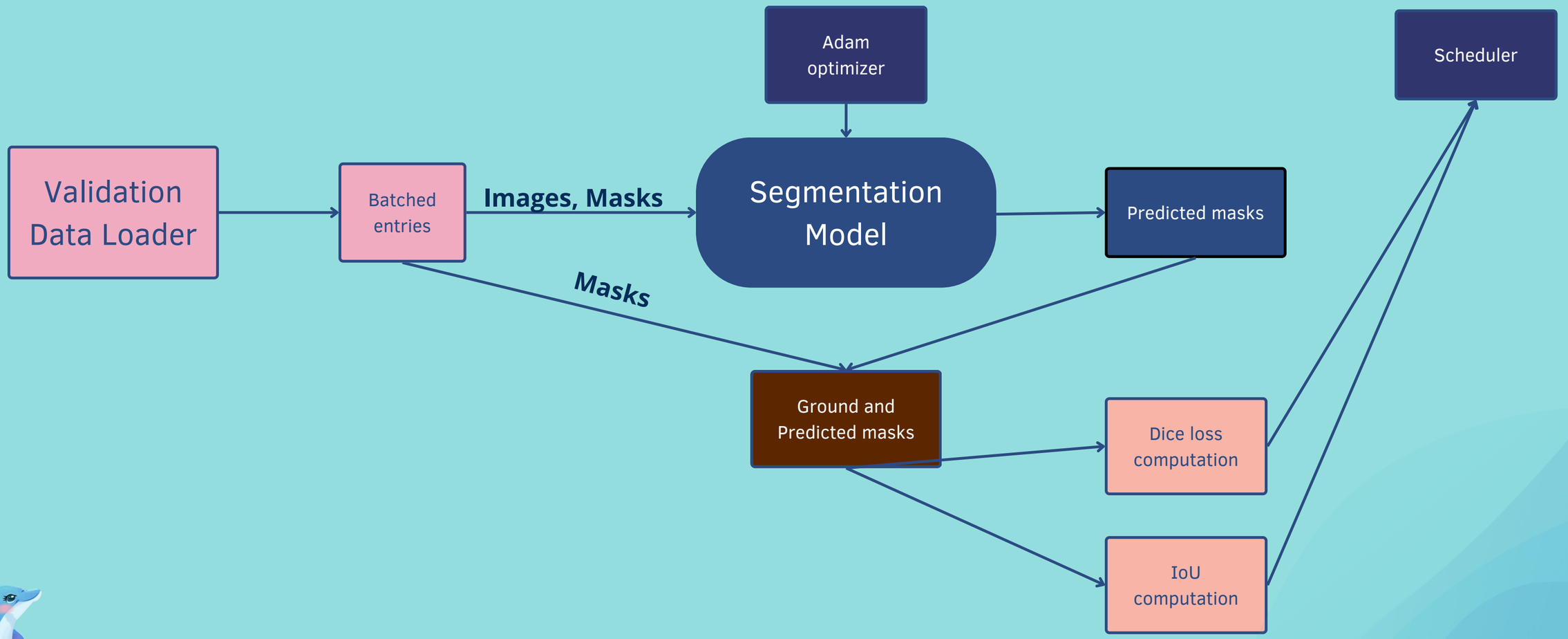
The rest of the UNet architecture remains unchanged.

```python
ENCODER = 'resnet18'
ENCODER_WEIGHTS = 'imagenet'
ACTIVATION = 'softmax2d'  # Use sigmoid  if doing one-single-class segmentation
DEVICE = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

class UNetWithDropout(smp.Unet):
    def __init__(self, encoder_name, encoder_weights, classes, activation, in_channels=3, dropout_prob=0.5):
        super().__init__(
            encoder_name=encoder_name,
            encoder_weights=encoder_weights,
            classes=classes,
            activation=activation,
            in_channels=in_channels
        )
        self.dropout = nn.Dropout2d(p=dropout_prob)  # Dropout layer

    def forward(self, x):
        """Forward method with dropout added to the encoder and decoder outputs."""
        features = self.encoder(x)
        features = [self.dropout(feature) for feature in features]  # Apply dropout after each encoder layer
        decoder_output = self.decoder(*features)  # Decode features
        masks = self.segmentation_head(decoder_output)  # Generate segmentation mask
        return masks

model = UNetWithDropout(
    encoder_name=ENCODER,
    encoder_weights=ENCODER_WEIGHTS,  # Use 'imagenet' pretrained weights for encoder initialization
    classes=len(CLASSES),  # Number of classes in your dataset
    activation=ACTIVATION,  # Activation function for the output
    in_channels=3,  # Model input channels (1 for gray-scale images, 3 for RGB, etc.)
    dropout_prob=0.3
)
```

Running a training epoch



Running a validation epoch



Metrics

HOW THE MODEL EVALUATES

Weighted Dice Loss

Dice loss is a similarity measure used in image segmentation quantifying the overlap between predicted and ground truth binary masks.

Due to the high imbalance of the classes in masks, the WeightedDiceLoss was implemented by assigning weights to each class. It ensures that contributions from classes are balanced, particularly addressing imbalances where some classes might be underrepresented in the data.

IoU

The IoU (Intersection over Union) metric calculates the ratio of the intersection (the overlapping area) between the predicted and actual object regions to the union (the total area covered by both the predicted and actual objects).
A higher IoU score indicates better model performance, with the prediction closely matching the ground truth.

Parameters

HOW THE MODEL LEARNS

Optimizer

An optimizer is an algorithm that adjusts a model's weights and biases to minimize the loss function and improve its performance.

The optimizer used here is the Adam optimizer, known for its efficiency and self-tuning capabilities.
It is configured with a learning rate and weight decay of 1e-4 to prevent overfitting.

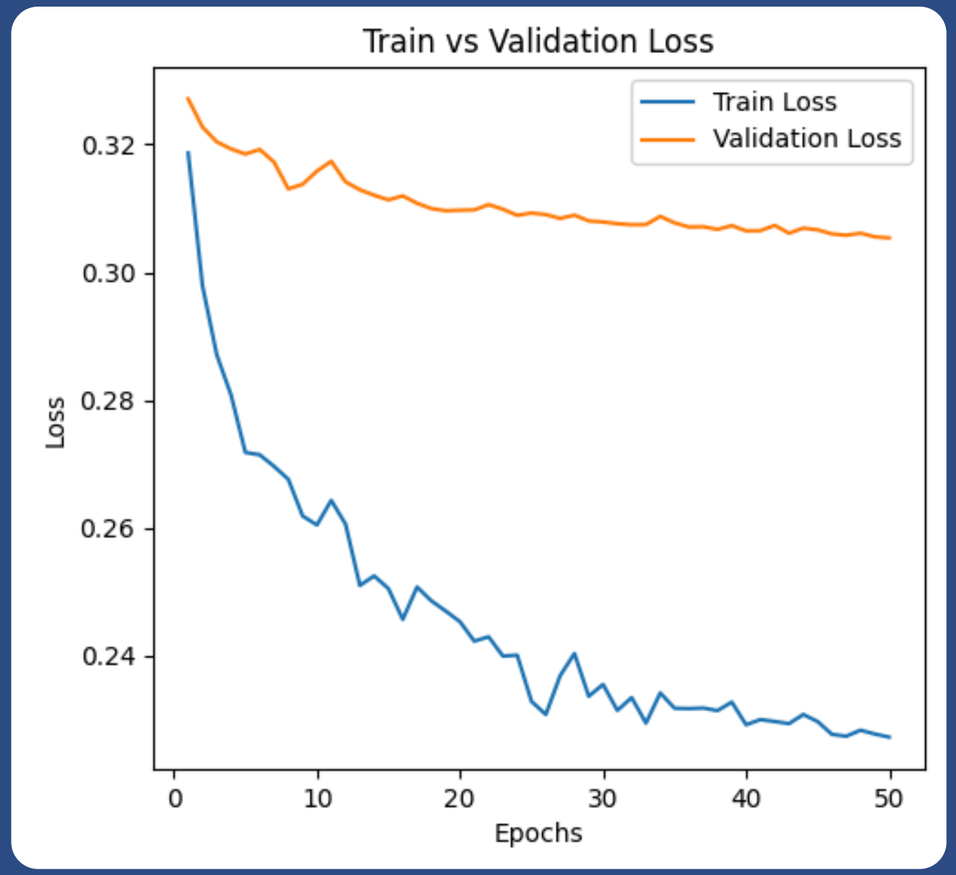
Scheduler

A learning rate scheduler, ReduceLROnPlateau, is used to adjust the learning rate during training.

It reduces the learning rate by a factor of 0.5 when the validation loss plateaus (not improves) for 3 consecutive epochs.

Computational Results

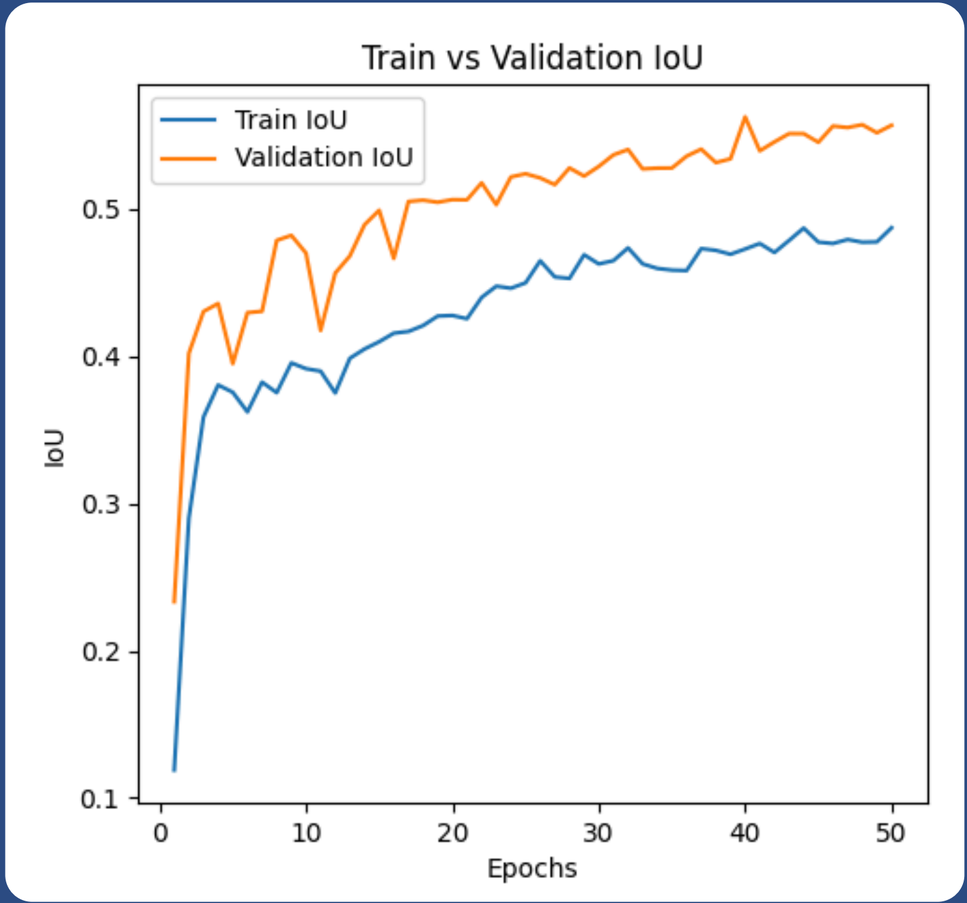
Train and Validation Loss



After 50 epochs

- Loss measures how well the model is performing its predictions. Lower values indicate better performance.
- The training loss consistently decreased over 50 epochs, showing steady learning and improvement.
- Validation loss followed a similar trend with minor fluctuations.
- The small gap between training and validation loss towards the end suggests the model generalizes well to unseen data, indicating minimal overfitting.

Train and Validation IoU

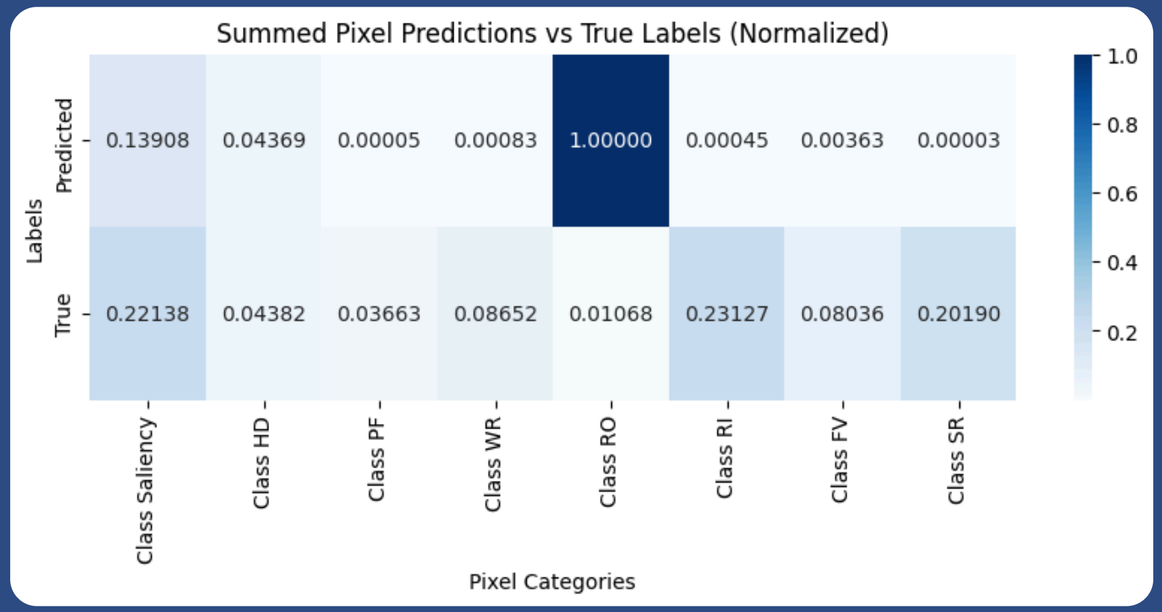


After 50 epochs

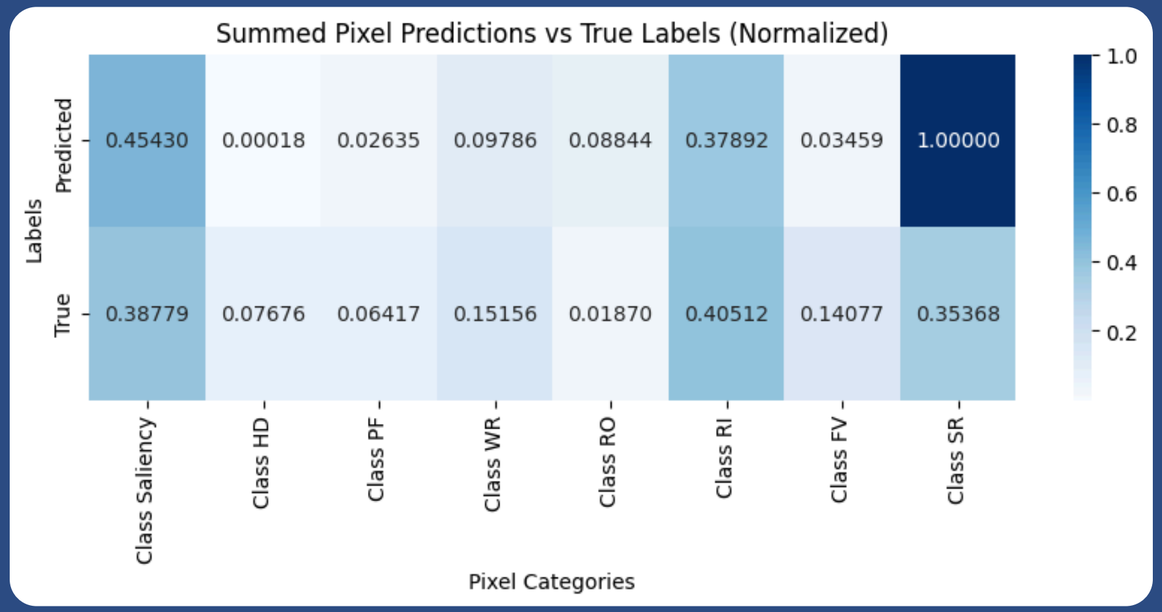
- IoU measures the overlap between predicted and true segments, where higher values indicate better performance.
- Training IoU improved significantly, reflecting enhanced model accuracy.
- Validation IoU saw a similar rise, showing strong generalization capabilities.
- Slight fluctuations in Validation IoU are visible but remain within an acceptable range, indicating robustness.
- Steady increases in both metrics across epochs indicate effective learning.

Link to tableau dashboard

Heatmap



After 50 epochs



After 100 epochs

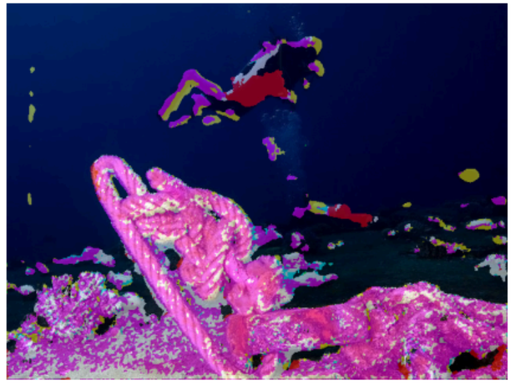
We notice a big improvement in predicted classes but still some classes are underpredicted and others are overpredicted

Some segmentation results

Unet after 100 epochs

Original Image

Image with Segmentation Overlay

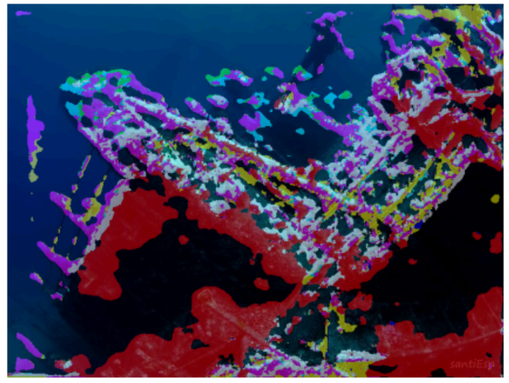


Original Image

Image with Segmentation Overlay

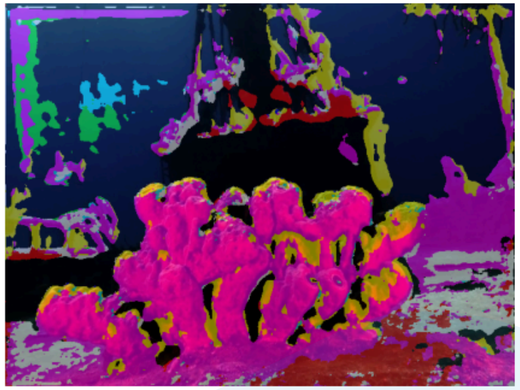
Original Image

Image with Segmentation Overlay



Original Image

Image with Segmentation Overlay



Comparison of different models

We compared several models over 10 epochs to evaluate their performances.

The models generally show decreasing loss and improving IoU across training.

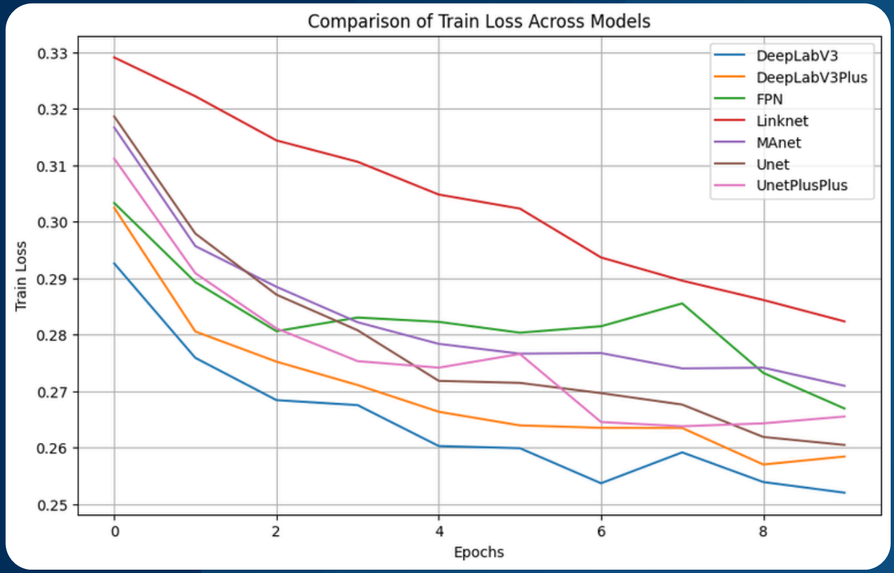
DeepLabV3, DeepLabV3Plus, and MAnet demonstrate strong performance, while FPN and Linknet have slower IoU gains.

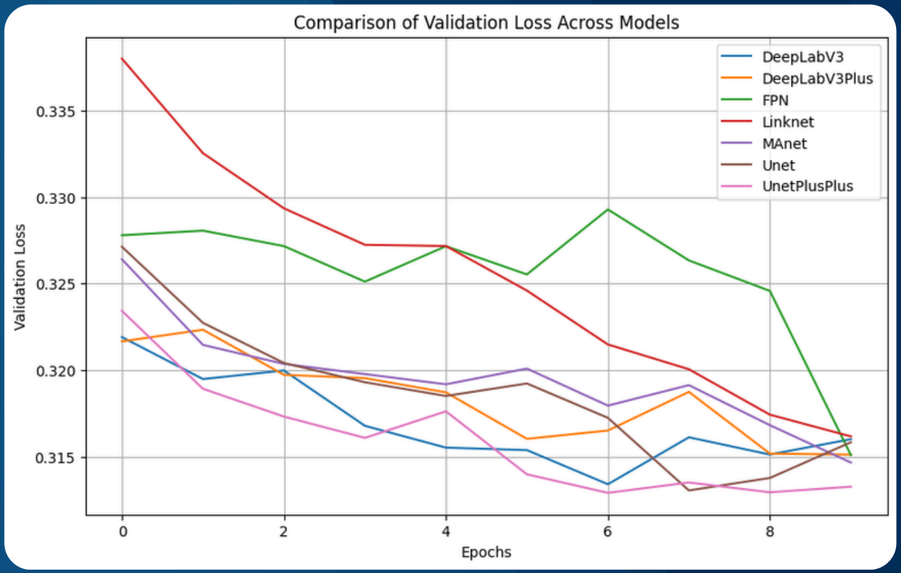
Unet and UnetPlusPlus improve consistently, with UnetPlusPlus slightly ahead.

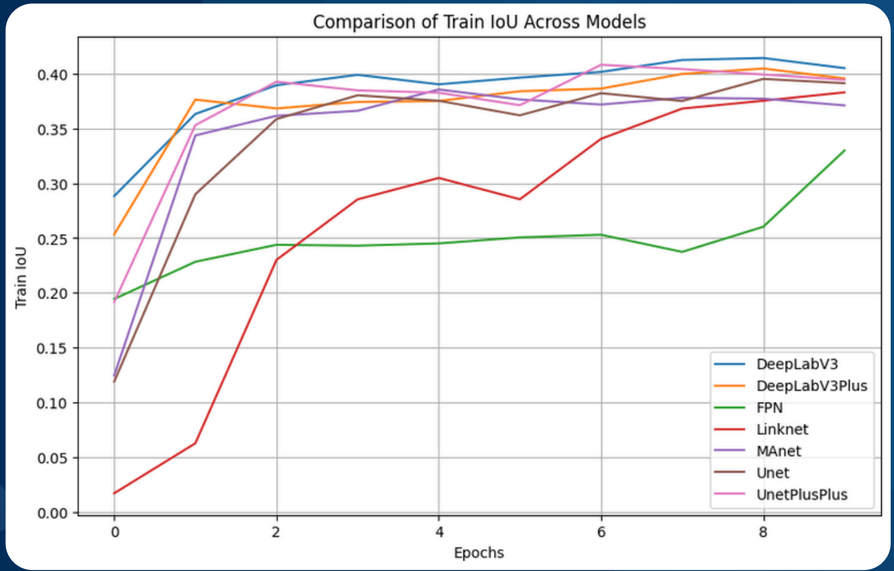
Overall, DeepLabV3Plus achieves the best balance between training and validation, highlighting its robustness.

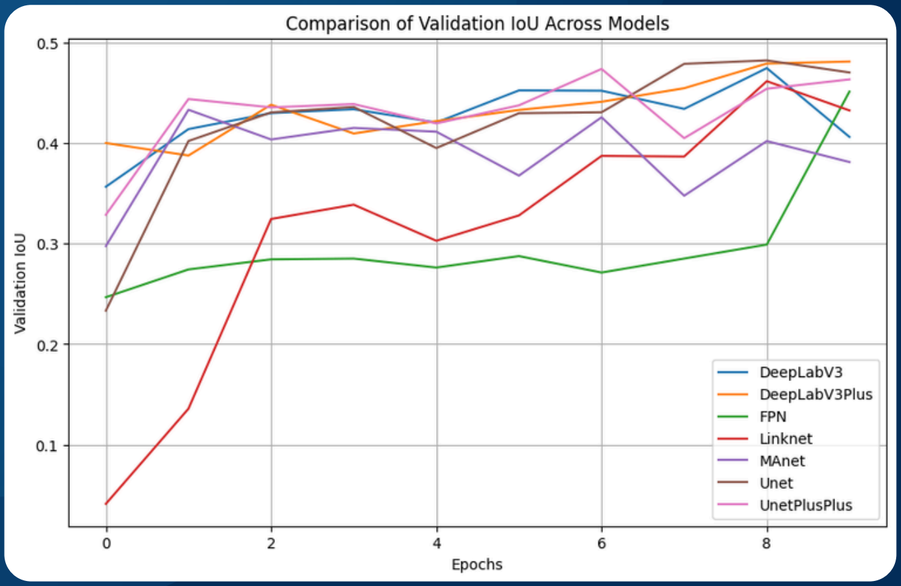
In the next slide we present the plots of the values.

Comparison of different models

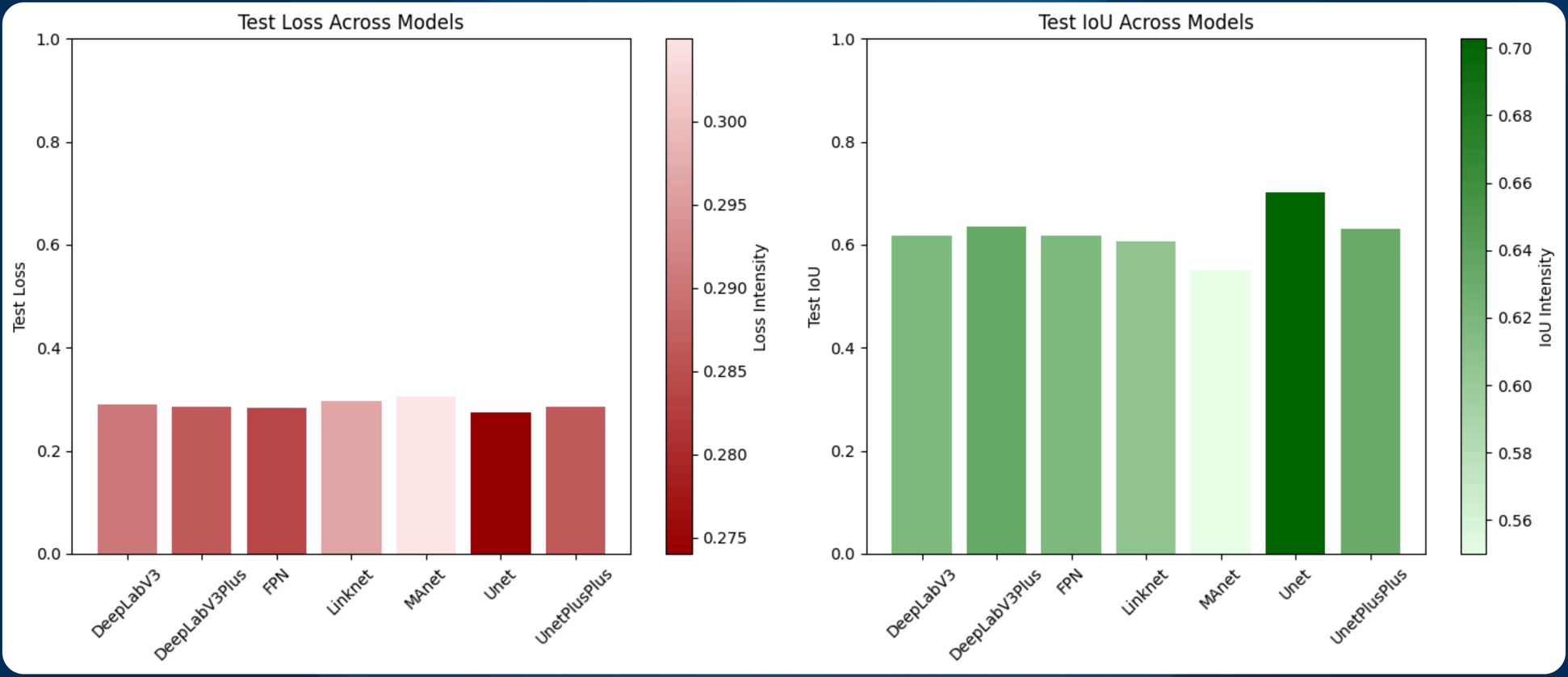








Comparison of different models



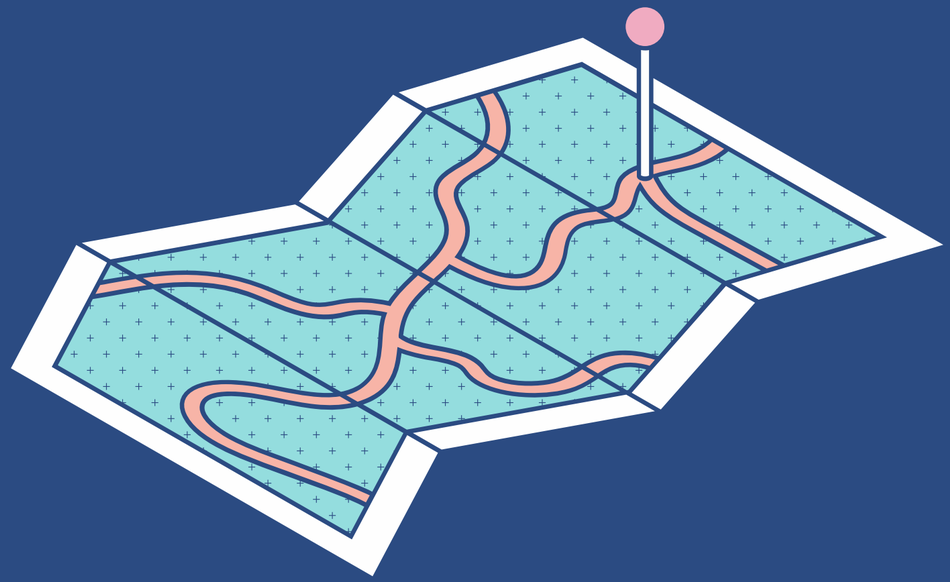
The test metrics provide an evaluation of model performance on unseen data.

Unet achieves the highest Test IoU and the lowest Test Loss

Can we do better?

To improve results, several strategies can be applied:

- **Hyperparameter Tuning:**
  - Fine-tuning learning rates, batch sizes, and other hyperparameters can help the models converge more efficiently and avoid overfitting.
- **Combining a Similar Dataset:**
  - Integrating additional, similar datasets can provide more diverse training examples, helping the model generalize better to unseen data.
- **Transfer Learning:**
  - Leveraging pretrained models on related tasks or datasets can improve performance by transferring learned features, especially when training data is limited.
- **Advanced Augmentation Techniques:**
  - Crop the regions with minority classes in the masks and apply aggressive augmentation to those areas, providing the model with more samples of underrepresented classes.



Thank you!