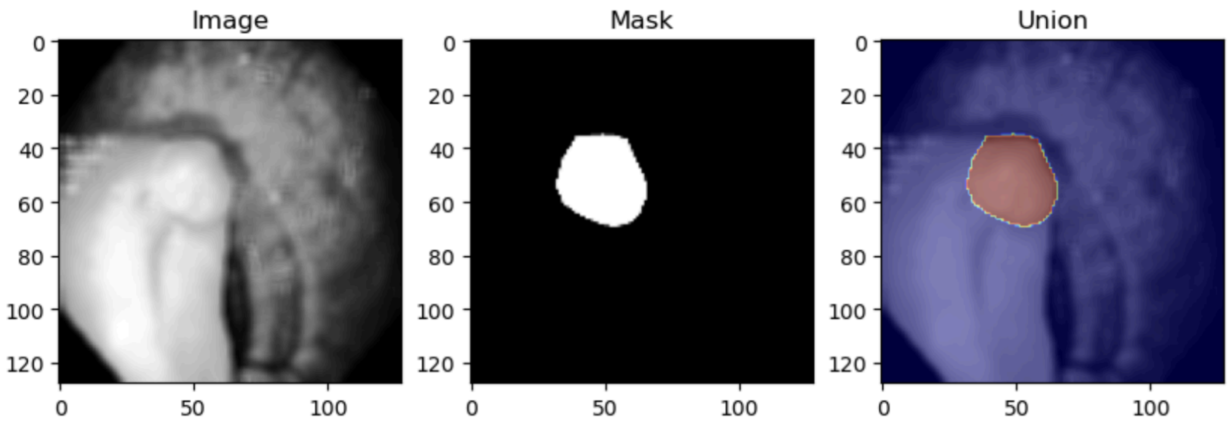
Ashkan Zarkhah
610399196

- **Abstract:**
In the referenced study, researchers attempted to leverage pre-trained models on the Kvasir-SEG dataset; however, these parameters were not publicly available. Consequently, we decided to construct our models from scratch. Our initial approach involved training a U-net model with randomly initialized parameters. Subsequently, utilizing the predictions generated by this model, we synthesized images augmented with their corresponding predicted masks, thus refining our dataset to enhance segmentation performance. We then subjected this augmented dataset to two distinct models: a ResNet model and an additional U-net model, facilitating a comparative analysis of their segmentation capabilities. Finally, we explored the potential impact of iteratively refining our dataset using predictions from the ResNet model, investigating whether a sequential application of three models—U-net, ResNet, and U-net—yielded improved or deteriorated results.
- **Dataset Familiarization:**
The Kvasir-SEG dataset comprises 1000 images accompanied by their respective masks. Despite claims made by the dataset publishers regarding the removal of clinic signatures from all images, some instances retain this identifying information, along with hardcoded dates and times. These remnants of clinic signatures and temporal data have the potential to adversely affect model training and may lead to decreased performance. Additionally, the dataset includes a JSON file containing original information provided by healthcare professionals, which serves as the basis for generating the masks associated with the images.

- **Preprocessing:**
In the preprocessing stage, distinct procedures were applied to both the images and masks. For the images, initial processing involved converting them to grayscale, a step taken to reduce model parameters and enhance learning efficiency given the constraints of computational resources. Subsequently, all images were resized to a uniform dimension of 128 x 128 pixels, facilitating consistent encoding by ensuring a power-of-two dimensionality so that our encoders can go long by dividing features by two. Following resizing, histogram equalization was employed to enhance contrast and reveal latent details within the images, compensating for variations in brightness and ensuring uniformity in model perception. Additionally, a 4x4 square-based blurring technique was applied to remove sharp edges and prevent model overfitting. To augment the

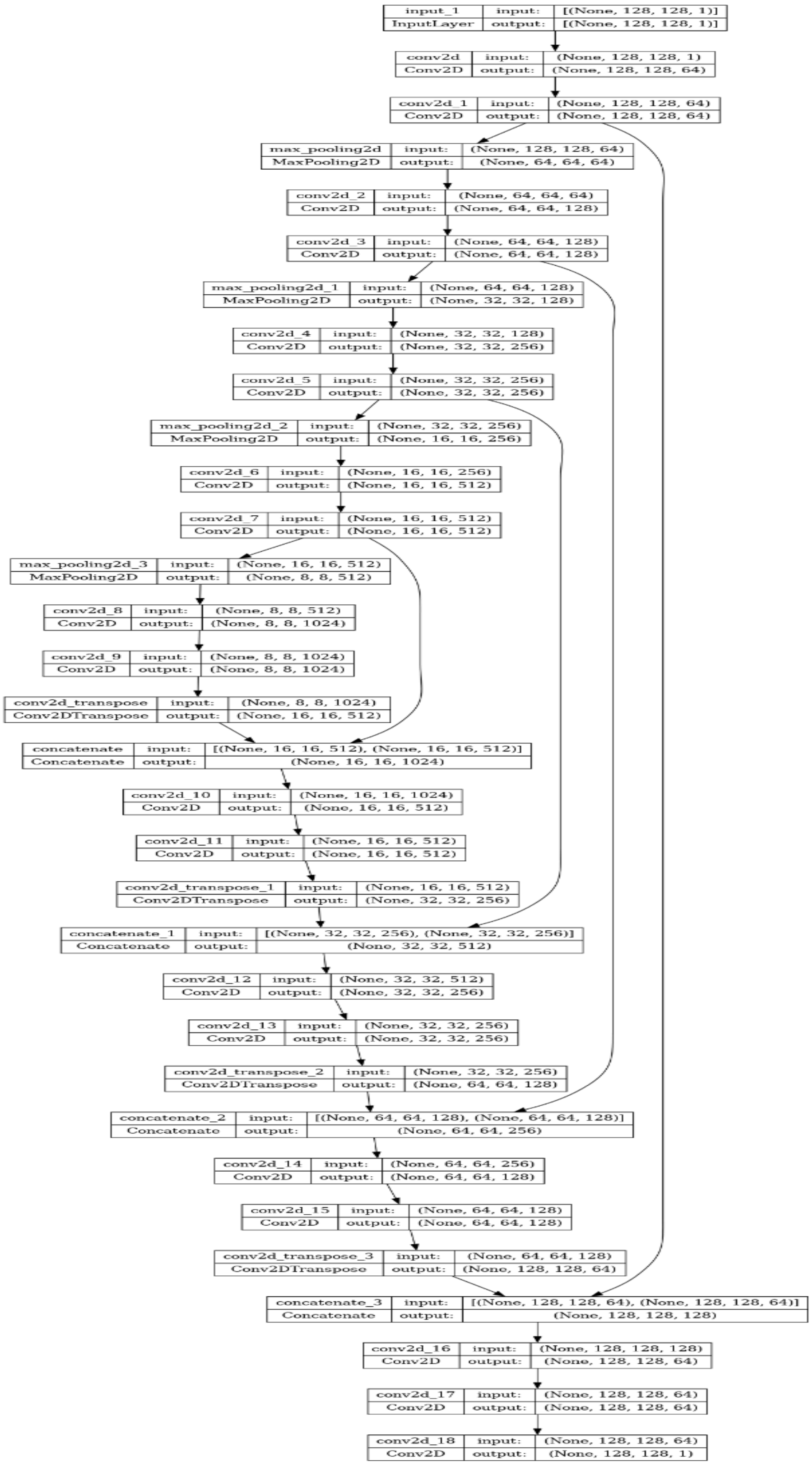
dataset, images were further rotated by 90, 180, and 270 degrees, enabling the model to capture variations in perspective and angle.

Similar preprocessing steps were applied to the masks, excluding histogram equalization and blurring to preserve the integrity of the ground truth data.



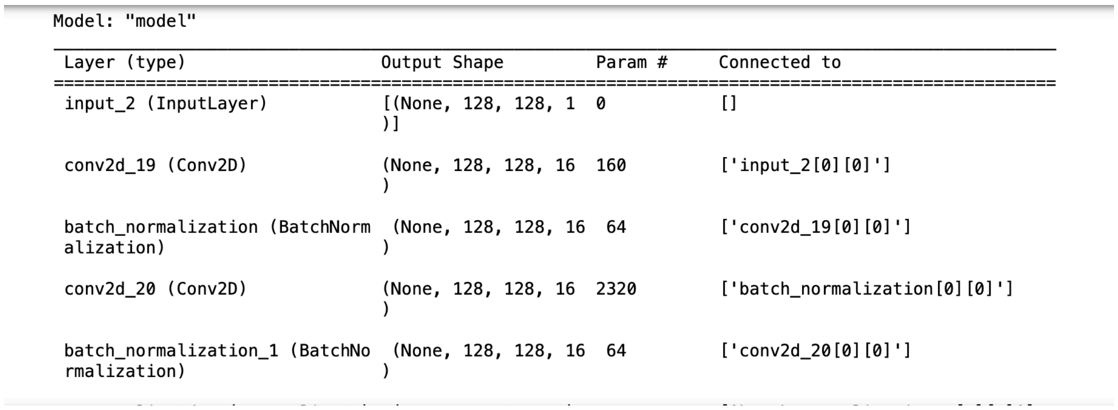
- **Model1: U-net**
  Our initial U-net model processes preprocessed images and comprises four distinct layer types:
  1. Encoder Blocks (Downsampling): The input images undergo processing by the encoder blocks, with the first block consisting of two convolutional layers followed by a max-pooling operation. Each subsequent block continues this process, resulting in feature maps with reduced dimensions due to pooling.
  2. Bottleneck Layer: Following the encoder blocks, a bottleneck layer is introduced, characterized by a convolutional block with an increased number of filters (1024). This layer is pivotal in capturing the most abstract features learned from the input images.
  3. Decoder Blocks (Upsampling): The decoder blocks initiate the upsampling process by reconstructing feature maps from the bottleneck layer and incorporating features from the corresponding encoder blocks. This process aims to recover spatial information lost during downsampling.
  4. Output Layer: Finally, the model concludes with an output layer comprising a single-filter convolutional layer with a sigmoid activation function. This layer generates pixel-wise binary predictions, indicating the presence or absence of the target object in each image pixel.

  The model’s architect can be described in the following picture too (more clearer picture in the ipynb file):


input_1 | input: | [(None, 128, 128, 1)]
InputLayer | output: | [(None, 128, 128, 1)]
conv2d | input: | (None, 128, 128, 1)
Conv2D | output: | (None, 128, 128, 64)
conv2d_1 | input: | (None, 128, 128, 64)
Conv2D | output: | (None, 128, 128, 64)
max_pooling2d | input: | (None, 128, 128, 64)
MaxPooling2D | output: | (None, 64, 64, 64)
conv2d_2 | input: | (None, 64, 64, 64)
Conv2D | output: | (None, 64, 64, 128)
conv2d_3 | input: | (None, 64, 64, 128)
Conv2D | output: | (None, 64, 64, 128)
max_pooling2d_1 | input: | (None, 64, 64, 128)
MaxPooling2D | output: | (None, 32, 32, 128)
conv2d_4 | input: | (None, 32, 32, 128)
Conv2D | output: | (None, 32, 32, 256)
conv2d_5 | input: | (None, 32, 32, 256)
Conv2D | output: | (None, 32, 32, 256)
max_pooling2d_2 | input: | (None, 32, 32, 256)
MaxPooling2D | output: | (None, 16, 16, 256)
conv2d_6 | input: | (None, 16, 16, 256)
Conv2D | output: | (None, 16, 16, 512)
conv2d_7 | input: | (None, 16, 16, 512)
Conv2D | output: | (None, 16, 16, 512)
max_pooling2d_3 | input: | (None, 16, 16, 512)
MaxPooling2D | output: | (None, 8, 8, 512)
conv2d_8 | input: | (None, 8, 8, 512)
Conv2D | output: | (None, 8, 8, 1024)
conv2d_9 | input: | (None, 8, 8, 1024)
Conv2D | output: | (None, 8, 8, 1024)
conv2d_transpose | input: | (None, 8, 8, 1024)
Conv2DTranspose | output: | (None, 16, 16, 512)
concatenate | input: | [(None, 16, 16, 512), (None, 16, 16, 512)]
Concatenate | output: | (None, 16, 16, 1024)
conv2d_10 | input: | (None, 16, 16, 1024)
Conv2D | output: | (None, 16, 16, 512)
conv2d_11 | input: | (None, 16, 16, 512)
Conv2D | output: | (None, 16, 16, 512)
conv2d_transpose_1 | input: | (None, 16, 16, 512)
Conv2DTranspose | output: | (None, 32, 32, 256)
concatenate_1 | input: | [(None, 32, 32, 256), (None, 32, 32, 256)]
Concatenate | output: | (None, 32, 32, 512)
conv2d_12 | input: | (None, 32, 32, 512)
Conv2D | output: | (None, 32, 32, 256)
conv2d_13 | input: | (None, 32, 32, 256)
Conv2D | output: | (None, 32, 32, 256)
conv2d_transpose_2 | input: | (None, 32, 32, 256)
Conv2DTranspose | output: | (None, 64, 64, 128)
concatenate_2 | input: | [(None, 64, 64, 128), (None, 64, 64, 128)]
Concatenate | output: | (None, 64, 64, 256)
conv2d_14 | input: | (None, 64, 64, 256)
Conv2D | output: | (None, 64, 64, 128)
conv2d_15 | input: | (None, 64, 64, 128)
Conv2D | output: | (None, 64, 64, 128)
conv2d_transpose_3 | input: | (None, 64, 64, 128)
Conv2DTranspose | output: | (None, 128, 128, 64)
concatenate_3 | input: | [(None, 128, 128, 64), (None, 128, 128, 64)]
Concatenate | output: | (None, 128, 128, 128)
conv2d_16 | input: | (None, 128, 128, 128)
Conv2D | output: | (None, 128, 128, 64)
conv2d_17 | input: | (None, 128, 128, 64)
Conv2D | output: | (None, 128, 128, 64)
conv2d_18 | input: | (None, 128, 128, 64)
Conv2D | output: | (None, 128, 128, 1)

- **Model2: ResNet**
  To train our ResNet model, we merged our preprocessed images with the predictions from our initial U-net model. After creating our new training, validation, and test datasets, we proceed with the training of our ResNet model, which is structured as follows:
  1. Stage 1: Input images undergo processing through two convolutional layers with 16 filters each, followed by batch normalization and ReLU activation functions. Subsequently, max pooling with a pool size of (2,2) is applied to downsample the feature maps.
  2. Stages 2-4: These three stages adopt a uniform structure, utilizing a residual block function (resblock) to process the feature maps. These blocks consist of multiple convolutional layers with batch normalization and ReLU activation functions, followed by max pooling to further downsample the feature maps.
  3. Stage 5 (Bottleneck): Serving as the bottleneck layer, this stage employs a residual block to process the feature maps with 256 filters. Unlike the preceding stages, no max pooling is applied here to preserve spatial information.
  4. Upscale Stages 1-4: These stages are pivotal in gradually reconstructing the spatial information lost during downsampling. Each stage involves upsampling the feature maps and concatenating them with the corresponding feature maps from earlier stages using the upsample_concat function. The concatenated feature maps then undergo processing through a residual block with decreasing numbers of filters (128, 64, 32, 16).
  5. Final Output: The model concludes with a convolutional layer comprising a single filter and a sigmoid activation function. This layer yields pixel-wise binary predictions, indicating the presence or absence of the target object in each image pixel.

  Also, the model's structure can be described in the following picture (more complete picture in the ipynb file):



```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_2 (InputLayer)          [(None, 128, 128, 1  0           []
                               )]

 conv2d_19 (Conv2D)            (None, 128, 128, 16  160         ['input_2[0][0]']
                               )

 batch_normalization (BatchNorm  (None, 128, 128, 16  64          ['conv2d_19[0][0]']
 alization)                    )

 conv2d_20 (Conv2D)            (None, 128, 128, 16  2320        ['batch_normalization[0][0]']
                               )

 batch_normalization_1 (BatchNo  (None, 128, 128, 16  64          ['conv2d_20[0][0]']
 rmalization)                  )
```

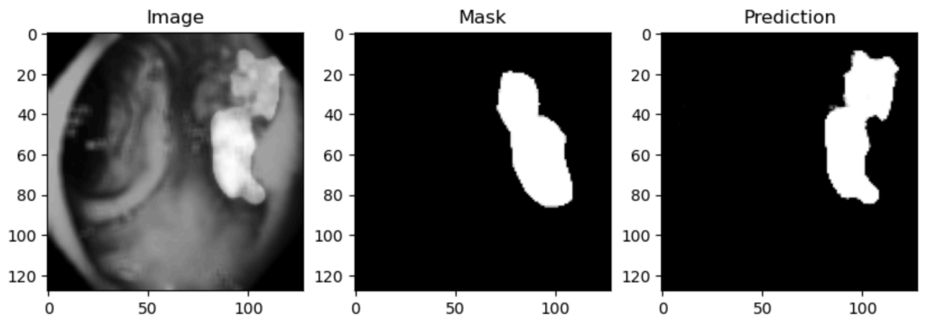
- **Model3: U-net**
  We developed two additional U-net models subsequent to our ResNet model. The first U-net model utilized the same input data as the ResNet model, facilitating a direct comparison between the performance of ResNet and U-net models on the identical preprocessed dataset. In contrast, the second U-net model differed in approach. Here, we integrated our newly generated dataset with predictions from our ResNet model, subsequently training a new U-net model on this combined dataset. This approach aimed to assess whether the ResNet model had effectively extracted additional information compared to our initial U-net model.

  Both U-net models maintained the same architectural structure as the first U-net model, primarily due to the promising results achieved by the initial model and computational constraints limiting further architectural complexity. Despite this consistency in structure, the comparative evaluation allowed us to discern any potential enhancements or insights gained from incorporating predictions from the ResNet model into the training process of the subsequent U-net model.

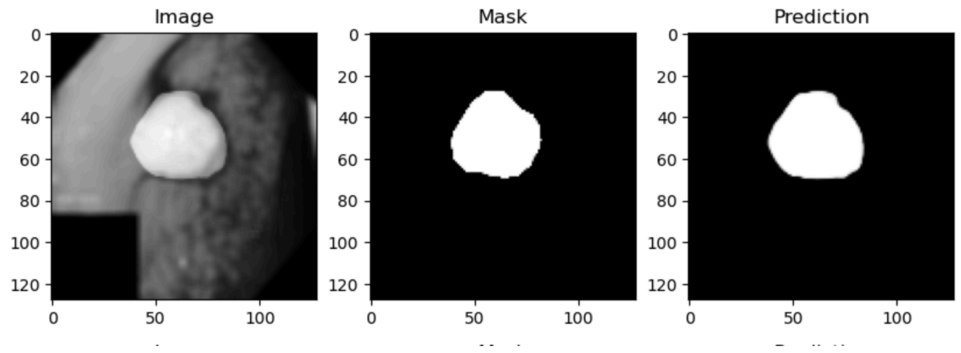
- **Results and Analysis**

	IoU	**Dice coefficient**	**Recall**	**Precision**
U-net alone	**0.73**	**0.71**	**0.84**	**0.62**
U-net -> ResNet	**0.73**	**0.72**	**0.84**	**0.63**
U-net -> U-net	**0.74**	**0.73**	**0.83**	**0.65**
U-net -> ResNet -> U-net	**0.46**	**0.42**	**0.30**	**0.68**

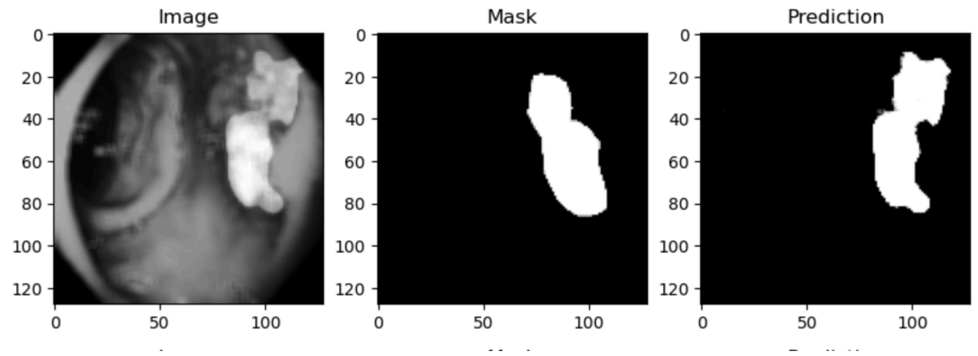
U-net alone: Our initial U-net model, trained from scratch, yielded commendable results, underscoring the inherent relationship between our images and corresponding masks, which the model successfully learned to discern.



U-net -> ResNet: Integrating ResNet into our model pipeline with the new dataset produced results akin to those of the standalone U-net model. A closer examination of the accompanying code reveals subtle improvements in mask prediction by the combined model, albeit with slightly increased padding. However, due to the heightened complexity of ResNet and resource constraints, the model's training was prematurely halted. Given sufficient computational resources and deeper models, even more promising outcomes are anticipated.



U-net -> U-net: Comparison with our standalone U-net model showcased remarkably similar results, suggesting that the initial U-net model had effectively learned the underlying patterns within the data. The marginal improvement observed in the subsequent U-net model trained on the learned data underscores the ResNet model's ability to learn as effectively as a simpler U-net model.



U-net -> ResNet -> U-net: Attempting to generate new datasets from model predictions resulted in overfitting, with our final model quickly converging and exhibiting signs of overfitting on the predictions of previous models. This underscores the challenge of integrating predictions from previous models into subsequent ones, as the errors of earlier models persist, hindering the correction of these mistakes. Consequently, it becomes apparent that deeper models rather than iterative dataset generation are imperative, as the latter approach perpetuates the propagation of errors without the opportunity for correction.

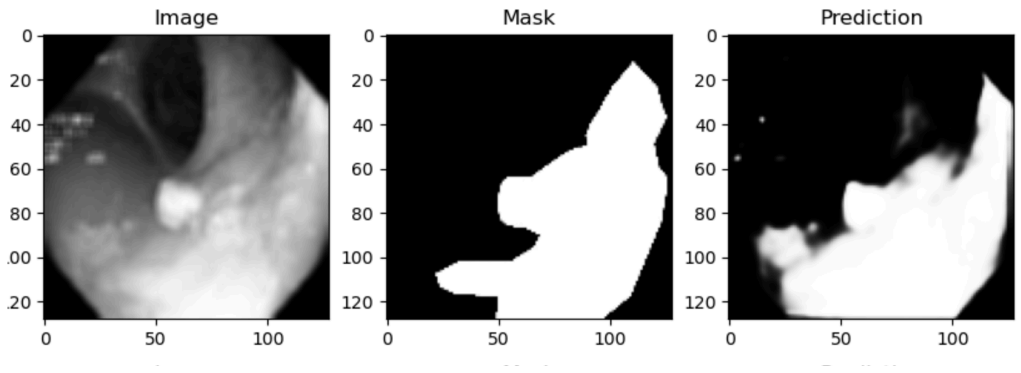


Image
Mask
Prediction
0
20
40
60
80
100
120
0
50
100