ZNEUS Project 2

Amal Akhmadinurov, Lukáš Šebök

December 5, 2025

1 Tools

- Torch
- TorchMetrics
- Pandas
- Other utility libraries

2 Dataset: TIGER Challenge dataset

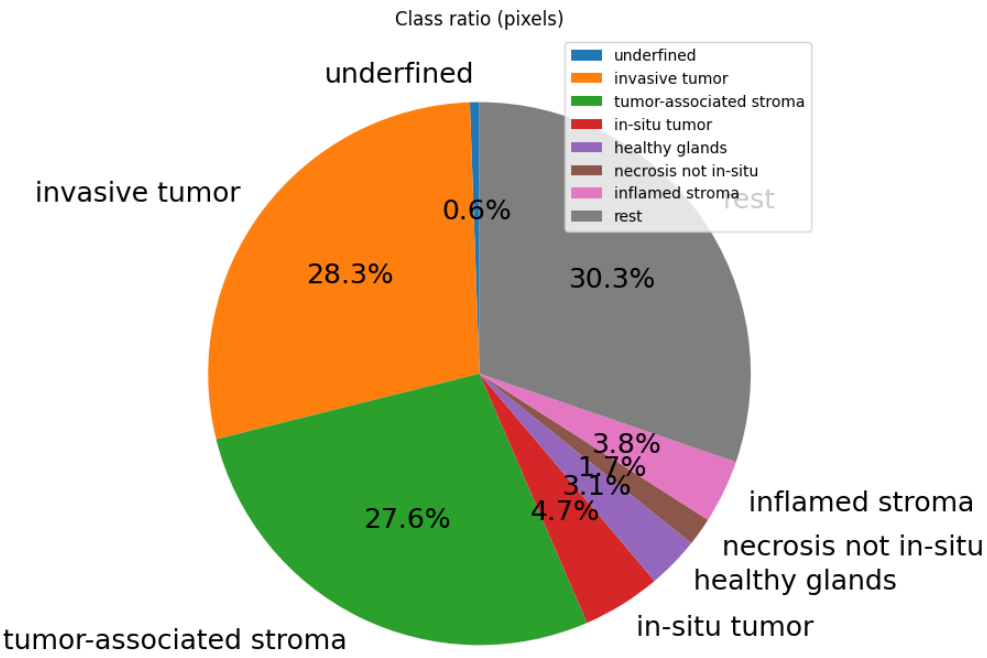

Figure 1: Class ratio

For this project we used ROI subfolder of this dataset

In each ROI, the following regions are annotated, with the corresponding labels:

- **Invasive tumor** (*label* = 1): this class contains regions of the invasive tumor, including several morphological subtypes, such as invasive ductal carcinoma and invasive lobular carcinoma.

- **Tumor-associated stroma** (*label* = 2): this class contains regions of stroma (i.e., connective tissue) that are associated with the tumor. This includes stromal regions within the main bulk of the tumor and its close surroundings. In some cases, the tumor-associated stroma may resemble the "healthy" stroma typically found outside the tumor bulk.
- **In-situ tumor** (*label* = 3): this class contains regions of in-situ malignant lesions, such as ductal carcinoma in situ (DCIS) or lobular carcinoma in situ (LCIS).
- **Healthy glands** (*label* = 4): this class contains regions of glands with healthy epithelial cells.
- **Necrosis not in-situ** (*label* = 5): this class contains regions of necrotic tissue that are not part of in-situ tumor. For example, DCIS often presents a typical necrotic pattern that is considered part of the lesion itself; such necrotic regions are annotated as "in-situ tumor" rather than "necrosis."
- **Inflamed stroma** (*label* = 6): this class contains tumor-associated stroma with a high density of lymphocytes (i.e., "inflamed"). For TIL assessment, inflamed stroma and tumor-associated stroma may be grouped together, but they were annotated separately to account for differences in visual patterns.
- **Rest** (*label* = 7): this class contains various tissue compartments not specifically annotated in the other categories; examples include healthy stroma, erythrocytes, adipose tissue, skin, nipple, etc.

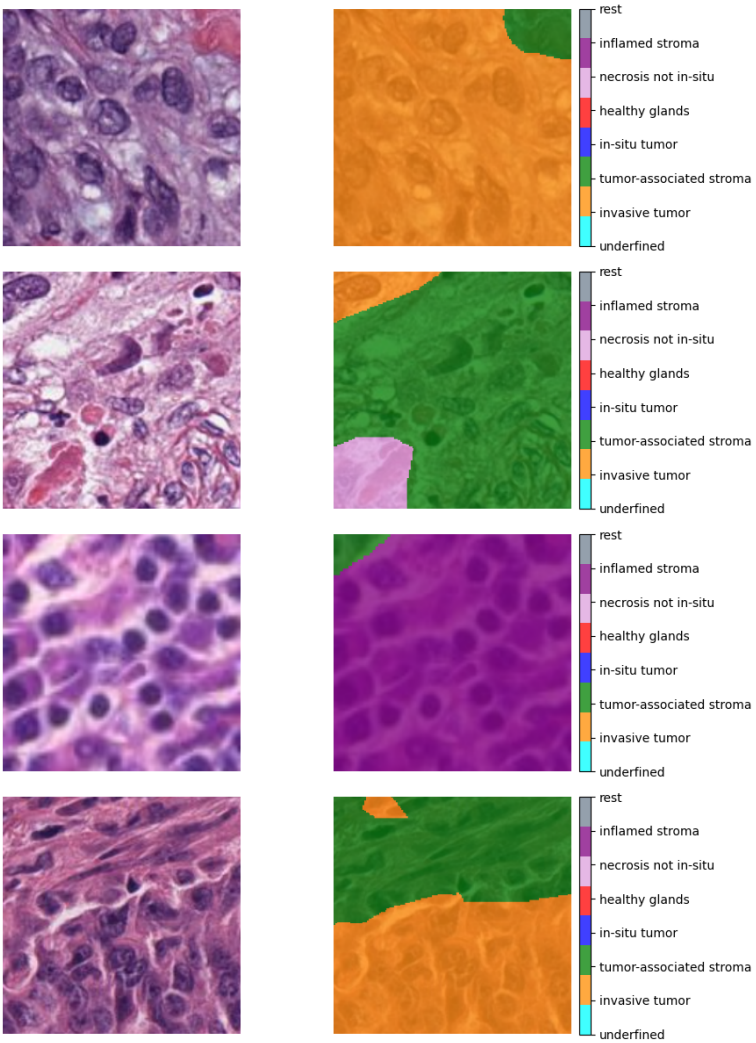

Figure 2: Visualization of TIGER dataset

2.1 Normalization and data transformation

Each image was normilized using division by 255.

Patches

To make dataset larger and leverage large images in the dataset we have decided to extract 256x256 non-overlapping patches from original images. Images that have size smaller than patch size were non patched, but only resized to 256x256.

Augmentations

For each image we have applied the following tranformations:

- **Rotate**: Random rotation with a limit of 45°, using the specified `border_mode` ($p = 0.5$).
- **ShiftScaleRotate**: Applies random shifting (up to 0.5), scaling (up to 0.15), and no rotation, with the given `border_mode` ($p = 0.5$).
- **HorizontalFlip**: Random horizontal flip applied with probability 0.5.
- **VerticalFlip**: Random vertical flip applied with probability 0.5.

BORDER REFLECT

During EDA we have identified that augmentations create a big amount of pixels with 0 value. To address this issue that will inevitably harm the performance of the model we have used border reflect mode (from OpenCV library). Examples of augmented images can be seen in figure 3

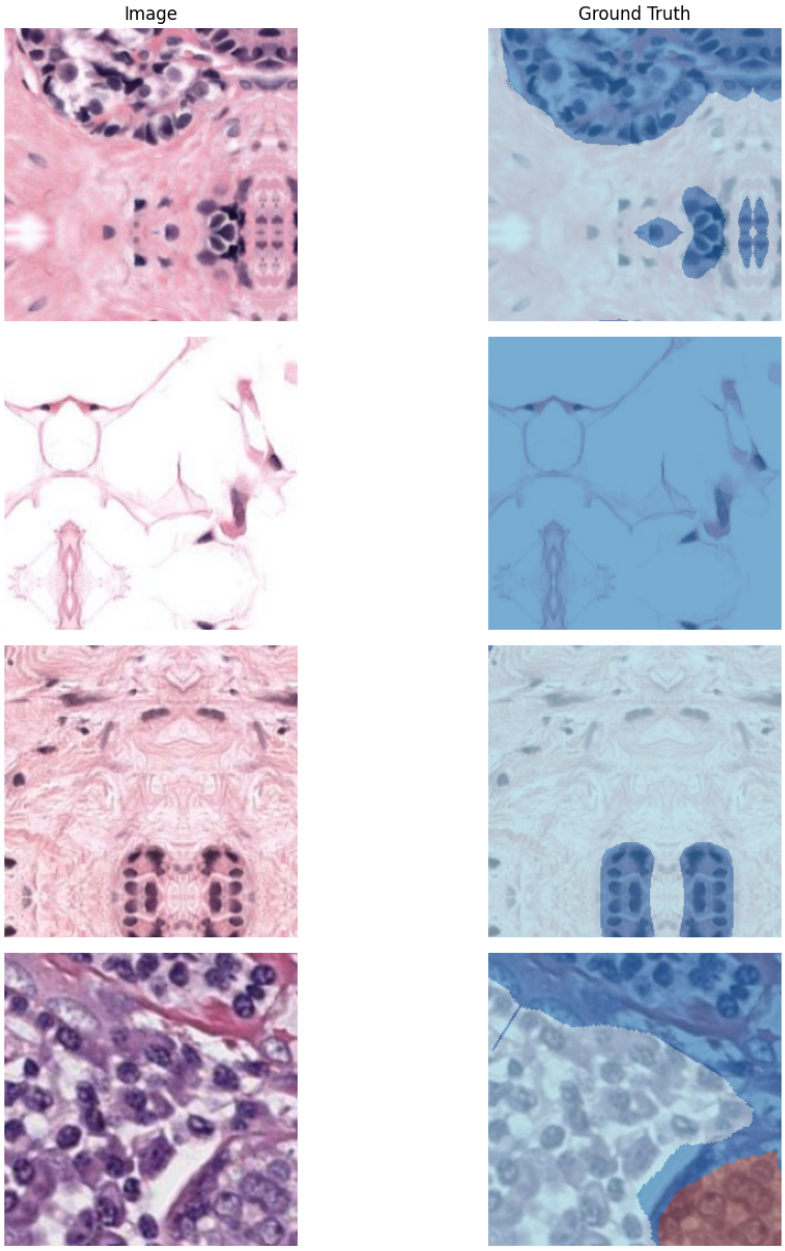

Figure 3: Model architecture without pretrained backbone

3 Evaluation Metrics

The model evaluates performance using the following metrics, computed for both training and validation sets:

- **IoU**: Mean Intersection over Union across all classes, computed using `MeanIoU` with `include_background=True` and `input_format="index"`.
- **Dice Score**: Macro-averaged Dice coefficient, computed using `DiceScore` with `include_background=True`, `average="macro"`, and `input_format="index"`.
- **Loss**: Segmentation loss defined by the chosen loss function $\mathcal{L}$, applied to logits and class-index masks.

4 Original Tiger Dataset experiments

Report is available here: Report

4.1 Model Architecture

Initially we have used model without any pretrained backbone.

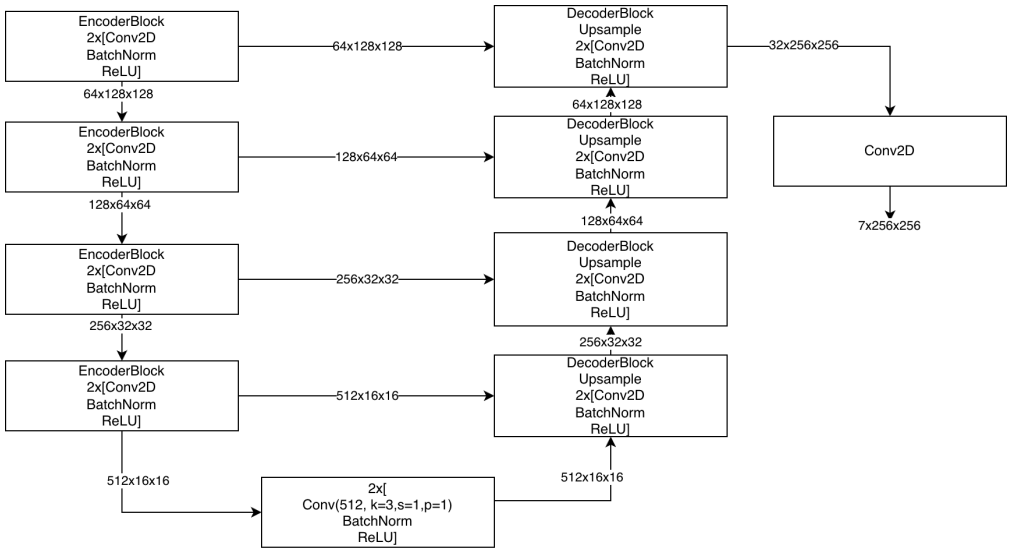

Figure 4: Model architecture without pretrained backbone

But for further experiments we used layers from a pretrained version of EfficiientNetB0

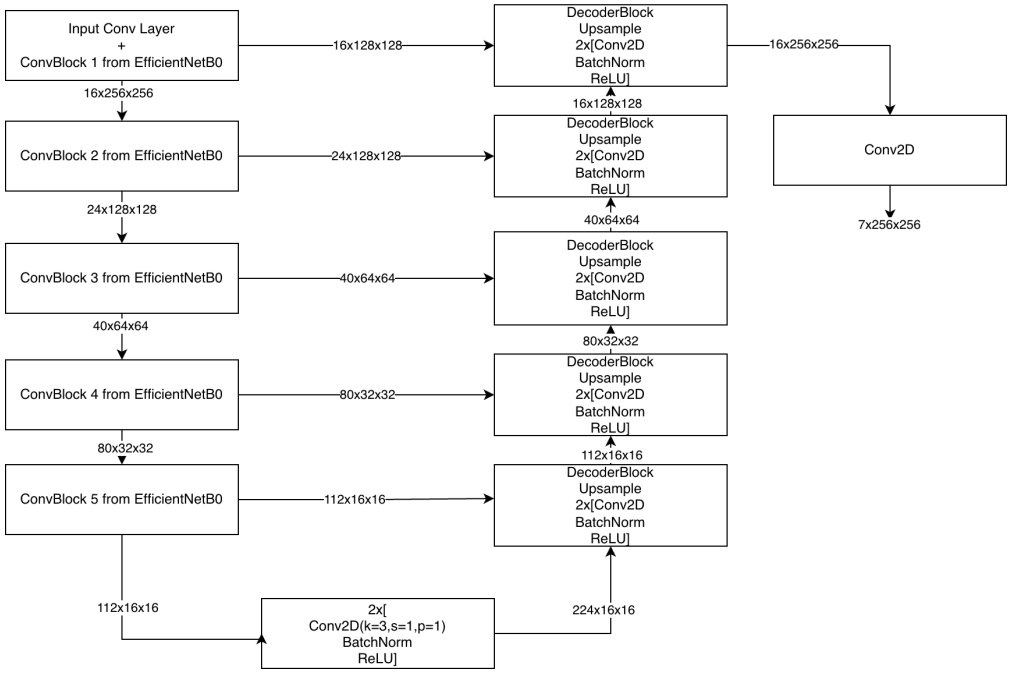

Figure 5: Model architecture with EfficientNetB0 based pretrained encoder (best model in experiments)

4.2 Hyperparameters

The best model uses EfficientNetB0 pretrained layers for the encoder part and custom decoder trained from the ground up (look at Figure 2 for architecture) with the following hyperparameters, defined as constants in the implementation:

Table 1: Hyperparameters of the Model

Parameter	**Value**
Batch Size	32
Learning Rate	0.001
Maximum Epochs	50
Reduce LR Patience	10
Early Stopping Patience	10
Loss function	JaccardLoss("multiclass", [1,2,3,4,5,6,7])
Image Size	256x256

Loss function is Jaccard Loss (class is imported from segmentation_models_pytorch library) includes only 7 classes in the computation as 0 class is undefined in the original dataset.

The model applies a argmax activation to the output logits during training and evaluation to produce probabilities for each class in the output mask.

Key takeaways from this group of experiments:

- Image size can seriously influence the performance
- Original Dataset is imbalanced
- In out case its makes sense to use pretrained backbone

4.3 Results

Class	**Count**
Loss:	0.48958
IOU	0.43021
Dice Score	0.5933

Table 2: Best metrics values achieved on validation set

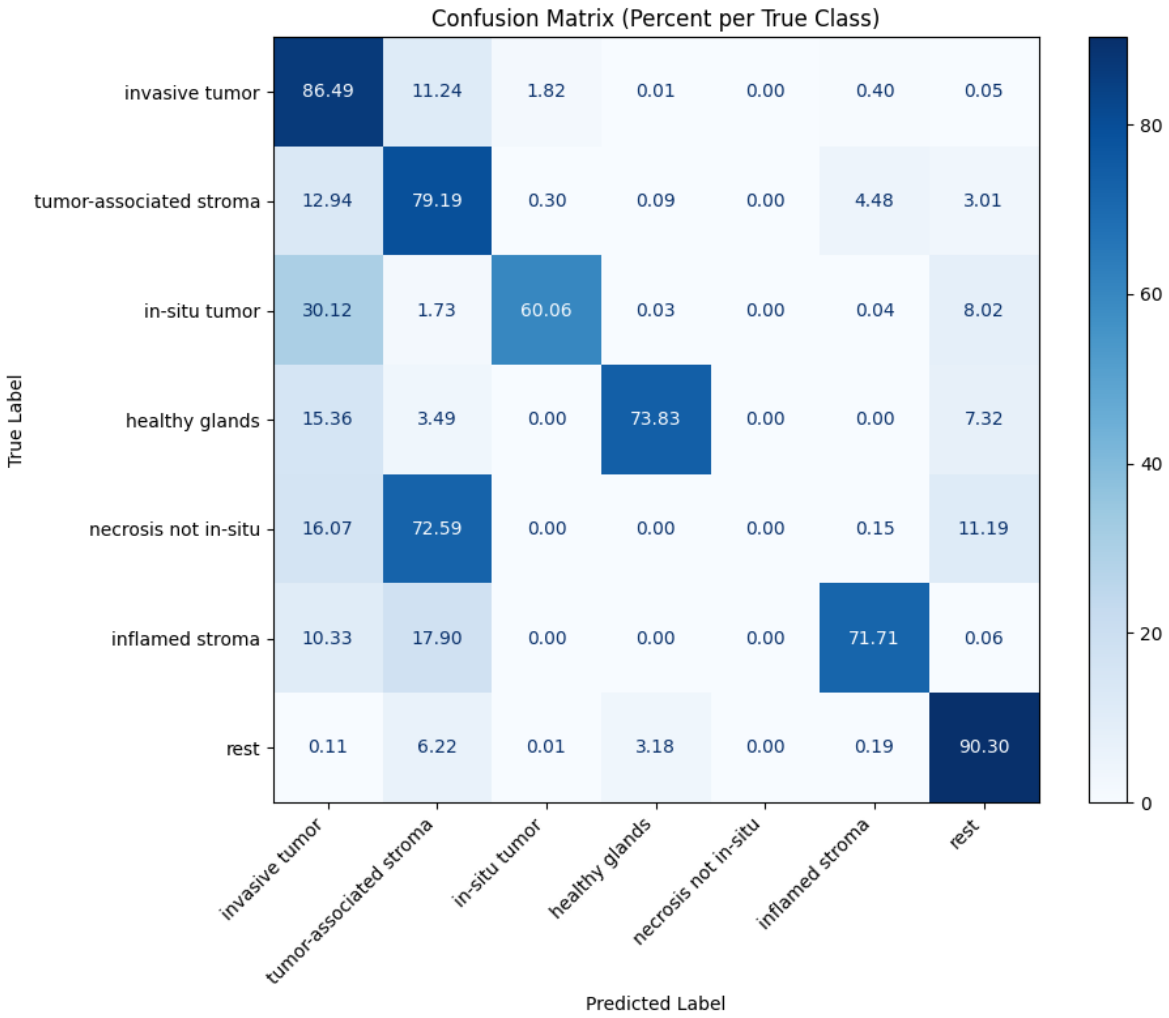

Figure 6: Confusion matrix in percents (per pixels)

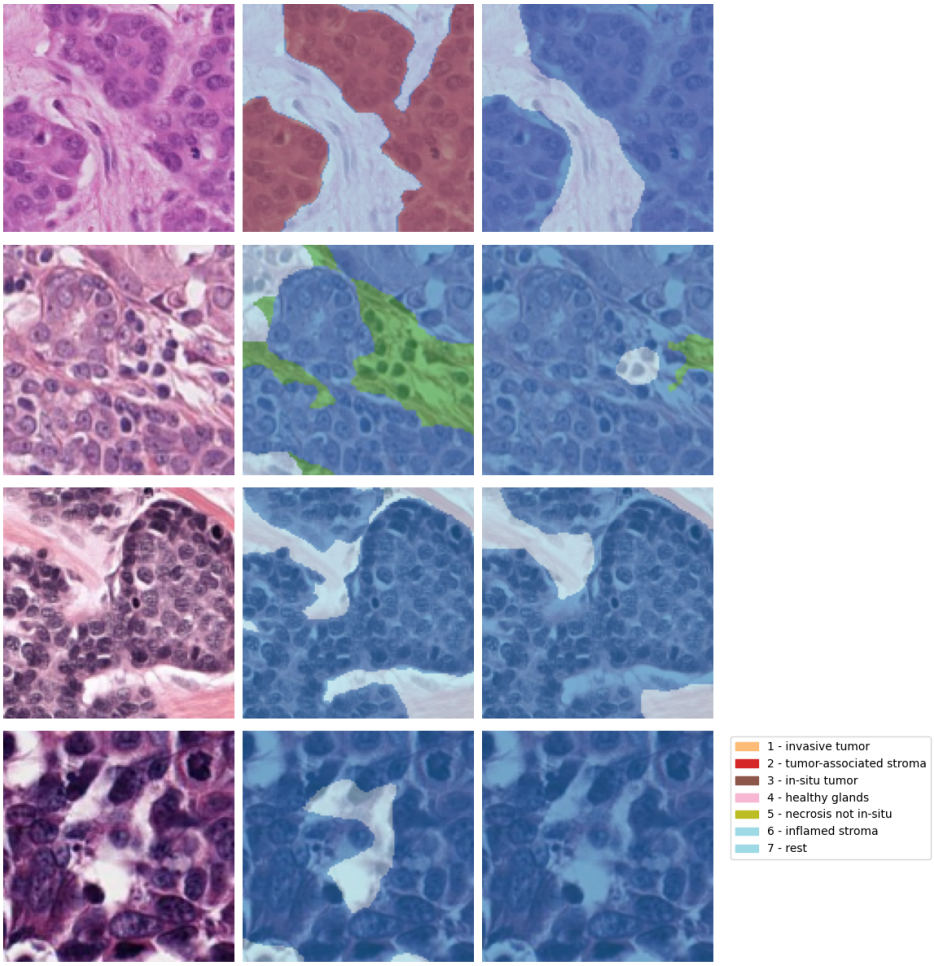

Figure 7: Example output on validation set

5 Modified Dataset: Unicorn Challenge Task 9. U-Net with EfficientNet encoder

Report is available here: Report

5.1 Data tranformation

The original seven annotated tissue classes from TIGER were merged into three:

- Invasive tumor and in-situ tumor → **tumor**
- Tumor-associated stroma and inflamed stroma → **stroma**
- Healthy glands, necrosis not in-situ, and rest → **rest**

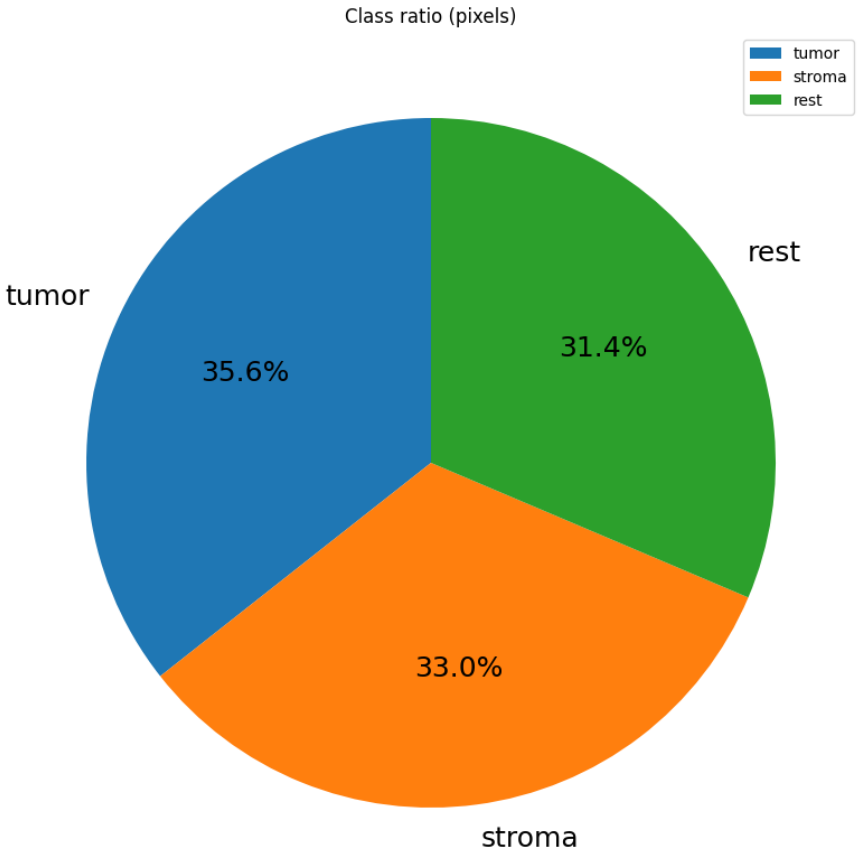

Figure 8: Best model architecture with EfficientNetB0 based pretrained encoder

5.2 Model Architecture

For these experiments we used layers from a pretrained version of EfficiientNetB0 for encoder of the U-Net and custom decoder.

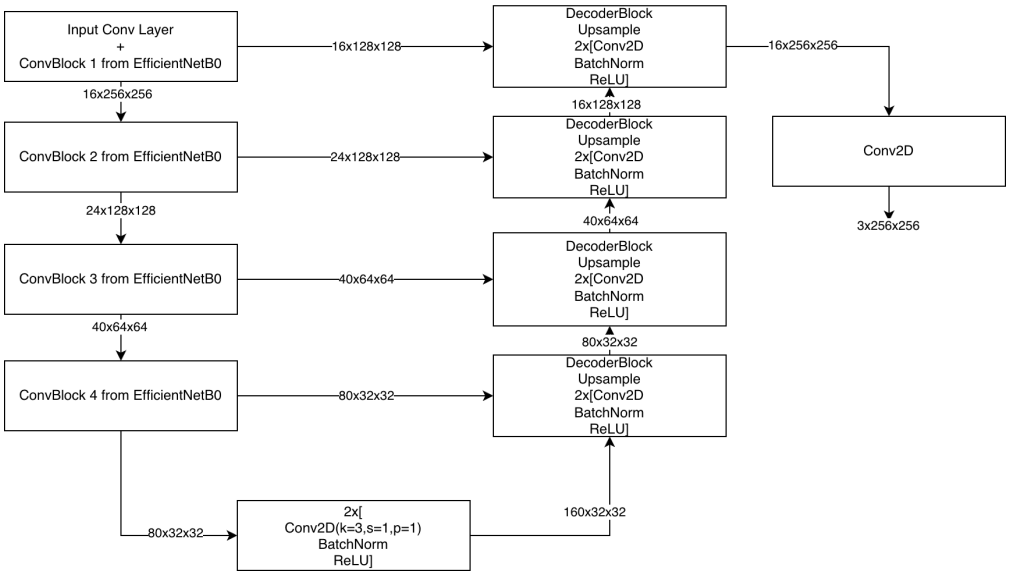

Figure 9: Best model architecture with EfficientNetB0 based pretrained encoder

5.3 Hyperparameters

The best model uses EfficientNetB0 pretrained layers for the encoder part and custom decoder trained from the ground up (look at Figure 8 for architecture) with the following hyperparameters, defined as constants in the implementation:

Table 3: Hyperparameters of the Model

Parameter	**Value**
Batch Size	32
Learning Rate	0.001
Maximum Epochs	50
Reduce LR Patience	10
Early Stopping Patience	10
Loss function	JaccardLoss("multiclass", [0,1,2])
Image Size	256x256

Loss function is Jaccard Loss (class is imported from segmentation_models_pytorch library) The model applies a argmax activation to the output logits during training and evaluation to produce probabilities for each class in the output mask.

Key takeaways from this group of experiments:

- Image size can seriously influence the performance
- Original Dataset is imbalanced
- In out case its makes sense to use pretrained backbone

5.4 Results

Class	**Count**
Loss:	0.24133
IOU	0.57794
Dice Score	0.713

Table 4: Best metrics values achieved on validation set

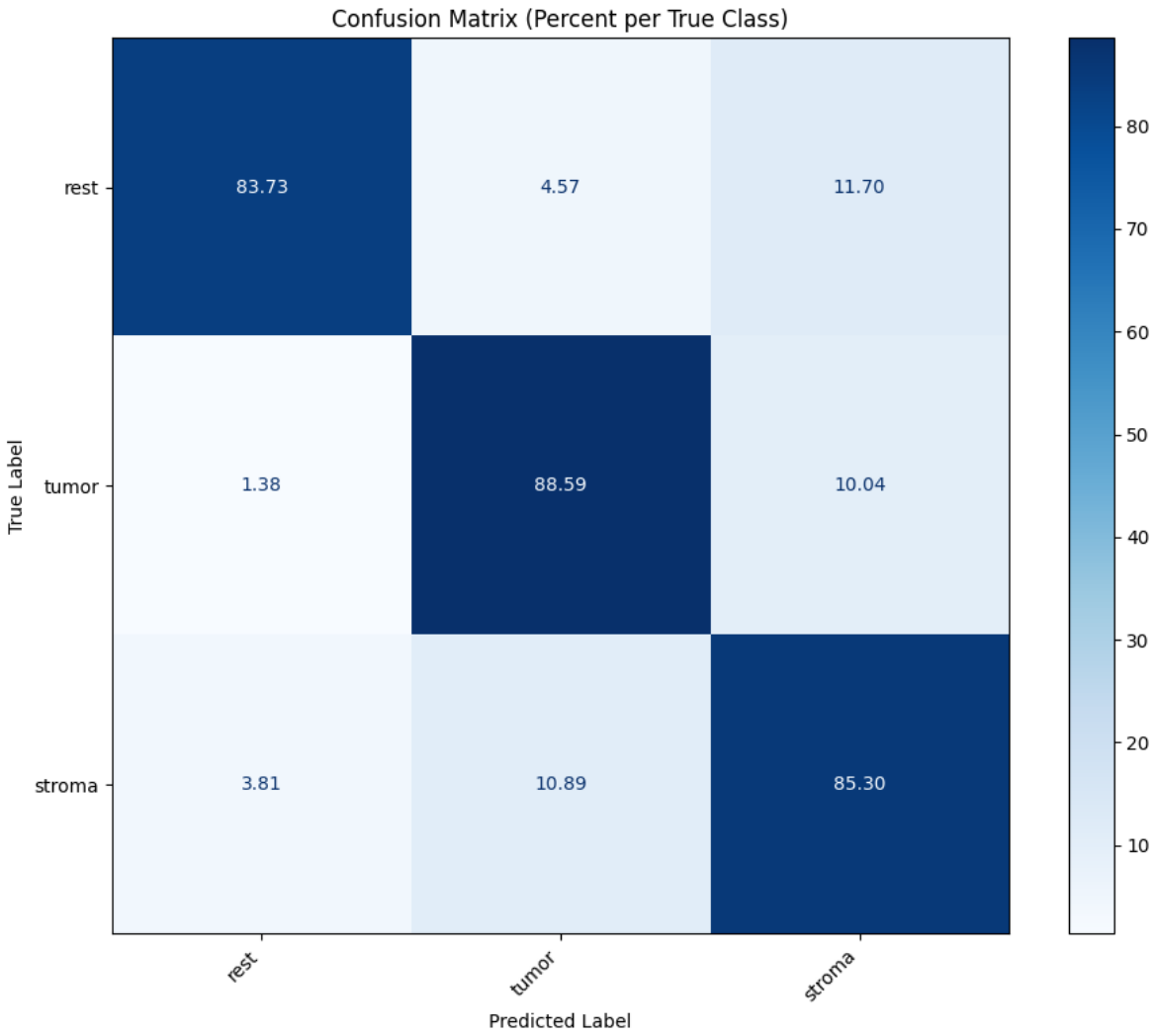

Figure 10: Confusion matrix in percents (per pixels)

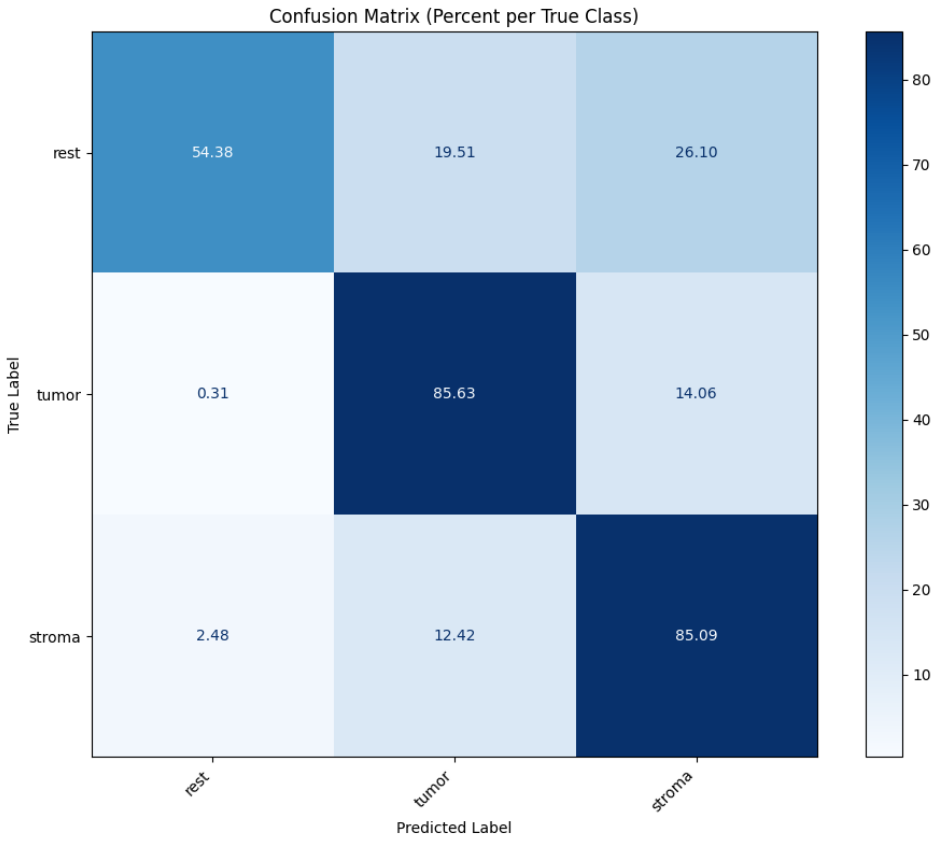

Figure 11: Confusion matrix in percents (per pixels) for only non-homogenious patches

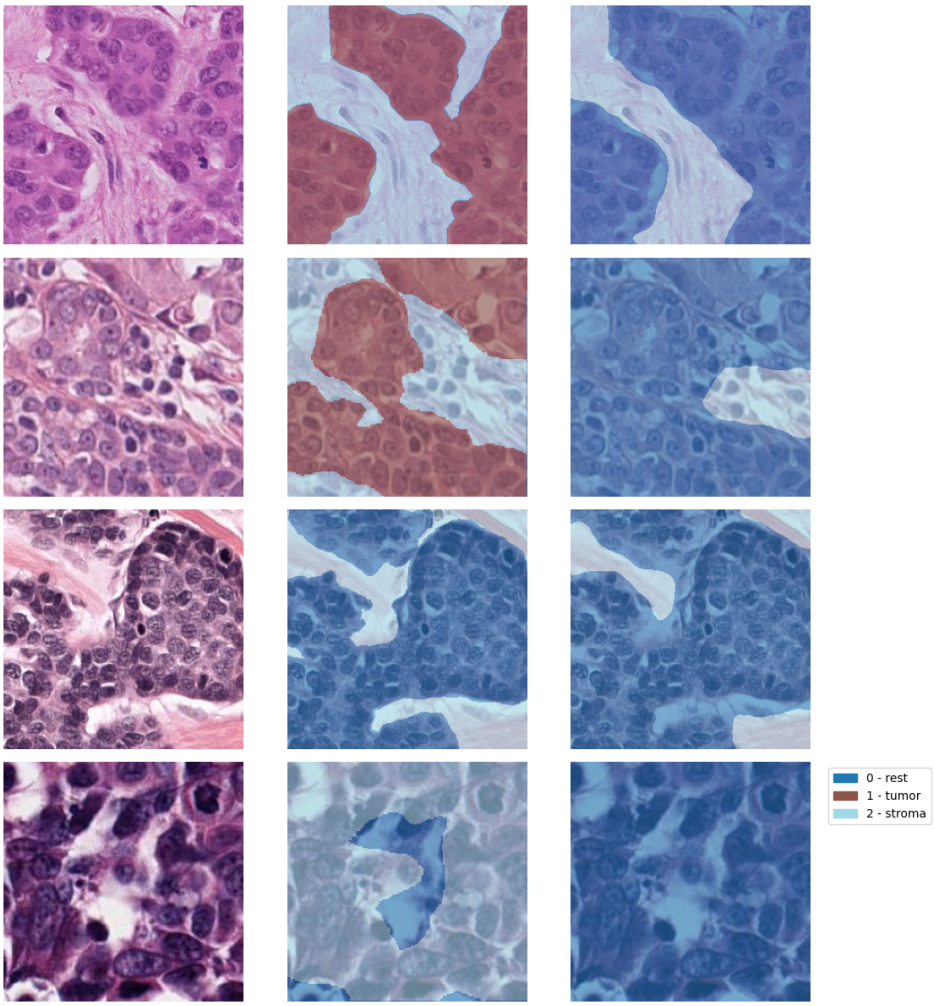

Figure 12: Example output on validation set

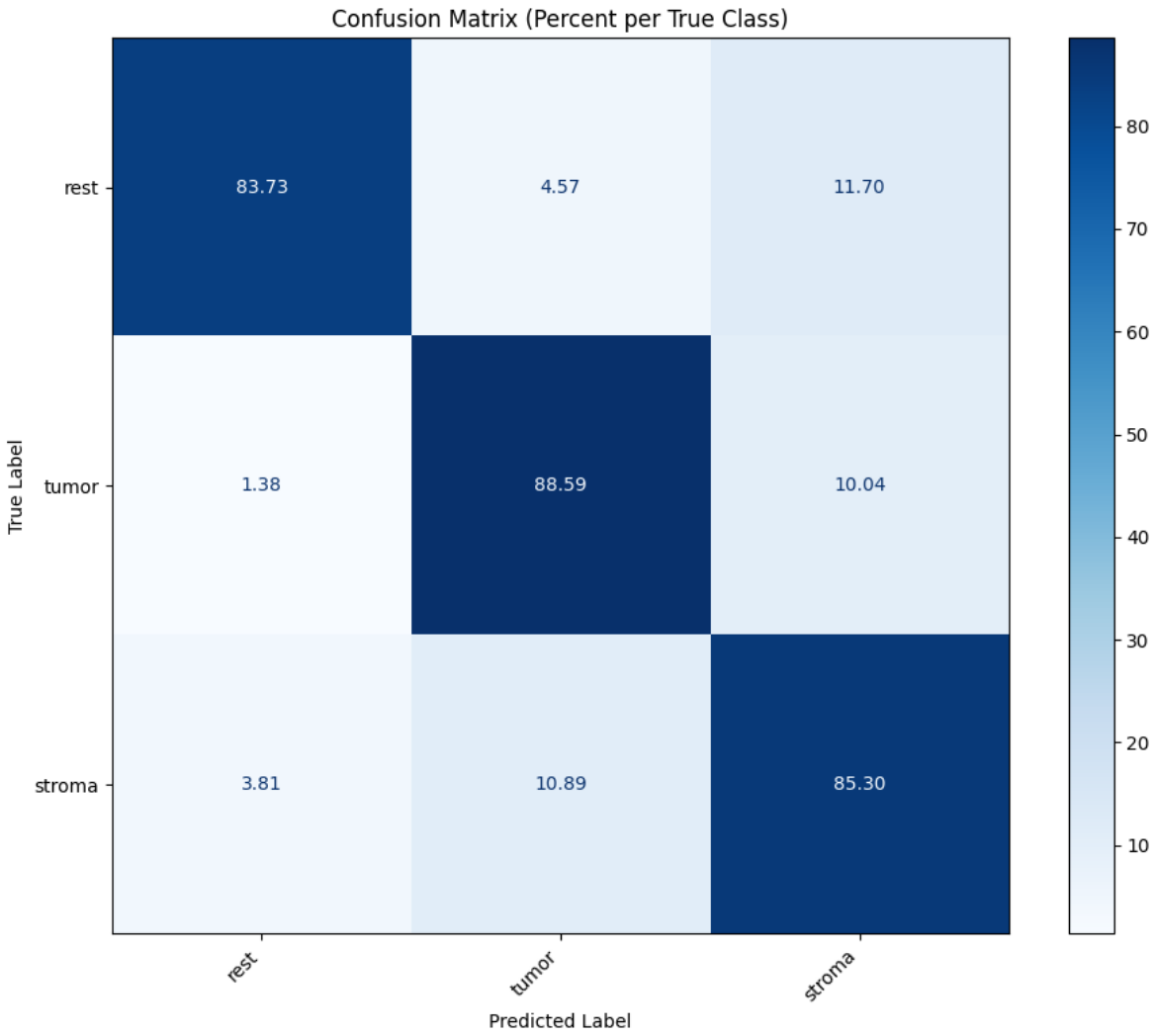

Figure 13: Confusion matrix in percents (per pixels)

6 Modified Dataset: Unicorn Challenge Task 9. Attention U-Net

Report is available here: Report (contains also future experiments)

As a part of this experiments we also tested training only on non-homogenious patches (patches that contains several classes) because there is a big amount of patches that contain only one class, that makes our model to do classification task rather than real segmentation.

6.1 Model Architecture

For these experiments we used layers from a pretrained version of EfficiientNetB0 for encoder of the U-Net and custom decoder with applied attention mechanism.

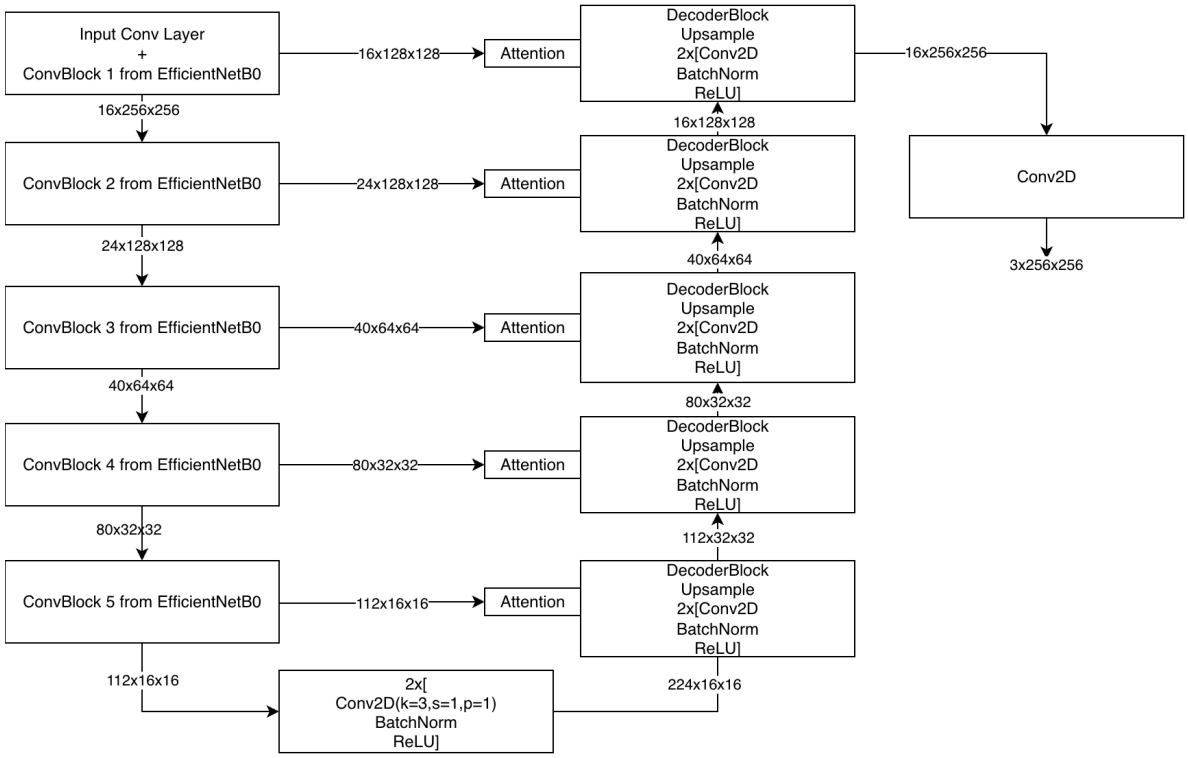

Figure 14: Confusion matrix in percents (per pixels)

6.2 Hyperparameters

The best model uses EfficientNetB0 pretrained layers for the encoder part and custom decoder trained from the ground up (look at Figure 8 for architecture) with the following hyperparameters, defined as constants in the implementation:

Table 5: Hyperparameters of the Model

Parameter	**Value**
Batch Size	32
Learning Rate	0.001
Maximum Epochs	50
Reduce LR Patience	10
Early Stopping Patience	10
Loss function	JaccardLoss(”multiclass”, [0,1,2])
Image Size	256x256

Loss function is Jaccard Loss (class is imported from segmentation_models_pytorch library) The model applies a argmax activation to the output logits during training and evaluation to produce probabilities for each class in the output mask.

Key takeaways from this group of experiments:

- Image size can seriously influence the performance
- Original Dataset is imbalanced
- In out case its makes sense to use pretrained backbone

6.3 Results

Class	Count
Loss:	0.24308
IOU	0.57851
Dice Score	0.71044

Table 6: Best metrics values achieved on validation set

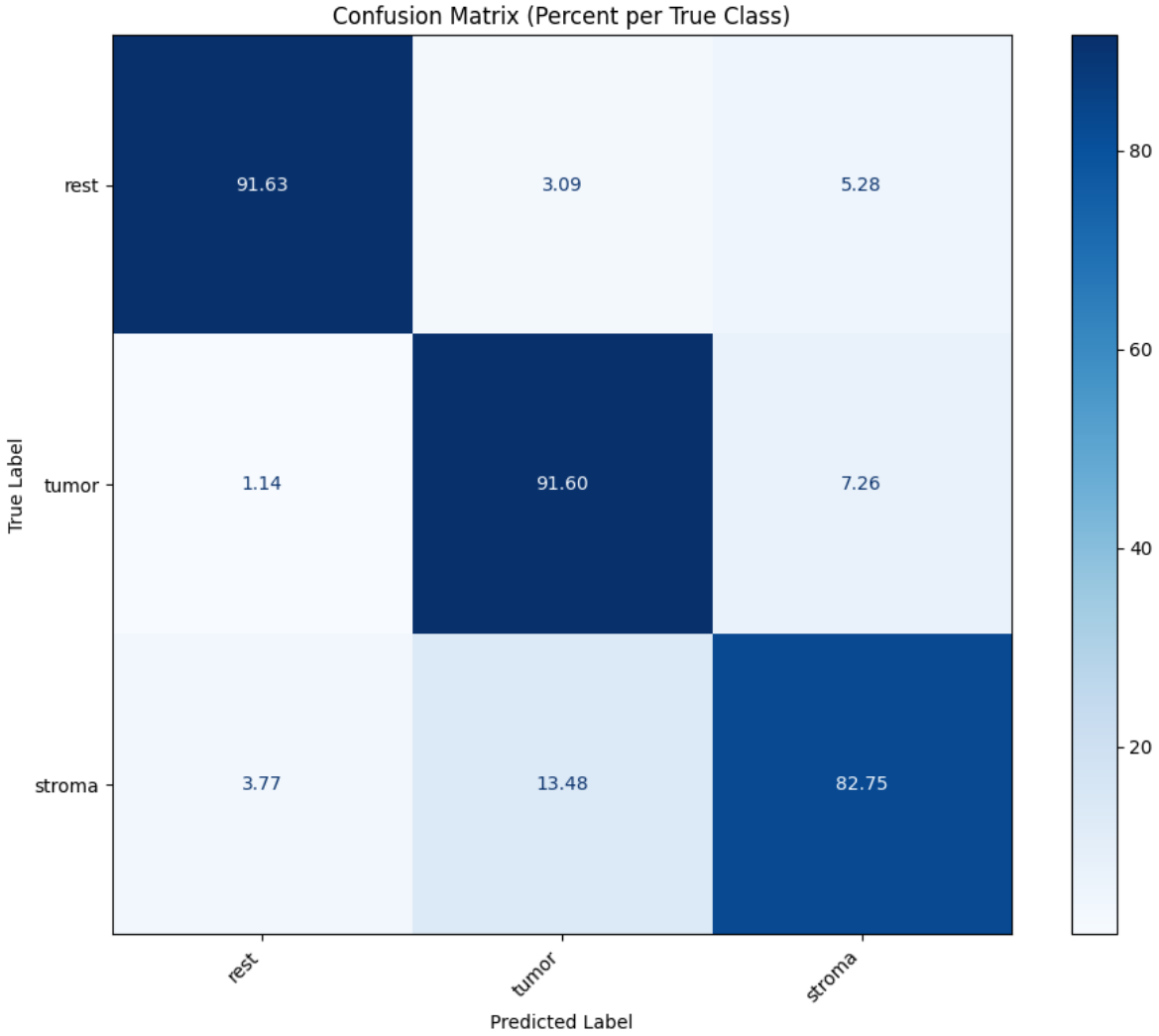

Figure 15: Confusion matrix in percents (per pixels)

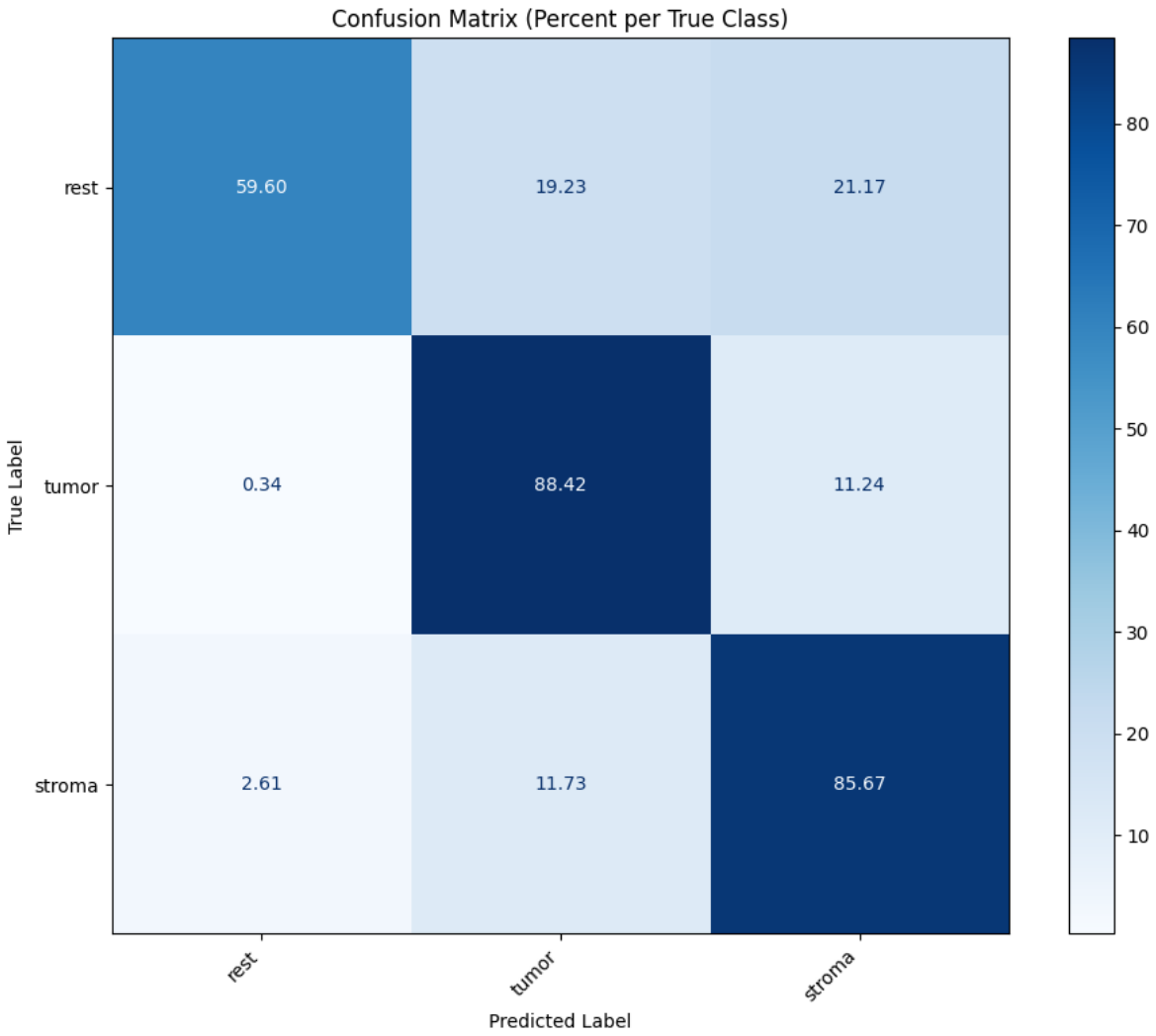

Figure 16: Confusion matrix in percents (per pixels) for only non-homogenious patches

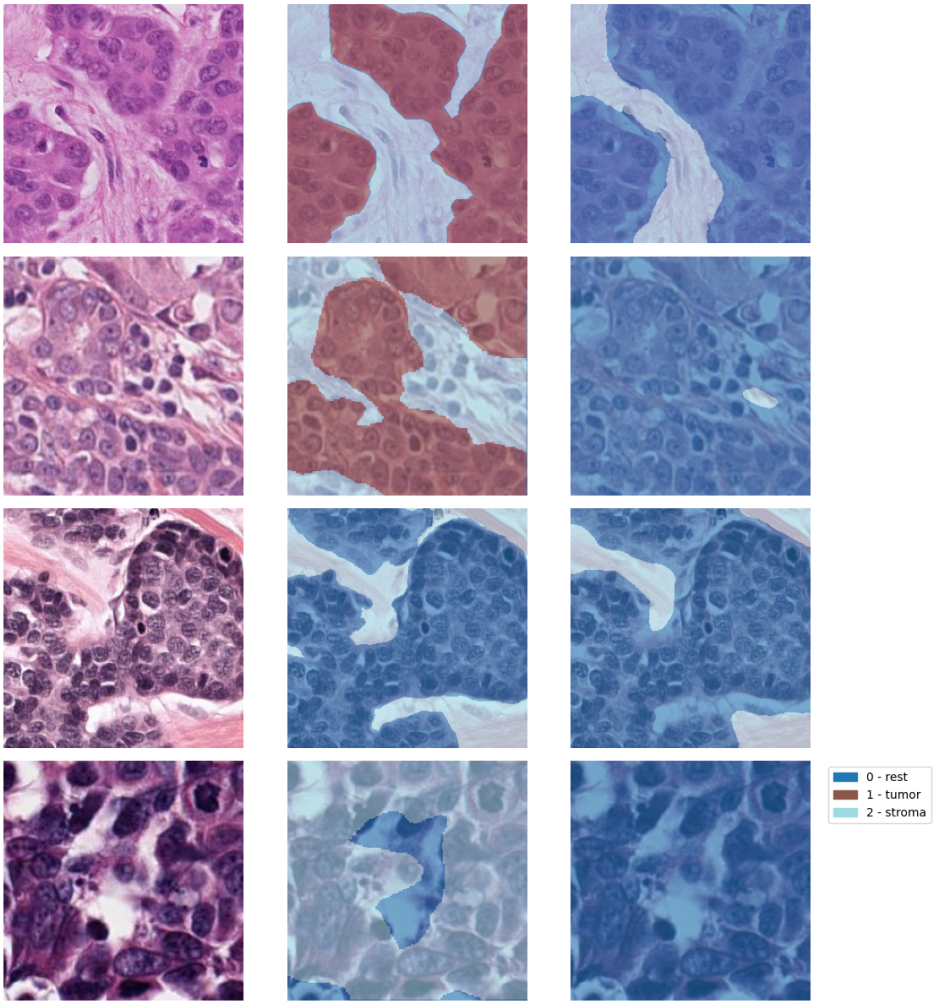

Figure 17: Example output on validation set

7 Final Analysis

We have applied GradCAM algorithm to identify whether the networks is able to extract respective areas in the images. As we can see in some cases we the network correctly identifies entitties in the image

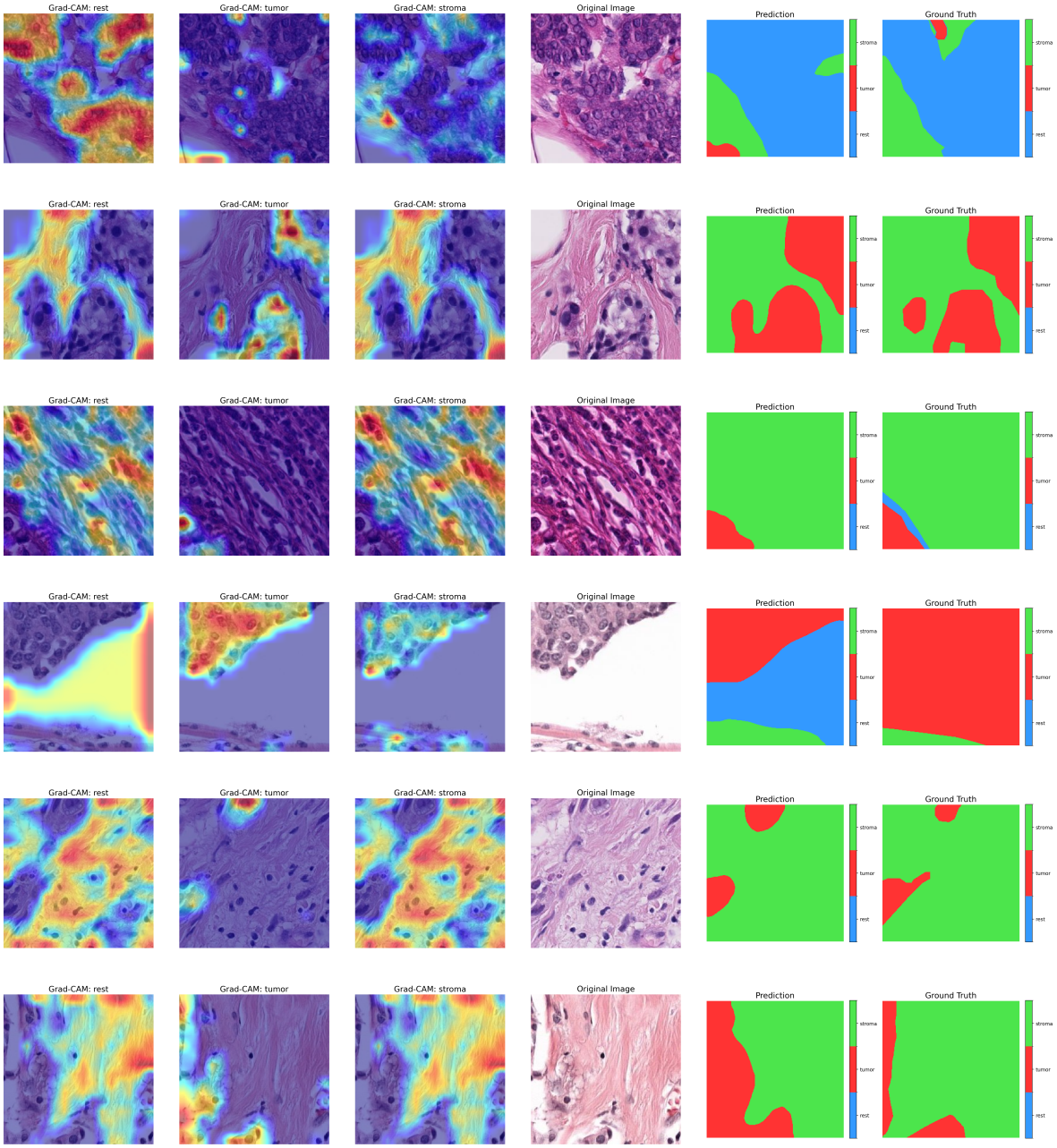

Figure 18: GradCAM of Attention U-Net