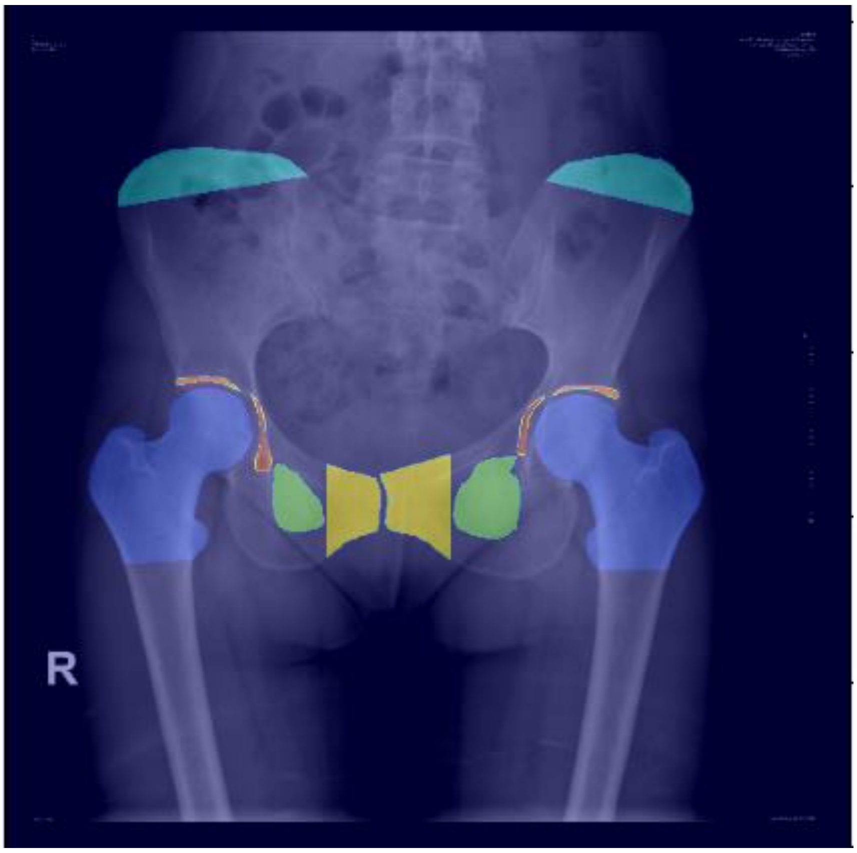
OPENCAMPUS.sh

Segmentation of Pelvic Bone for 2d X-Ray Images



Submitted By: Dikshyant Acharya

Submitted To: Nicolai Krekiehn and Niklas Koser

Submitted on: 06.02.2024

Course: Machine Learning für Medizin (OpenCampus Kiel)

Introduction

This project was completed as part of the "Machine Learning für Medizin" module, mentored by Nicolai Krekiehn and Niklas Koser at OpenCampus. The primary objective of the module was to segment multiple bones in the pelvic region. The project was for approximately three months, during which discussions were held weekly among three students (including myself) and two mentors. Each participant implemented their ideas and experiments, utilizing the same transformer model but with different segmentation models and hyperparameters.

The datasets were already available on the following site: https://universe.roboflow.com/ks-fsm9o/pelvis-ap-x-ray, where the training, validation, and testing datasets were readily accessible. This research is divided into the following sections:

1. Analysis of the original datasets
2. Selection of a transformer model for image augmentation
3. Selection of a segmentation model
4. Training of the model
5. Evaluation Metrics
6. Results
7. Discussion and Conclusion

1. Analysis of original datasets

The dataset was downloaded and extracted, creating dictionaries for the training, testing, and validation files. Each dictionary contained images as keys and labels as values. In total, there were 320 training images, 91 validation images, and 47 testing images. All data were in “.nii.gz” format. Using the "nibabel" Python library, the datasets were converted into "Nifti1Image" format and then into "ndarray", such that it can be processed further for the image augmentaiton . Final image of the dataset can be seen as:

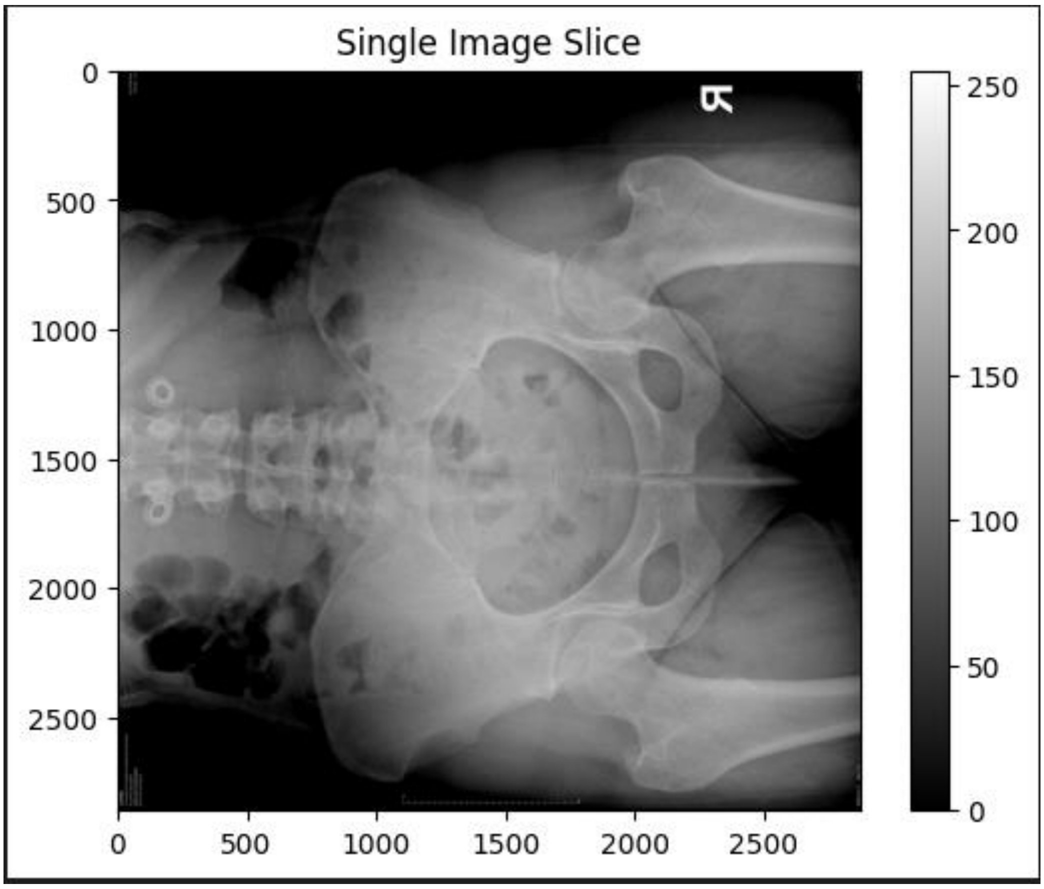


Figure 1: Single Image data in the dataset

The images varied in size, with dimensions ranging approximately from 2200x3000 to 1800x3000 pixels, as demonstrated in a bar diagram of image widths and heights.

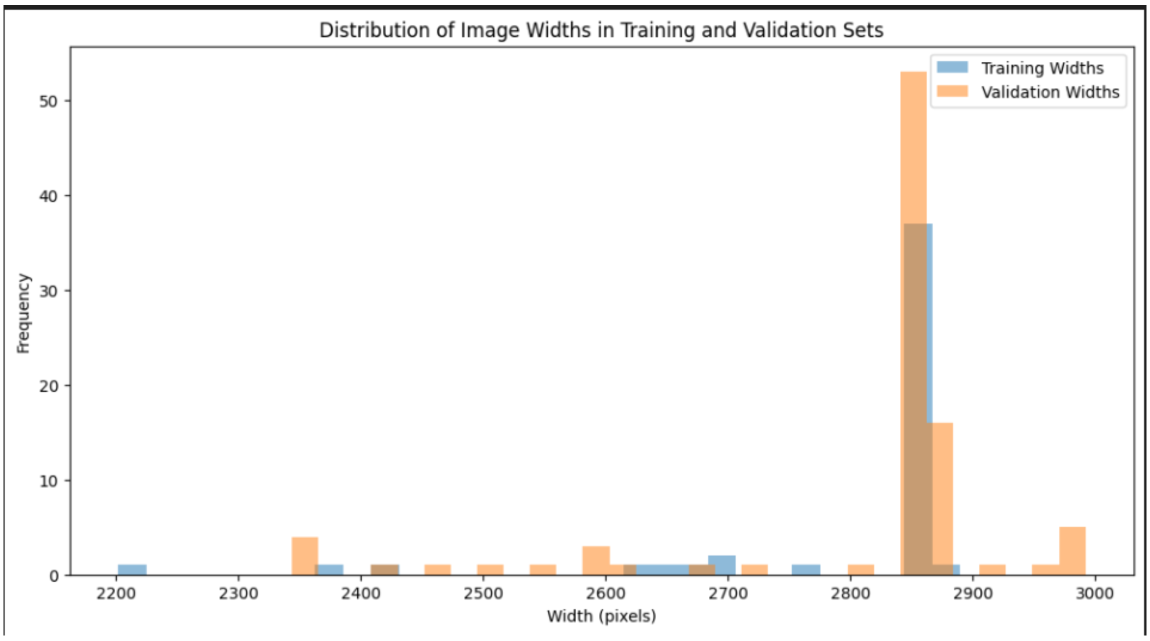


Figure 2: Different size of width of image in the dataset

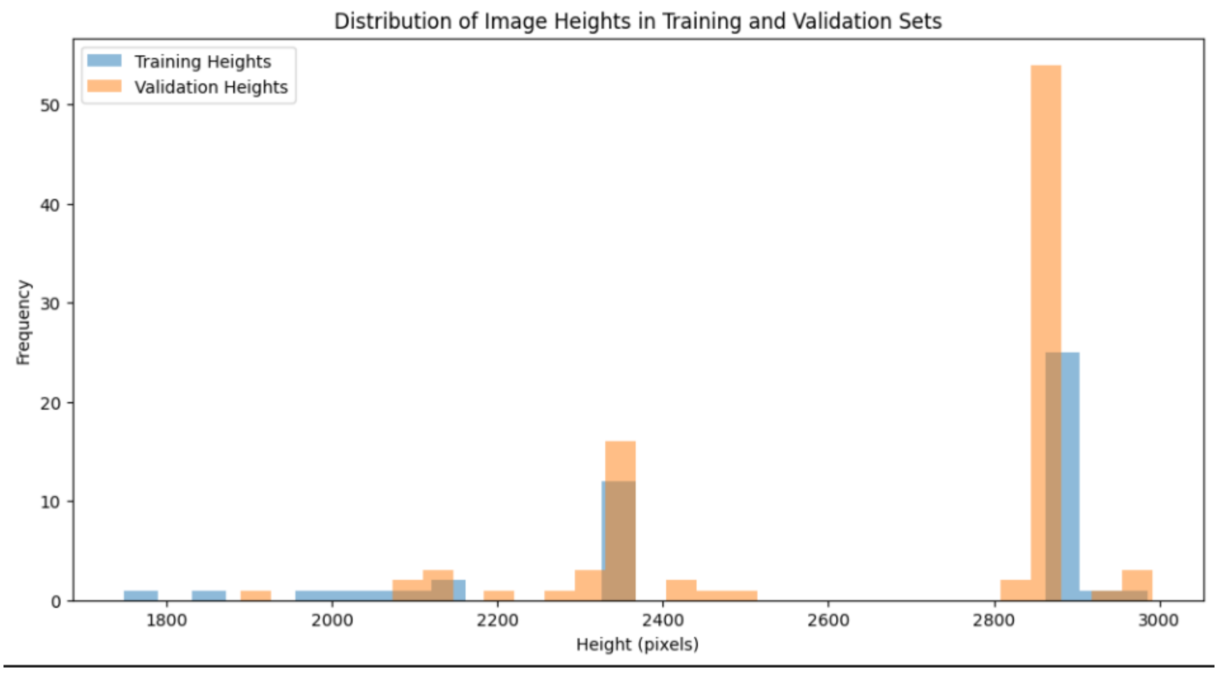


Figure 3:Different size of height of image in the dataset

Both labels and original image was printed where the labels were overlapped with the original image. And the following image was observed (excluding the arrow and number). The image is rotated for this diagram to have better illustration.

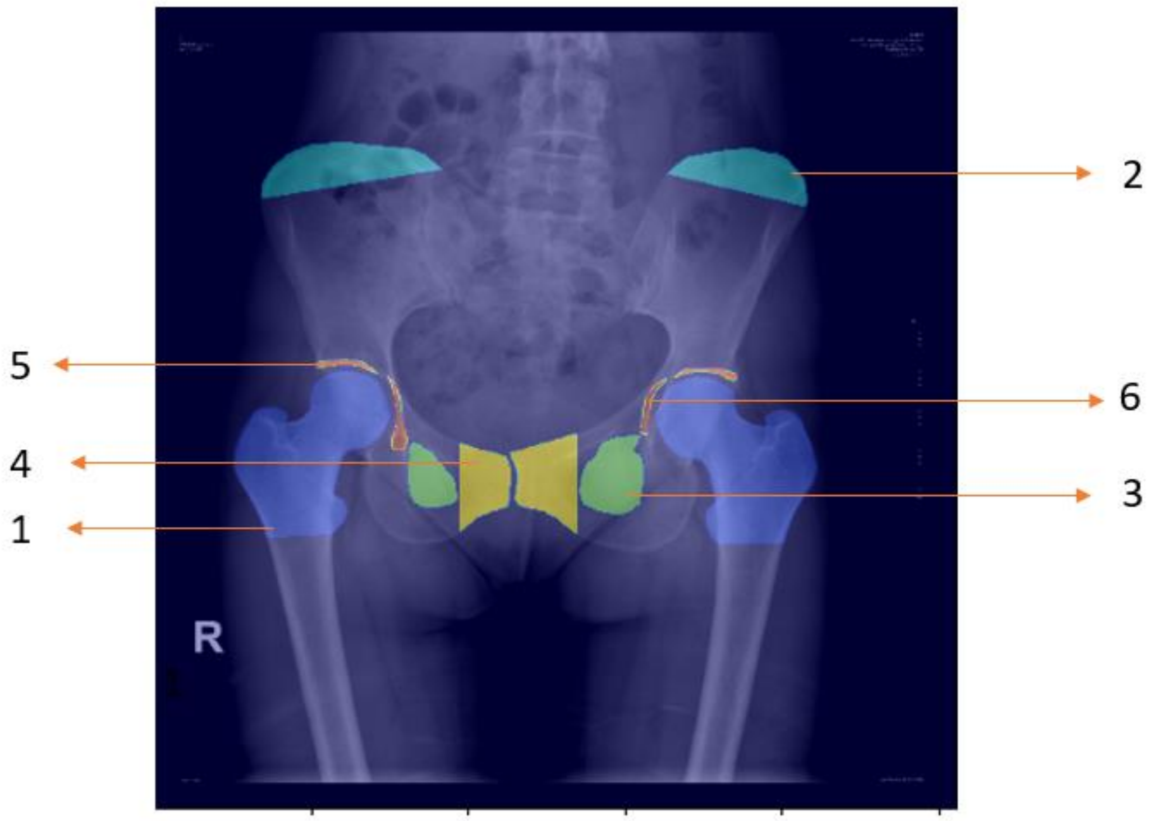


Figure 4: Training Set Image where labels (1 to 6) were mapped above original image

Upon close examination of the dataset, issues were found with the segmentation accuracy; notably, one training image had incorrect bone segmentation, and approximately 35 images in the training and testing datasets had inconsistent segmentation for bones 3 and 4. Two models are used for segmentation in this project segmenting bone (1, 3, 4) and bone (2, 4, 5) which are discussed later. These images were removed from the training set for training the model that detected bone 1, 3 and 4, improving the model's overall accuracy.

However, bones 5 and 6 presented a challenge due to their small size and inconsistent segmentation across different images. The incorrect labeling of the bone 5 (as in Fig:6) was removed, and then all other remaining datasets were used to train the second model segmenting bone (2, 5, 6). These two bones (5 and 6) proved difficult to segment accurately and yielded the lowest performance according to the dice metric.

Some of the wrong segmentations of images are:

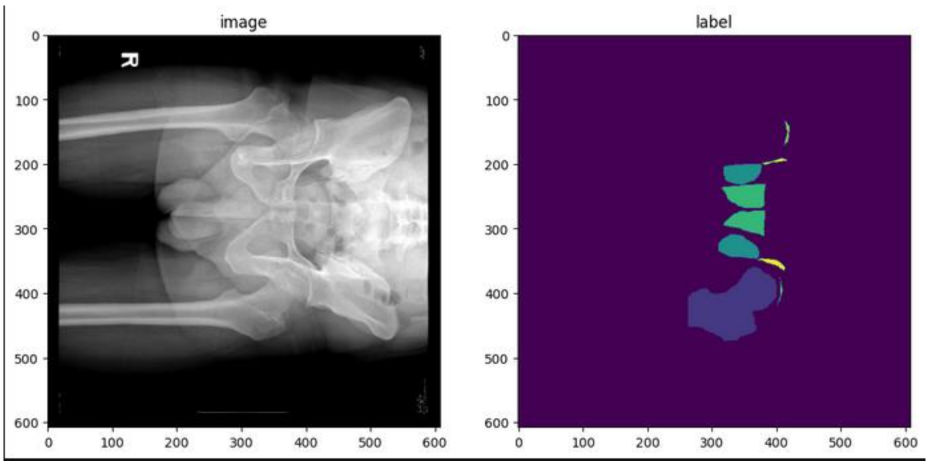


Figure 5: Failure of segmentation of bone 1

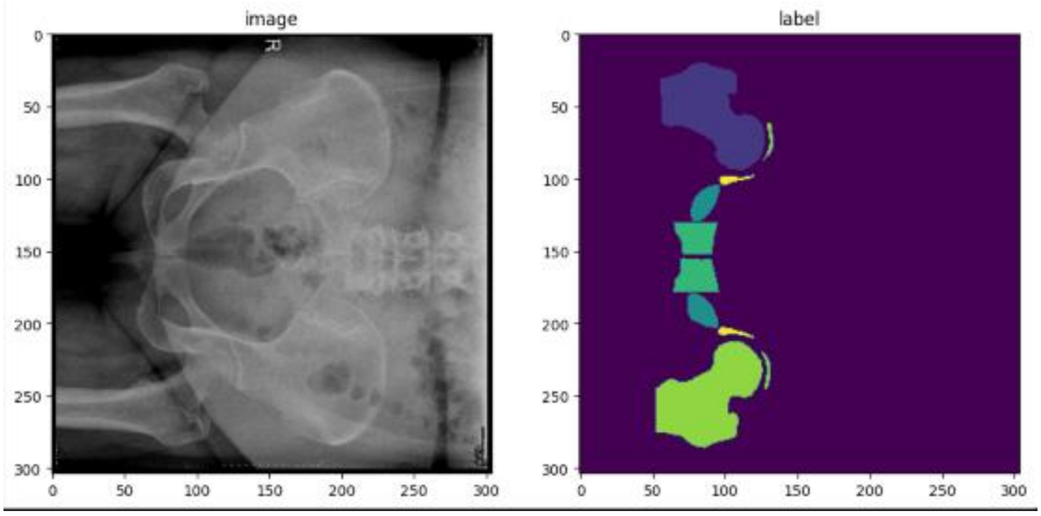


Figure 6: Mislabeling of segmentation of bone 1 as bone 5

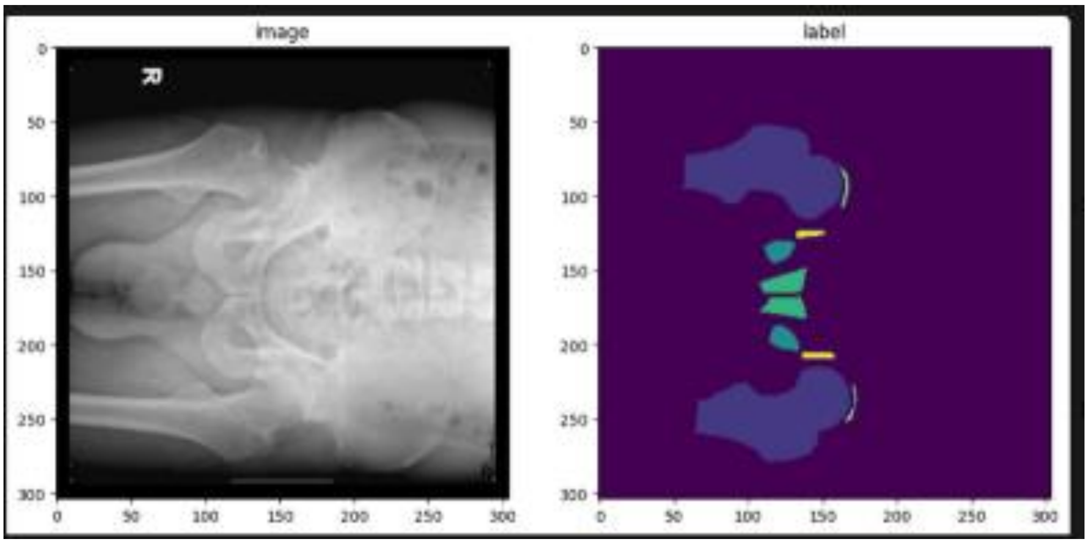


Figure 7: Poor Segmentation of bone 4

Such wrong segmentation (especially for bone 4) was found in the following index of datasets:

Training datasets:
[5, 10, 16, 34, 39, 45, 53, 58, 72, 78, 81, 83, 85, 86, 90, 103, 108, 115, 122, 140, 142, 178, 193, 195, 199, 214, 243, 247, 254, 255, 277, 286, 287, 312, 319]

Validation datasets:
[4, 29, 32, 33, 36, 40, 46, 59, 64, 68]

2. Transformer and image Augmentation

For image augmentation, MONAI (Medical Open Network for Artificial Intelligence) was selected as our transformer tool. MONAI is a free, open-source framework built on PyTorch, designed to enhance medical images through various transformations such as scaling, rotation, noise addition, and padding.

These transformations can be applied sequentially, allowing for comprehensive augmentation of the images to achieve the desired outcomes. The process is helped by the Compose function, which chains callable transformations to process images in a sequence, ensuring each transformation is applied in order.

The transformations used in this project include:

1. LoadImaged:
Loads image data and metadata, handling both images and labels.

2. EnsureChannelFirstd:
Ensures the image channels (e.g., grayscale or RGB) are processed in the format (Color_feature, width, height).

3. Orientationd:
Changes the orientation of the images to align them in a standard orientation. The possible combination can be made from following:
(L, R) -> Left, Right
(P, A) -> Posterior, Anterior
(I, S) -> Inferior, Superior

4. ReplaceValuesNotInList:
A custom function that limits the labels to six, replacing all others with background labels for segmentation purposes. (Mistakenly there were more than 6 lables in some dataset, so this function was necessary. Furthermore, this allowed later to train the model for labels only)

5. SpatialPadd:
Applies padding to standardize image dimensions across the dataset.

6. ConditionalRandCropByPosNegLabel:
Randomly crops images based on label presence, aiding in feature learning without processing large images.

7. Spacingd:
this reduces the size of images as given in the pixdim. If the image is (100, 100), and I have a pixdim as (10, 10), then the image that I will get at then will be (10, 10) size. This reduces the image size, thus reducing the computational time required.

8. DivisiblePadd:
Ensures image dimensions are divisible by 16, compatible with the U-Net Architecture.

9. RandRoatated:
Rotates images to diversify the dataset and improve model learning. Probability of occurring this transformer was set to 0.6.

10. Rand2DElasticd:
Distorts images to introduce variability in label shapes.

11. ScaleIntensityRanged:
Normalizes pixel values from grayscale to a range between 0 and 1.

12. RandGaussianNoised:
Adds random noise to images to enhance feature learning in unclear images.

All the above augmentations were not done instantly, but rather were added one after another in an attempt to improve the accuracy of the model. These were the best series of augmentations, that gave better accuracy after multiple tryouts.

After these augmentations, some of the output images looked like this.

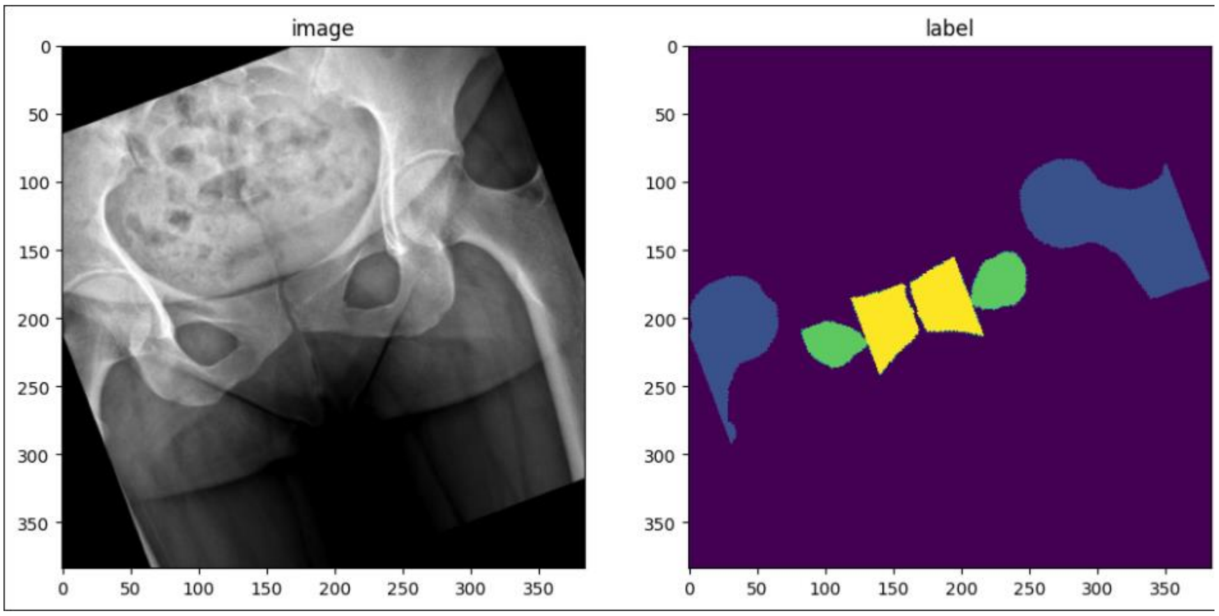


Figure 8: Transformed image using transformer of: 1_134.ipynb

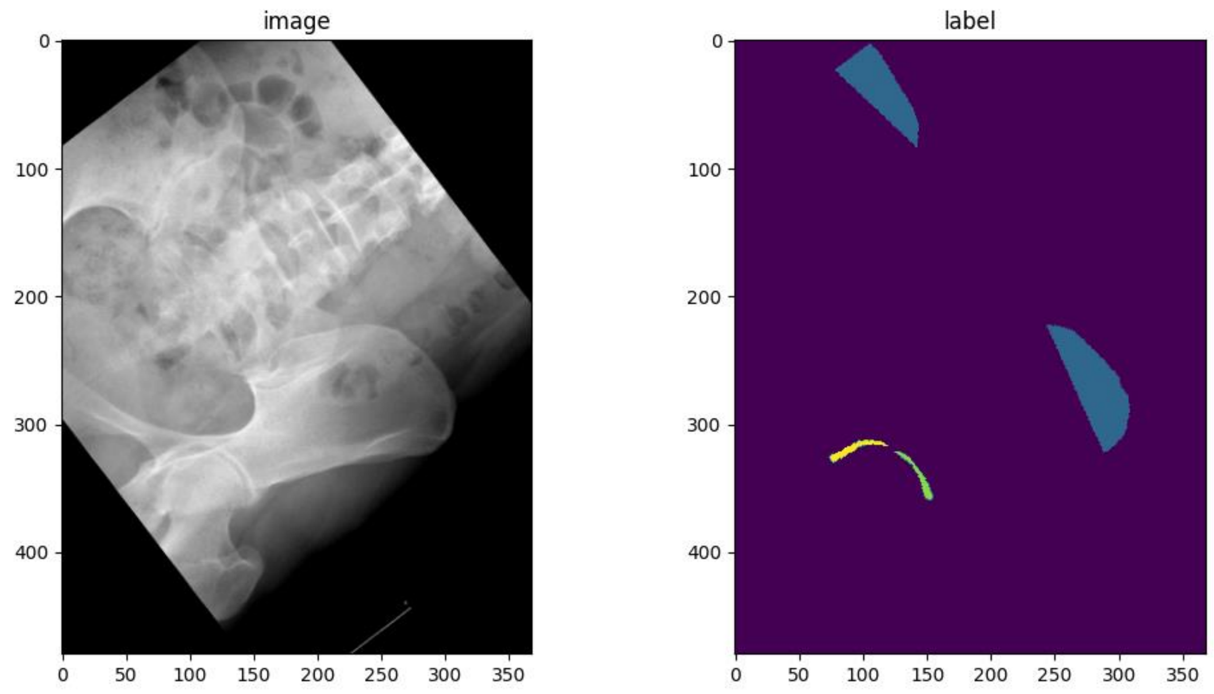


Figure 9: Transformed image using transformer of: 1_256.ipynb

In the above 2 diagrams, we can see the orientation, replacement of values (allowing only certain labels), spatial padding, cropping, spacing, divisible padding, rand rotation andrand2delastic in action. The Gaussian noise is also applied, but due to probability of occurring, above image didn't get the noise.

3. Choice of Segmentation Model

A U-Net model was chosen for this project. A transformer would have been great for this purpose, but the datasets were not sufficient. And for small medical datasets, U-Net was the best option. More information about detail of the U-Net and how it works are available in this link [https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/].

For a U-Net architecture, several parameters needed to be chosen. Some of them are:
spatial_dimenstion: This is the dimension of the input data, which is 2 in our case.

in_channel: Number of channels for input data, which is 1

out_channel: Number of output channel. This should align with the number of clases of segmentation.

channels: These are the tuples that define the number of channels at each level of encoder and Decoder.
strides: It control the down sampling factor at each level. We are halving the dimensions at each level.

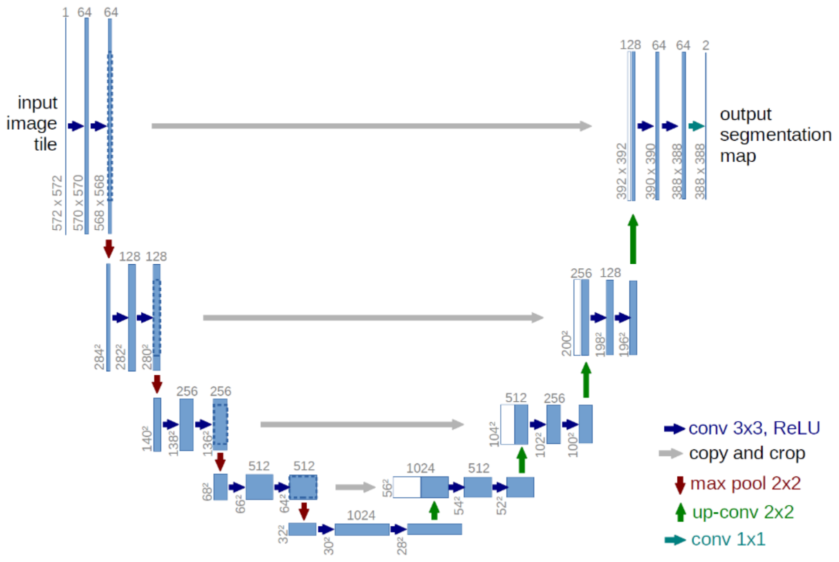


Figure 10: U-Net   https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/u-net-architecture.png

4. Training of the model

The M1 Pro was utilized with its Metal Performance Shaders ("MPS") for training the model. The training time depended on two main factors:

1. Image size
2. Channels of UNet Model

The fastest training occurred with images of around 300x300 pixels and channels ranging from 16 to 256. However, the disadvantage was that, the model could not capture the features of the bones, thereby failing to detect bone segmentation. Increasing the pixel size to 400x400 increased the computational time but still did not achieve the desired accuracy, which was around a 0.75 Dice coefficient. Increasing the channels from 32 to 512 proved to be the best for detecting the required features from the image but came at a computational cost. It took about whole day (10 hours) to run epoch of 100 for the channels (32 to 512) with pixel size (400 * 400). So, it was bit difficult to try hyperparameter tuning.

5. Evaluation Metrics

For the evaluation, training was conducted on one set of data, and the model was evaluated at regular epoch intervals. The model was immediately saved if its accuracy exceeded that of the previous version. The validation set consisted of data that our model had never seen before. Both the overall average Dice coefficient and the individual Dice coefficients for each bone were analyzed. For image evaluation, the images were resized to approximately 600x600, and a sliding window with the size of the training images (300x300) was used to obtain the segmentation of the image.

6. Results

Initially, achieving a Dice coefficient of 0.6 was easy with random parameters in the transformer, such as training image size, and using various channels for the transformer and U-Net models. However, it became increasingly challenging to enhance the accuracy. Following parameters were crucial in improving accuracy:

1. RandRotaion:
This transformer increased the dice metric where the accuracy was then observed to be from 0.6 to 0.67

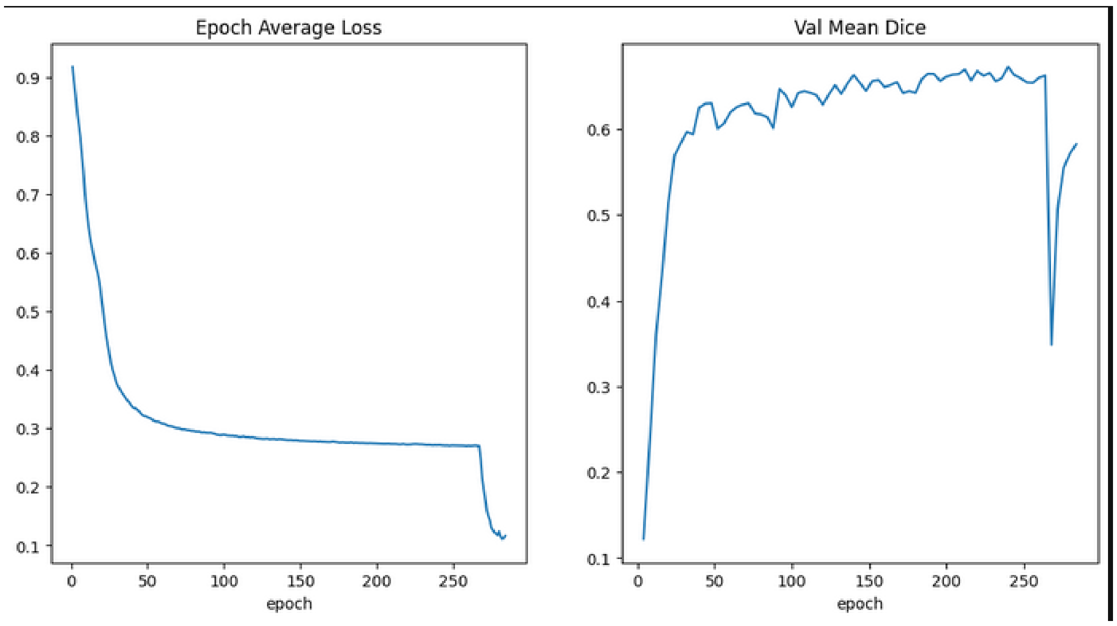


Figure 11: Loss (in training set) and Dice value (in validation set) during training

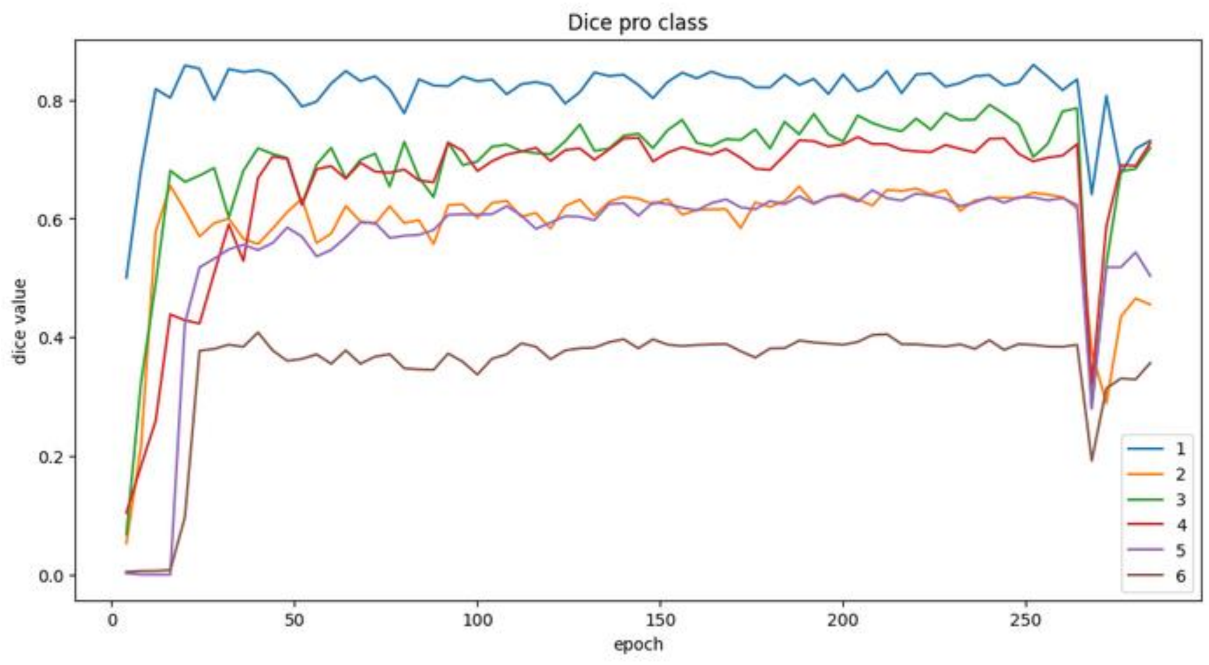


Figure 12: Dice value of each class during training

2. Using RandCrop:

This transformer was key in increasing the Dice coefficient from 0.7 to approximately 0.82. During this phase, bones 1, 3, and 4 achieved a Dice coefficient of about 0.93, while other bones had lower Dice coefficients—approximately 0.82 for bone 2, and around 0.65 for bones 5 and 6.

No amount of parameter tuning was able to increase Dice coefficient beyond 0.82. Different learning rates were tried (10^-3 and 10^-4), along with the addition of schedulers under different conditions, but these adjustments did not give better results. Consequently, a decision was made to use two separate models for image segmentation since bones 1, 3, and 4 were already achieving higher Dice values and the others were not scoring good.

So, a similar transformer will be used for the segmentation of the bone 1, 3, 4, where it achieved higher dice value as above and hyperparameter tuning will be done to enhance the Dice score. And another model will be used to segment the rest, where different transformation models can be tried out specifically targeting the bones 2, 5, 6.

The Rand Crop transformer played a crucial role in increasing accuracy. The challenge was that the large image size around 1000x1000 were impractical for training. Therefore, Rand Crop was utilized to generate randomly cropped images of 350x350.

This transformer allowed for the selection of the probability of cropping images in such a way that the labels would be present or absent. This feature was essential because setting the probability of positive (labels appearing in the cropped image) to 1 and negative (labels not appearing) to 0 meant that all images in the training set contained labels. This approach led to a scenario during sliding window inference, where the model attempted to segment bones that were not present, having learned that every image contains a segmentable object. To address this, a balance between positive and negative probabilities was established to ensure the model could accurately detect the required bones.

Two different transformers were used, and the data was trained separately for different sets of bones. One model, dedicated to the detection of bones 1, 3, and 4, achieved an overall accuracy of about 0.95. Another model, focusing on bones 2, 5, and 6, reached a Dice coefficient of 0.74 with hyperparameter tuning. This dual-model approach resulted in a slight improvement compared to training a single model for all bones.

Model for detection of Bone 1, 3 and 4

The parameter for transformer and model can be found in the notebook: 1_134.ipynb
Average Mean Dice of 0.9522 in epoch 96 was found.

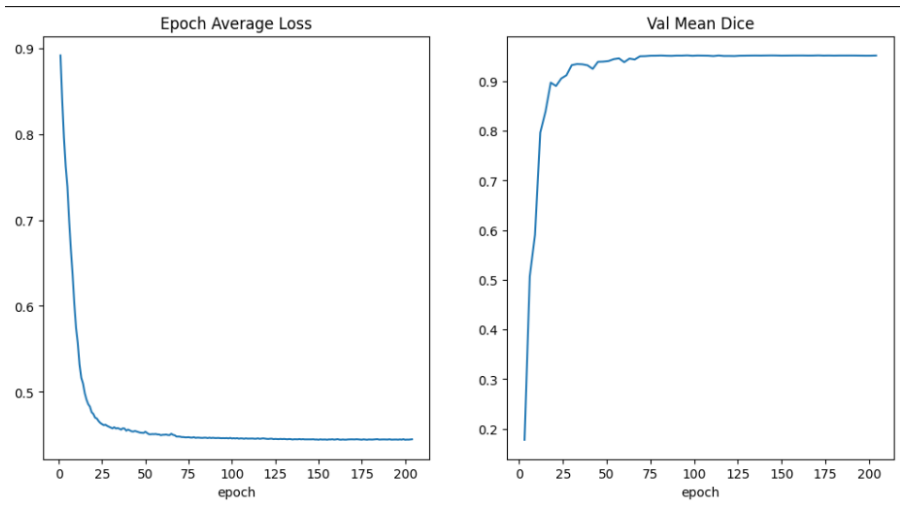


Figure 13: Training loss and Validation Dice Value for bone 1,3 and 4

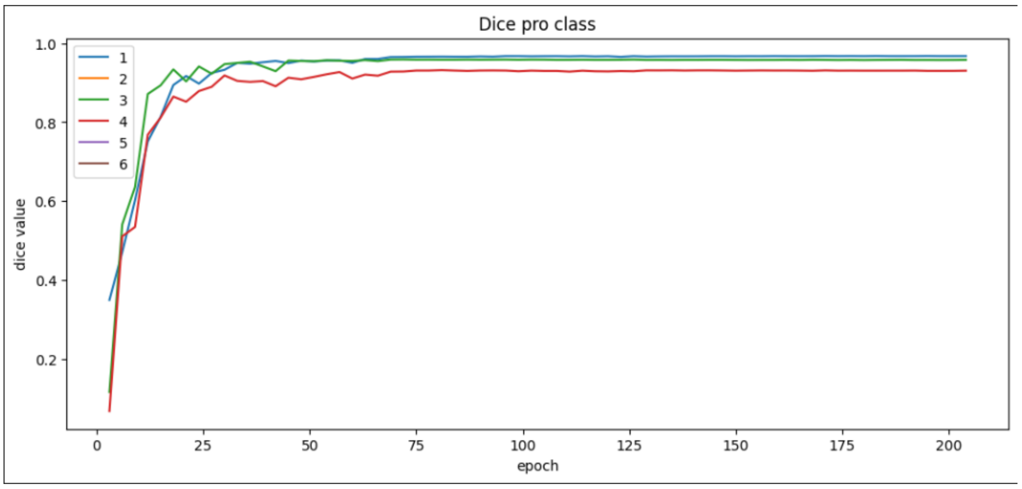


Figure 14: Dice value of each class in validation dataset measured during training

Prediction of segmentation using the best trained model:

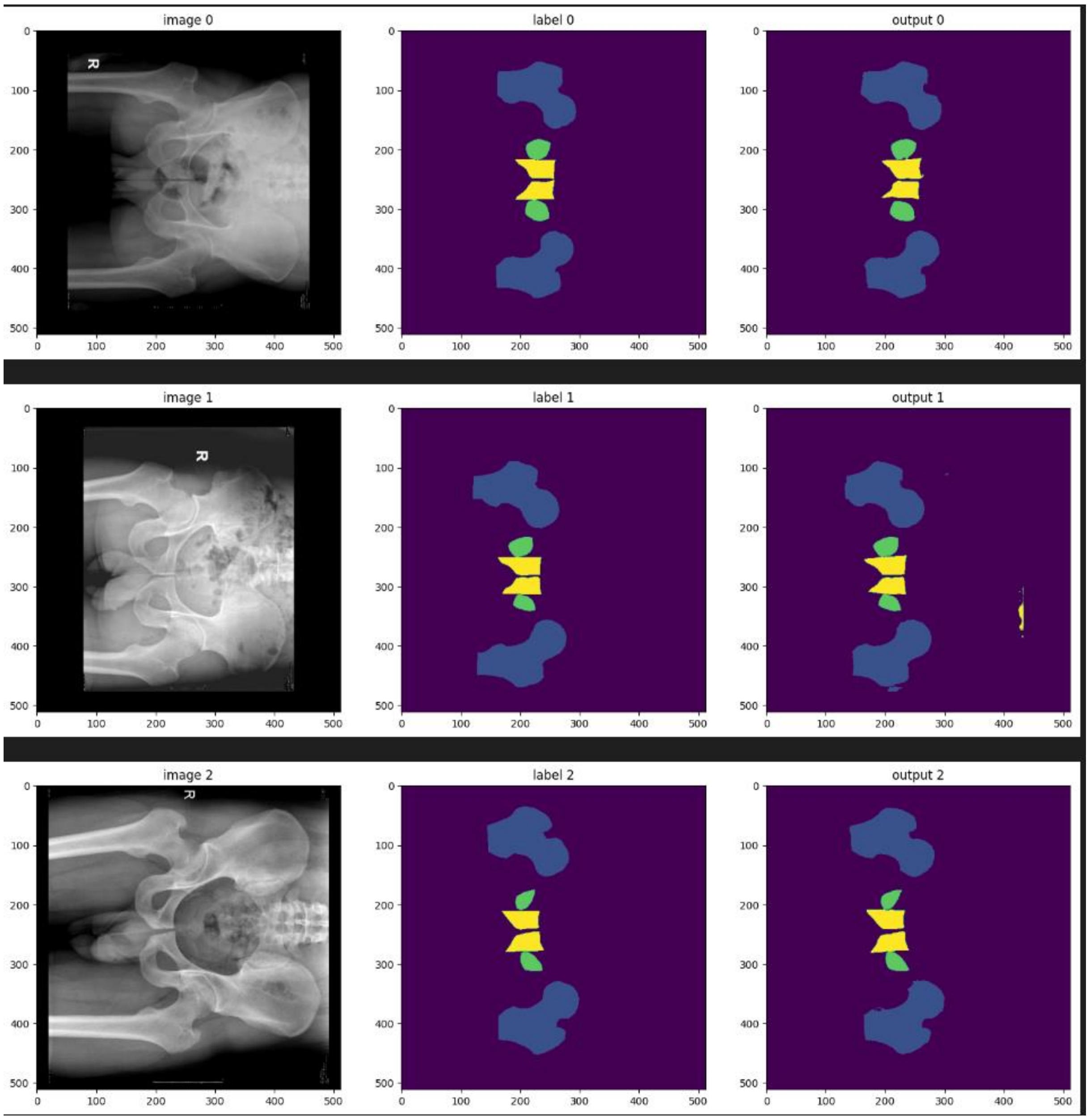


Figure 15: Prediction of segmentation using the best trained model

Overlapping view of original image with predicted segmentation over it

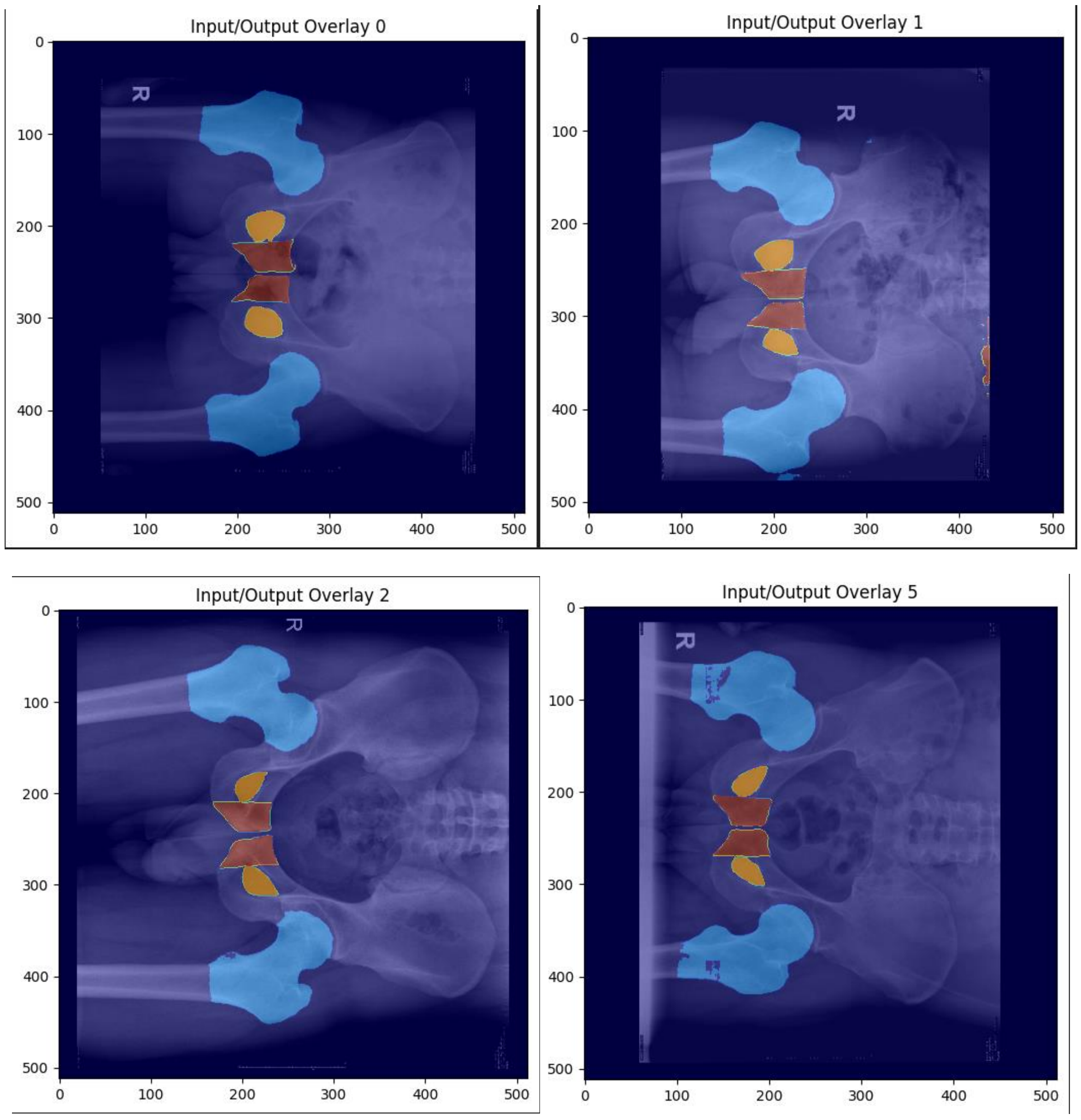


Figure 16: Overlapping view of original image with predicted segmentation over it

Model for detection of Bone 2, 5 and 6

The parameter for transformer and model can be found in the notebook: 1_256.ipynb
Average Mean Dice of 0.7454 in epoch 156 was found.

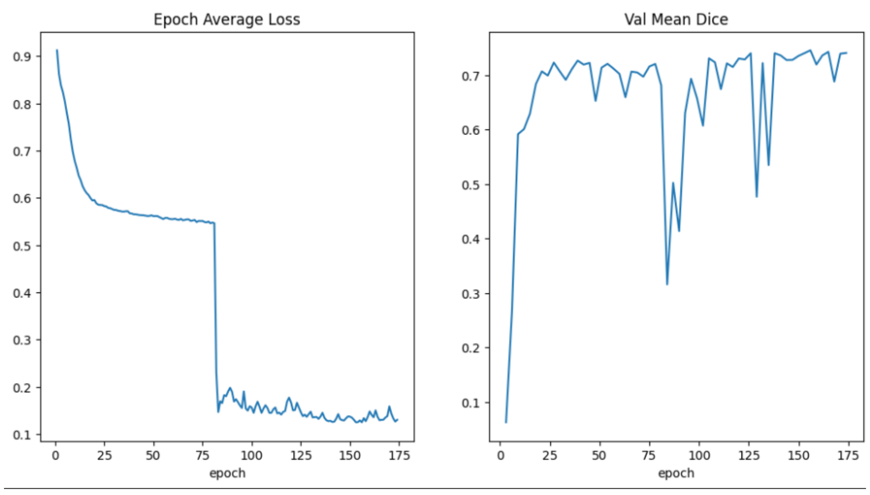


Figure 17: Training loss and Validation Dice Value for bone 2,5 and 6

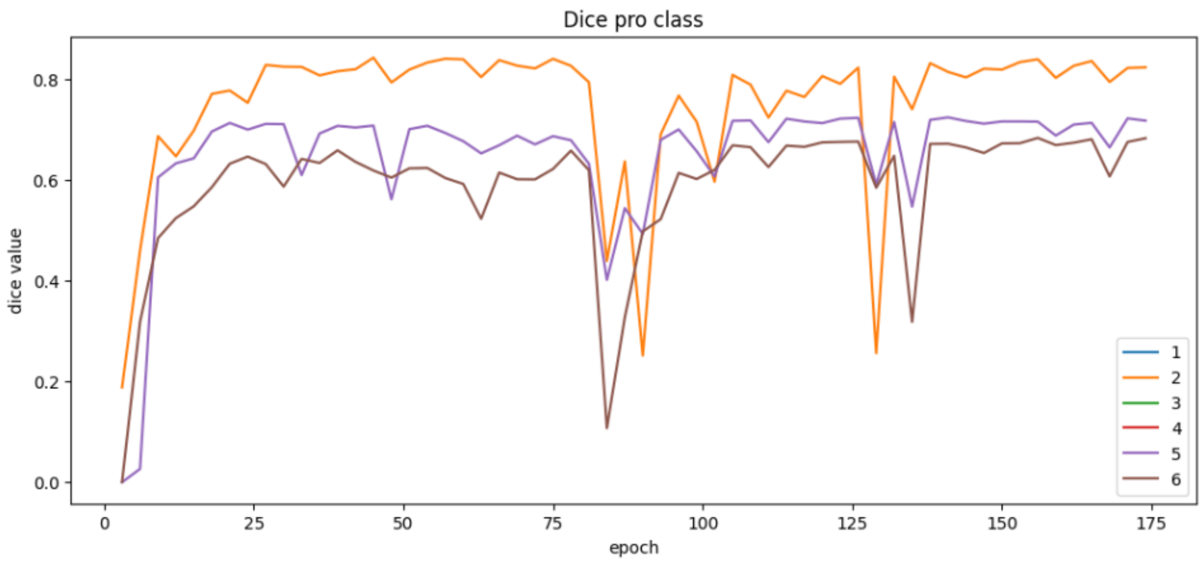


Figure 18: Dice value of each class in validation dataset measured during training

Prediction of segmentation using the best trained model:

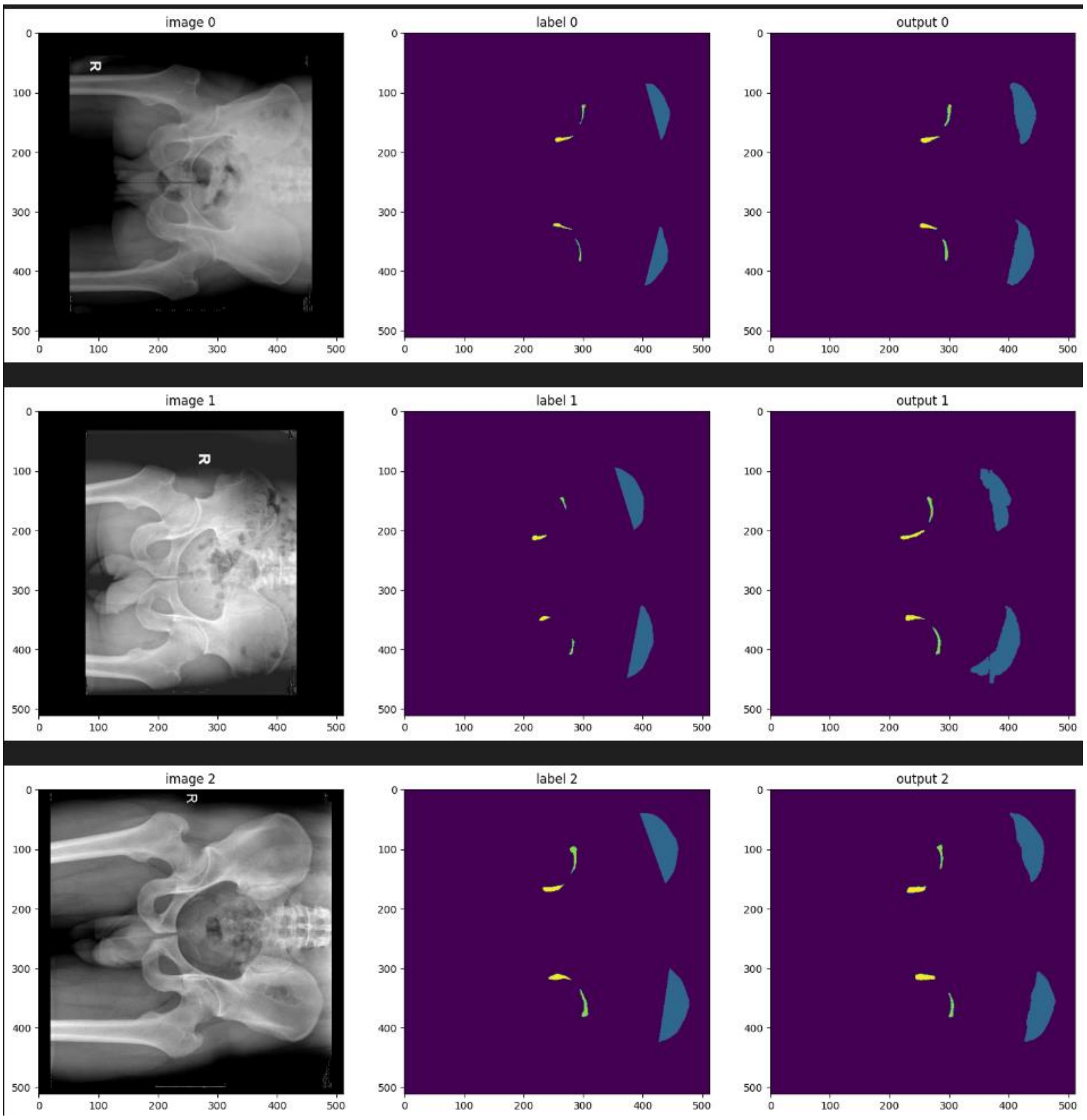


Figure 19: Prediction of segmentation using the best trained model

Overlapping view of original image with predicted segmentation over it

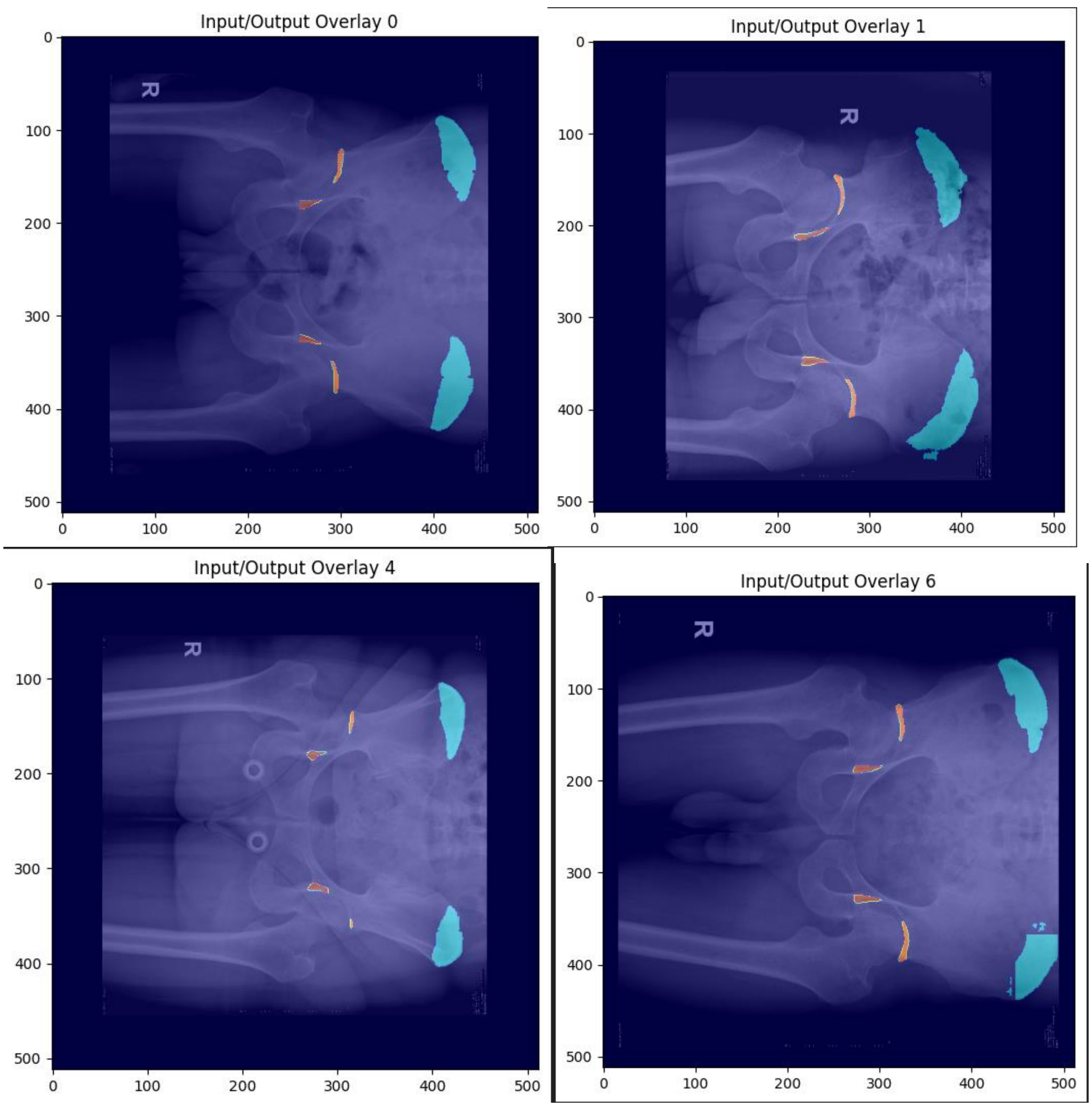


Figure 20: Overlapping view of original image with predicted segmentation over it

Combination of both model for a single image:

When combining both model, an average of both 0.8488 was to be expected. The combination of the both model is in the notebook 1.combined.ipynb file

Prediction of segmentation using the best combined trained model:

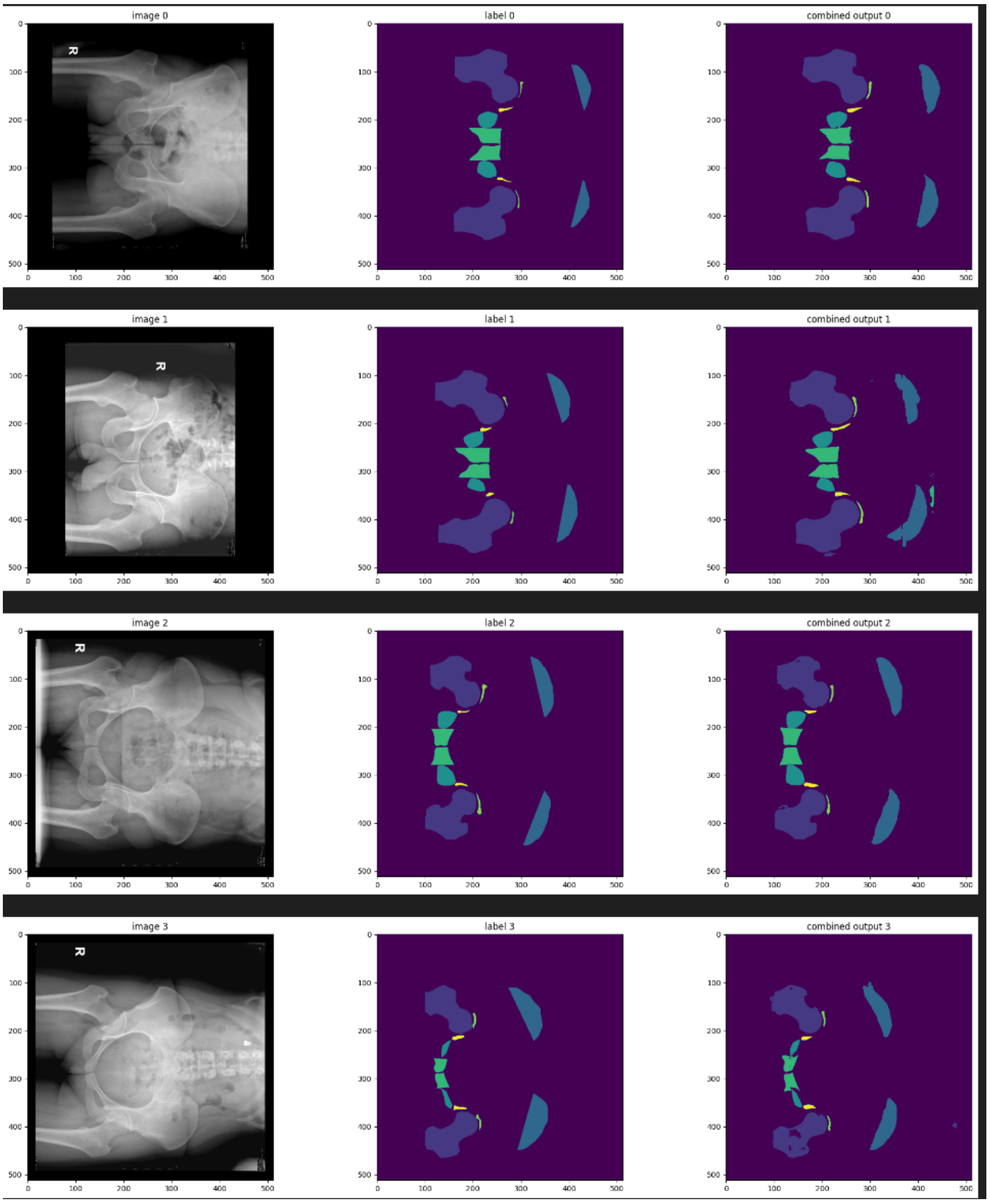
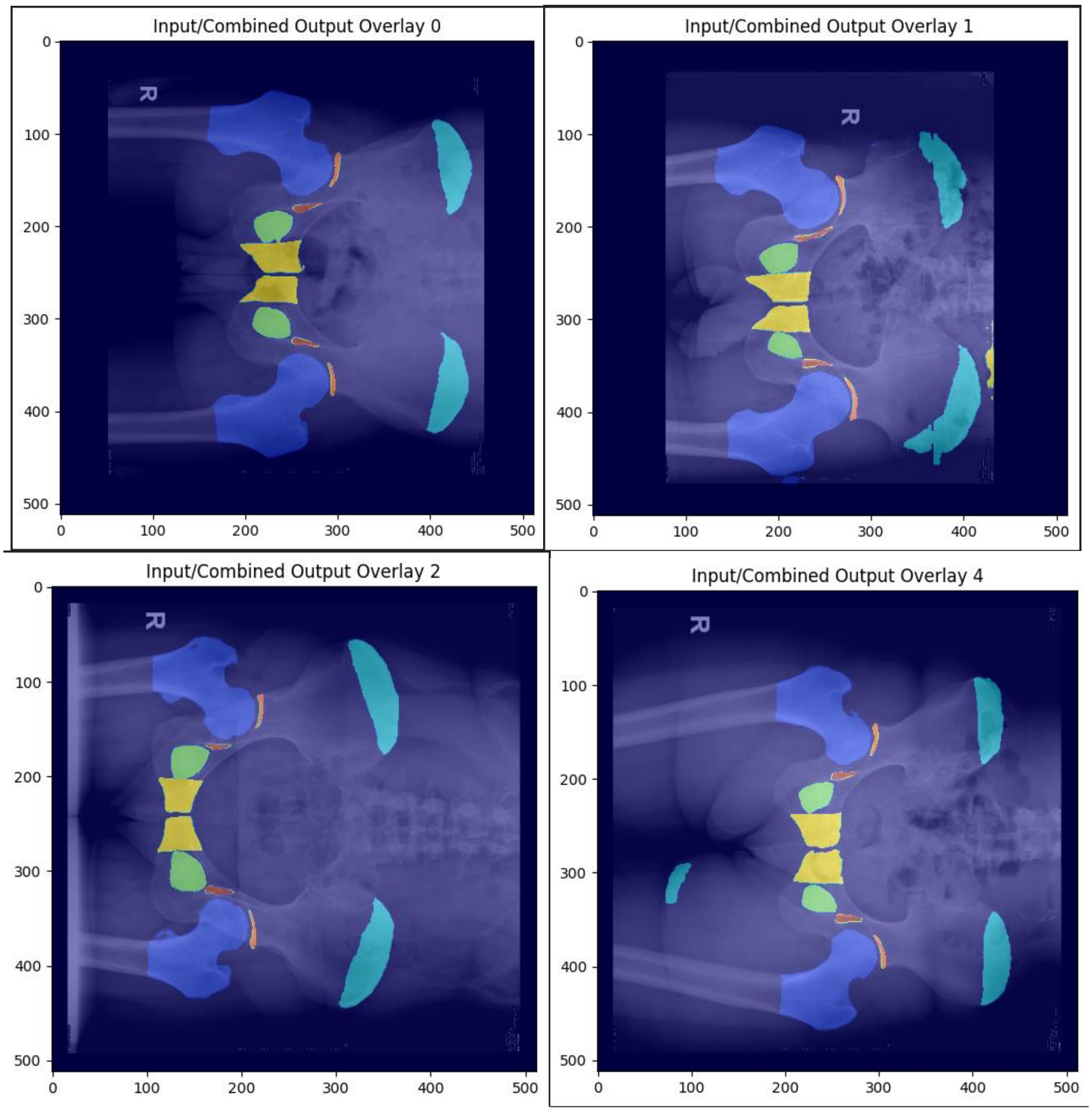


Figure 21: Prediction of segmentation using the best combined trained model

Overlapping view of original image with combined predicted segmentation over it



Input/Combined Output Overlay 0
Input/Combined Output Overlay 1
Input/Combined Output Overlay 2
Input/Combined Output Overlay 4
R

Figure 22: Overlapping view of original image with combined predicted segmentation over it

7. Discussion and Conclusion

U-Net was considered a better choice than other models due to the limited number of training samples. However, the small segmentation area of bones 5 and 6 presented challenges in segmentation, as the training images were not consistent with each other. Two models were trained, ultimately achieving a Dice coefficient of about 0.84. There was potential for improvement by experimenting with different model architectures and employing transfer learning, although these approaches were not tested. Various numerical values for transformer parameters, model architectures, batch sizes, and learning rates were tried. Future improvements might focus on the correction of the segmentation for bones 5 and 6, as well as the exploration of alternative models.