Art of Machine Learning Main Project

Introduction:

In this particular project we are given a dataset created by Yoon et al (Yoon-2019.pdf). This dataset of cells of the eye(cornea). And using the images, we have to perform the segmentation. Segmentation gives us borders and interior of the cell. And we are performing this task using U-net architecture which was given in the paper.

U-NET Architecture:

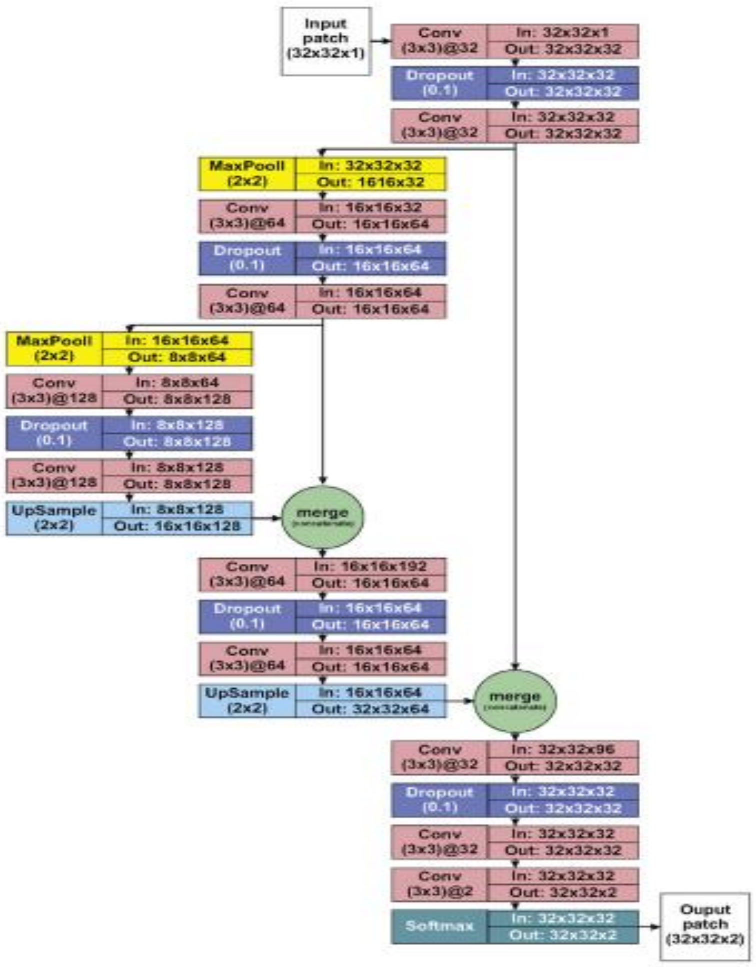


Figure 1. the above figure shows the u-net architecture given in paper

U-net is an architecture in the deep learning methodology, this architecture is used to do segmentation. U-net works by doing a combination of down sampling and up sampling and doing convolution in between. And by doing all these we also merge the old images in the network through concatenation, to introduce the concept of regularization we use dropout layers and at the last (exit point of the network) we use SoftMax activation. We do all these to make the model learn to distinguish between the cell borders and interior of the cells.

A] Dividing the dataset:

1] As mentioned in the dataset, if the number(label) is odd number we have taken it as training dataset and even number is taken as testing dataset. There were total of 180 images, in which 73 images went for testing and the remaining (87) were marked for training.

The libraries that we have choose the below libraries:

Import os- to read the dataset from the folder.

Import cv2 – to read the image as a NumPy array.

2] In this segment we will extract 200 random images from the image that has a size of 500x500 and here, we randomly extract(using np.random) 32x32 patches(here we have to note that the patches where the cell does not comprise of 50% of the patch is rejected and we still continue the loop to find the patches to get 200 of them per image)

The libraries that we have used here are:

Import numpy as np – to use the numpy arrays.

Import np.randint – to compute the random values.

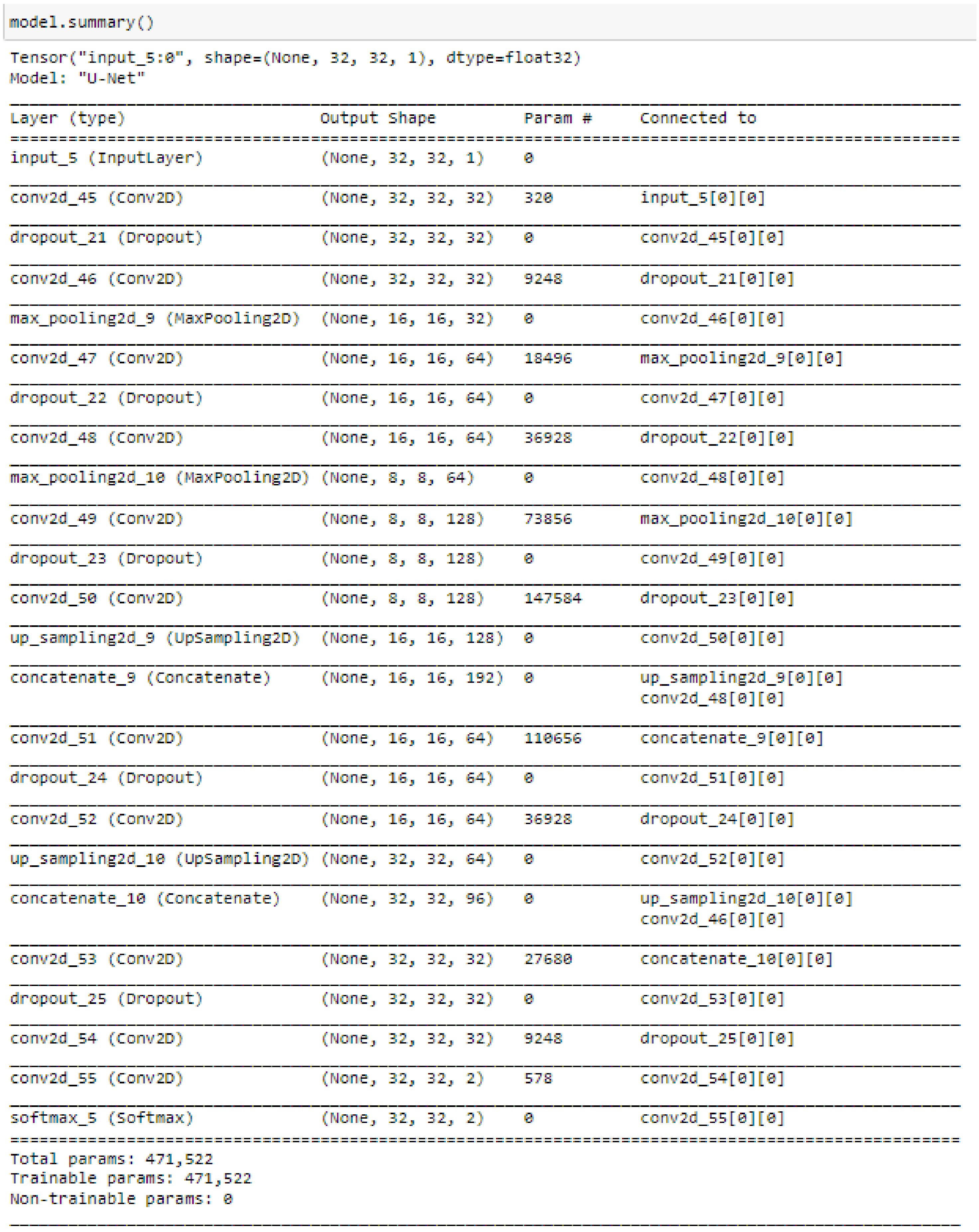
B] Implementing the model:

As we have mentioned above that the U-net model is given in the paper and we have used the same architecture to segment the images, in implementing the architecture, we ran into some issues and that were, we were using the last layer as convolution + SoftMax, whereas we were required to use only SoftMax layer. And apart from that we have seen the architecture in the paper and implemented the same thing. One of the interesting things that we encountered in the above implementation was the use of concatenate function as we were not familiar with this thing Initially. And we also got a good learning experience as we were searching for hoe the above up-sampling and max-pooling works in the background. In the below lines I will attach the architecture that we used (same as paper but we are attaching this for your evaluation and we used model. Summary () to get this)

The libraries that we used here are:

Keras, Tensorflow – using this we imported layers, optimizer, loss function, activation function and etc.

We also implement the model checkpointing using the above libraries. This saves the model which shows improvement in validation loss above a particular threshold.



```
model.summary()
```

```
Tensor("input_5:0", shape=(None, 32, 32, 1), dtype=float32)
Model: "U-Net"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_5 (InputLayer)            (None, 32, 32, 1)    0
__________________________________________________________________________________________________
conv2d_45 (Conv2D)              (None, 32, 32, 32)   320         input_5[0][0]
__________________________________________________________________________________________________
dropout_21 (Dropout)            (None, 32, 32, 32)   0           conv2d_45[0][0]
__________________________________________________________________________________________________
conv2d_46 (Conv2D)              (None, 32, 32, 32)   9248        dropout_21[0][0]
__________________________________________________________________________________________________
max_pooling2d_9 (MaxPooling2D)  (None, 16, 16, 32)   0           conv2d_46[0][0]
__________________________________________________________________________________________________
conv2d_47 (Conv2D)              (None, 16, 16, 64)   18496       max_pooling2d_9[0][0]
__________________________________________________________________________________________________
dropout_22 (Dropout)            (None, 16, 16, 64)   0           conv2d_47[0][0]
__________________________________________________________________________________________________
conv2d_48 (Conv2D)              (None, 16, 16, 64)   36928       dropout_22[0][0]
__________________________________________________________________________________________________
max_pooling2d_10 (MaxPooling2D) (None, 8, 8, 64)     0           conv2d_48[0][0]
__________________________________________________________________________________________________
conv2d_49 (Conv2D)              (None, 8, 8, 128)    73856       max_pooling2d_10[0][0]
__________________________________________________________________________________________________
dropout_23 (Dropout)            (None, 8, 8, 128)    0           conv2d_49[0][0]
__________________________________________________________________________________________________
conv2d_50 (Conv2D)              (None, 8, 8, 128)    147584      dropout_23[0][0]
__________________________________________________________________________________________________
up_sampling2d_9 (UpSampling2D)  (None, 16, 16, 128)  0           conv2d_50[0][0]
__________________________________________________________________________________________________
concatenate_9 (Concatenate)     (None, 16, 16, 192)  0           up_sampling2d_9[0][0]
                                                                 conv2d_48[0][0]
__________________________________________________________________________________________________
conv2d_51 (Conv2D)              (None, 16, 16, 64)   110656      concatenate_9[0][0]
__________________________________________________________________________________________________
dropout_24 (Dropout)            (None, 16, 16, 64)   0           conv2d_51[0][0]
__________________________________________________________________________________________________
conv2d_52 (Conv2D)              (None, 16, 16, 64)   36928       dropout_24[0][0]
__________________________________________________________________________________________________
up_sampling2d_10 (UpSampling2D) (None, 32, 32, 64)   0           conv2d_52[0][0]
__________________________________________________________________________________________________
concatenate_10 (Concatenate)    (None, 32, 32, 96)   0           up_sampling2d_10[0][0]
                                                                 conv2d_46[0][0]
__________________________________________________________________________________________________
conv2d_53 (Conv2D)              (None, 32, 32, 32)   27680       concatenate_10[0][0]
__________________________________________________________________________________________________
dropout_25 (Dropout)            (None, 32, 32, 32)   0           conv2d_53[0][0]
__________________________________________________________________________________________________
conv2d_54 (Conv2D)              (None, 32, 32, 32)   9248        dropout_25[0][0]
__________________________________________________________________________________________________
conv2d_55 (Conv2D)              (None, 32, 32, 2)    578         conv2d_54[0][0]
__________________________________________________________________________________________________
softmax_5 (Softmax)             (None, 32, 32, 2)    0           conv2d_55[0][0]
==================================================================================================
Total params: 471,522
Trainable params: 471,522
Non-trainable params: 0
__________________________________________________________________________________________________
```

Figure 2. The above figure shows our model architecture

C]

In this part we trained for 150 epochs from the randomly generated patches of the trained sets and we also created random patches for the test set and then used it as validation data. The training started with a training set accuracy of 68 percent and validation accuracy of 71 percent at the end of first epoch. It was noteworthy that the training loss and the accuracy was linearly improving through the training and the final accuracy obtained on the train set was 79.9 percent and validation accuracy of 72.6 percent was accomplished.

D]

Implementing Section 3.3 of paper

We have tried to make the same operations that was shown in the paper. In the first segment of 3.3, we are doing padding of 25 pixels on the right and the bottom of the image. Since we want to perform sliding window operations with window size 32X32 with an overlap with 3 pixels in adjacent windows, which requires 525x525 image size, which is an integral multiple of the sliding block. In next operation we will take the input of the neural network and we take output of the neural network even that is 32x32 pixels. Whereas, In the overlapping images we take the average of the values predicted by the neural network on those cells.

Implementing section 3.4 of paper

In this method we consider the 5x5 neighborhood of the cell. So, the method used in paper gives us the threshold that needs to be set for every single pixel in the image which will be based on the value of model prediction in the neighborhood of 5x5 pixels. We will then use the threshold to rest if the cell intensity is greater than the threshold and if it is true the value of the threshold is set to 1. If not, it is set to 0. This method essentially allows us to remove the noise from the predicted results from the neural network. Since the threshold considers the neighboring cells standard deviation and mean values when setting the threshold.

$$T = m\left[1 + k\left(\frac{\sigma}{\sigma_{\max}} - 1\right)\right]$$

Figure 3. The above figure shows the threshold formula

Here in the above figure m refers to the mean intensity value of the 5x5 pixels present around the cell. Sigma is the standard deviation of the 5x5 neighborhood and sigma max is the highest computed 5x5 standard deviations thought the entire matrix.

Our 3.3 and 3.4 results:

In most cases the output of 3.3 seemed to be very accurate and often even seemed visually more accurate than the given labels. It was observable from the output of 3.4 that some of the blurry pixels and low value pixels were completely removed and the segmentation which contains proper 1 and 0 values, which represent the cell boundary and cell interior.

Instead of applying the formula to every individual pixel, we came up with a smart matrix approach which boosts the speed of the execution by a lot. We created a function called avg_filter which returns the matrix where every cell contains the average of the 5x5 pixels surrounding them. Using this function, we will be easily able to compute the mean, mean square and standard deviation on 5x5 channels applied on the matrix. And after obtaining the mean standard deviation and max of standard deviation we simply apply the above-mentioned formula to get the threshold using which we will compute our post-prediction result.

E]

In this good and bad segmentation visualization we are comparing the good and bad results obtained in individual 32x32 segment's output prediction of 3.3 method, and post processing result of 3.4 method. Finally, we get the visualization of the images that I am going to show in the below page.

Good patches:

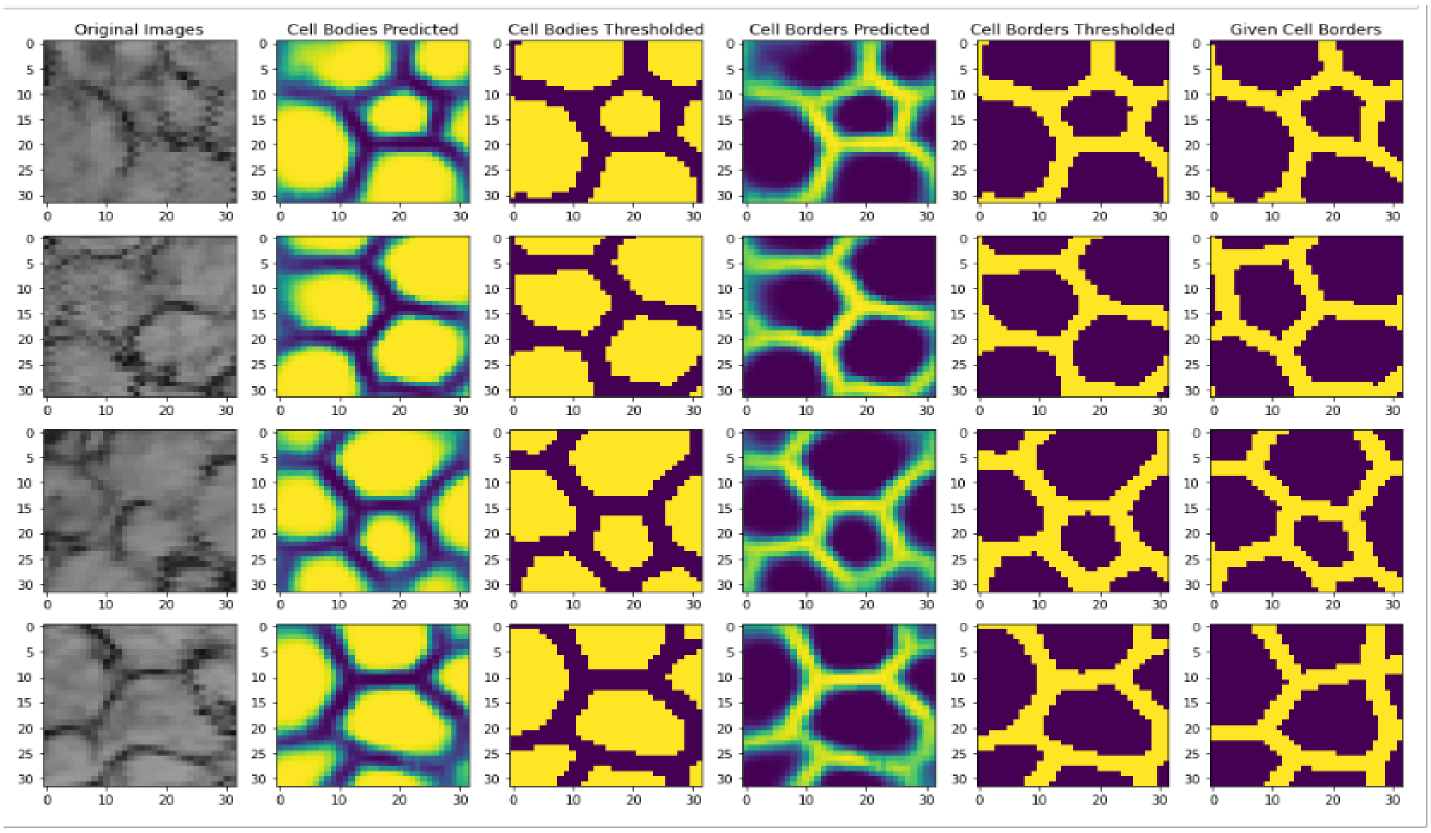


Figure 4. the above figure shows the good patches

Bad patches:

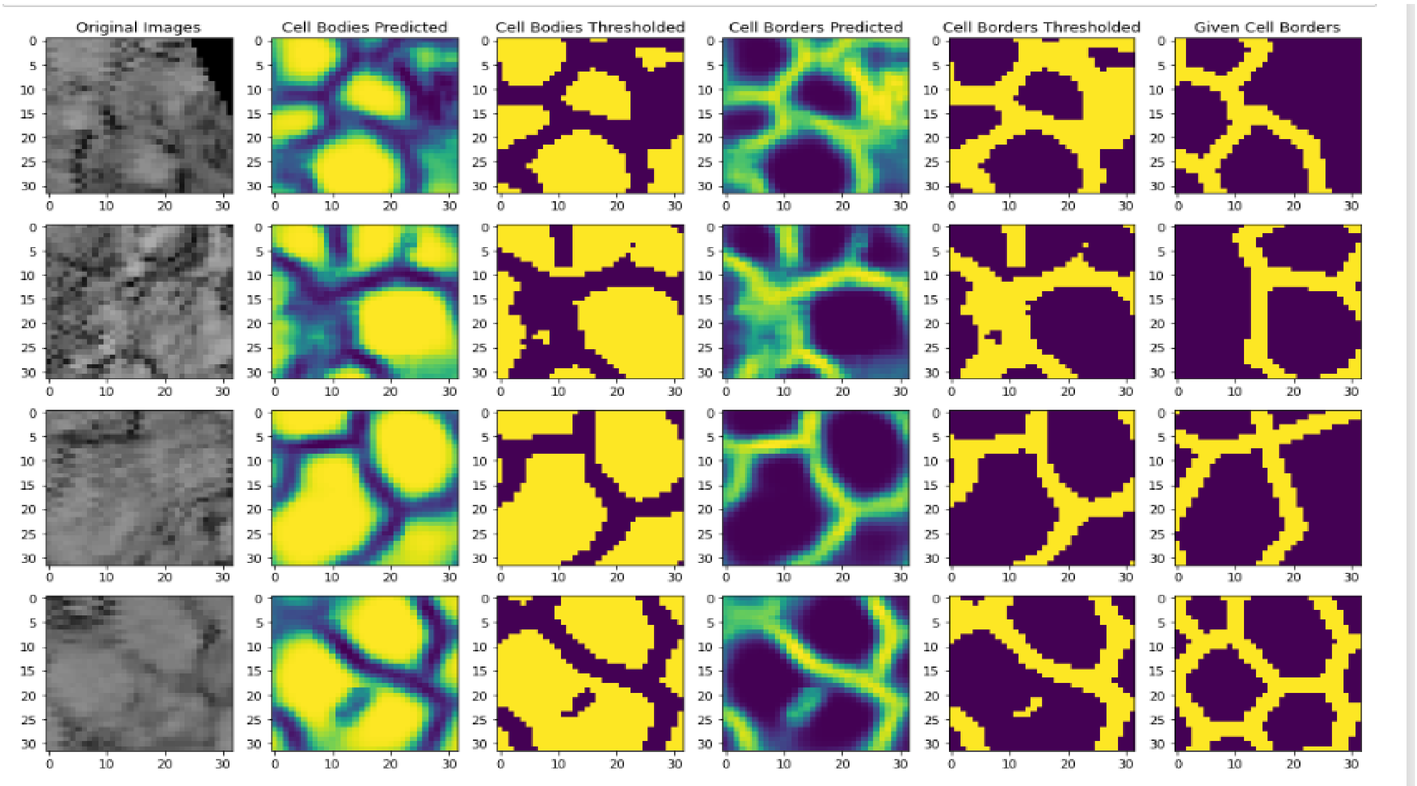


Figure 5. The above figure shows the bad patches

Good Predictions:

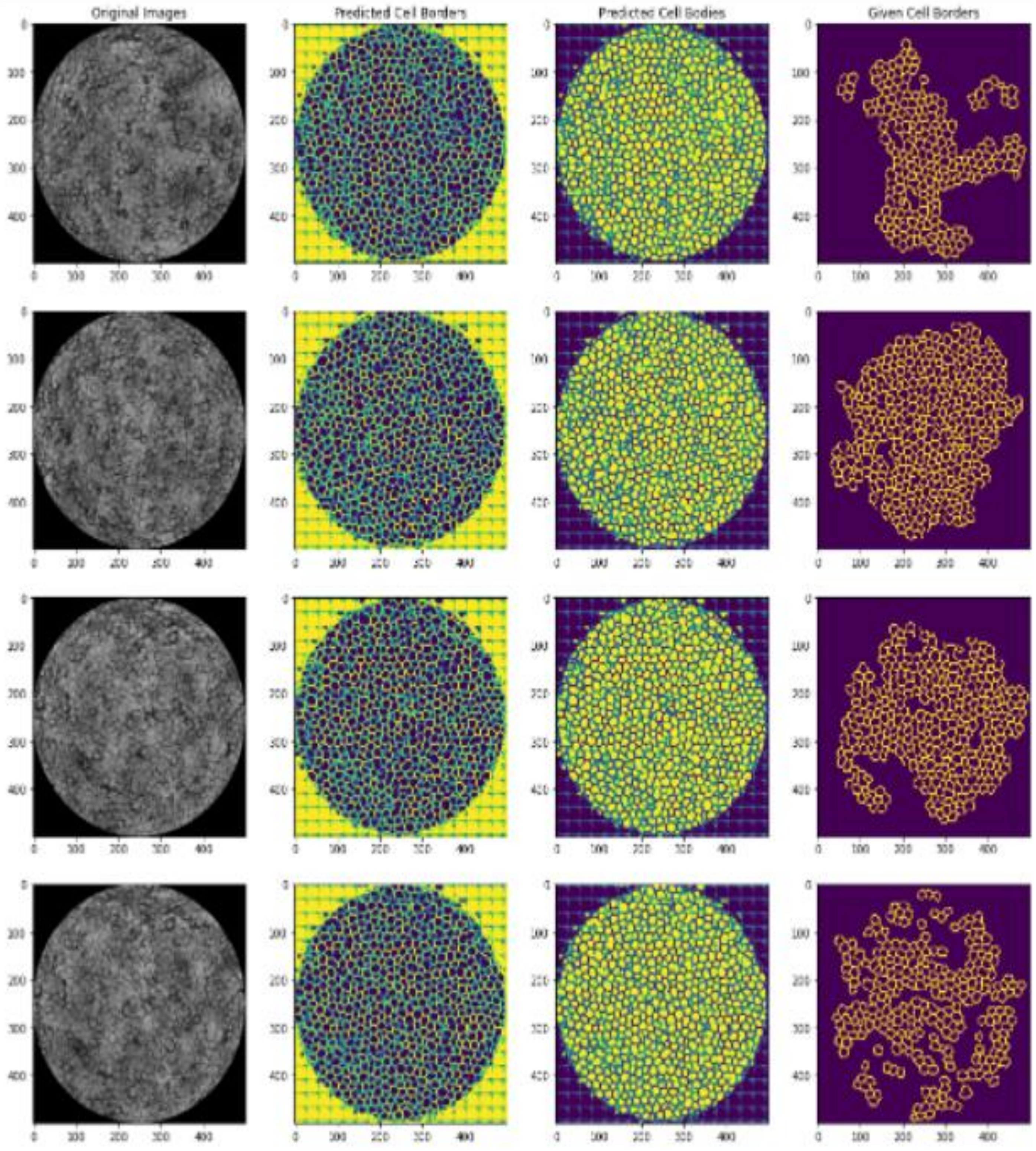


Figure 6. The above figure shows good predictions

Bad predictions:

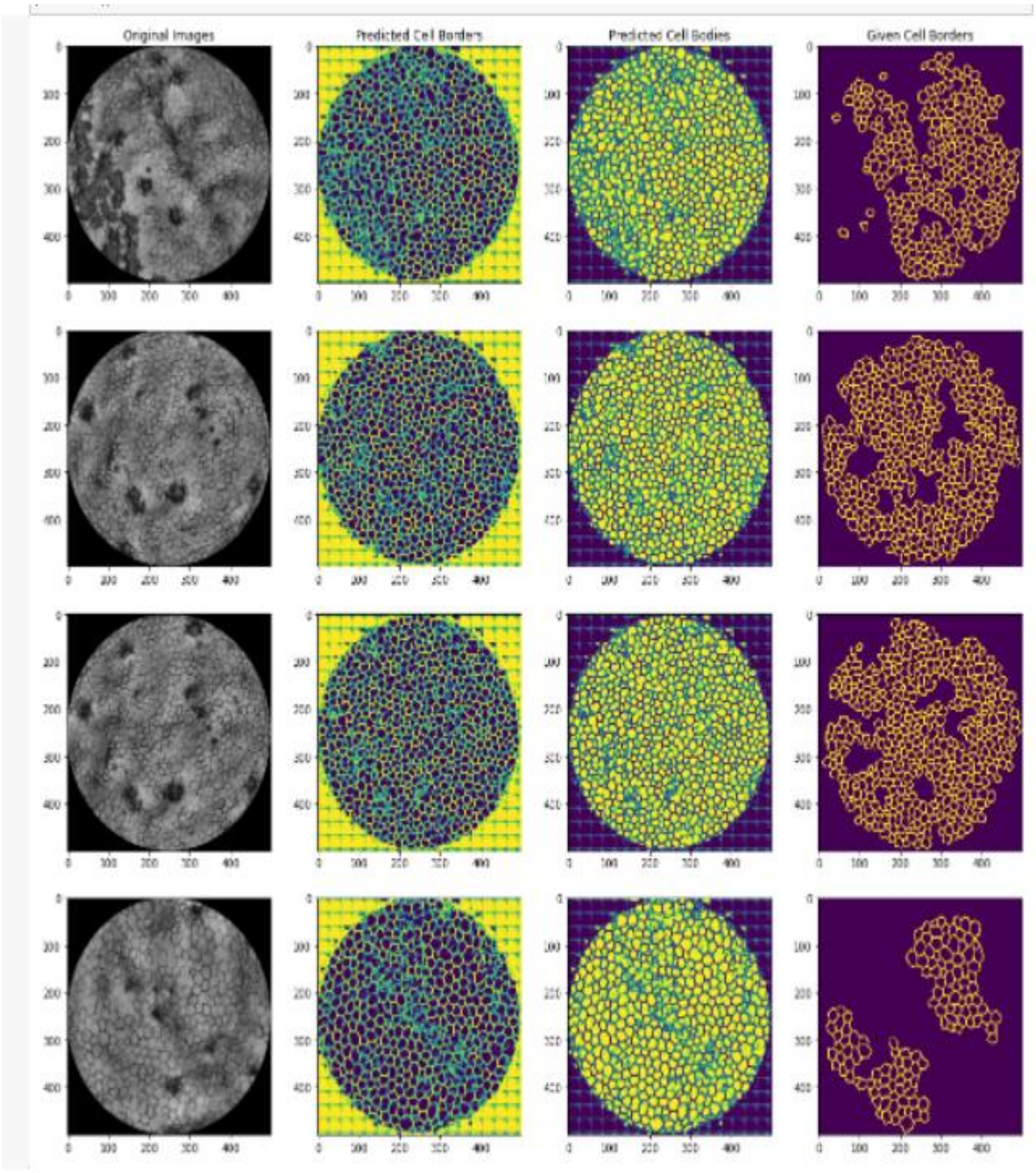


Figure 7. The above figure shows the bad predictions

How did we classify good and bad predictions?

We have observed that the in the good and bad patches the first image (dataset), labels and the predictions and we are looking at the similarity of them, if we see that as similar, we are noting it as good segmentation, whereas if it(predictions) is not similar to labels, we are noting it as bad predictions.

F]

The ROC and AUC of the model is calculated by using the sklearn learn functions. Here we input Predicted data and given labels and in-turn we get the ROC and AUC of the model. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. We use this to distinguish the difference between predicted and actual values. comparing the ROC and AUC values we get good values but not as good as the paper. As in paper the values were close to (0.919) for odd and (0.927) for even. Whereas we got around (0.869) for odd and (0.839) for even. Which is close to the paper but not as good.

G]

As we mentioned earlier the AUC values that we got vs the paper was around

AUC (ODD) PAPER	AUC (EVEN) PAPER	AUC (ODD)	AUC (EVEN)
0.919	0.927	0.869	0.839

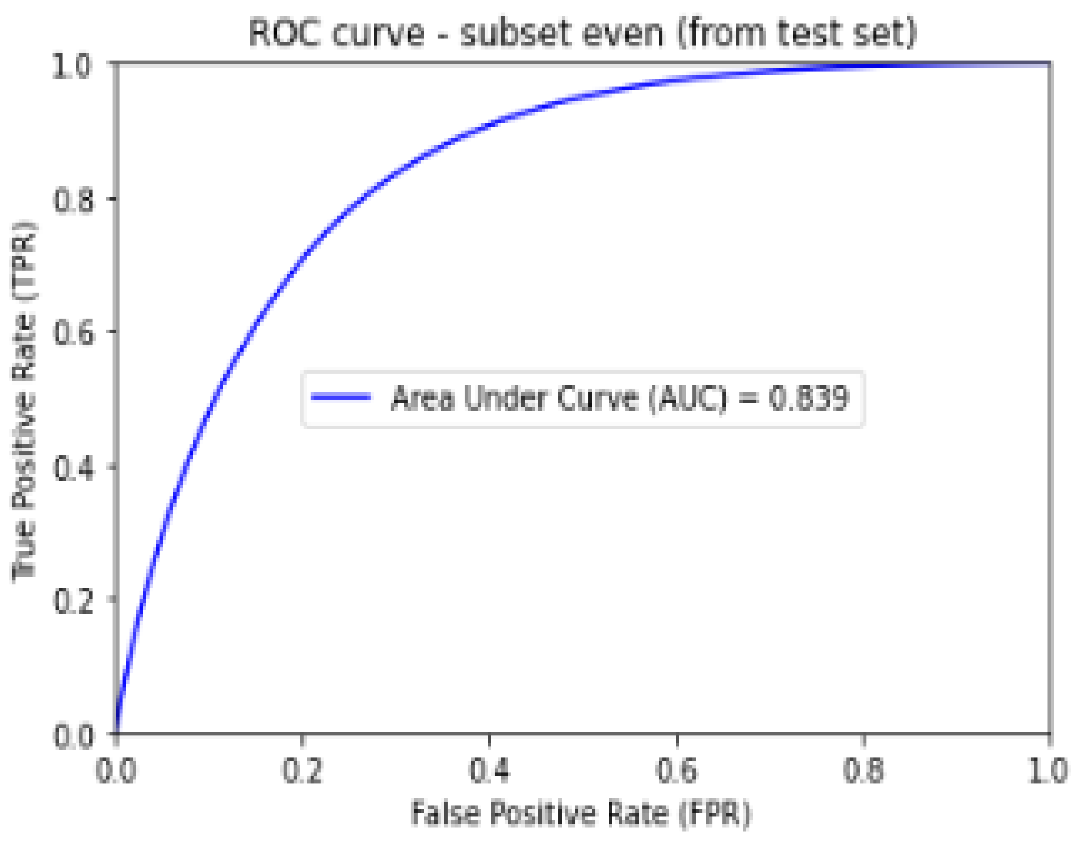


Figure 8. The above figure shows the ROC curve from test data(even)

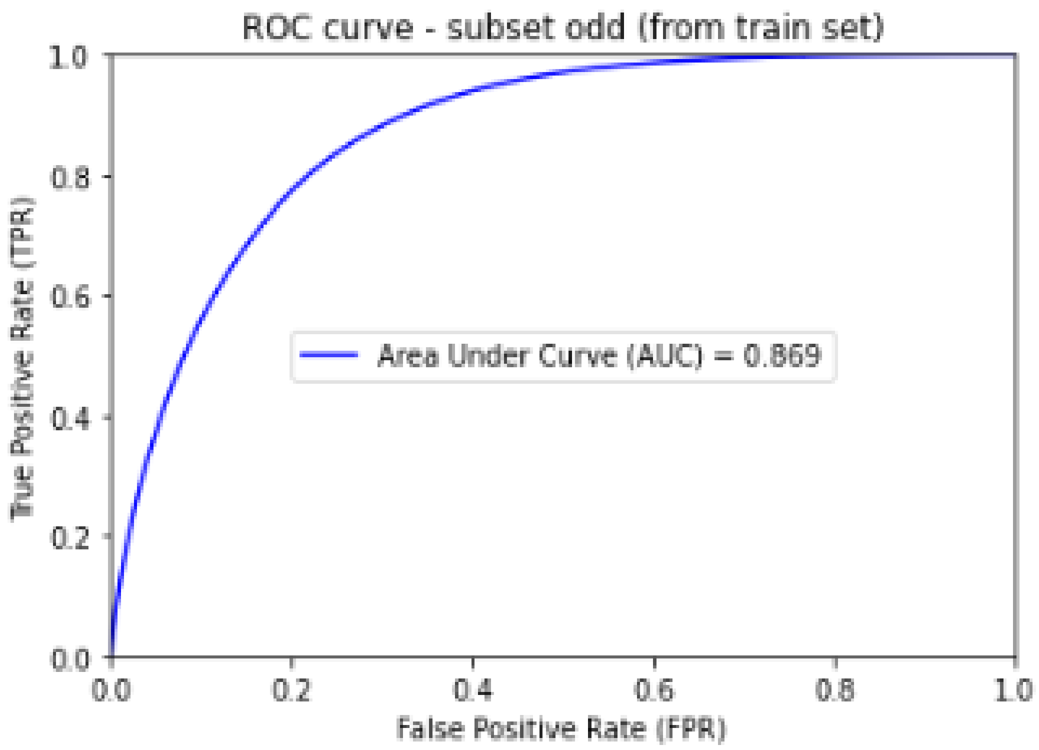


Figure 9. The above figure shows the ROC curve on train set(odd)

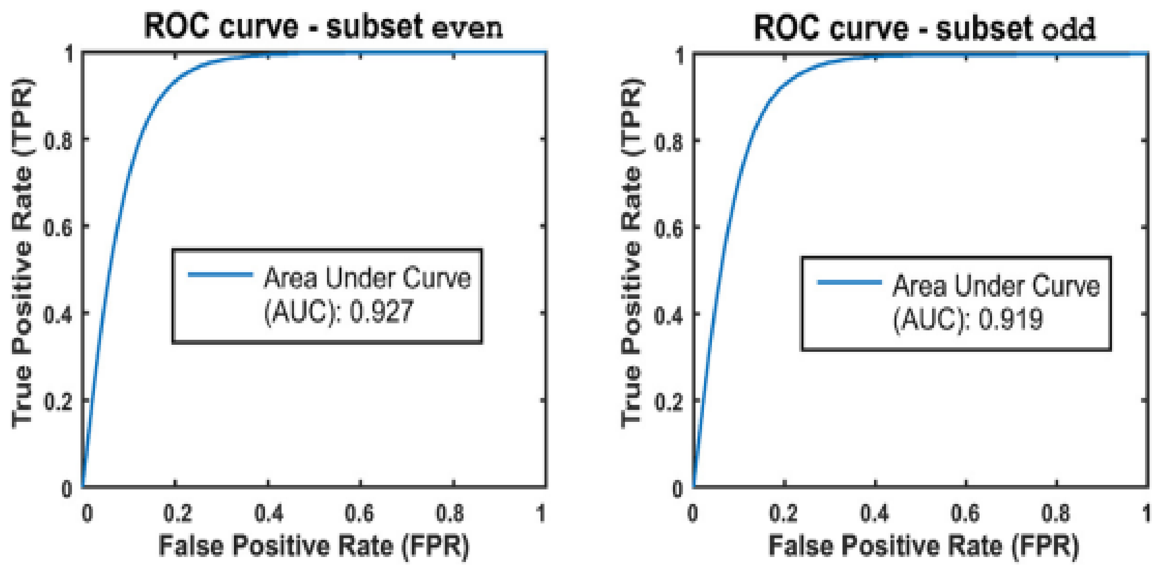


Figure 10. The above figure shows the ROC in the paper on even set vs odd set

In the above table we compared the AUC values that we got against the values that were estimated in the paper. From this as I mentioned earlier, we can estimate the difference between predicted values and actual values. Apart from that we got around 79.9 percent training accuracy and the loss of training (loss on the training set) was around 0.44(at the start of training (after first epoch)) and it improved to around 0.2637 at the end of the training (after 150 epochs). Apart from this the validation accuracy on the test set was around 72.6 percent and the loss improved from around (0.719 (at the start of the first epoch)) to (0.4519(at the end of 150 epochs)).In the above figures we can see that the curve is leaned on the left side and that should the ideal output and we have seen that in the paper and in our implementation, the curve seems to be the same. But yeah the AUC values are slightly different.

We were able to visually infer that the prediction and the 3.4 postprocessing method in the paper gave very similar outputs and most of them found the cell boundaries and the cell bodies accurately as shown in the section **E]** (in our implementation). Hence, we have successfully predicted the outputs but the AUC values are a bit less compared to the paper and that can be due to a different image size that was used in the paper(575x575).

Thank You.

Team-member information

Name -Babu Aman Rai Saxena (UR ID: 32103519)

Name -Deepak velumani Subramani (UR ID: 32132007)

Email-address:

Aman: bsaxena@ur.rochester.edu

Deepak: dsubrama@ur.rochester.edu