Spring 2019: Deep Learning and its applications to Signal and Image Processing and Analysis

Dr. Tammy Riklin Raviv

Yahel Salomon- 303079396
Netanel Biton- 305435059
Teodor Linnik- 312728017

Final Report- Microscopy Cells Segmentation Project

Table of contents

Abstract 2
System Description 3
  Architecture 3
  Training 4
Assumptions and limitations 6
Experiments 6
  Image augmentation 6
  Image patches 7
  Blank images 7
  Challenge “Broad Bioimage Benchmark Collection” dataset 8
  Different architecture 8
  Weighted cross-entropy 9
  Hyperparameters 9
Overcoming difficulties 10
Results 11
Conclusions 12
Bibliography 13

Spring 2019: Deep Learning and its applications to Signal and Image Processing and Analysis
Dr. Tammy Riklin Raviv

Yahel Salomon- 303079396
Netanel Biton- 305435059
Teodor Linnik- 312728017

Final Report- Microscopy Cells Segmentation Project

Abstract

Machine Learning algorithms used for classification tasks, commonly called classifiers, have already achieved human level classification capabilities in many problems and datasets. The training process of the classifiers usually depends on large amounts of labeled samples, unfortunately, in real-world scenarios, labeling large amounts of samples is impossible, due to lack of skill and large cost of human analysts, a common example is doctors labeling cells. Fully Convolutional Networks are very popular in image segmentation field. UNet [1] is a baseline network of many other applications in image segmentation. In this research, we chose UNet as baseline and we implemented an improved network and called it Small-Unet. In addition, we examine the impact of several preprocessing techniques on the dataset and suggested hyper-parameters and network architecture for the task.

Yahel Salomon- 303079396
Netanel Biton- 305435059
Teodor Linnik- 312728017

Final Report- Microscopy Cells Segmentation Project

System Description

Architecture

UNET is a Fully Convolutional Network Model which is very popular in the field of image segmentation. We took the UNET architecture and made some important changes. We called it *'small-UNET'*. UNET is very deep network with a lot of layers, Training the UNET with a large bunch of images spent a lot of time and resources, so we removed layers as shown in fig. 1. In addition, we add three layers with neurons hopping to get better segmentation. The results will be shown in Results paragraph.

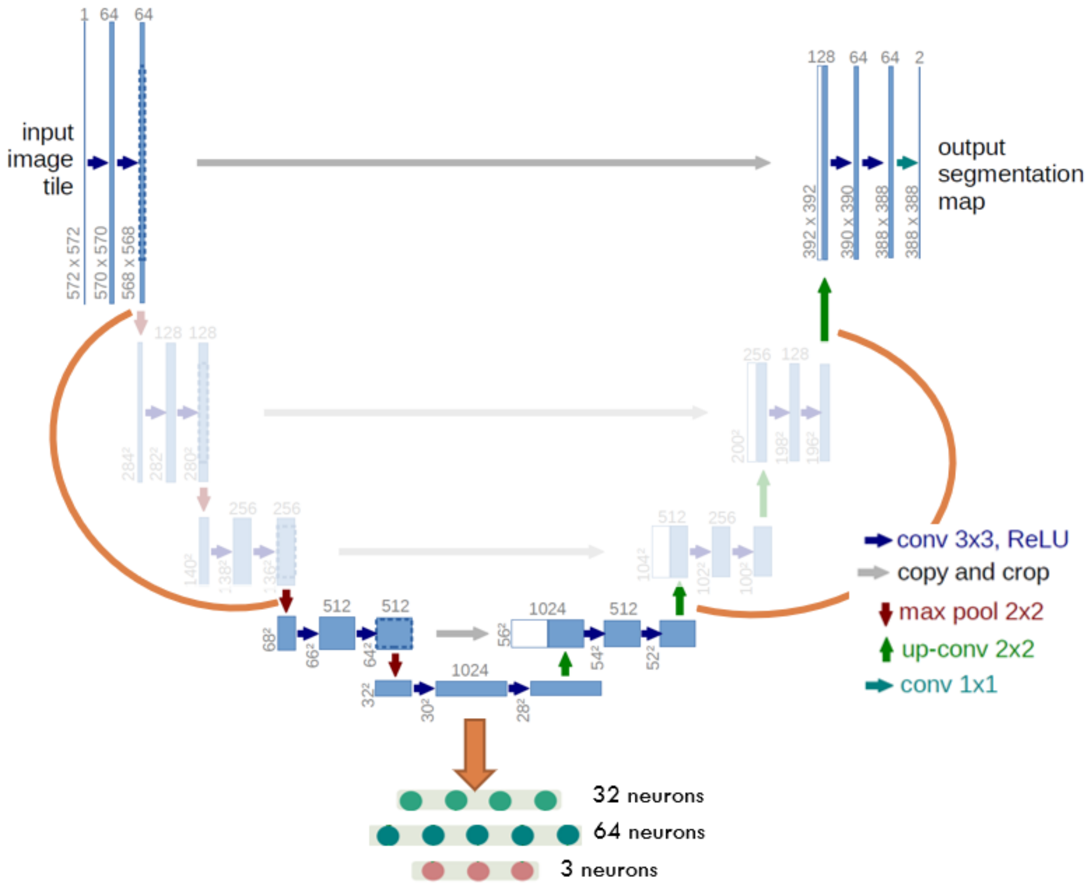


Figure 1: small-UNET architecture

Yahel Salomon- 303079396
Netanel Biton- 305435059
Teodor Linnik- 312728017

Training

Data preparation:

The dataset contains a unique 15 microscopy cell images. We divide the dataset to training set and testing set. 11 images for the training set and 4 images for the testing set.
In order to improve the model performances, we did a data preparation for the training set:

1. Augmentation: Enlarge the dataset by adding images with augmentation: rotation, zoom in, mirroring. From each of the 11 train images, we were created 10 new images. i.e., 121 images including the original images.
2. Small images: Enlarge the dataset by cropping each image to 64 pieces.
3. Background images: Add to the dataset several blank images with background only.
4. Adding a new image data set from the challenge called “Broad Bioimage Benchmark Collection” [2]. Presented in figures 2,3. We crop the images to be with the same size of our dataset.

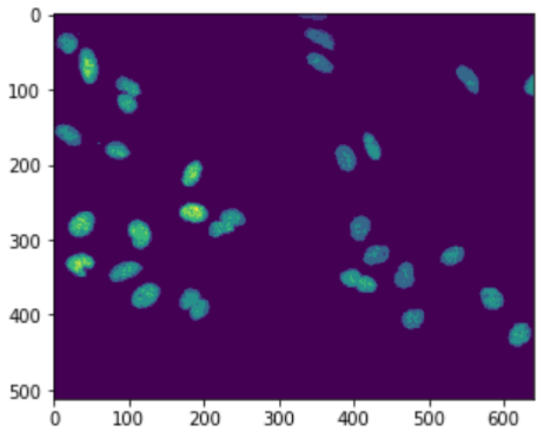


Figure 2: new image

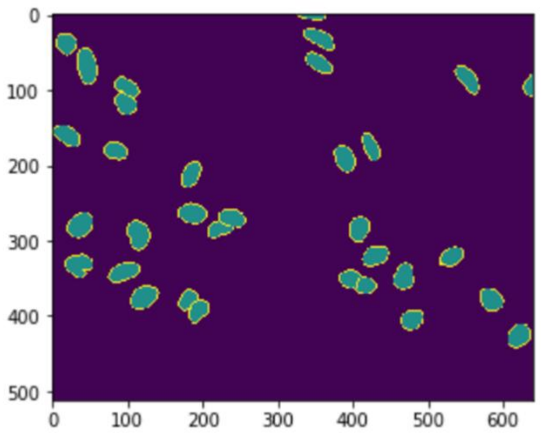
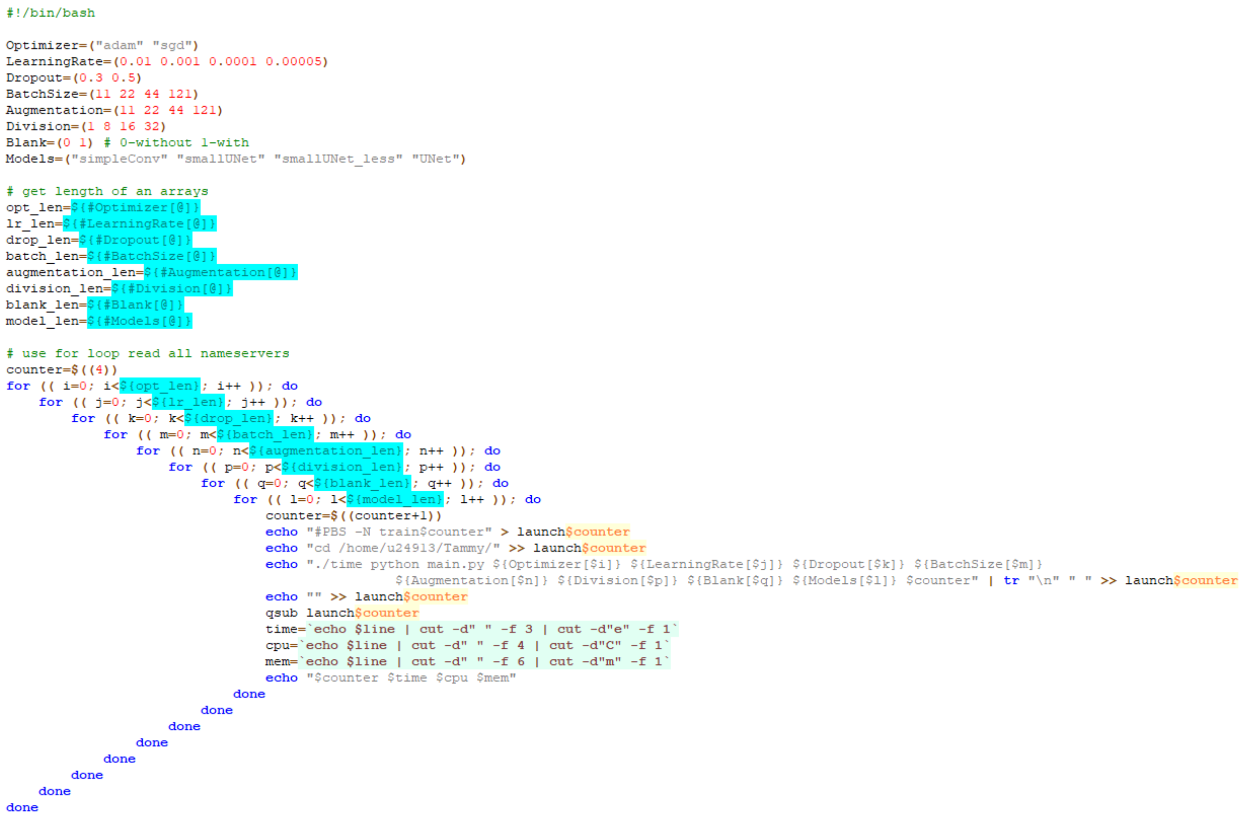


Figure 3: new segmentation

Parameters:

We wanted to train the model with the best hyperparameters. First, by Callbacks we tried the model by using a function that decrease the learning rate when there is no improvement after couple of epochs, while the model is training. However, in all of the models we tried, the learning rate was converged to a local minimum, so we didn't get the desired results. Since we had the option to train the model on big computational power, on Intel's servers, instead of training one model every time, we wrote a script as shown in fig. 4 and fig. 5, that's training several models with different hyperparameters simultaneously, so we could take the best hyperparameters.



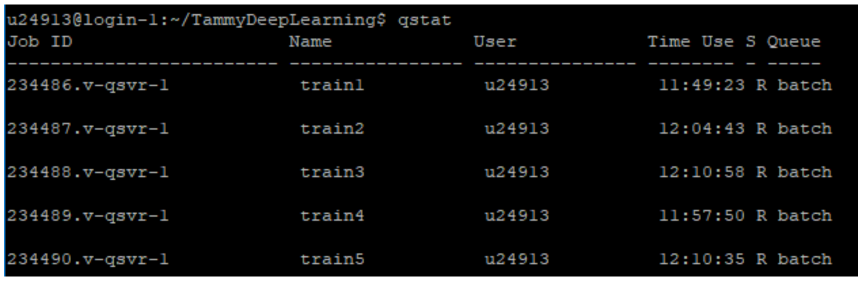
```bash
#!/bin/bash

Optimizer=("adam" "sgd")
LearningRate=(0.01 0.001 0.0001 0.00005)
Dropout=(0.3 0.5)
BatchSize=(11 22 44 121)
Augmentation=(11 22 44 121)
Division=(1 8 16 32)
Blank=(0 1) # 0-without 1-with
Models=("simpleConv" "smallUNet" "smallUNet_less" "UNet")

# get length of an arrays
opt_len=${#Optimizer[@]}
lr_len=${#LearningRate[@]}
drop_len=${#Dropout[@]}
batch_len=${#BatchSize[@]}
augmentation_len=${#Augmentation[@]}
division_len=${#Division[@]}
blank_len=${#Blank[@]}
model_len=${#Models[@]}

# use for loop read all nameservers
counter=$((4))
for (( i=0; i<${opt_len}; i++ )); do
    for (( j=0; j<${lr_len}; j++ )); do
        for (( k=0; k<${drop_len}; k++ )); do
            for (( m=0; m<${batch_len}; m++ )); do
                for (( n=0; n<${augmentation_len}; n++ )); do
                    for (( p=0; p<${division_len}; p++ )); do
                        for (( q=0; q<${blank_len}; q++ )); do
                            for (( l=0; l<${model_len}; l++ )); do
                                counter=$((counter+1))
                                echo "#PBS -N train$counter" > launch$counter
                                echo "cd /home/u24913/Tammy/" >> launch$counter
                                echo "./time python main.py ${Optimizer[$i]} ${LearningRate[$j]} ${Dropout[$k]} ${BatchSize[$m]}
                                            ${Augmentation[$n]} ${Division[$p]} ${Blank[$q]} ${Models[$l]} $counter" | tr "\n" " " >> launch$counter
                                echo "" >> launch$counter
                                qsub launch$counter
                                time=`echo $line | cut -d" " -f 3 | cut -d"e" -f 1`
                                cpu=`echo $line | cut -d" " -f 4 | cut -d"C" -f 1`
                                mem=`echo $line | cut -d" " -f 6 | cut -d"m" -f 1`
                                echo "$counter $time $cpu $mem"
                            done
                        done
                    done
                done
            done
        done
    done
done
```

Figure 4: script for training models simultaneously



```
u24913@login-1:~/TammyDeepLearning$ qstat
Job ID                    Name             User            Time Use S Queue
------------------------- ---------------- --------------- -------- - -----
234486.v-qsvr-1            train1           u24913          11:49:23 R batch

234487.v-qsvr-1            train2           u24913          12:04:43 R batch

234488.v-qsvr-1            train3           u24913          12:10:58 R batch

234489.v-qsvr-1            train4           u24913          11:57:50 R batch

234490.v-qsvr-1            train5           u24913          12:10:35 R batch
```

Figure 5: example of training 5 models simultaneously

Assumptions and limitations

First assumption is that the data is one-dimensional PNG images. To increase the dataset and shortening the network training time we defined the size of the picture being inserted into the network is 32 on 40. Before you perform prediction to input image, the image must be divided into a few small images to the appropriate size for the network input. The images must be inserted in a known order So that the result images could be put back together. In addition, we were using a lost weight function. Therefore, it is necessary to check that the constant classes weights are fit before starting the training, otherwise we will get the wrong result.
We call seg-Measure score to the Jaccard index score we got from the file segMeasure.py

Experiments

First, with callbacks we were looking for the number of epochs that will be enough for training the model using a lot of images.
We did a lot of experiments to get the best seg-Measure score. We did multiple data manipulations and examine some different networks. Here are some of the results and the conclusions we got:

Image augmentation

In this experiment we train the same model with the same hyperparameters, to examine the profitability of training a model on multiple examples. As it can be seen, adding image augmentation improves seg-Measure score.

#	**Architecture**	**Weights**			**Epochs**	**Training batch size**	**Image training size**	**Training Dataset size**	**segMeasure Score**
		bg	Cell nucleus	Cell contour					
1	“small-Unet”	0.001	1.05	0.93	30	44	32x40	11x16x16	**0.577**
2	“small-Unet”	0.001	1.05	0.93	30	44	32x40	22x16x16	**0.717**
3	“small-Unet”	0.001	1.05	0.93	30	44	32x40	44x16x16	**0.758**
4	“small-Unet”	0.001	1.05	0.93	30	44	32x40	121x16x16	**0.823**

Image patches

We cut the image into small pieces. We did that to enlarge the dataset and to train the model on image patches. In this experiment we train the same model with the same hyperparameters, for choosing the best images size dataset. i.e., getting better prediction and shortening the training runtime. As it can be seen, adding image patches improves seg-Measure score and the running time decreases.

#	Architecture	Weights			Epochs	Training batch size	Image training size	Training Dataset size	Elapsed time	Seg Measure Score
		bg	Cell nucleus	Cell contour						
1	“small-Unet”	0.001	1.05	0.93	30	44	512x640	121x1x1	20:39.19	**0.76**
2	“small-Unet”	0.001	1.05	0.93	30	44	64x80	121x8x8	44:01.9	**0.829**
3	“small-Unet”	0.001	1.05	0.93	30	44	32x40	121x16x16	53:28.37	**0.823**
4	“small-Unet”	0.001	1.05	0.93	30	44	16x20	121x32x32	55:29:39	**0.72**

Blank images

We add an information to our training by adding an image without nucleus cells. we added an original size blank image to the dataset, and then if we train the model with image patches we have removed images without nucleus cells. as it can be seen, the model is learning also from the blank image.

#	Architecture	Weights			Epochs	Training batch size	Image training size	Training Dataset size	With/ Without bg images	Seg Measure Score
		bg	Cell nucleus	Cell contour						
1	“small-Unet”	0.001	1.05	0.93	30	44	32x40	121x16x16	With	**0.81**
2	“small-Unet”	0.001	1.05	0.93	30	44	32x40	19246	Without	**0.776**
3	“small-Unet”	0.001	1	1	30	44	32x40	121x16x16	With	**0.823**
4	“small-Unet”	0.001	1	1	30	44	32x40	19246	Without	**0.78**
3	“small-Unet”	1	1	1	30	44	32x40	121x16x16	With	**0.79**
4	“small-Unet”	1	1	1	30	44	32x40	19246	Without	**0.765**

Yahel Salomon- 303079396
Netanel Biton- 305435059
Teodor Linnik- 312728017

Challenge “Broad Bioimage Benchmark Collection” dataset

We made some tests on the new information we found. There was no big influence on the data.

#	Architecture	Weights			Epochs	Training batch size	Image training size	Training Dataset size	With/ Without the new dataset	Seg Measure Score
		bg	Cell nucleus	Cell contour						
1	“small-Unet”	0.001	1	1	20	44	32x40	321x16x16	With	**0.76**
2	“small-Unet”	0.001	1	1	20	44	32x40	121x16x16	Without	**0.78**

Different architecture

We compare between our new model architecture, “small-Unet” performances to anther architectures. We compare it to the original model that we were inspired by. Another architecture was a simple convolution model. We added to the simple-Unet a fully connected layers that we thought it might help the machine to learn better.

#	Architecture	Weights			Epochs	Training batch size	Image training size	Training Dataset size	Elapsed time	Memory (%CPU)	Seg Measure Score
		bg	Cell nucleus	Cell contour							
1	“small-Unet”	0.001	1.05	0.93	30	44	64x80	121x8x8	53:03.3	1199	**0.829**
2	“Unet”	0.001	1.05	0.93	30	44	64x80	121x8x8	1:53:37	1294	**0.835**
3	“conv model”	0.001	1.05	0.93	30	44	64x80	121x8x8	23:06.9	1454	**0**
4	“small-Unet no fully con.”	0.001	1.05	0.93	30	44	64x80	121x8x8	50:55.6	1175	**0.79**

Weighted cross-entropy

The loss function we chose to work with is Weighted Categorical Cross-Entropy function:

$$L = -\sum_{x=0}^{W}\sum_{y=0}^{H}\sum_{c=1,2,3} w_c \cdot \delta(c, G(x,y)) \cdot log(P_\Theta(x,y,c))$$

The weights for every class have impact on the ability of the network to converge and the decision bounds of the cell classification (as presented in the table below). We finally chose used weights of Background: 0.01, Foreground: 1, Edge: 0.95.

Hyperparameters

1. We trained our network with a 121x16x16 mini image patches and with 30 epochs. We chose that number after training the network with 200 epochs and we noticed that it is better to enlarge the dataset instead of train the model by repeating the same examples. Presented below in fig.4.

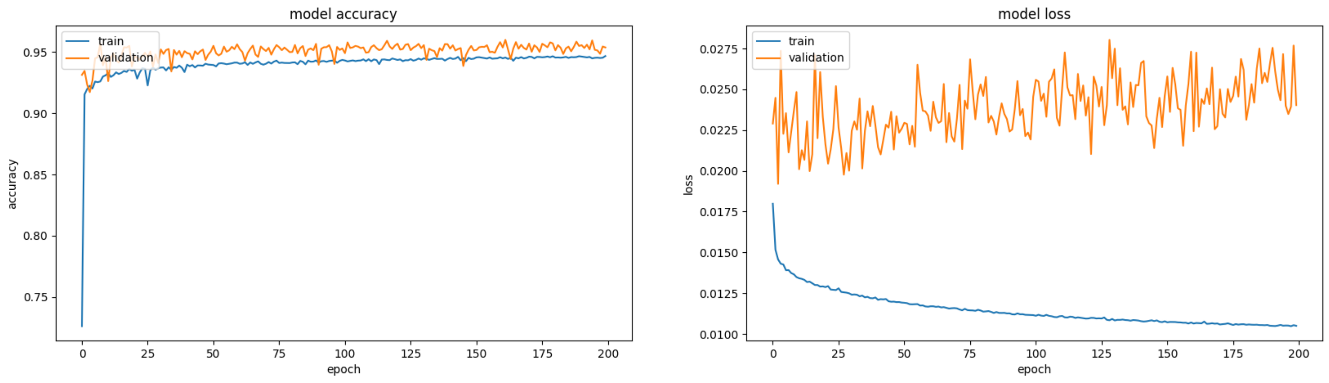


Figure 6: small-UNet training with 200 epochs

Yahel Salomon- 303079396
Netanel Biton- 305435059
Teodor Linnik- 312728017

2. Total training batches- we examine a couple of batch size values (1,11,22,44,121) for training the network. We chose 44 batch size. As it can be seen below it gave us the best results and the fastest runtime.

#	Architecture	Weights			Epochs	Training batch size	Image training size	Training Dataset size	Seg Measure Score
		bg	Cell nucleus	Cell contour					
1	“small-Unet”	0.001	1.05	0.93	30	1	32x40	121x16x16	**0.736**
2	“small-Unet”	0.001	1.05	0.93	30	11	32x40	121x16x16	**0.762**
3	“small-Unet”	0.001	1.05	0.93	30	22	32x40	121x16x16	**0.772**
4	“small-Unet”	0.001	1.05	0.93	30	44	32x40	121x16x16	**0.823**
5	“small-Unet”	0.001	1.05	0.93	30	121	32x40	121x16x16	**0.81**

3. The solver is “Adam”. We also tested the “SGD” solver, but we got a very bad result with it.
4. Dropout is 0.3
5. Callbacks- we were trying to train the model using a callback that decreases the learning rate while training the network. That experiment failed because the learning rate decreases fast and stabilized at local minimum.

Overcoming difficulties

The mayor difficulty we've encountered during the work is the very little dataset we got. Our first solution was to produce augmentations from dataset (110 augmentations + original image=121). In addition, each image we cut into a of small images. So, we got from one 512-by-640 image, 30976images 32-by-40(121*16*16=30976). The third solution was to use an external dataset we found online and try to match it to our data. Another problem was working with a Colab, which could not work and collapsed every time during a training phase. The solution was to work on our PCs or alternatively use Intel’s servers. Another problem is length training time, and because we wanted to test how a change of the dataset or weights would affect the final result. The solution was to use an Intel’s server that allowed us to run multiple networks at the same time, allowing us to test multiple changes at the same time.

Results

The best "small-UNET" results got were with these hyperparameters:

Test num	Architecture	Weights			Epochs	Training batch size	Image training size	Dataset size	segMeasure Results
		bg	Cell nucleus	Cell contour					
3	"small-Unet"	0.01	1	0.95	30	30	64x80	121x8x8	**0.829**

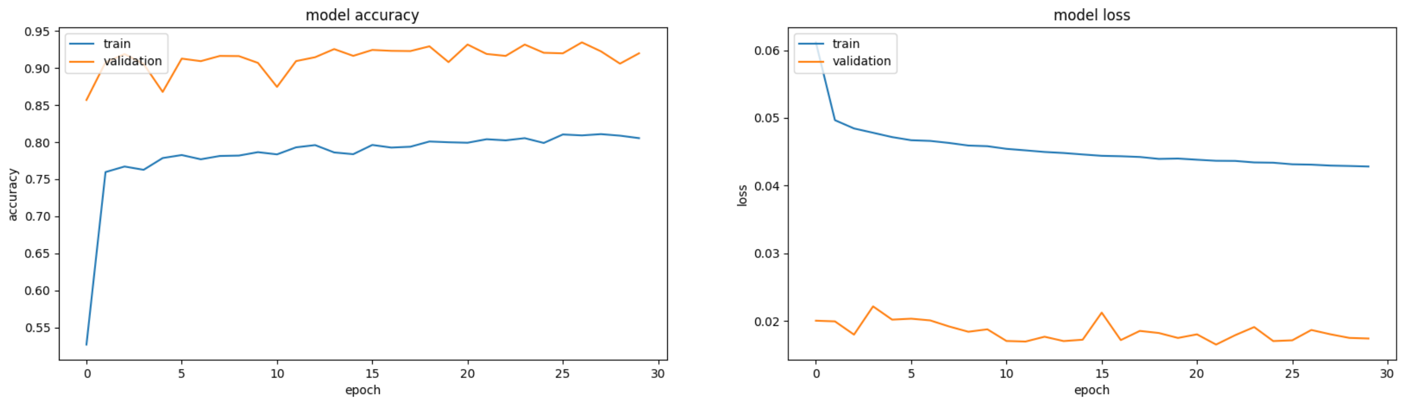


Figure 7:small-UNET best results

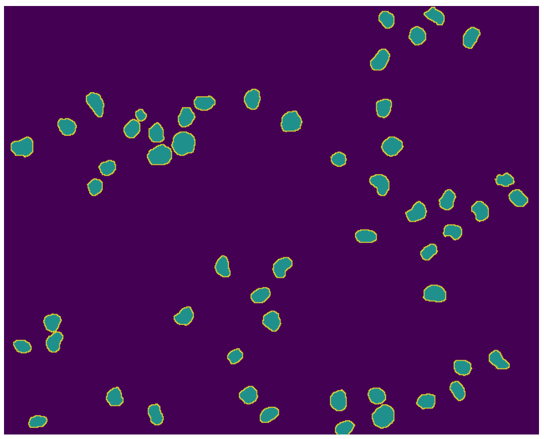


Figure 8: ground truth

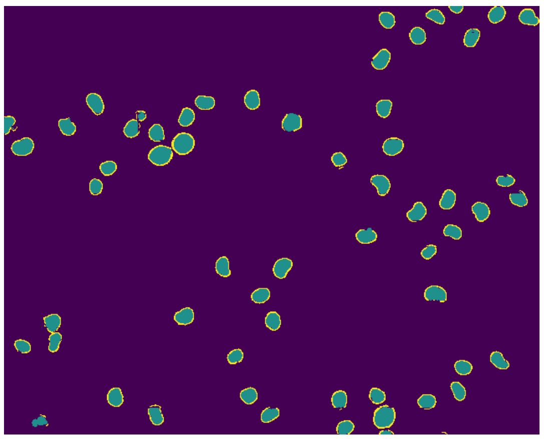


Figure 9: small-Unet segmentation

Yahel Salomon- 303079396
Netanel Biton- 305435059
Teodor Linnik- 312728017

Conclusions

We considered the problem of using “small-UNET” for choosing the best training samples in the problem of cell image segmentation. We tested different datasets, Hyper-Parameters, and architectures. After a comprehensive research, we present our conclusions:

1. Enlarging the dataset by augmentation improves performances.
2. Training the network with smaller images improves performances and shortening the running time.
3. Using Blank images in the data set improves performances.
4. Adding the challenge new dataset after we already did the augmentation, achieves equal.
5. Cell segmentation is a different than the known classification task, every image has all the classes, all the images are almost equal in uncertainty value.
6. It is faster to train the ”small-UNET” than to train the full “UNET”. And the results are almost equal.
7. The constant weights of the weighted loss cross entropy function are very important and had big influence on the results.
8. Network architecture, features (dropout, normalization), loss functions, solver, and activations have great impact on the ability of the network to perform well.

Bibliography

[1] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in Proc. Int. Conf. Medical Image Comput. Comput.-Assisted Intervention, 2015, pp. 234–241.
[2] https://data.broadinstitute.org/bbbc/BBBC039/