*Final Project Challenge - Microscopy Cell Segmentation*

**Jul 2018**

***Ben Gurion University of the Negev***

*Electrical Engineering Department*

*Deep Learning for Signal and Image Processing and Analysis*

*361-2-1120*



**Lecturer:** Dr. Riklin Raviv Tammy

**Students:**

Maor Gaon 301308821

Aharon Kalantar 306304379

Ran Zvi Bezen 201085032

# תוכן עניינים

[תוכן עניינים 1](#_Toc520878328)

[Detailed explanation of the project 2](#_Toc520878329)

[Specs 3](#_Toc520878330)

[Architecture 1 4](#_Toc520878331)

[פרמטרים, תכונות ואימון הרשת: 4](#_Toc520878332)

[Architecture 2 7](#_Toc520878333)

[תובנות בין האינטרציות השונות 9](#_Toc520878334)

[post-prossesing 10](#_Toc520878335)

[תוצאות ה Jaccard של המודל הטוב ביותר 11](#_Toc520878336)

[Analysis and conclusions 12](#_Toc520878337)

[Bibliography 13](#_Toc520878338)

[נספחים 14](#_Toc520878339)

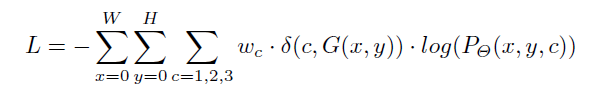
[קוד ארכיטקטורה ראשונה: 14](#_Toc520878340)

[קוד ארכיטקטורה שנייה: 16](#_Toc520878341)

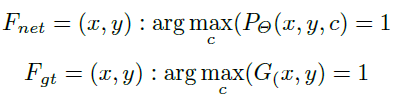
# **Detailed explanation of the project**

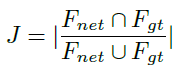
בפרויקט זה נדרשנו לבצע סיגמנטציה בעבור תמונות של תאים שהופקו ממיקרוסקופ אלקטרוני, הסגמנטציה בוצע pixel-wise וכל פיקסל סווג לאחד מ-3 קבוצות : רקע, תוכן התא, קווי המתאר של התא.

הסגמנטציה בוצע בעזרת שימוש ברשת נוירונים עמוקה, תוך שימוש ברקע התיאורטי שניתן במהלך הקורס, במסגרת הפרויקט התבקשנו למזער את הפונקציית ה-loss הבא:



כאשר המדד שבעזרתו נעריך את איכות המודל הוא Jaccard index שאותו נרצה להביא למקסימום.





במסגרת הפרויקט מימשנו מספר סוגי רשתות בעלי פרמטרים שונים עד שהתכנסו לפתרון האופטימלי, רשתות מסוג U-Net בעלי עומק שונה, רשת מסוג fusion net המבוססת על  
U-net ובעלת שיפורים. בכל הרשתות השתמשנו בטכניקה ה-Identity function מארכיטקטורת ה-resNet

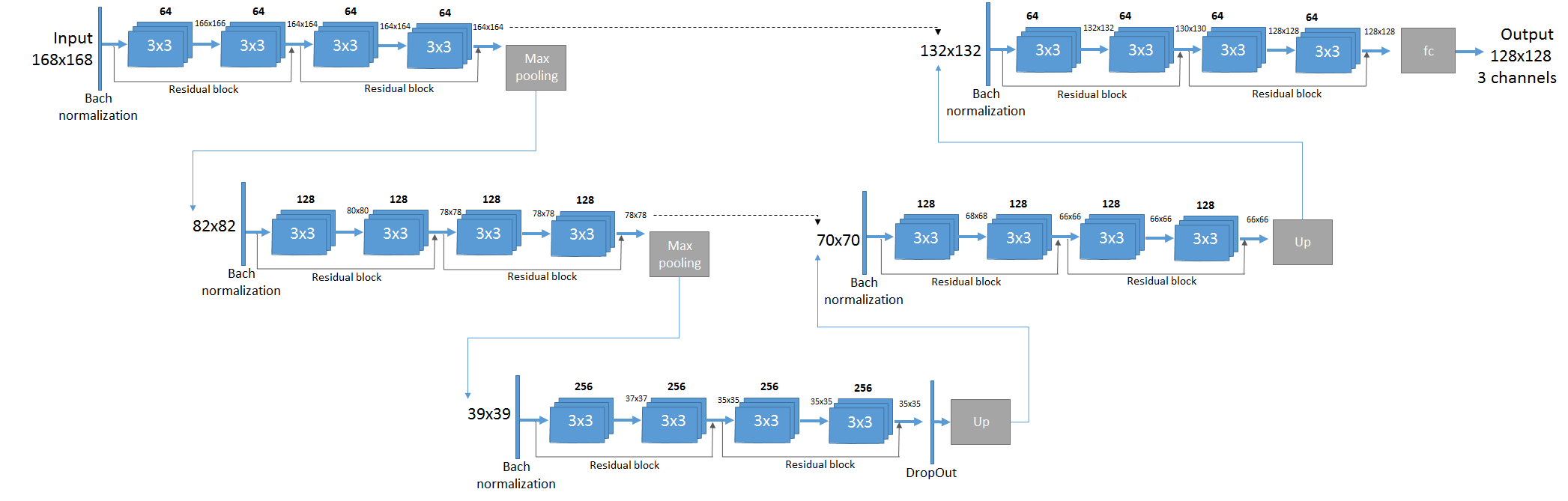
# **Specs**

השתמשנו במחשב הנייד שברשותנו, בעל המפרט הבא:

|  |  |
| --- | --- |
| CPU | Intel Core i7 (6th Gen) 6820HQ / 2.7 GHz |
| Graphics Processor | NVIDIA Quadro M1000M - 2 GB GDDR5 SDRAM |
| RAM | 32 Gb |

# **Architecture 1**

הארכיטקטורה הראשונה שניסינו היא שילוב של Unet עם residual blocks. הרשת מקבלת את התמונות ועושה להם resize לגודלן המקורי עם תוספת של 40 פיקסלים לאורך ולרוחב. שינוי הגודל קריטי לקבלת תמונת פלט בגודל המקורי של התמונה מכיוון חלק משכבות הקונבולוציות הן ללא zero padding.



איור 1: דוגמה לרשת עם input של 128x128

## פרמטרים, תכונות ואימון הרשת:

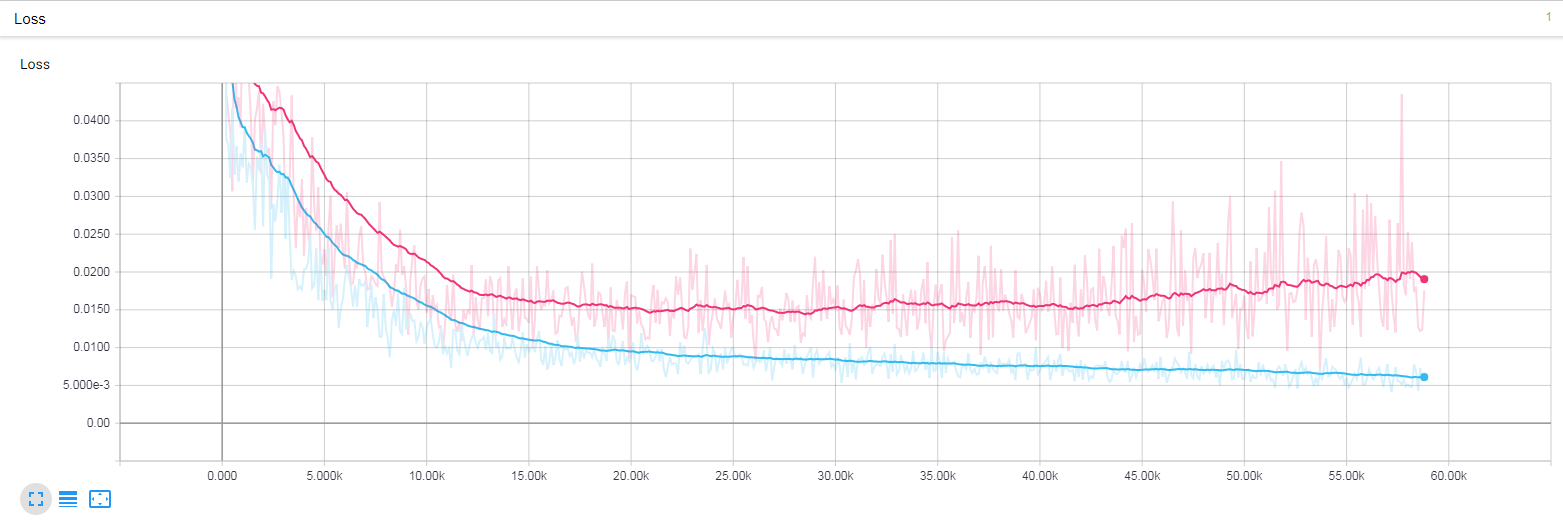
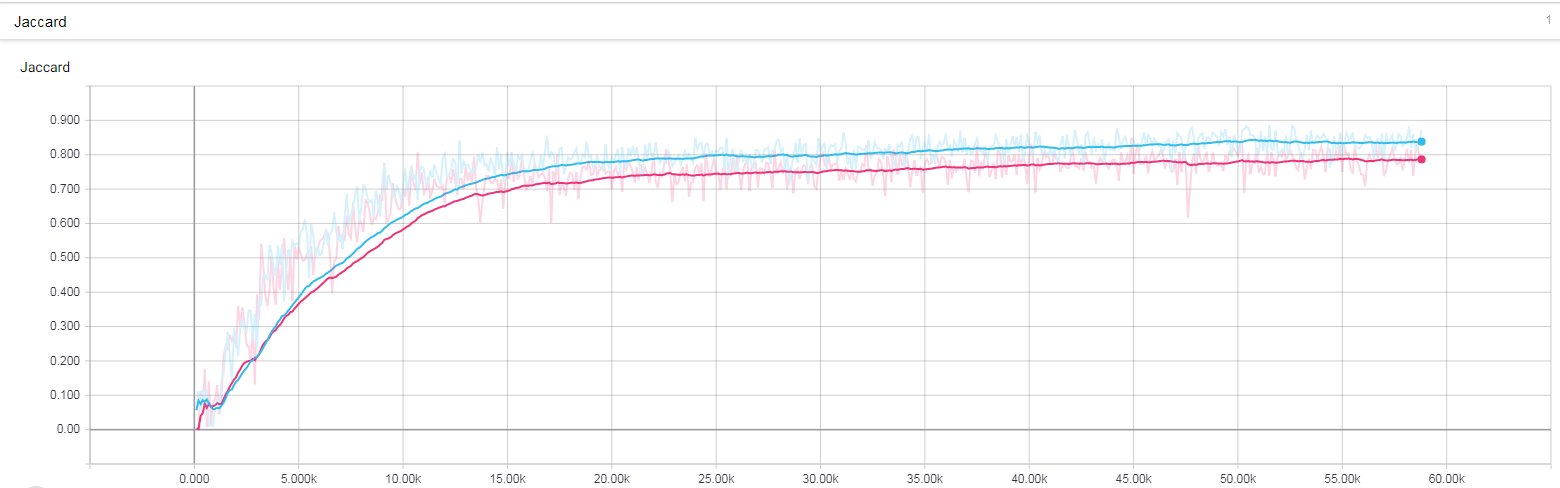
תכונות:

* לפני כל שכבת Down sample וUp sample- יש שכבה של Bach Normalization
* ברמה התחתונה ביותר של הUnet מבוצע Drop out
* כל הקרנלים של שכבות הקובולוציה הן בגודל קבוע של 3X3
* כל שכבת קונבולוציה מאותחל בעזרת Xavier, ומבוצע בה רגולריזצית L2

Params:

* Augmentation: Provieded
* Bath Normalization : at the beginning of each layer
* Drop-out : In the bottom of the U layers
* Learning rate: 0.0001
* Weights for segmentation : (0.14 , 0.16 , 0.7)
* Crop size: 128x128 , 10 crops per image
* Stride: 1 in each Convolution
* Batch size: 18
* Loss function : provided cross-entropy
* Optimizer : AdamOptimizer

Results – U-Net iteration 1



3: LOSS GRAPH Red line= validation Blue line= Training

2: Jaccard graph Red line= validation Blue line= Training

בארכיטקטורה זו הגענו לתוצאות הבאות:

Loss train: 0.0065 Loss Val: 0.018

Jaccard train : 0.89 Jaccard Val: 0.79

ניתן לראות כי לאחר K20 איטרציות החלק להיות overfitting למודל, כמו כן התוצאות ברשת זו לא היו מספקות.

# Architecture 2

במטרה לשפר את תוצאות הסגמנטציה, בנינו ארכיטקטורה נוספת המבוססת על FusionNet. בדומה לארכיטקטורה הקודמת, ה-FusionNet מבוססת על Unet עם בלוקים של resNet. בניגוד ל-Unet הקלאסי, בארכיטקטורה זו מעבר המידע בין הdown sampling ל-up sampling מבוצע על ידי חיבור element wise במקום concatenate, גודל הקרנלים אינו קבוע ומשתנה בהתאם לרמה של הרשת.   
כחלק משיפור תהליך הלימוד, עשינו שימוש במספר סוגי loss שונים, בין היתר ב - "L1 Haber Loss", בנוסף ניסינו לפתח loss משלנו. האינטואיציה בעבור הפיתוח היתה, הרצון למשקל באופן אדפטיבי את השגיאה שמתקבלת מכל סגמנט בנפרד, כך שנוכל לתת משקל גבוהה יותר למחלקה שאנו מעוניינים למזער את השגיאה שלה באופן יותר חד. לשם כך פתחנו את פונקציית ה- Loss האדפטיבית הבאה:

cl – Cross Entropy Loss per pixel.

el – Sum of cl pixels loss where the pixels are marked as edge.

bl – Sum of cl pixels loss where the pixels are marked as background.

fl – Sum of cl pixels loss where the pixels are marked as foreground.

de – The decay loss average of the edge class.

db – The decay loss average of the background class.

df – The decay loss average of the foreground class.

tl = el + bl + fl

ce – The constant weights for edge as defined on Params.py

cb – The constant weights for foreground as defined on Params.py

cf – The constant weights for background as defined on Params.py

עבור , אנו לומדים decay-moving-average, כך שנותנים 1% לערך החדש ו 99% לערך הנוכחי – כך שלמעשה מקבלים החלקה של אחוזי השגיאה.

באמצעות המדדים הנ"ל נגדיר את ה loss החדש כך:

ניתן לראות תיאור מפורט של קוד הפונקציה בנספחים.

Params:

* Augmentation: default + random angel rotation + random resize and center of 0.5 percent on each dimension + randomimage crop.
* Drop-out : On the bridge layers and after each decoding layer.

(Drop percent 0.3)

* Learning rate: 1e-3
* Weights for segmentation : (0.04, 0.06, 0.9) + adaptive weight learning based on the relative error of each classify category.
* 5 crop per image.
* Crop size + Batch Size:
  + for the first 64k iterations (batch size: 1)
  + from iterations 64k to 88k (batch size: 5)
  + from iterations 88k to 118k (batch size: 20)
  + from iterations 118k to 170k (batch size 50)
* Stride: 1 in each Convolution in the resnet.
* Max Pooling with strides of for the encoders.
* Transpose convolutions wit stride of for the decoder.
* Kernel L2 regularization for convolutions and transpose convolutions with scale of 0.3.
* Optimizer : AdamOptimizer

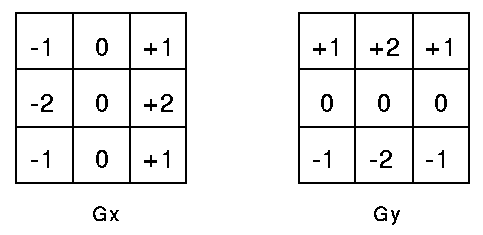
# **תובנות בין האינטרציות השונות**

בזמן אימון ארכיטקטורה זו ביצענו בחינה של מספר ואריאציות בשימוש פרמטרים שונים, כחלק משינוי המאפיינים השונים של הרשת הגענו לתובנות הבאות:

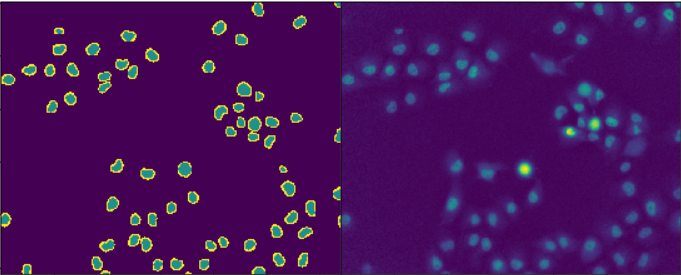
* יותר מדי שכבות downsampling/upsampling מכניס למודל שגיאה נצברת בעקבות חוסר דיוק בשחזור המימד (upsampling)
* מספיק שלוש רמות downsampling/upsampling בכדי להגיע לדיוק גבוהה מבלי להכנס ל overfitting.
* התמודדות עם overfitting:
  + ככל שהגרעין של כל convolution גדל, יש גידול רב בפרמטרים המכניס סכנת overfitting.
  + שילוב של dropout, L2 regularization, multiple res-net layers מורידים את ה overfitting.
  + שכלול ה augmentation מוריד את ה overfitting.
* כדאי להגדיל את ה batch-norm בשלבים מתקדמים של הלימוד מאשר להקטין את ה learning-rate, מכיוון שאנו עבדנו עם gpu של GB2 הורדנו את גודל ה slice מהתמונה ככל שה batch גדל.
* fusion-net מאפשר להשתמש ביותר שכבות מכיוון שהוא מבצע סכימה במקום שרשור של ה skip-connect, ועדיין מאפשר למודל להשתמש בצורה טובה במידע המועבר ע"י ה skip-connect.
* למרות שניסינו מספר losses שונים, הגענו למסקנה ש cross-entropy מניב את התוצאות הטובות ביותר – אם כי הגדרנו cross-entropy loss קצת שונה מהפונקציה המקורית, כך שהיא מנצלת בצורה טובה יותר את השגיאה של כל קטגוריה.
* בנוסף למשקל הקבוע המוגדר לכל קטגוריה, הגדרנו משקל אדפטיבי ע"פ השגיאות בכל קטגוריה לאורך היסטוריה אחרונה (דואך מעריכית).
* בכדי לעבוד עם batch-norm בצורה הטובה ביותר יש להכניס אותו לפני ה activation-layer.
* במקום ReLU ו leakyReLU השתמשנו ב Parametric ReLU – וריאציה של leakyReLU שלומד כפרמטר כמה להדליף.
* ככל שהמשכנו להגדיל את ה batch-size והקטנו את ה crop-size קיבלנו תוצאות טובות יותר, הגענו לדיוק של 93% בולידציה. אולם בזמן הרצת ה- test הסופי, התוצאות נראו לא תקינות ולכן הגשנו את הגירסה של ה 89% אחוזי הדיוק. ככל הנראה הסיבה תמונה בכך שה batch-size היה גדול מאוד וה crop-size קטן, לכן ה jaccard הניב תוצאה טובה בעוד שהמודל לא תפקד טוב על תמונה מלאה. הדבר נבע מכך שדירוג הרקע גבר על שאר המחלקות.

## post-prossesing

* כדי לקבל מסגרת אופטימלית – הוספנו לפלט הסופי חידוד ע"י sobel filter. פילטר זה מבצע נגזרת מסדר שני לתמונה על מנת לחדד Edges.



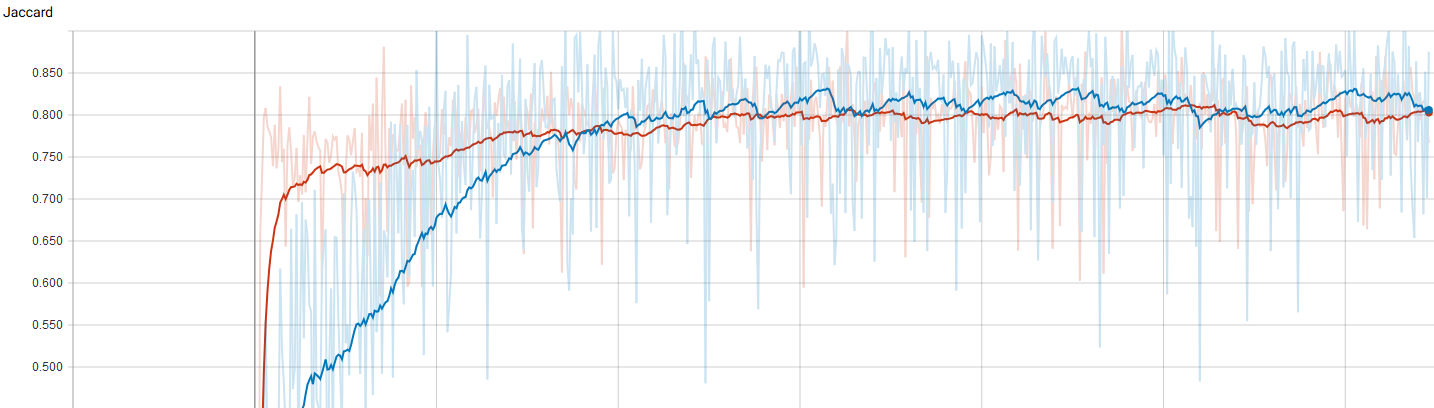
sobel filter



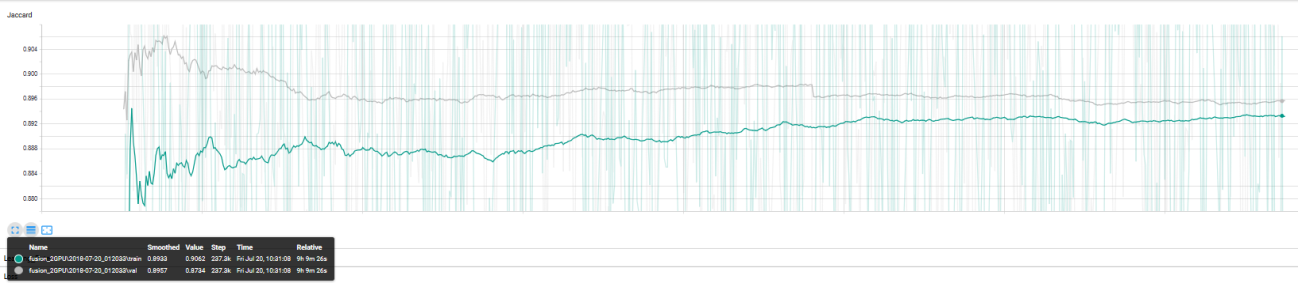
מצד ימין- לפני sobel filtter, מצד שמאל אחרי sobel filtter

# תוצאות ה Jaccard של המודל הטוב ביותר

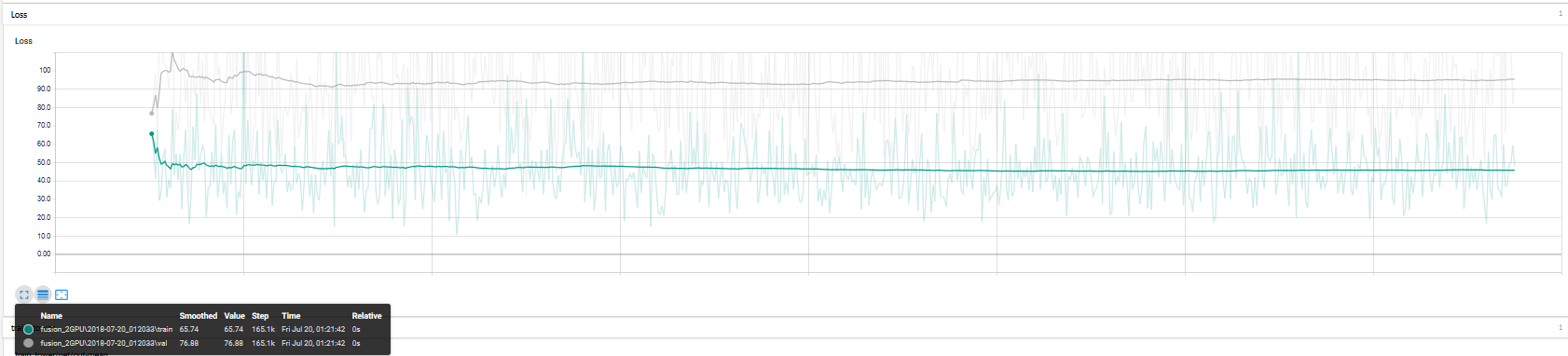
במודל זה הגענו לתוצאות של 89% דיוק במדד ה-Jaccard על סט הולידציה, ו93% דיוק על סט האימון



jaccard on 237k iteration scale. Blue -trainning, Red - validation



jaccard lasted iterations. green -trainning, gray - validation



loss graph on lasted iterations. green -trainning, gray - validation

# **Analysis and conclusions**

במסגרת פרויקט זה התנסנו בפתרון בעיית סגמנטציה בעבור תאים ביולוגיים. במהלך הפרויקט ממשנו שני סוגי רשתות עקריים : U-net, FusionNet, כאשר כל רשת ניסינו לאמן בעזרת מאפיינים שונים.  
התוצאות הטובות ביותר התקבלו עבור רשת מסוג FusionNet כאשר במהלך האימון המודל הגענו לתוצאות של %89 דיוק בסט הואלידציה עבור ה jaccard. בשל המשקול האדפטיבי קוי המתאר של התאים נעלמו,בכדי להתמודד עם בעיה זו עשינו post-prossesing למציאת ה- edjes בעזרת sobel filter.

טבלת השוואה:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Properties | Training Loss | Jaccard Training | Loss Validation | Jaccard Validation |
| 1 | Regular U-net | 0.094 | 0.8 | 0.021 | 0.74 |
| 2 | [Architecture 1 - resUnet](#_Architecture_1) | 0.0065 | 0.89 | 0.018 | 0.79 |
| 3 | Architecture 2 – Parametric ReLU, vanilla cross entropy | 0.002 | 0.91 | 0.0046 | 0.85 |
| 4 | Architecture 2 – vanilla Augmentation + L1 regularization + Huber loss | 13.7 | 0.95 | 69.38 | 0.84 |
| 5 | [Architecture 2 – final resFusionNet](#_Architecture_2) + solve overfitting + adaptive cross enthropy | 48 | 0.93 | 94 | 0.89 |

הערה: כל התוצאות התקבלו עם החלקה של 0.975, כאשר התוצאות מגיעות להתכנסות (לא מקסימום מקומי)

# Bibliography

* Quan, T. M., Hildebrand, D. G., & Jeong, W. K. (2016). Fusionnet: A deep fully residual convolutional neural network for image segmentation in connectomics. arXiv preprint arXiv:1612.05360.‏
* Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... & Sánchez, C. I. (2017). A survey on deep learning in medical image analysis. Medical image analysis, 42, 60-88.‏
* Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017, February). Inception-v4, inception-resnet and the impact of residual connections on learning. In AAAI (Vol. 4, p. 12(
* Jin-Yu, Z., Yan, C., & Xian-Xiang, H. (2009, April). Edge detection of images based on improved Sobel operator and genetic algorithms. In Image Analysis and Signal Processing, 2009. IASP 2009. International Conference on (pp. 31-35). IEEE.‏

# נספחים

לנוחיותכם בבדיקה, הוספנו את הקוד של כל מודל

## קוד ארכיטקטורה ראשונה:

import tensorflow as tf  
from tensorflow import concat, variable\_scope  
from functools import partial  
regularizer = tf.contrib.layers.l2\_regularizer(scale=0.1)  
xavier\_init = tf.contrib.layers.xavier\_initializer  
conv2d = partial(tf.layers.conv2d,  
 activation=tf.nn.relu,  
 kernel\_initializer=xavier\_init(),  
 kernel\_regularizer=regularizer,  
 padding="VALID")  
conv2d\_linear = partial(tf.layers.conv2d,  
 kernel\_initializer=xavier\_init(),  
 kernel\_regularizer=regularizer,  
 padding="SAME")  
Cropping2D = tf.keras.layers.Cropping2D  
UpSampling2D = tf.keras.layers.UpSampling2D  
max\_pooling2d = tf.layers.max\_pooling2d  
dropout = tf.layers.dropout  
batch\_norm = partial(tf.layers.batch\_normalization, training=True)  
  
  
  
  
def residualDownsample(in\_data, filters, cropping=None, last=False,  
 name='DownSample', reuse=False):  
  
 with variable\_scope(name, reuse=reuse):  
 in\_data = batch\_norm(in\_data, name='batchNormalization')  
 conv1 = conv2d(in\_data, filters, 3)  
 conv2 = conv2d(conv1, filters, 3)  
 identity = Cropping2D(cropping=2)(in\_data)  
 bottle=conv2d\_linear(identity, filters, 1)  
 identityShortcut=bottle+conv2  
 #identityShortcut = concat([identity, conv2], axis=3)  
 resBlock = residualBlock(identityShortcut, filters, 3)  
 if last:  
 return dropout(resBlock)  
 crop = Cropping2D(cropping=cropping)(resBlock)  
 pool = max\_pooling2d(resBlock, 2, 2)  
 return crop, pool  
  
  
def residualUpsample(in\_data, crop, filters, name='UpSample', reuse=False):  
  
 with variable\_scope(name, reuse=reuse):  
 in\_data = batch\_norm(in\_data, name='batchNormalization')  
 up = UpSampling2D(size=(2, 2))(in\_data)  
 #conv1 = conv2d(up, filters, 2, padding="SAME")  
 merge6 = concat([crop, up], axis=3)  
 bottle=conv2d\_linear(merge6, filters, 1)  
 conv2 = conv2d(merge6, filters, 3)  
 conv3 = conv2d(conv2, filters, 3)  
 identity = Cropping2D(cropping=2)(bottle)

identityShortcut = identity+conv3

resBlock = residualBlock(identityShortcut, filters, 3)  
  
 return resBlock  
  
  
def residualBlock(input, channels, kernel, name='residualBlock', reuse=False):  
 with tf.variable\_scope(name, reuse=reuse):  
 conv = conv2d\_linear(input, channels, kernel)  
 conv = conv2d\_linear(conv, channels, kernel)  
 shortcut=conv+input  
 conv = tf.nn.relu(shortcut)  
 return conv  
  
  
def unet(in\_data, name='UNet', reuse=False):  
  
 assert in\_data is not None  
 with variable\_scope(name, reuse=reuse):  
 h = in\_data.get\_shape().as\_list()[1]  
 w = in\_data.get\_shape().as\_list()[2]  
 in\_data = tf.image.resize\_images(in\_data, [h + 40, w + 40])  
  
 crop1, pool1 = residualDownsample(in\_data, 64, 16, name='DownSample1', reuse=reuse)  
 crop2, pool2 = residualDownsample(pool1, 128, 4, name='DownSample2', reuse=reuse)  
 drop = residualDownsample(pool2, 256, name='DownSample3', reuse=reuse, last=True)  
 up1 = residualUpsample(drop, crop2, 128, name='UpSample1', reuse=reuse)  
 up2 = residualUpsample(up1, crop1, 64, name='UpSample2', reuse=reuse)  
 conv = conv2d(up2, 3, 3, padding="SAME", name='Conv')  
 out = conv2d\_linear(conv, 3, 1, name='Out')  
 return out

## קוד ארכיטקטורה שנייה:

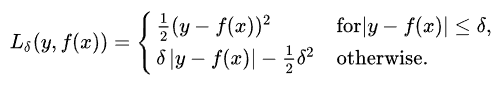
**פונקציית Loss:**

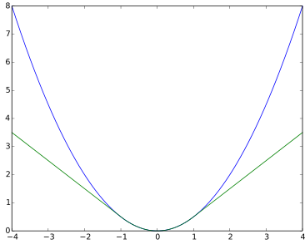
valid\_pixel\_loss = tf.multiply(pixel\_loss, gt\_valid)  
pixel\_loss\_weighted = 0  
loss\_bg = tf.reduce\_sum((valid\_pixel\_loss \* gt\_bg))  
loss\_fg = tf.reduce\_sum((valid\_pixel\_loss \* gt\_fg))  
loss\_edge = tf.reduce\_sum((valid\_pixel\_loss \* gt\_edge))  
total\_loss = loss\_bg + loss\_fg + loss\_edge + tf\_eps  
if decay\_bg is None:  
 decay\_bg = loss\_bg / total\_loss  
else:  
 decay\_bg = decay\_bg \* 0.99 + (loss\_bg / total\_loss) \* 0.01  
if decay\_fg is None:  
 decay\_fg = loss\_fg / total\_loss  
else:  
 decay\_fg = decay\_fg \* 0.99 + (decay\_fg / total\_loss) \* 0.01  
if decay\_edge is None:  
 decay\_edge = loss\_edge / total\_loss  
else:  
 decay\_edge = decay\_edge \* 0.99 + (decay\_edge / total\_loss) \* 0.01  
cw = params.class\_weights  
loss = (loss\_bg \* decay\_bg \* cw[0] + loss\_fg \* decay\_fg \* cw[1] + loss\_edge \* decay\_edge \* cw[2])

**fusionNet**

*"""  
Fusion-Net Implementation based on: https://arxiv.org/abs/1711.07341 with some modifications and our adaptations.  
"""*from \_\_future\_\_ import absolute\_import, division  
import tensorflow as tf  
from tensorflow import variable\_scope  
from functools import partial  
from utils import bn\_conv\_layer, bn\_conv2d\_transpose\_layer, conv2d\_linear  
  
UpSampling2D = tf.keras.layers.UpSampling2D  
max\_pooling2d = tf.layers.max\_pooling2d  
dropout = partial(tf.layers.dropout, rate=0.1)  
  
  
class FusionNet(object):  
 def \_\_init\_\_(self, layers, kernel\_size, kernel\_num, num\_repeat):  
 *"""  
 Create the fusion-net model instance.* ***:param*** *layers: The number of down and up layers.* ***:param*** *kernel\_size: The size of the kernel window.* ***:param*** *kernel\_num: The size of the initial resulting kernel count which grow by two on each depth.* ***:param*** *num\_repeat: Number of internal convolutions between the resnet connection.  
 """* self.layers = layers  
 self.kernel\_size = kernel\_size  
 self.kernel\_num = kernel\_num  
 self.num\_repeat = num\_repeat  
  
 def conv\_res\_conv\_block(self, prev\_layer, output\_dim,  
 name="ConvResConvBlock", training=True):  
 *"""  
 Resnet block.* ***:param*** *prev\_layer: Previous incomming layer.* ***:param*** *output\_dim: The dimension of the each convolusion kernel dimension.* ***:param*** *name: The name for this layer.* ***:param*** *training: Whether or not it is a train run.  
 """* with variable\_scope(name):  
 conv1 = bn\_conv\_layer(prev\_layer, output\_dim, self.kernel\_size, name="Conv1", training=training)  
 output = conv1  
 for i in range(self.num\_repeat):  
 conv = bn\_conv\_layer(output, output\_dim, self.kernel\_size, name="conv\_{0}".format(i), training=training)  
 output = conv  
 res = conv1 + output  
 conv2 = bn\_conv\_layer(res, output\_dim, self.kernel\_size, name="Conv2", training=training)  
 return conv2  
  
 def encoder(self, prev\_layer, name='Encoder', training=True):  
 *"""  
 The encoding layer block.* ***:param*** *prev\_layer: Previous incomming layer.* ***:param*** *name: The name for this layer.* ***:param*** *training: Whether or not it is a train run.  
 """* with variable\_scope(name):  
 pool = prev\_layer  
 skip\_links = []  
 for i in range(self.layers):  
 down = self.conv\_res\_conv\_block(pool, self.kernel\_num \* (2 \*\* i),  
 name="DownSample{0}".format(i + 1),  
 training=training)  
 bn = down  
 pool = max\_pooling2d(bn, 2, 2, name="Pool{0}".format(i + 1))  
 skip\_links.insert(0, down)  
 return pool, skip\_links  
  
 def decoder(self, prev\_layer, skip\_links, name='Decoder', training=True):  
 *"""  
 The decoding block.* ***:param*** *prev\_layer: The encoded output.* ***:param*** *skip\_links: The skip\_links saved by the encoder block.* ***:param*** *name: The name for this layer.*  ***:param*** *training: Whether or not it is a train run.  
 """* with variable\_scope(name):  
 d = prev\_layer  
 l = len(skip\_links)  
 for i, link in enumerate(skip\_links):  
 cur\_kernel\_num = self.kernel\_num \* (2 \*\* (l - i - 1))  
 bn = d  
 res = 0.5 \* (bn\_conv2d\_transpose\_layer(bn, cur\_kernel\_num, self.kernel\_size, strides=2,  
 name="Res{0}".format(l - i), training=training) + link)  
 up = self.conv\_res\_conv\_block(res, cur\_kernel\_num, name="UpSample{0}".format(l - i),  
 training=training)  
 d = dropout(up, name="Dropout{0}".format(l - i), training=training)  
 return d  
  
 def fusion\_net(self, input\_data, training):  
 *"""  
 Create the fusion-net model.* ***:param*** *input\_data: Input to the layer.* ***:param*** *training: Whether or not it is a train run.* ***:return*** *The model.  
 """* encode\_vec, skip\_links = self.encoder(input\_data, training=training)  
 bridge = self.conv\_res\_conv\_block(encode\_vec, self.kernel\_num \* (2 \*\* self.layers),  
 name="Bridge", training=training)  
 d = dropout(bridge, training=training)  
 decode\_vec = self.decoder(d, skip\_links, training=training)  
 output = conv2d\_linear(decode\_vec, 3, self.kernel\_size, name="Output")  
 return output

Huber loss (not used on best model architecture 2):





def smooth\_l1(y\_true, y\_pred):  
 *"""  
 Smooth L1 Huber loss.* ***:param*** *y\_true: The ground-truth.* ***:param*** *y\_pred: The predictions.* ***:return****: The loss result.  
 """* x = tf.abs(tf.to\_float(y\_true) - tf.to\_float(y\_pred))  
 x = tf.where(x < HUBER\_DELTA, 0.5 \* x \*\* 2, HUBER\_DELTA \* (x - 0.5 \* HUBER\_DELTA))  
 return tf.reduce\_sum(x)