Bacteria segmentation for medical images

COMPUTER ENGINEERING FOR ROBOTICS AND SMART INDUSTRY

MACHINE LEARNING & ARTIFICIAL INTELLIGENCE - 2020/2021

Francesco Taioli
VR465453
francesco.taioli@studenti.univr.it

Sebastiano Scodro
VR457975
sebastiano.scodro@studenti.univr.it

Contents

1 MOTIVATION AND RATIONALE

When choosing the dataset and the area of interest for the Artificial intelligence & machine learning course project, we wanted to tackle a solution to a problem that could be useful in a real scenario. After some research, we came across Semantic segmentation, a computer vision task used in a lot of applications where we label specific regions of an image according to what's being shown. In short, semantic segmentation is the task of linking each pixel in an image to a class label. It is a form of pixel-level prediction because each pixel in an image is classified according to a category. In the examples below you can see some real-world applications:

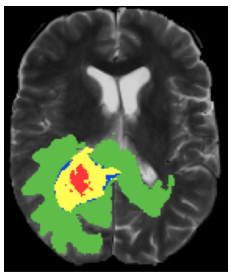


Figure 1: Tumor segmentation of brain

Figure 2: Background removal - bokeh effect - editing of images and videos

Figure 3: Street segmentation for autonomous vehicles

Figure 4: Street segmentation for autonomous vehicles

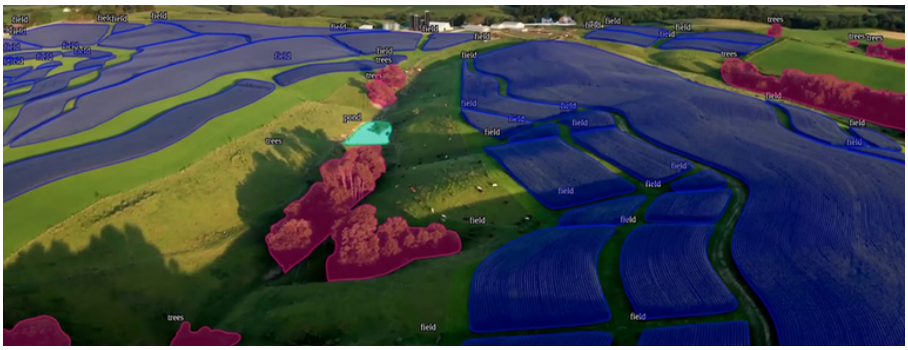


Figure 5: Segmentation of aerial views (in this case for agriculture)

Semantic segmentation is extremely useful in the field of autonomous vehicles. In fact, it allows the autonomous car to understand the objects in the scene such as other cars, pedestrians, bicycles, traffic signs and so on. While looking for a dataset we have encountered some problems. First of all, the commonly used datasets used for autonomous vehicles, as you can imagine, are huge (more than 30 GB). Secondly, our computational power is limited (only one of us could train in local, the other had to use google colab). Moreover, the datasets that we have found with a size that allows us to take them into consideration, were damaged or with some problems with respect to the quality of the image. After some research, we have found another interesting problem: bacteria and cells segmentation. Here, given darkfield microscopy images, our aim is to segment the blood cells, the bacteria and the background.

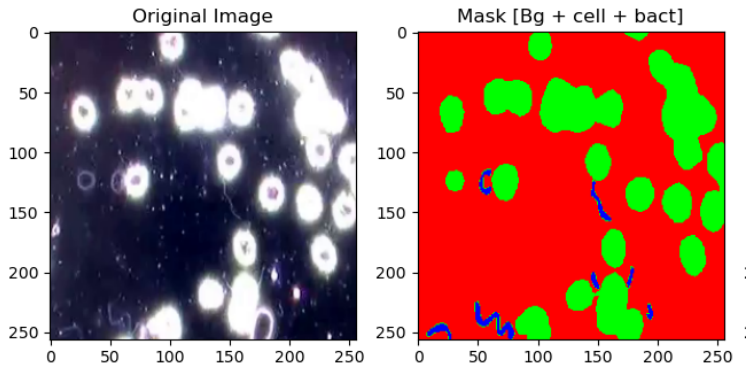


Figure 6: Example of image + mask (red, green and blue as background, blood cell and bacteria)

The dataset is available **here**. As you can see this is a challenging problem, significant for research purposes. In the dataset home page we have found: *"Spirochaeta is a genus of bacteria classified within the phylum Spirochaetes. Included in this dataset are 366 darkfield microscopy images and manually annotated masks which can be used for classification and segmentation purposes. Detecting bacteria in blood could have a huge significance for research in both the medical and computer science field"*

2 STATE OF THE ART

The problem of semantic segmentation is gaining an increasing amount of attention in these years: as a consequence, we could find many interesting papers dealing with the problem applied to some famous datasets like those of cityscapes (segmentation for autonomous driving), ADE20K (landmark image segmentation dataset, containing both indoor and outdoor images) and many others.
We have spent some time looking for state-of-the-art architectures to perform image segmentation and the most common techniques to get good results in this kind of problem.
We have found that the most common models for image segmentation are the following ones:

U-Net

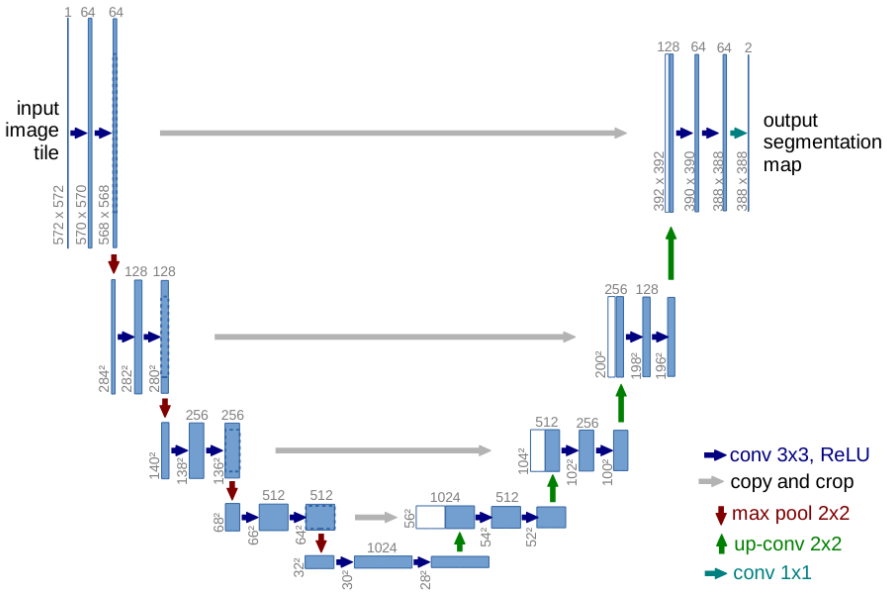


Figure 7: U-net architecture

U-Net [7] is a convolutional neural network architecture (that expanded with few changes in the CNN architecture) originally developed by Olaf Ronneberger for segmenting biomedical images where few training sample are present and the localization is important. The architecture consists of a contracting path (left side) to capture context and a symmetric expanding path (right side) that enables precise localization. The expansive path is more or less symmetric to the contracting path, and yields a u-shaped architecture. Moreover, in the original paper, we found some interesting hints regarding data augmentation when few images are present.

PSPNet

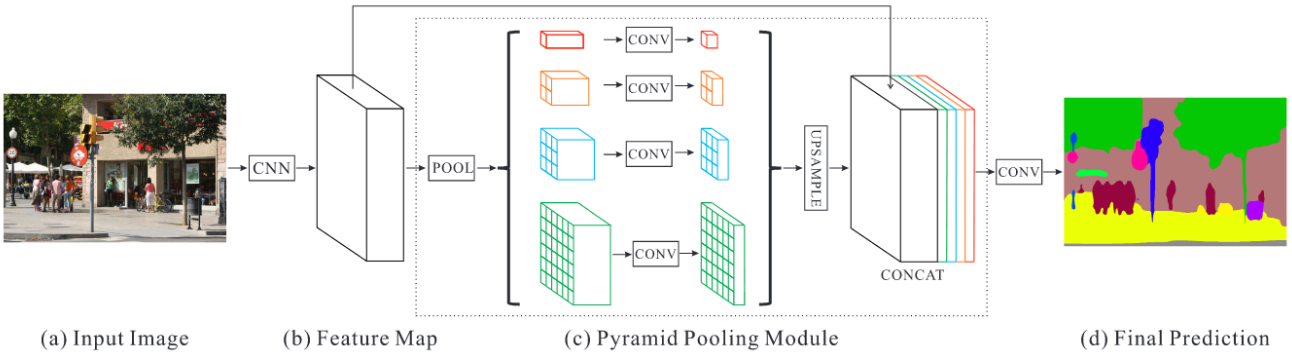


Figure 8: PSPNet architecture

PSPNet [3], or Pyramid Scene Parsing Network, is a semantic segmentation model that utilises a pyramid parsing module that exploits global context information by different-region based

context aggregation. The local and global clues together make the final prediction more reliable.
From the paper: *Given an input image (a), we first use CNN to get the feature map of the last convolutional layer (b), then a pyramid parsing module is applied to harvest different sub-region representations, followed by upsampling and concatenation layers to form the final feature representation, which carries both local and global context information in (c). Finally, the representations are fed into a convolutional layer to get the final per-pixel predictions (d).*

LinkNet

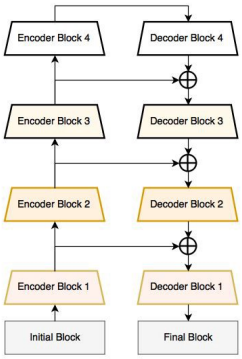


Figure 9: LinkNet architecture

LinkNet [1] is a fully convolution neural network used for semantic image segmentation. It consists of encoder and decoder parts connected with skip connections. Encoder blocks extract features of different spatial resolutions (skip connections) which are used by decoder blocks to define accurate segmentation mask. There are different implementations that differ in the number of skip connections (3, 4 or 5).

3 OBJECTIVES

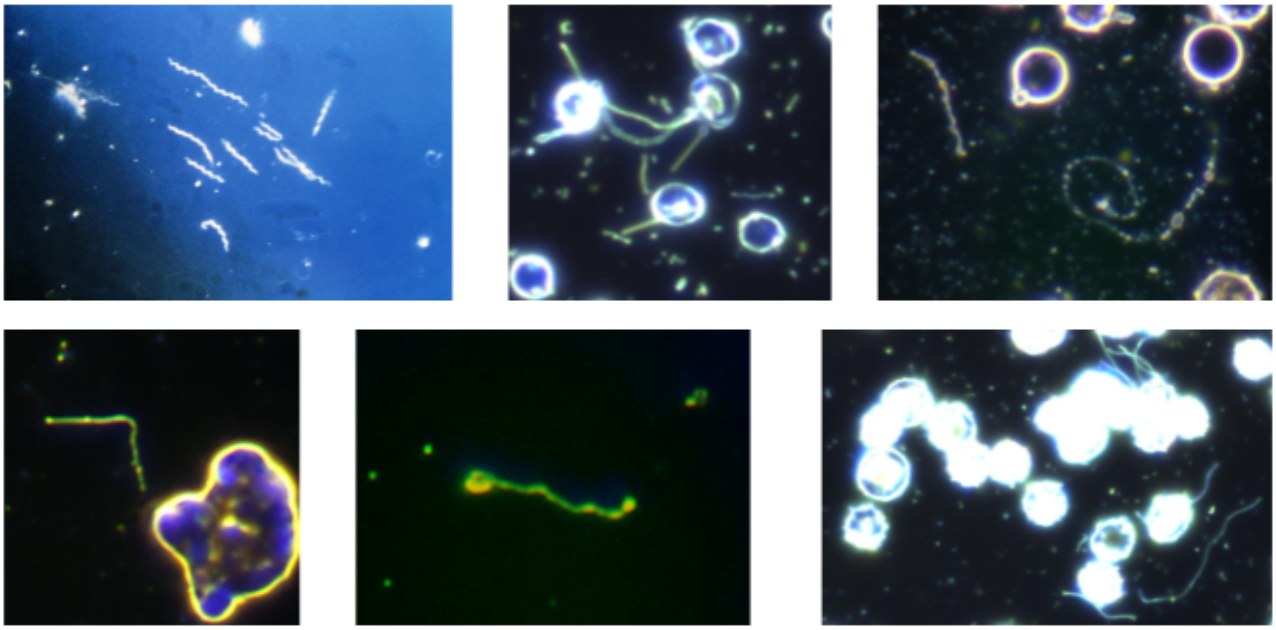
The aim of our project is, given a darkfield microscopy image, to segment it into background, blood cell and the specific type of bacteria. Doing this, we want to find the network architecture that gives us the best result (measured with the appropriate metric like MeanIoU). In order to do this, we have tried several different combinations of architectures and exploited several known techniques to improve the training. Moreover, as we are dealing with a very small dataset, we have used cross validation with the best model in order to get a better idea of the real performance of the model. We should have done it for all the models but, due to our computational power and consequently the time for all the tests, it wasn't feasible.

4 METHODOLOGY

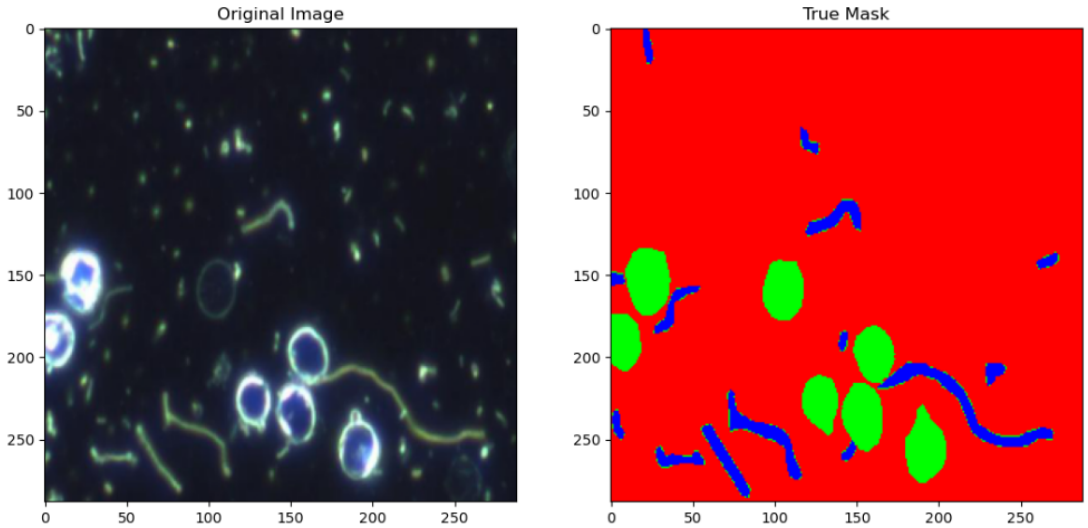
4.1 Dataset

The dataset is composed of 366 images and the corresponding masks. For the images we have essentially three problems: the images have different sizes, the dataset is quite small and finally,

there are a lot of different light conditions, contrast and content. In the dataset we can find a broad range of different sizes: 1014x774, 454x404, 1500x840 and so on. To standardize the images, we have decided to resize them all to 288x288. We choose this specific size because with our test we found that these particular dimensions give us the best results. To get a better idea of the various images, you can find some raw examples below:



Moreover, during the first analysis, we have found also some problems in the masks. In fact, the borders of the different regions in the masks are not always perfect, probably because the dataset was initially created only to differentiate background from cells/bacteria (which were combined into a single class). In the image below you can find one example.



As you can see, we have some green pixels on the border of the bacteria. We think that this problem is created by the gathering method: it was gathered and annotated by students (hand-on experience) who probably didn't pay too much attention to these little but very important details. Moreover, we think that not all bacteria and blood cells were annotated. However,

despite these problems, we found the dataset was good enough to perform segmentation and obtain meaningful results.

4.2 Tools

Computational

We started by using Google Colab as our only computational tool but it quickly became really difficult to take note of all the architectures and other experiments that we made. After that, we have refactored that python notebook into a more structured python project. For the training we are using one pc with AMD ryzen 7 3700x, 32 GB of ram and RTX 2060 super. Moreover, we have prepared a git repository with all the code.

Libraries

Apart from the standard machine learning libraries (numpy, matplotlib, sklearn) we have used:

- Albumentations: a famous Python library for image augmentation
- Segmentation model: Python library with useful metrics, losses and 4 state of the art neural net architectures.
- Tensorflow Keras
- Kaggle for the dataset

4.3 Methods

Pre-processing

The first operation when using deep network architectures is to prepare the data in the right way. In our case we have a segmentation problem so we need images and their corresponding masks. If we consider a 288 x 288 rgb image we can write it with a numpy notation, namely (288, 288, 3) where the three channels stand for the red, green and blue pixel, respectively. For the mask instead, we have to do some pre-processing in order to reshape it in the right way. In fact, the mask from the dataset is loaded as grey scale image, in numpy format (288 x 288) with pixels' values that vary from 0 to 2. As a result of our image segmentation pipeline we want that the image contains 3 classes (remember: background, blood cell and bacteria) so we need also the mask in a three-channel format. In order to do this, we create a temporary mask with shape (288, 288, 3) where in the first channel we can find all the pixels for the background, in the second channel all the pixels for the blood cell and, finally, in the third and last channel we can find the bacteria pixels. Note that now in every channel we can find either 0 or 1 as pixels' value.

```python
def create_layer_of_color(mask):
    img = np.asarray(mask)
    tmp = np.zeros((HEIGHT, WIDTH, 3), dtype=int)

    tmp[:, :, 0] = img == 0  # background
    tmp[:, :, 1] = img == 1  # blood cell
    tmp[:, :, 2] = img == 2  # bacteria
    return tmp
```

Moreover, after loading the images in a numpy array, we normalize them dividing the value of each pixel (in each channel) by 255.

Net Architectures

To solve this image segmentation problem we have tried different architectures: in particular, we have tried U-Net, LinkNet, PSPNet, SegNet and FCN8 (we didn't report results of this last one because they were extremely low). Most of them were already described in the state-of-the-art section. We have trained them (note: they differ in training time since the number of trainable parameters ranges from $\sim$ 1 million to up to $\sim$ 50 million) multiple times with different parameters in order to see which one gives us the best result. Moreover, we saved the best model for each architecture using the ModelCheckPoint Callback of Keras. You can find more details in the Experiment & result section.

Data Augmentation

To perform augmentation on images we have decided to use a public available python library called Albumentations. We do this in order to mitigate the problem of the low amount of images: doing augmentation with many different filters and operations (like horizontal and vertical flips, changes of gamma/contrast/brightness/saturation, gaussian noise) we are able to obtain variations of our original images to be used for training our models.
Here is an example of image augmentation:

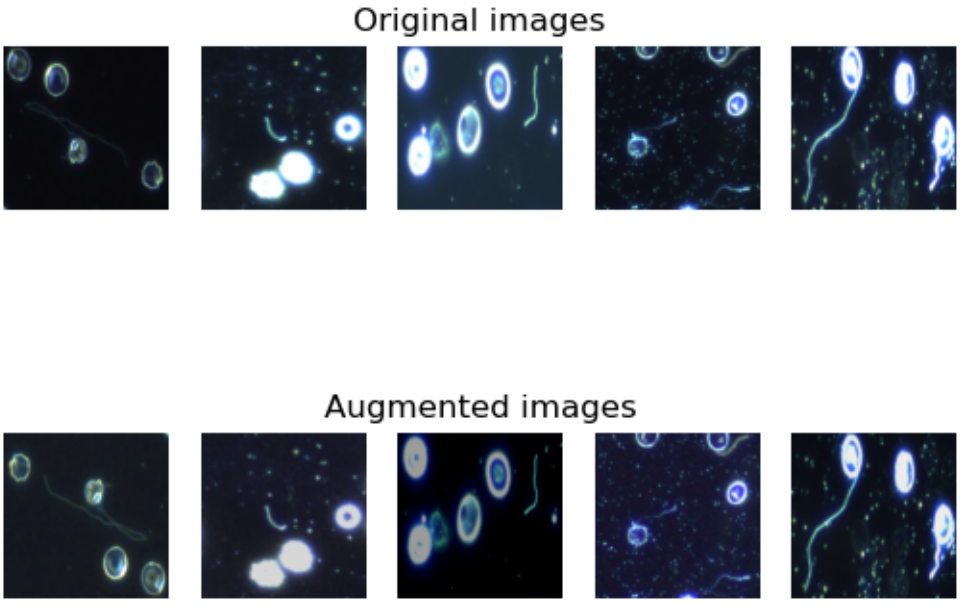


Figure 10: Example of augmentation

The code for the augmentation pipeline is shown here:

```
def augment(image, prob=0.2):
    """
    :param image: rgb image (dtype uint8)
    :return: augmented image (dtype uint8)
    """
    transform = A.Compose([
        A.RandomBrightnessContrast(p=prob),
        A.RandomContrast(p=prob),
        A.RandomGamma(p=prob),
        A.HueSaturationValue(p=prob),
        A.GaussNoise(p=prob)
    ])

    transformed = transform(image=image)
    return transformed["image"]
```

Note: there are also horizontal and vertical flips that aren't shown here.

4.4 Optimizers

SGD

Stochastic gradient descent with momentum[5].
The update rules for the weights are the following:

```
    Without momentum:
    w = w - learning_rate * g

    With momentum:
    velocity = momentum * velocity - learning_rate * g
    w = w + velocity

    where 'g' is the gradient
```

SGD performs a parameter update for each training example. The updates are frequent with a high variance that cause the objective function to fluctuate heavily.

RMSprop

RMSprop (root mean square propagation) is an adaptive learning rate method proposed by Geoff Hinton[5]. It uses an exponential moving average (to give more importance to recent values) of squared gradients to normalize the gradient balancing the step size (momentum), decreasing the step for large gradients to avoid exploding and increasing the step for small gradients to avoid vanishing. RMSProp automatically decreases the size of the gradient steps towards minima when the steps are too large (to avoid overshooting).
The update rule for the weights w works as follows:

$$v_t = \rho v_{t-1} + (1 - \rho) \cdot g_t^2$$

$$\Delta w_t = -\frac{\eta}{\sqrt{v_t + \epsilon}} \cdot g_t$$

$$w_{t+1} = w_t + \Delta w_t$$

where:

- η learning rate
- v_t exponential average of squares of gradients
- g_t gradient at time t of a certain weight w

Adam

Adaptive Moment Estimation[5]. In addition to storing an exponentially decaying average of past squared gradients v_t like RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum.
The decaying averages of past gradients m_t and past squared gradients v_t are computed as follows:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Since m_t and v_t are initialized as vectors of 0's, they are biased toward zero especially in the initial time steps. For this reason m_t and v_t undergo correction:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

The Adam update rule is the following:

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

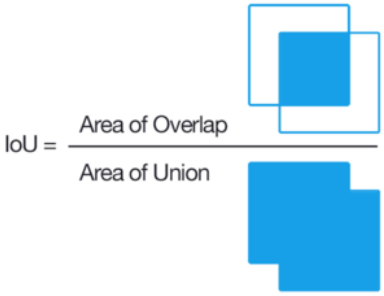
4.5 Metrics considered

Pixel accuracy

Pixel accuracy is probably the most commonly used metric to measure performance in semantic segmentation problems (in the cases of balanced datasets where the number of pixels for each class doesn't differ too much). It measures the percentage of pixels in an image that are classified correctly. It's easy to understand why this isn't the best metric around: in cases where the background fills most of the images and, consequently, the model predicts as background the whole image, we will still obtain good pixel accuracy since most of the images are in fact filled with background. If the classes of our images are imbalanced, we should use other metrics to get a better representation of the real performance.

MeanIoU (or Jaccard Index)

The Intersection-Over-Union [6], despite being a very straightforward metric, is another commonly used metric in semantic segmentation. As you can see from the figure, the IoU is the area of overlap between the predicted segmentation and the ground truth divided by the area of union between the predicted segmentation and the ground truth: for this reason its value ranges from 0% to 100% (the higher the better). When dealing



with multiclass segmentation (like in our case) the metric becomes "MeanIoU", taking the IoU of each class and averaging them. MeanIoU is a much better indication of the success of our segmentation compared to pixel accuracy since predicting all the image as background is highly penalized (because of the average of the 3 different IoU for each class it is somewhat more "fair" with respect to those classes which have a much lower amount of pixels in the masks).

4.6 Losses considered

Categorical cross entropy loss

It measures the Categorical Cross Entropy between the ground truth and the prediction. Cross-entropy loss[4] increases as the predicted probability for a pixel of being of a certain class diverges from the actual label. In multiclass problems (like in our case) we calculate a separate loss for each class label per observation and sum the result:

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- M - number of classes (background, blood cell, bacteria)
- log - the natural logarithm
- y - binary indicator (0 or 1) if class label c is the correct classification for observation o
- p - predicted probability observation o is of class c

Pixel wise loss

As suggested by the name, the pixel wise loss[8] measures the pixel-to-pixel loss of the target image and the prediction. It is calculated by giving an "equal learning" to each pixel in the image. Since the pixel wise losses average the loss across all pixels, training can be dominated by the most prevalent class hence it suffers from the class imbalance problem (same as pixel wise accuracy) and thereby it is most suited for balanced datasets.

Jaccard loss

The Jaccard Loss uses the Jaccard Index (or MeanIoU) in order to compute the loss function. It is measured as:

$$L(A, B) = 1 - \frac{A \cap B}{A \cup B}$$

where A is the ground truth and B is the prediction

4.7 Learning rate schedulers

The learning rate controls how quickly the model is adapted to the problem. High learning rate means fast convergence to a solution that could not be optimal, while a value that is too low can either cause the training to get stuck or extremely long and useless training times. We have tried 3 different ways to manage the learning rate during the training of the models (other than using a constant value for the entire training).

- Cyclic learning rate scheduler with decay

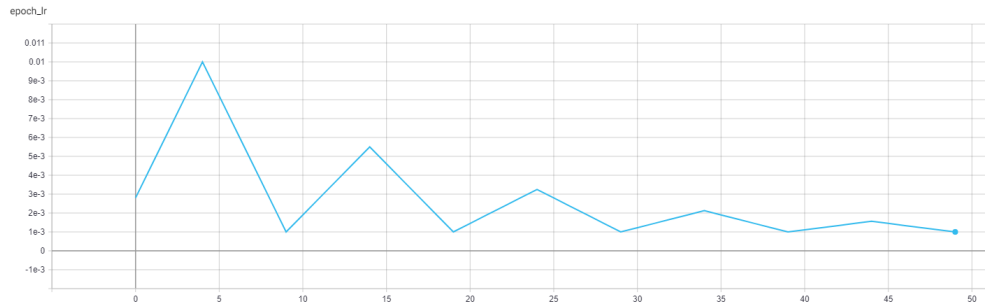


Figure 11: Cyclic with decay

- Warm up learning rate scheduler with cosine function

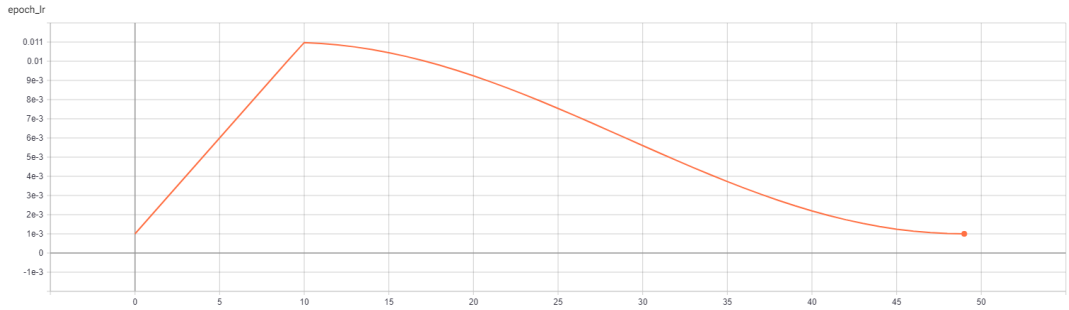


Figure 12: Warm up + cosine function

- Reduce on plateau learning rate scheduler

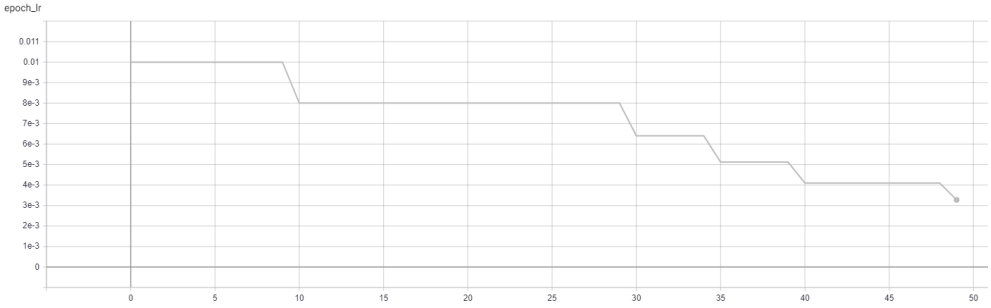


Figure 13: Reduce on plateau

4.8 Cross Validation

Since our dataset is small, cross validation comes in help in order to validate the performance of the model. Due to our limited computational power, we have decided to perform cross validation only on the best model (it is an expensive process in terms of time: it's like training n different models where n is the number of parts in which the dataset is split). For doing this, we have opted for the k-fold approach[2].

The process is composed as follows:

1. The dataset is shuffled

2. The dataset is split in n groups
3. For each group:
    - Take the group as a hold out (only once in the entire process)
    - Take the remaining groups as a training data set
    - Fit a model on the training set and evaluate it on the hold out
    - Retain the evaluation score and discard the model
4. Summarize the evaluation scores obtained

In this way, each sample is given the opportunity to be used in the hold out set 1 time and used to train the model $n-1$ times. The idea behind this approach is that the mean estimate of any metric is less biased than a one-shot estimate since every sample will be used to perform tests and training at different times. Since the performance of the model is estimated with a mean it is a good idea to report its variance too.
The test on the cross validation were all carried out with k=3 since in the original tests 70% of the dataset was used for training and 30% was used for validation and we wanted to maintain a similar proportion between the two sets.

```
Model: LinkNet with 288x288 dataset, RMSprop optimizer,
       cyclic learning (0.01 to 0.001), Jaccard Loss
Mean of final MeanIoUs recorded: 0.6763
Variance: 0,00011183
```

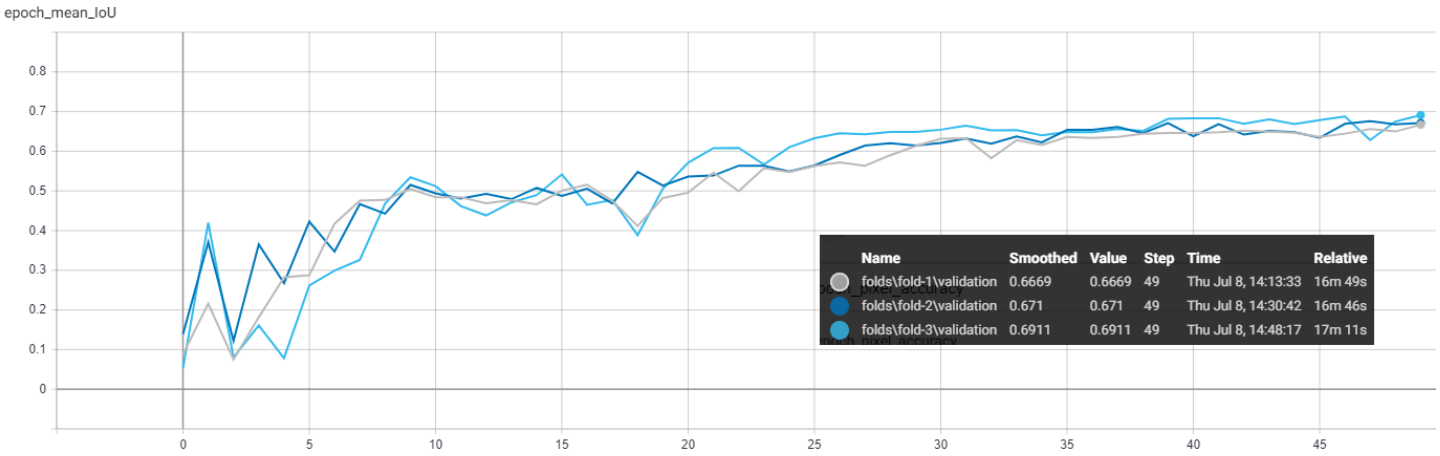


Figure 14: Reduce on plateau

5 EXPERIMENTS & RESULTS

These are the results obtained with the models and the validation set.
Note: On each plot's legends we have used a special notation in order to give more information: {*model*}_{*img_size*}_{*learning_rate_scheduler*}_{*optimizer*}_{*loss_function*} where CL stands for cyclic learning, WU for warm up and finally Rop for reduce on plateau.

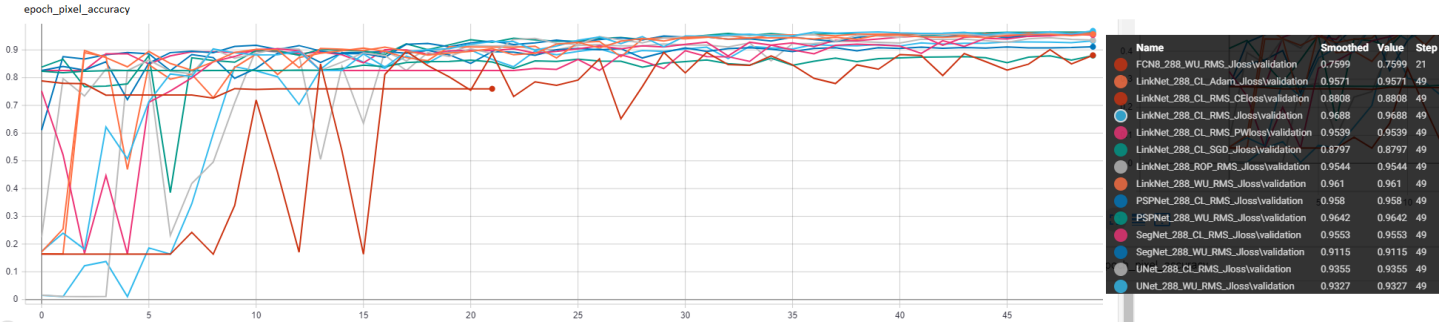


Figure 15: Pixel accuracy (validation set) for all the models tested

As you can see, some models can reach a really high accuracy (LinkNet ~ 0.95). In a more general case, the models reach a plateau at around 0.9. The pixel accuracy, in this case, isn't a good metric to decide which one of the models are performing better due to the class imbalance problem. In fact, the images don't contain an equal amount of pixels that belong to the same class; we have a lot of background and a relatively small amount of bacteria and blood cells. With these considerations, a model can achieve a relatively good result predicting a large portion as background.
The objective of the segmentation though is to identify bacteria and blood cells in the images: these two classes should have a bigger impact on the performance of a model when "miss-classified". In order to do that we had to use the mean-IoU metric.

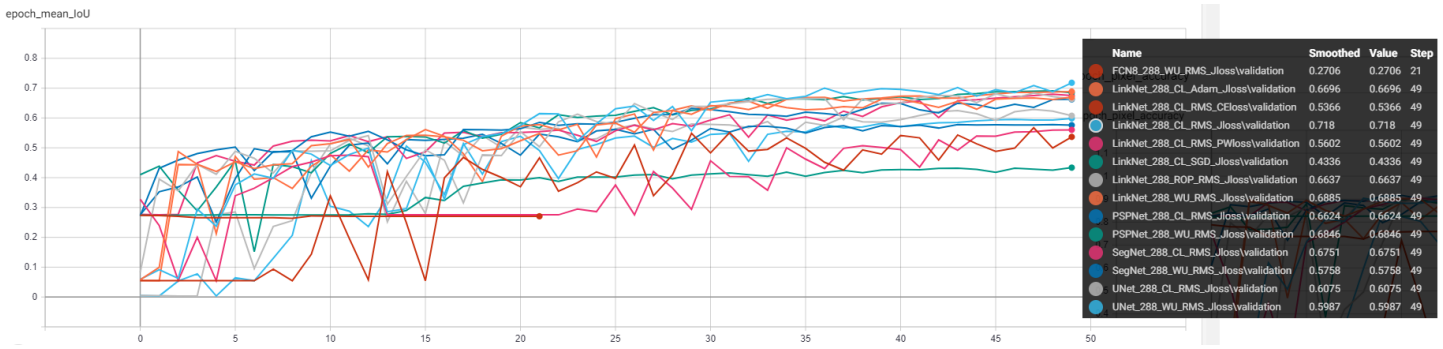


Figure 16: Mean IoU (validation) for all models

In figure 16 we have tested each model with different learning rate schedulers, loss functions and optimizers (which can be distinguished in the name of the model itself). We have also tested the FCN8 architecture but the results were extremely disappointing (the model predicted all pixels as background).

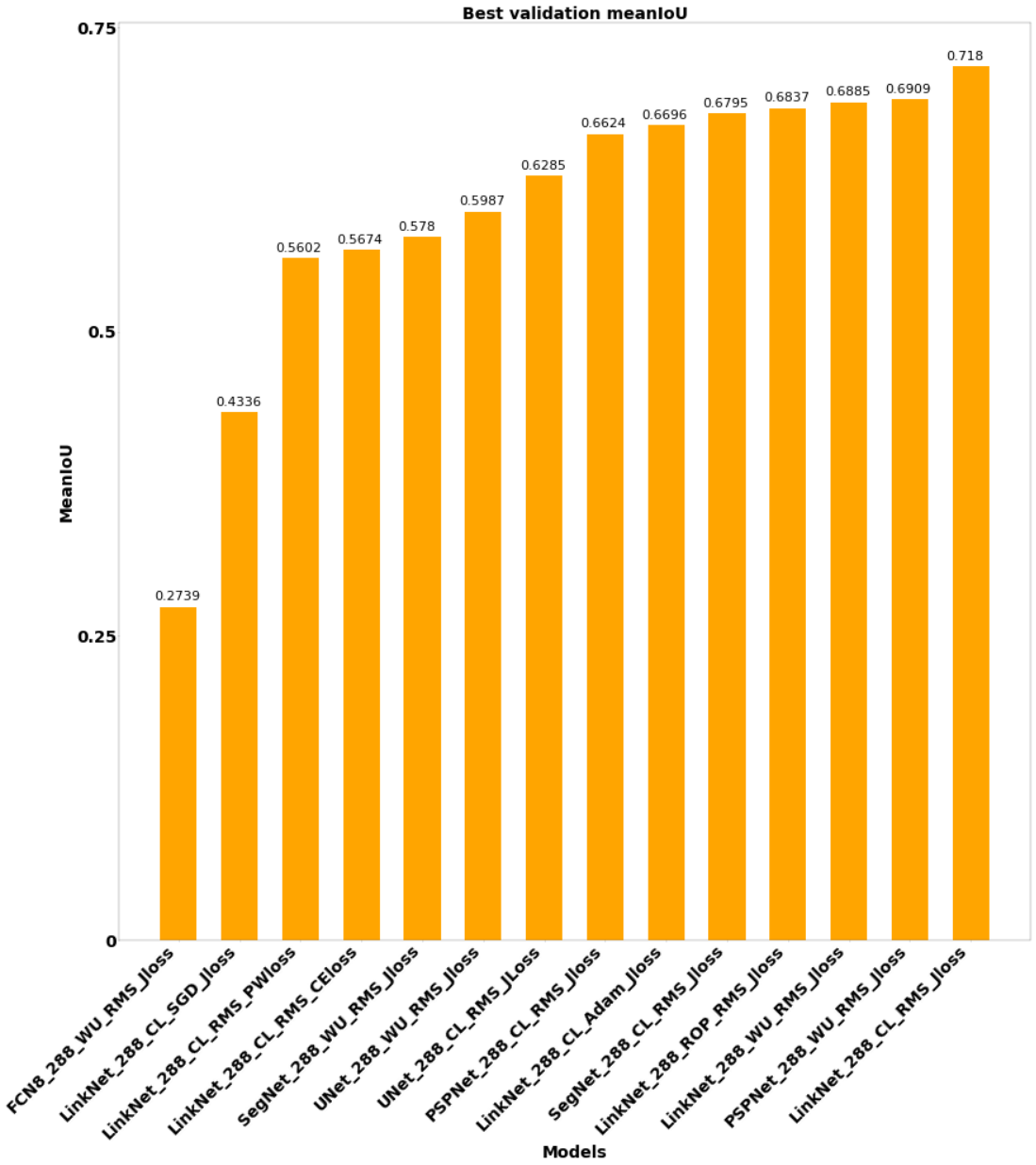


Figure 17: Best mean IoU (validation) recorded for each model tested

In figure 17 we have reported the bar plot of all the best meanIoU scores obtained with each model tested. All the models were tested with a maximum of 50 epochs on the datasets of 288x288 images. As we can see the architecture which gives the best results is **LinkNet**. For this reason, we have decided to focus on this architecture trying to squeeze the best result we could obtain.

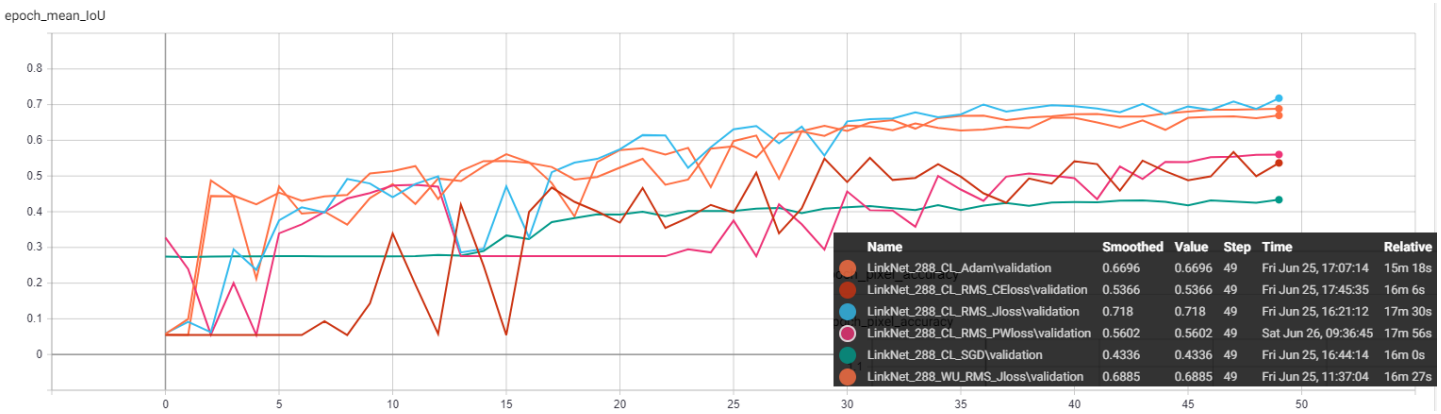


Figure 18: Mean IoU (validation) for all LinkNet models (with different hyperparameters)

In figure 18 we reported the results obtained with various tests on the LinkNet architecture. We can reach a value of 0.71 considering meanIou as a metric after 50 epochs using the cyclic learning rate scheduler, RMSprop as optimizer and the Jaccard loss as loss function.

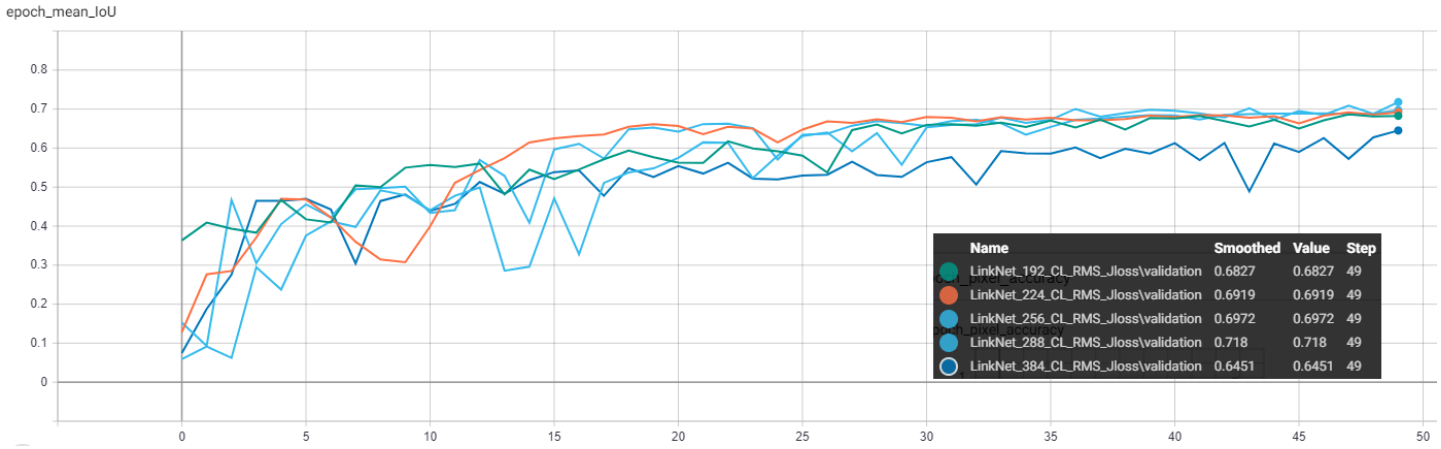


Figure 19: MeanIoU of LinkNet models with different image sizes

Moreover, in figure 19 we have also tried to resize the images we are using to train the network in order to see how the change would affect the performance of the models. As you can see from the graph it is indeed causing some difference between the performance of the models: with a smaller image (192x192) we get almost the same performance, with a bigger one (384x384) the performance takes a considerable hit (we should also consider that all the experiments were done with a limit of 50 epochs so increasing the number of epochs could increase the performance of the model further).

Finally, as the last test, we have trained the LinkNet model with no data augmentation. Here, we have noticed an unexpected result. The model reached 0.725 as the highest mean-IoU on the validation set as you can see in the figure 20.

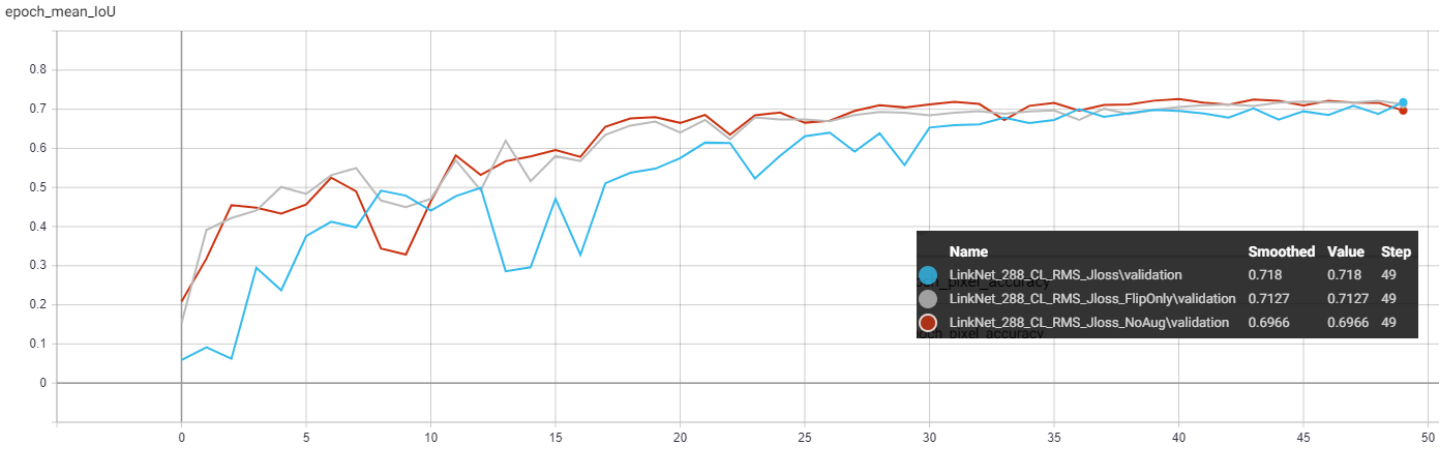
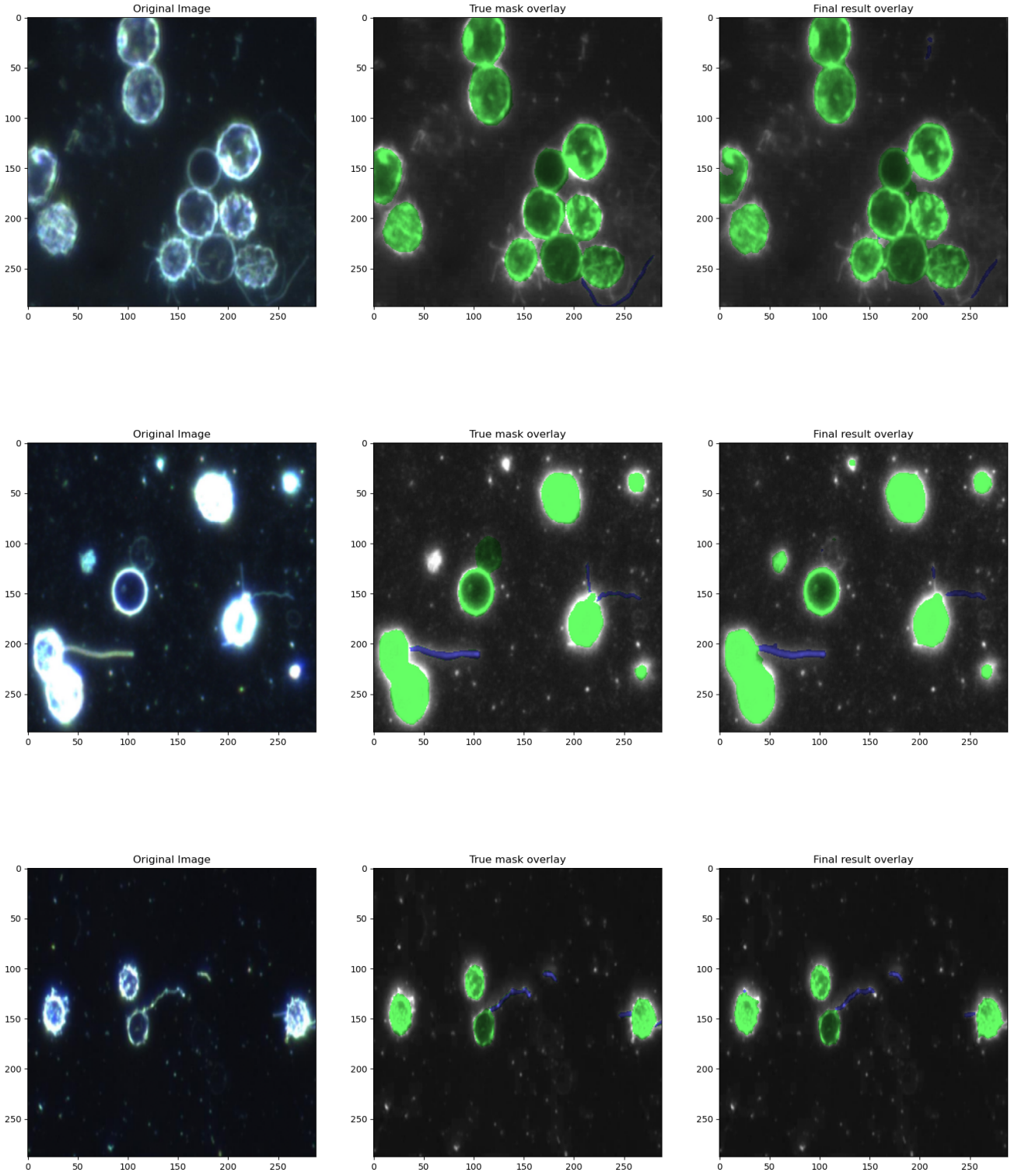


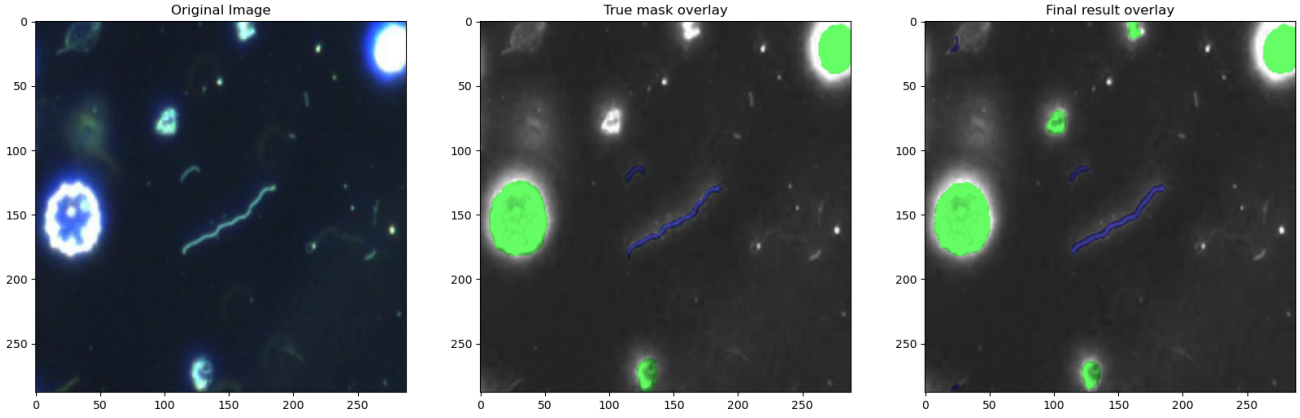
Figure 20: LinkNet with augmentation vs LinkNet with no augmentation

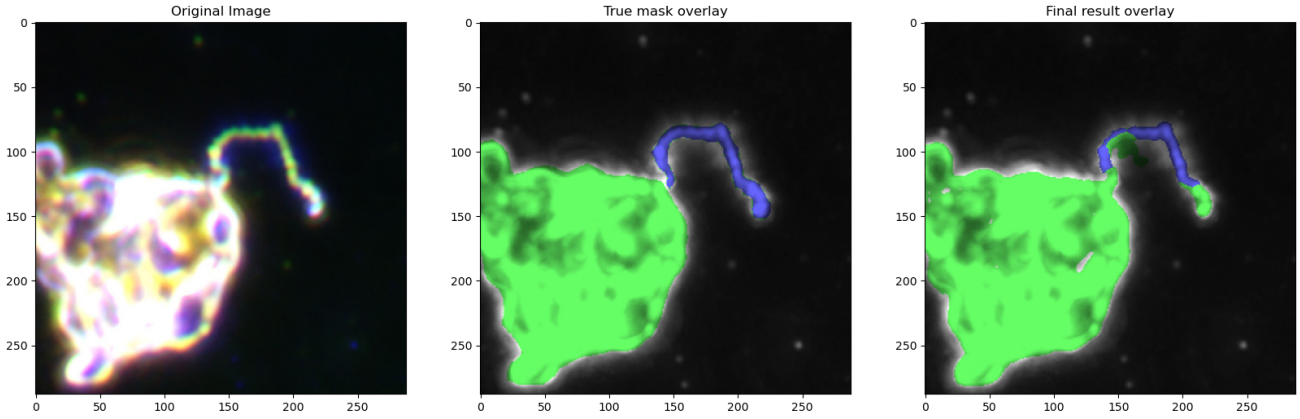
These results could be caused by the data-augmentation pipeline itself, which may be too harsh for this type of image. In fact, just by applying vertical and horizontal flips, we can obtain slightly better results compared to the "Not augmented" model (the difference would probably increase with more epochs).

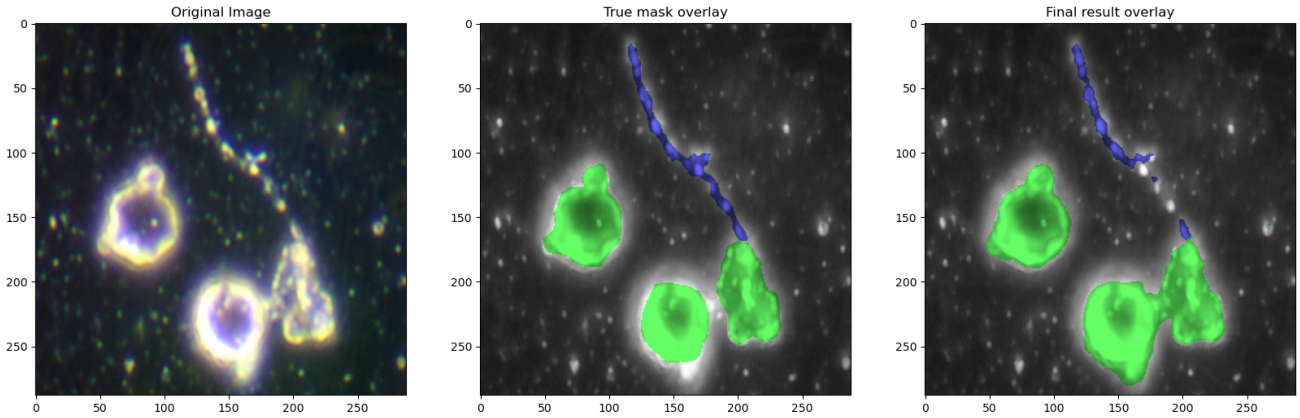
6 CONCLUSIONS

In conclusion, we have found that the best model is the LinkNet trained with 50 epochs. Moreover, we have found that other models were very similar in terms of results (mean-IoU score). Here you can find some result after the segmentation pipeline. The images belong to the validation set. The green parts refer to the blood cell, instead the blue ones refer to the bacteria.




Original Image
True mask overlay
Final result overlay


Original Image
True mask overlay
Final result overlay


Original Image
True mask overlay
Final result overlay

7 REFERENCES

[1] Eugenio Culurciello Abhishek Chaurasia. *LinkNet*.
URL: https://codeac29.github.io/projects/linknet/.

[2] Jason Brownlee. *A Gentle Introduction to k-fold Cross-Validation*.
URL: https://machinelearningmastery.com/k-fold-cross-validation/.

[3] Hengshuang Zhao Jianping Shi Xiaojuan Qi Xiaogang Wang Jiaya Jia.
Pyramid Scene Parsing Network. URL: https://paperswithcode.com/method/pspnet.

[4] *Loss Functions*.
URL: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html.

[5] Sebastian Ruder. *An overview of gradient descent optimization algorithms*.
URL: https://ruder.io/optimizing-gradient-descent/index.html#stochasticgradientdescent.

[6] Ekin Tiu. *Metrics to Evaluate your Semantic Segmentation Model*.
URL: https://towardsdatascience.com/metrics-to-evaluate-your-semantic-segmentation-model-6bcb99639aa2.

[7] Xiangjian He & Paul Kennedy Wenjing Jia. *Deep Learning Techniques for Medical Image Segmentation: Achievements and Challenges*.
URL: https://link.springer.com/article/10.1007/s10278-019-00227-x.

[8] Sowmya Yellapragada. *Understanding Loss Functions in Computer Vision!*
URL: https://medium.com/ml-cheat-sheet/winning-at-loss-functions-2-important-loss-functions-in-computer-vision-b2b9d293e15a.