Blood Cell Segmentation Using U-Net and Convolutional Neural Network

Othman Elkhazraje

Abstract - The aim of this paper is to segment blood Cell using U-Net architecture, from blood smear images captured through a compound microscope. This paper presents a method to digitally analyze the image of blood cells and find the cells from the blood smear microscopic images. Plane Extraction of the microscopic images is done by the convolutional neural network under U-Net architecture. The obtained results of the experiment are achieved 98% of accuracy.

1 - INTRODUCTION

Blood cells are the cells which are produced during hematopoiesis and found mainly in the blood. Blood is composed of the blood cells which accounts for 45% of the blood tissue by volume, with the remaining 55% of the volume composed of plasma, the liquid portion of the blood. Accurate segmentation and classification of RBCs and WBCs play a vital role in the identification of blood-related diseases like blood cancer, syndrome, leukemia, anemia, AIDS, and malaria.

In this paper, we are going to introduce our work on the segmentation of some different images from the dataset that we have of blood cell. The basic determination about segmentation of blood cells is the correct identification of blood constituents and extraction of useful information and it is capable of segmenting an unknown image into different portions or objects.

2 - The Convolutional Neural Network Architecture

The Convolutional Neural Network Convolutional Neural Networks (**ConvNets** or **CNNs**) are a category of Neural Networks that have proven very effective in areas such as image recognition and classification.

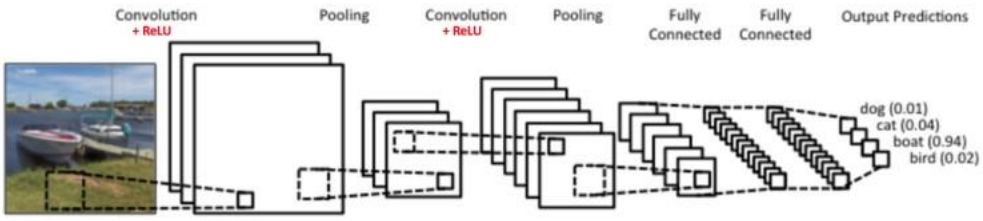


Figure 1: A simple ConvNet

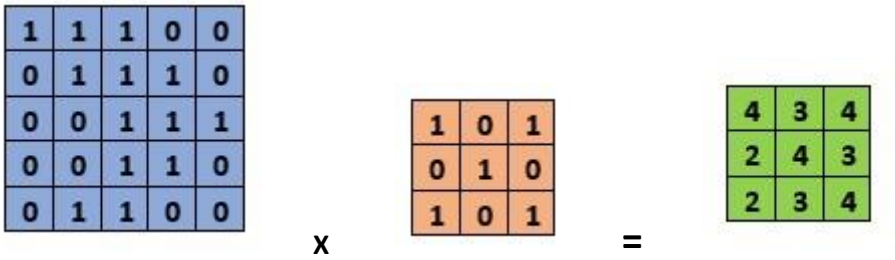
There are four main operations in the ConvNet shown in Figure above:

- Convolution
- Non Linearity (ReLU)
- Pooling or Sub Sampling
- Classification (Fully Connected Layer)

These operations are the basic building blocks of every Convolutional Neural Network so We will try to understand the intuition behind each of these operations.

The Convolution Step

ConvNets derive their name from the "convolution" operator. The primary purpose of Convolution in case of a ConvNet is to **extract features** from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data.



Every image can be considered as a matrix of pixel values. Consider a **5 x 5 image** and also, consider another **3 x 3 matrix**.
We slide the 3 x 3 matrix over our original image by 1 pixel (also called '**stride'**) and for every position, we **compute element wise multiplication** (between the two matrices) and add the multiplication outputs to get the final integer which forms a single element of the output matrix. Note that the 3×3 matrix "sees" only a part of the input image in each stride.

In CNN terminology, the 3×3 matrix is called a **'filter'** or **'kernel'** or **'feature detector'** and the matrix formed by sliding the filter over the image and computing the dot product is called **the 'Convolved Feature'** or **'Activation Map'** or the **'Feature Map'**. It is important to note that filters acts as feature detectors from the original input image.

The size of the Feature Map (Convolved Feature) is controlled by three parameters that we need to decide before the convolution step is performed:

Depth: Depth corresponds to the number of filters we use for the convolution operation. In the network shown in **Figure 2**, we are performing convolution of the original boat image using three distinct filters, thus producing three different feature maps as shown. You can think of these three feature maps as stacked 2d matrices, so, the 'depth' of the feature map would be three.

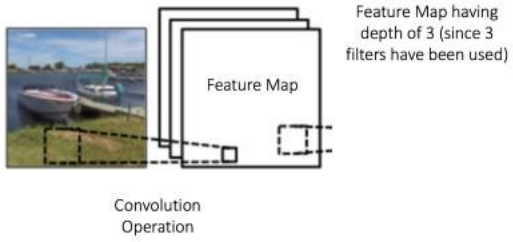


Figure 2

Stride: Stride is the number of pixels by which we slide our filter matrix over the input matrix. When the stride is 1 then we move the filters one pixel at a time. When the stride is 2, then the filters jump 2 pixels at a time as we slide them around. Having a larger stride will produce smaller feature maps.

Zero-padding: Sometimes, it is convenient to pad the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix. A nice feature of zero padding is that it allows us to control the size of the feature maps. Adding zero-padding is also called wide convolution, and not using zero-padding would be a narrow convolution.

Introducing Non Linearity (ReLU)

An additional operation called ReLU has been used after every Convolution operation in Figure 1 above. ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by:

$$\text{Output} = \text{Max}(\text{zero}, \text{Input})$$

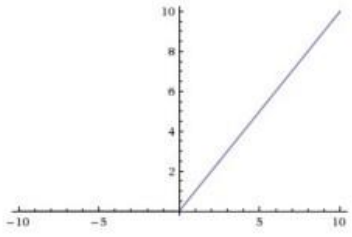


Figure 3: The ReLU operation

ReLU is **an element wise operation** (applied per pixel) and **replaces all negative** pixel values in the feature map **by zero**. The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear (Convolution is a linear

operation – element wise matrix multiplication and addition, so we account for non-linearity by introducing a non-linear function like ReLU).

Other non linear functions such as **tanh** or **sigmoid** can also be used instead of ReLU, **but** ReLU has been found to **perform better** in most situations.

The Pooling Step

Spatial Pooling (also called **subsampling** or **downsampling**) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.

In case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, **Max Pooling has been shown to work better**.

Figure 4 shows an example of Max Pooling operation on a Rectified Feature map (obtained after convolution + ReLU operation) by using a 2×2 window.

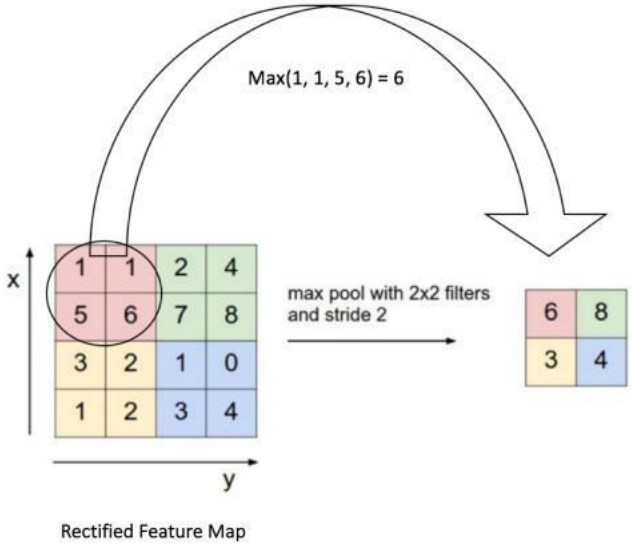


Figure 4: Max Pooling

We **slide** our 2 x 2 window by 2 cells (also called 'stride') and **take the maximum value** in each region.

Fully Connected Layer

The Fully Connected layer is a traditional **Multi Layer Perceptron** that uses a **softmax** activation function in the output layer. The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer.

The **output from the convolutional and pooling layers represent high-level features** of the input image. The purpose of the Fully Connected layer is to **use these features for classifying** the input image into various classes based on the training dataset. For example, the image classification task we set out to perform has four possible outputs as shown in **Figure 5 below** (note that Figure does not show connections between the nodes in the fully connected layer)

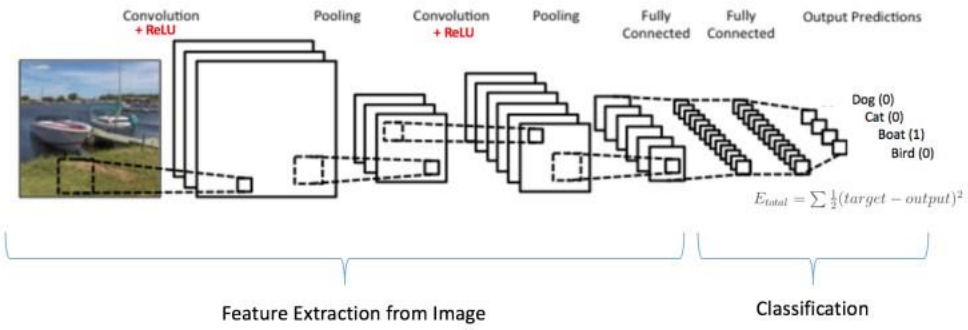


Figure 5

Apart from classification, adding a fully-connected layer is also a (usually) cheap way of learning nonlinear combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better.

The sum of output probabilities from the Fully Connected Layer is 1. This is ensured by using the Softmax as the activation function in the output layer of the Fully Connected Layer. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a **vector of values between zero and one that sum to one.**

3 -

U-Net Architecture

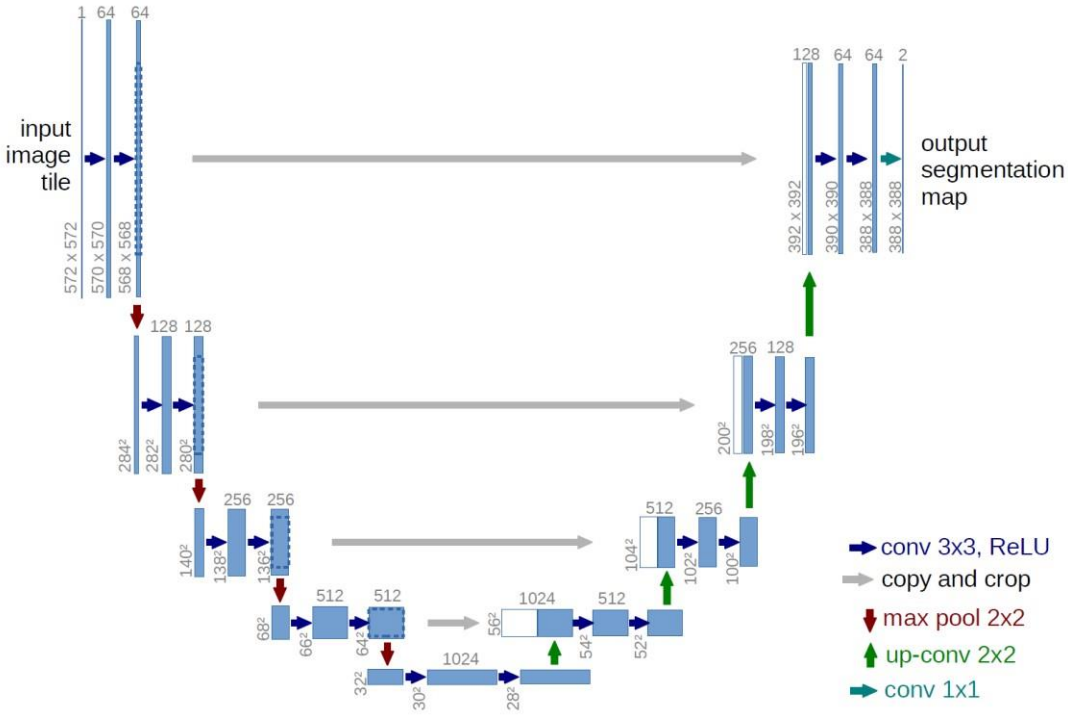


Figure6: U-net architecture

The architecture looks like a **'U' which justifies its name**. This architecture consists of three sections: **The contraction**, **The bottleneck**, and **the expansion** section.

The contraction section is made of many contraction blocks. Each block takes an input **applies two 3X3 convolution** l**ayers followed by a 2X2 max pooling**. The number of kernels or feature maps after each **block doubles** so that architecture **can learn the complex structures effectively.** The bottommost layer mediates between the contraction layer and the expansion layer. It uses **two 3X3 CNN** layers followed by **2X2 up convolution** layer.

But the heart of this architecture lies in the **expansion** section. Similar to contraction layer, it also consists of several expansion blocks. Each block passes the input to **two 3X3 CNN** layers followed by a **2X2 upsampling** layer. Also after each block number of feature maps used by convolutional layer **get half to maintain symmetry**. However, every time the input is also get appended by feature maps of the corresponding contraction layer. This action would ensure that the features that are learned while contracting the image will be used to reconstruct it. The number of expansion blocks is as same as the number of contraction block. After that, the resultant mapping passes through **another 3X3 CNN layer** with the number of feature maps equal to the number of segments desired

4 - Architecture

Our approach is based on Convolutional Neural Network under U-Net Architecture in order to segment the blood cells images with **Adam method for the optimization**, and **in activation method we use sigmoid function**.

The overall approach summarized as below:

Step1: transform all image (original and masks) in a gray scale.

Step2: Division lists to list of training and testing

Step3: checking the lists if there is a mismatch between the original images and the masked one.

Step4: feeding the model with initial parameters.

Step5: start the training

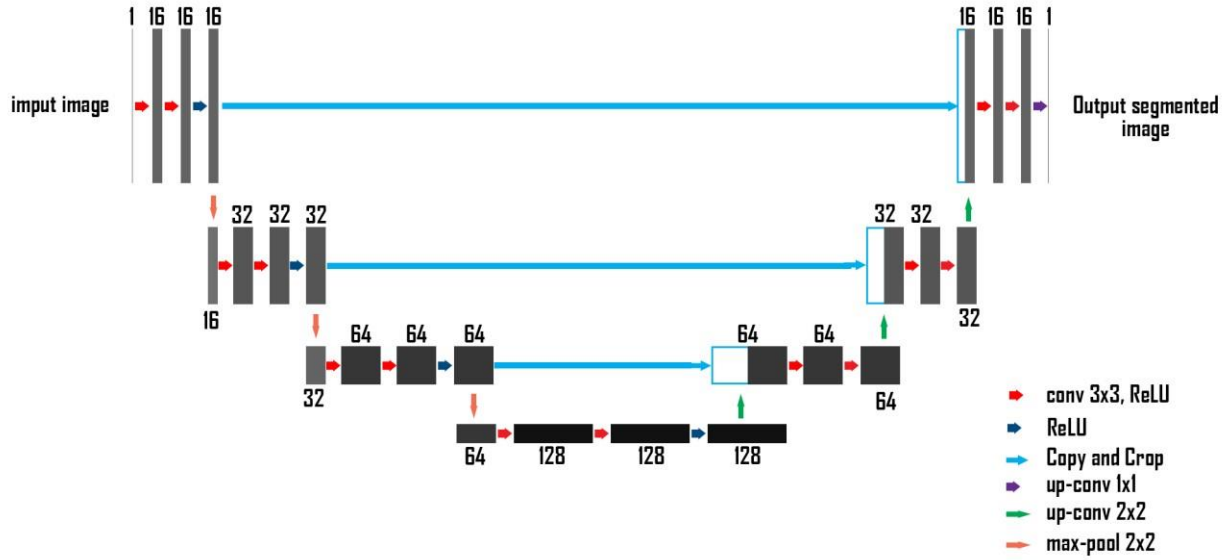


Figure 7: Model architecture

This is our model, is based on U-net but with We add convolution matrix before the max pooling.

5 -

Training

we use 90% of images in training part and we left 20% for testing.

Before training process, we resize our images because of their sizes and limited GPU memory. And we creation we creation another mask Binarize additional to the grayscale mask.

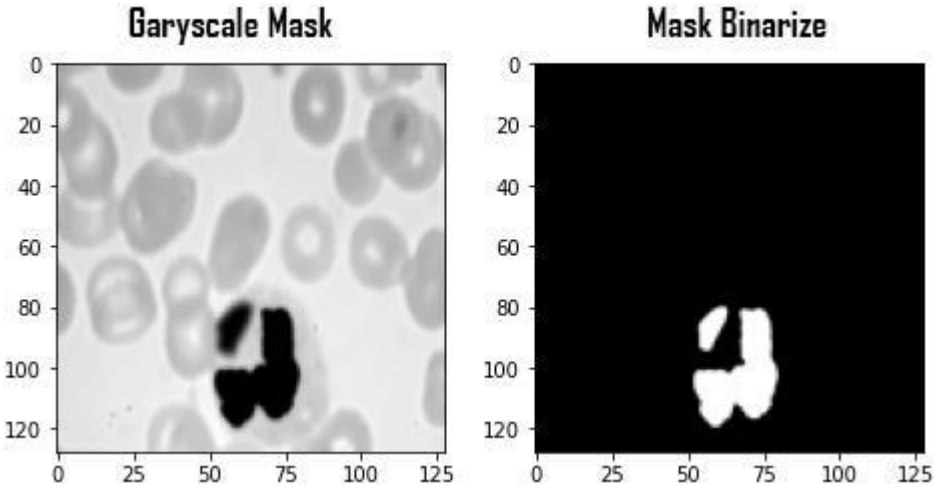


Figure 8

After training we get two models with the following results:

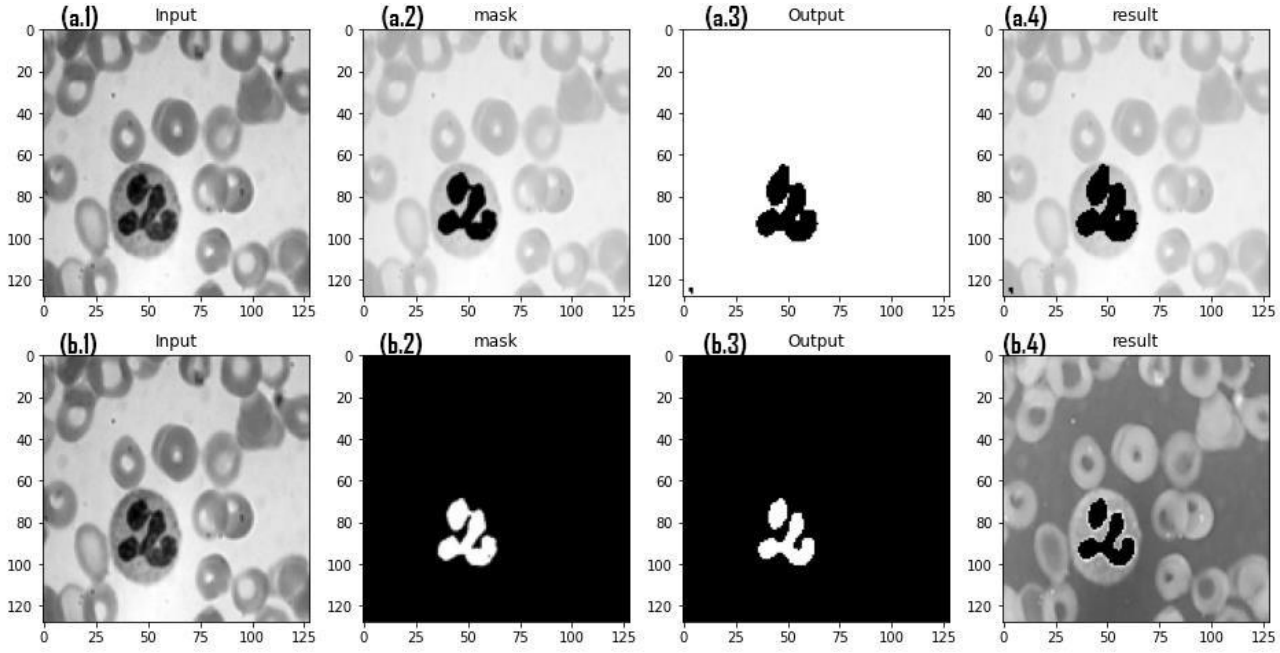


Figure 9: Blood Cells images and models predictions

6 -

Model 1 (M1): model training based on grayscale mask image

(a.1): The input image given to the model

(a.2): The mask given to the model in the training

(a.3): Model 1 prediction for input (a.1)

(a.4): element wise multiplication between (a.1) and (a.3)

Model 2 (M2): model training based on mask image Binarize

(b.1): The input image given to the model

(b.2): The mask given to the model in the training

(b.3): Model 2 prediction for input (b.1)

(b.4): element wise multiplication between (b.1) and (b.3)

Note: (a.1) and **(b.1)** are the same image

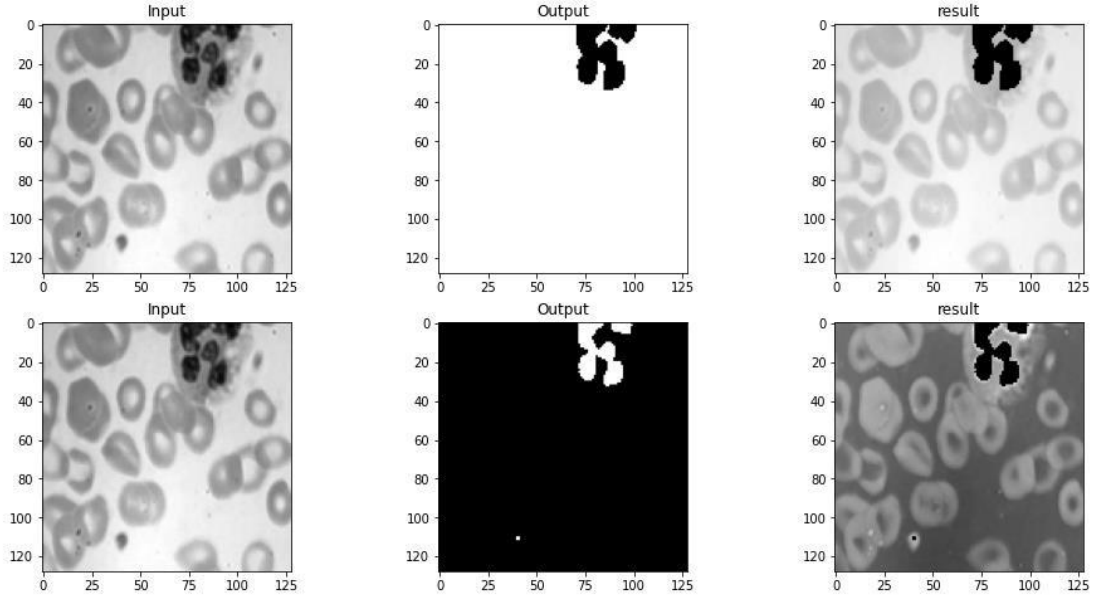
6 - result explanation

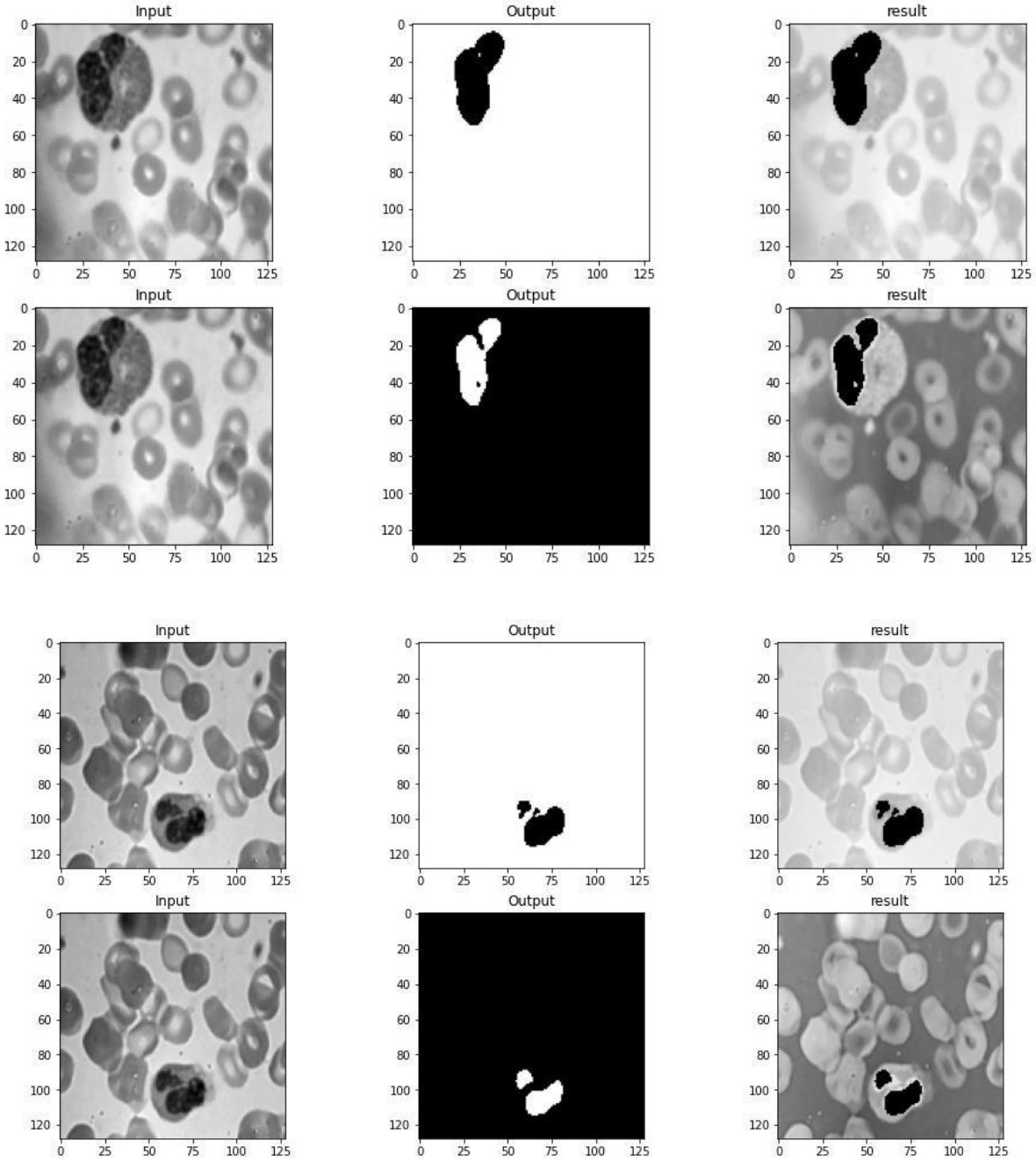
after finishing the training, we get the value of **loss** and **accuracy** for both Models M1 and M2:

the values of M1 are **[loss: 0.5721 - accuracy: 0.0190]** and that mean the output prediction **(a.3)** is so far from the Ideal output **(a.2).** even if the segmentation is good. This is due to the ReLU the we add at and of every block in **The contraction** section.

in another hand the values of M2 are **[loss: 0.0210 - accuracy: 0.9843]**. and those are a good result. Because the predicted image **(b.3) is** similar to the **original mask.**

Another prediction result





7 - Conclusion

In this paper we try to show that u-net is really good architecture it only needs some changes and very few annotated images and has a very reasonable training time.it can achieves very good result on very different image segmentation applications.