**University of Campinas**

**Institute of Mathematics, Statistics and Scientific Computing**

**(IMECC - Unicamp)**

**Project Title :**

# Semantic segmentation using region fully connected networks: application to medical images

Advisor: Prof Dr João Batista Florindo
Student: Gabriel Borin Macedo

June 2019

## Abstract

The manual location of cellular nuclei in an image of human epithelial tissue is an exhausting and slow process, making this work highly susceptible to failures. This project investigates the use of a specific neural network entitled Region Mask Convolutional Neural Network (Mask-RCNN) to perform the localization, segmentation and classification of different types of cells in this type of microscopy images. The accuracy obtained by Mask-RCNN, using the Kaggle 2018 Data Science Bowl database, was approximately 62.00%, which is a satisfactory result. This model shows a high potential to automatically localize the human epithelial cells in an image, making the overall localization process faster and less susceptible to errors.

# 1 Introduction

The easier access to high performance hardware, especially graphics cards (GPUs), allowed the application of more convolutional neural networks to a variety of areas in science and industry, allowing, for example, significant advances in the task of locating and placing objects in images in the literature. These advances also made possible for important results to be achieved in applications such as autonomous cars [2], games like chess and go [9], medical diagnostics [5], and many others.

Particularly in the area of medical imaging, an example of application using this model is addressed in [5] to the automatic and accurate recognition of cervical nuclei using a neural network. More precisely, a neural network called "convolutional network per region with mask" (*Mask-RCNN*) is used in association with semantic segmentation techniques for recognition of objects of interest, which in this case are the nuclei of cells.

This dense neural network model extends the *Faster R-CNN* [7], which adds a branch to predict the segmentation masks on each *Region of Interest* (RoI), simultaneously with the existing branch for classification and bounding box regressor. This mask branch is an aplication of a *Fully Connected Network* (FCN) [8] on each RoI, with the aim of predicting a pixel-wise segmentation mask. In addition, a so-called *RoiAlign* quantization layer is added to fix the misalignment through preservation accurate precise spatial locations. It is worth pointing out that *Mask-RCNN*, unlike *Faster R-CNN*, was designed to make a pixel-based alignment between the network inputs and outputs.

Besides that, the architecture of *Mask RCNN* makes implementation and training more simple than in *Faster R-CNN*, which facilitates a wide range of applications of such neural network structure for object classification. Furthermore, the mask has a comparatively smaller computational overhead, which accelerates the computational time, both for training as well as for the validation of new data.

Based on these results previously reported in the literature, this project aims at describing the method *Mask-RCNN*, its performance and accuracy in the task of classifying cell nuclei in medical images. ( falta colocar depois : conclusao )

# 2 Theoretical background

This section presents some topics that will be of extreme importance to understand the operation of neural networks of the Mask-RCNN type.

## 2.1 Neural networks

Neural networks are mathematical models used to predict or classify a given data (input) and its operation is very simple. The neural network is composed by neurons that work by receiving and processing an input data $x_1, x_2, ..., x_n$. The processing basically consists in a linear combination of the inputs, followed

by the application of a non-linear activation function f(v). The image bellow represents this idea.

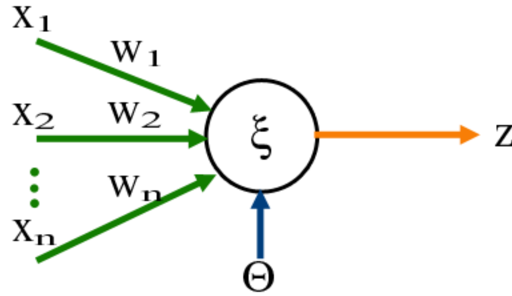



Figure 1: Generic representation of a perceptron

and

$$v = \sum_{i=1}^{n} x_i w_i + \theta \tag{1}$$

where $w_i$ is the weight associated with each neural connection and $\theta$ is a constant. This structure composed by one neuron (image x) is called perceptron in the literature. $\theta$ is a parameter called *bias*.

Bias is a measure of how easy it is for the perceptron to "activate". A perceptron with large bias has higher probability of producing an output 1.

The image bellow represents a step function :

$$f(x) = \begin{cases} 0 & \text{if Eq(1)} \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

### 2.1.1 Softmax Neurons

A network of perceptrons can be used to solve a problem by making them learning weights and biases so that the output from the network correctly classifies, for example, an object. A good principle for the learning process is that making a small change in some weight or bias should cause only a small corresponding change in the output of the network.

This principle is the inspiration to modify the weights and biases of neural networks. This strategy of changing the weights and biases repeatedly to produce better and better output is what is called learning.

However, networks that contain only perceptrons can have severe changes, for instance flips from 0 to 1. This flip may cause the behaviour of the rest

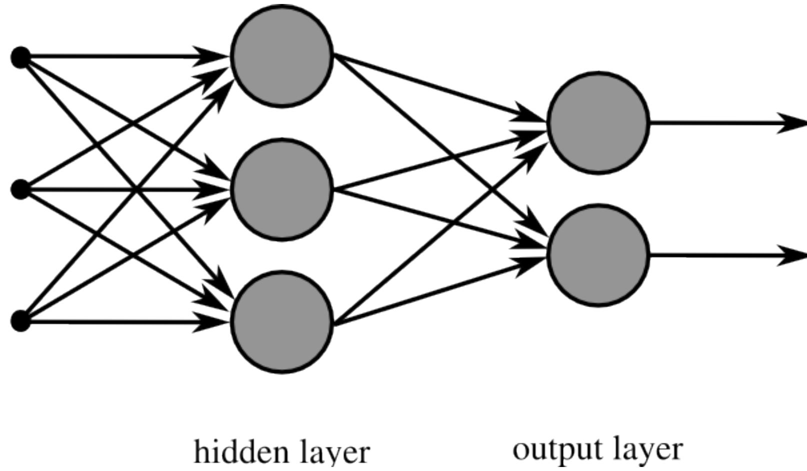



Figure 2: Generic representation of a Multi Layer neural network

of a network to change dramatically in a very complicated way, preventing the network from being able to perform correct classification. This process makes difficult to see how to gradually modify the weights and biases so that the network gets closer to the desired behaviour.

In this context, softmax neurons are used, since they are more consistent with changes of weights and bias, making a more concise and precise classification. Softmax neurons have a behavior similar to perceptrons, where such neurons recive as input $x_1, x_2, ..., x_n$. But instead of outputting only 0 or 1, these inputs can also output any real value between 0 and 1.

Softmax activation function that are also of great importance in these neural models, as their outputs behave like probabilities for the potential outcomes. The equation bellow represents this idea.

$$y = [y_1, y_2, ..., y_i, ..., y_n] = S(y_i) = \frac{\exp y_i}{\sum_{j=1}^{n} \exp y_j} = V_p = [p_1, p_2, ..., p_i, ..., p_n]$$

$$\sum_{k=1}^{n} p_k = 1$$

where $y_i$ refers to each element in the output vector y and $V_p$ is the vector of probabilities.

and the bias expression is :

$$\frac{1}{1 + \exp(-v)} \tag{2}$$

where v comes from from Eq. 1.

The smoothness $S(y_i)$ implies that small changes in the weights $\delta w_i$ and bias $\delta b$ produce small change $\delta_{out}$ in the neuron output:

$$\delta_{out} \approx \sum_i \frac{\partial out}{\partial w_i} \delta w_i + \frac{\partial out}{\partial b} \delta b \tag{3}$$

where the sum covers every weights $w_i$ and $\frac{\partial out}{\partial w_i}$ and $\frac{\partial out}{\partial b}$ are partial derivatives of the output with respect, respectively, to $\delta w_i$ and $\delta b$, as $\delta out$ is a linear function of the changes of weights ( $\delta w_i$ ) and bias ( $\delta b$ ).

These neurons will be important to understand the behavior of fully connected networks.

#### 2.1.2 Sigmoid neurons

This kind of neurons have the same principe of the softmax neurons described in the last section. However, the most important difference is its activation function, described as:

$$\sigma(v) = \frac{1}{1 + \exp^{-v}}$$

where v is the same as in Eq (1). The bias and $\delta out$ expression is, respectively, the same as ( 2 ) and ( 3 )

An intriguing aspect of such function is its perceptron like behavior. While in the perceptron, the ouput is a binary value (exactly 0 or 1), for a sigmoid, the value is between 0 and 1. Such ouput can be interpreted as the probability of the input to belong to a particular class. Besides, when $v- > \infty$, then $\exp -v- > 0$ and consequently $\sigma(v) \approx 1$. Moreover, in the case of $v- > -\infty$, then $\exp -v- > \infty$ and $\sigma(v) \approx 0$. The graphic bellow represents the behaviour of a sigmoid function

Another interpretation of the data provided by a sigmoid function is that output values represent the average intensity of the pixels in an image input to a neural network. However, there are many situations where this type of output requires more complex data analysis. An example is the output of a network that indicates wether “the input number is 3” or “the input image is not 3”. Obviously, it would be easier to do this if the output was 0 or 1, as in a perceptron. Because of that, there are particular conventions to deal with this, for example, by deciding to interpret any output of at least 0.5 as indicating a “3”, and any output less than 0.5 as indicating “not a 3”.

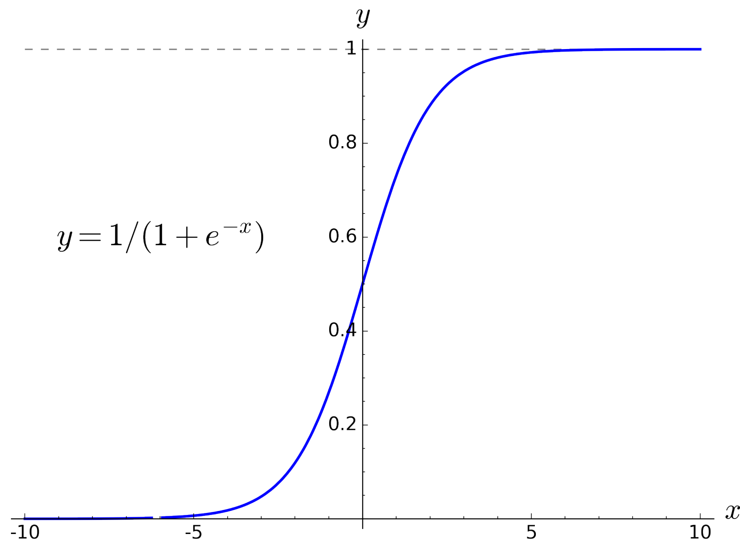



Figure 3: Sigmoid Function

#### 2.1.3 The architecture of a neural network

The neural network is separated into three parts. The first is called the input. The second part comprises the so-called hidden layers. The neurons in hidden layers are neither inputs nor outputs of the neural network. Finally, the last layer contains the output neurons.

The processing accomplished in all the inputs and ouputs of this network is straightforward. An example of this is a handwritten grey-scale image exhibiting the number 3 suposed to have dimension n x n. Therefore we will have $n^2 = nxn$ input neurons with the grey intensities appropriately scaled between 0 and 1. The output layer will contain only a single neuron, with output values of less than, for example, 0.5, suggesting that the image is not the number 3 or with values greater than or equal to 0.5, indicating that the image is a number 3.

The straightforwardness in input and output layers of a neural network is associated with a design heuristics for the hidden layers, which allows the user to obtain the intended behaviour of the network. An example of this is to determine how to trade off the number of hidden layers against the time required to train the network.

It is important to note that feedfoward neural networks are those that do not present looping alongside the network. Consequently, all the information is fed forward and never fed back.

# 3 Convolution Neural Networks

The great problem with using previous neural networks with fully connected layers is that they do not analyze the spatial structure of the images, such that those networks treat input pixels that are far apart and close together on exactly the same footing. Consequently, not taking into account this characteristic, may negatively affect the classification process of the object.

Convolutional Neural Networks (CNN) resolve this problem. They take into account the spatial structure of images by making convolutional operations to carry out the classification. CNNs can be separated into three stages: local receptive fields, shared weights, and pooling.

### 3.0.1 Local receptive fields

The fully connected layers can be represented by a vertical line of neurons. Besides that, the input will be n x n neurons that represent the pixel intensities.

Usually, the input pixels will be connected to a layer of hidden neurons and will make connections in small, localized regions of the input image, instead of connecting every input pixel to every hidden neuron. Therefore every neuron in the first hidden layer will be connected to a small region of the input neurons k x k times per region that correspond to n x n pixels.

This region in the input image is called local receptive field for the hidden neuron and has size k x k. In the literature these small windows are usually called kernels. All the connections learn weights, and the hidden neuron learns an overall bias.

Each channel of a CNN can detect only a single kind of localized feature and for image recognition, more than one feature map is necessary. A robust convolutional neural network consists of several different feature maps

Furthermore, an extra operation involving the kernel is defined by sliding a local receptive field across the entire input image.

Then the local receptive field slides by one pixel to the right to connect to a second hidden neuron.

It is also possible to define the step of the convolution (striding) with values greater than 1 to obtain better results.

### 3.0.2 Sharing the weights and biases

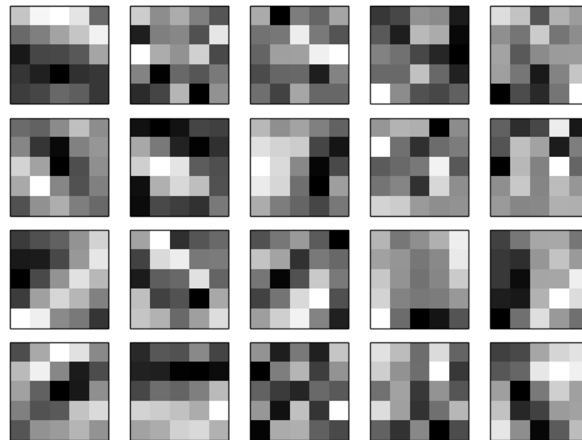
Hidden neurons have a bias and k x k weights connected to their local receptive field. It is important to emphasize the use of the same weights and bias for each of the n x n hidden neurons. That is, for the $j^{th}$ element and the $r^{th}$ hidden neuron, the output will be obtained from the following convolutional operation :

$$\sigma(b + \sum_{l=0}^{k} \sum_{m=0}^{k} w_{l,m} a_{j+l,r+m}), \tag{4}$$

where $\sigma$ is the neural activation function. For CNNs, it is common practice to use the RelU activation function. This is defined as $y(x) = max(0, x)$, where $x$ is the result of the following operation : $b+\sum_{l=0}^{k}\sum_{m=0}^{k} w_{l,m}a_{j+l,r+m}$. Besides that, b is the shared value for the bias, $w_{l,m}$ is a k x k array of shared weights and $a_{j+l,r+m}$ denotes the input activation at position ( x,y ).

Thus all the neurons in the first hidden layer detect exactly the same feature at different locations in the input image. In other words, the weights and bias are such that the hidden neuron can identify, for example, a vertical edge in a particular local receptive field. This methodology also can be applied to the same feature detector everywhere in the image. Therefore convolutional networks are well adapted to the translation invariance of images. Because of that, the maps from the input layer to the hidden layer are named feature maps. The weights and bias defining the feature map are the same shared weights and bias. These are often said to define a kernel or filter.

In the example below, there are 3 feature maps. Each one is defined by a set of 55 shared weights, and a single shared bias. The result is that the network can detect 3 different kinds of features, with each feature being detectable across the entire image. But in practice, CNN uses more feature maps. As an example, we have f features maps in the figure below:



The f images correspond to f different feature maps, filters or kernels. Every map is represented by k x k components. The f images correspond to f different feature maps, filters or kernels. Every map is represented as k x k block images, which correspond to the same k x k weights on the local receptive field. Moreover, the whiter blocks usually have the most negative weights and are associated with lower intensity response to the corresponding input pixels. In contrast, the black pixels correspond to larger weights and consequently the feature maps have higher responses to the corresponding input pixels.

The features maps composition of sub-regions with light and dark pixels make the network really learning things related to the spatial structure. How-

ever, it is difficult to visually see what these feature detectors are learning. Still, a great advantage of sharing weights and biases is the significant reduction in the number of parameters involved in a convolutional network. An example where this is visible is assuming that each feature maps has 25 = 5 x 5 shared weights, plus a single shared bias. So each feature map requires 26 parameters and having, for example, 20 feature maps, a total of 20  26 = 520 parameters defines the convolutional layer. On the other hand, a fully connected first layer, with 784 = 28  28 input neurons and a relatively small 30 hidden neurons, will have a total of 784  30 = 23,520 weights, plus an extra 30 biases, having 23,550 parameters in total.

Besides using fewer parameters, we also preserve translation invariance property on the convolutional layers. This reduces the number of parameters necessary to provide the same performance as the fully-connected model, yielding a faster training for the convolutional model.

One interesting point is that Equation (4) is the convolution and may be written in the following equivalent form:

$$a^1 = \sigma(b + w * a^0),$$

where $a^1$ denotes the set of output activations from one feature map, $a^0$ is the set of input activations, and $iscalledaconvolutionoperation$.

#### 3.0.3 The Pooling layers

The convolutional layers contain pooling layers. These pooling layers are used after convolutional layers, in order to facilitate the flow of information in the output from the convolutional layer.

In other words, the pooling layer takes every feature map output from the convolutional layer and prepares a condensed feature map. Usually, a frequent procedure for pooling is known as max-pooling. In this approach, a unit simply outputs the maximum activation in a $k^{'}xk^{'}$ input region.

Notice that having 24 x 24 neurons output from the convolutional layer implies that after pooling we will have 12 x 12 neurons.

The convolutional layer usually involves more than a single feature map and max-pooling is applied to each feature map separately.

The great advantage of using max-pooling is the reduction of the parameters needed in later layers. This operation is a way for the CNN to find any given feature in a region of the image and then throw away the exact positional information. Therefore, when a particular feature is detected, its location is not important as this rough location relative to other features.

#### 3.0.4 The general working of a CNN

CNNs have an architecture similar to the neural network previously discussed. But they have the addition of a layer having the same number of neurons as the number of h classes.

Convolutional neural networks behave similarly to the networks addressed in the previous topic. The objective remains the same, that is, to use training data to train the network weights and biases so that the network does a better classification of the input data.

The training of the network is accomplished by using the stochastic gradient descent and backpropagation. The backpropagation algorithm will be studied here in other neural networks.

## 3.1 Stochastic gradient descent

The stochastic gradient descent is a way to minimize an objective function $\psi(\theta)$ parameterized with a model's parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of theobjective function $\nabla_\theta \psi(\theta)$. Besides that, is defined a variable $\nu$, representing the learning rate, whitch determines the size of the necessary steps to reach the minimum local. that is, this method creates a function that downhill until reach a valley.

The Stochastic gradient descent (SGD) performs a parameter update every training example $x^i$ and $y^i$ :

$$\theta = \theta - \nu.\nabla\psi(\theta; x^i; y^i) \tag{5}$$

The batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. With this, SGD redundancy the error of the neural network by performing one update at a time, making the training more faster. Besides that, the SGD performs frequent updates with a high variance that cause the objective function tofluctuate heavily.

While batch gradient descent converges to the minimum of the basin the parameters are placed in. On the one hand, the SGD fluctuation allows to jump to new and potentially better local minima. However, this methodoly complicates convergence to the exact minimum, because the SGD will keep searching the minimal local. A way to prevent this from happening is slowly decrease the learning rate, making the SGD converge to to a local or the global minimum for non-convex and convex optimization respectively.

### 3.1.1 Momentum in SGDs

In surface curves, is commun to have steeply areas in one dimension than in another and this areas is common to stay around a local optima. In this situation, the SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottomtowards the local optimum. To resolve this kind of situacion, has created the momentum method that helps accelerate SGD in the relevant direction and dampens oscillations. This procedure uses a $\gamma$ variable , where $\gamma$ is a fraction of the update vector of thepast time step to the current update vector. The equation bellow represent this ideia :

$$v_t = \gamma v_{t-1} + \nu \nabla \psi(\theta)$$

$$\theta = \theta - v_t$$

And the default value of $\gamma$ is 0.9

Essentially, the behavior of a momentum is the same as that of pushing a ball down a hill. In this ball case, it's accumulates momentum as it rolls downhill, becoming faster and faster on the way, until it reaches its terminal velocity, if there a air resistance, in this case having $\gamma < 1$, this situation is similar on the SGDs updates parameters : The momentumterm increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. With that, the convergence is more faster and oscilation is reduced.

#### 3.1.2 Adam Optimizer

The Adaptive Moment Estimation (Adam) is a optimizer often used in Deep Learning. This optimizer that computes adaptive learning rates for each parameter and storing an exponentially decaying average of past squared gradients $v_t$. Besides that, Adam also keeps an exponentially decaying average of past gradients $m_t$ similar to momentum :

$$m_t = \beta_1 mt - 1 + (1 - \beta_1) g_t$$

$$v_t = \beta_2 vt - 1 + (1 - \beta_2) g_t^2$$

Where $m_t$ and $v_t$ are estimates of the first moment , that is the mean, is the second moment (the uncentered variance) of the gradients respectively, and $g_t$ is the current gradient. The variables $m_t$ and $v_t$ are initialized asvectors of zeros. This value was chosen because in practice, the bias has arround zero, especially during the initial time steps and especially when the decay rates are smal, like when $\beta_1$ and $\beta_2$ is close to value 1.

To revert this effect of bias, is compututed a bias-corrected first and second moment estimates in form :

$$m_t^{'} = \frac{m_t}{1 - \beta_1^t}$$

$$v_t^{'} = \frac{v_t}{1 - \beta_2^t}$$

And this parameters are used to o update the parameters, following the rule
:

$$\theta_{t+1} = \theta_t - \frac{\nu}{\sqrt{v_t^{'} + \epsilon}} m_t^{'}$$

Where $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

## 3.2 Residual Neural Network ( Resnet )

Deep Residual Learning or Residual neural network are neural networks that learn some residual intest of features at the end of its layers, like a convolutional neural network. Where this Residual can be simply understood as subtraction of feature learned from input of that layer. One of its advantages is to avoid the vanishing gradients that the most's neural networks are susceptible.

### 3.2.1 Residual learning

Be the H(x) an underlying mapping to befit by a few stacked layers, where x denotates the inputs to the first of these layers. Hypothesis, suppose that this network have multiples nonlinear layers with asymptotically and complicated functions. Then, is valid to supose that this functions can asymptotically approximate the residual functions in form H(x) - x, assuming that the input and ouput have the same dimension. With that, is viable to approximate a residual function in form F( x ) = H( x ) - x => F( x ) + x = H( x ). This 2 forms is also able to asymptotically approximate the desired functions, hypothesing this situacion ,the ease of learning might be different.

This reformulation is motivated by the counter intuitive phenomena about the degradation problem, as previously stated. Is possible to add new layers make to cosntructed a identity mapping with a deeper model to training error to stay no greater than its shallower counter part. The degradation problem suggests that the solvers can have difficulties to approximates a identity map by the multiple nonlinear layers. Using the residual learning re-formulation, the solvers can be simply drive the weights in multiples nonlinear layers toward zero to approach identity mappings.

In Pratical, it is unlikely that identity mappings are optimal. This reformulation help to precondition this problem. When the optimal is closer to an identity mapping than to a zero mapping, is might easy to the solvers find the perturbations with reference to an identity mapping, than to learn the function as a new one. Experimentally, this learned residual functions in general have small responses and intuitively indicates that identity map-pings provide reasonable preconditioning.

### 3.3 Identificate Mapping using shortcuts

The residual learning is adoted and used on every stacked layers. Besides that, the building block is defined as :

$$y = F(x, W_i) + x \tag{6}$$

Where, x and y are respectively the input and output vectors of the layers considered, $F(x, W_i)$ represents the residual mapping to be learned. For this work, is used a residual network with two layers. Therefore, F is defined as : $F = W_2\sigma(W_1x)$, where $\sigma$ is the RelU function ( the same activation function used on the convolution layers in CNN ) and the biases are omitted for simplifying no- tations. Futher more, the $F + x$ operation is computed using shortcuts connection and element-wise addition. Finally, is adopt the second non-linearity after the addition.

This shortcut connections in Eq ( 6 ) does not need any extra parameters and and does not increases the computation complexity. In practice, residual networks with the same number of parameters, depth, width, and computational cost has a better performance compared to a conventional convolutional neural network.

To this method, is necessary that x and F dimensions must be the same as Eq ( 6 ). When this not happens, hanging the input or output channels as example, is possible to make a linear projection $W_s$ using the shortcut connections to equal the dimensions. In other worlds, is used the following expression to that :

$$y = F(x, W_i) + W_sx \tag{7}$$

Is possible to use a square matrix $W_s$ in Eq ( 6 ). However in practice, the identity mapping is sufficient for addressing the degradation problem and is more economical compared to using a square matrix $W_s$ in Eq ( 6 ). Plus to this, $W_s$ is only used to match the dimensions.

The structure of the residual function F is easily modified, making it a flexible function to represents multiple convolutional layers. It is import to note that the element-wise addition is performed on two feature maps, channel by channel.

#### 3.3.1 Residual network structure and implementation

A generic residual network uses a generic CNN with shortcut connections added to it, turning the network into his counterpart residual version. This shortcuts, besides being described as Eq ( 6 ), is possible to use when the input and output are of the same dimensions. In the case of increasing the dimension, is possible to make to approaches. The first one make the shortcut keep performing dentity mapping with a extra zero entries padded for increasing dimensions. The

advantage of using this method is that it does not use extra parameters. The second method uses the projection shortcut describes in Eq ( 7 ) to equalize the dimensions using a 1 x 1 convolutions. The both options, in the case of shortcuts go across feature maps of two sizes, are performed with a stride of 2.

As for the structe, is use a resized image with its shorter side randomly sampled in 256 x 480. The crop is randomly sampled as 224 x 224 for the image or its horizontal flip, with the per-pixel mean subtracted. The standard RGB and batch normalization right after each convolution and before activation is used. The weight used is the Imagenet and the network as trained with all from scratch. A SGD is used with a mini-batch size of 256. For the learning rate, the network starts with 0.1 and divide by 10 when the error plateaus. This mode is rained for up to 60 x $10^4$ iterations. Finally, is use a weight decay of 0.0001 and a momentum of 0.9.

## 3.4 Fully convolutional neural networks

Fully convolutional neural networks are convolutional neural networks with every layer of data in a convnet is a three dimensional narray of size h x w x d.

Where h and w are patial dimensions and d is the feature or channel dimension. The first layer recive a image of size h x w with d channels. The locations in higher layers correspond to the locations in the path of the connected image. This locations is callend on leature as receptive fields.

A property of convolutional neural network is the invariance translation. In another words, convolution, pooling, and activation functions in this neural networks operate on local input regions and only depends on relative spatial coordinates. Is possible to write a data vector $x_{ij}$ at location ( i,j ) in a particular layer and a $y_{ij}$ for the following layer. The value of $y_{ij}$ is computed by the expression :

$$y_{ij} = f_{ks}(X_{si+\delta i, sj+\delta j})$$

that respects the relation :

$$0 \leq \delta i, \delta j \leq k.$$

Where k is the kernel size, s is the stride or subsampling factor and $f_{ks}$ determines the layer type. These layers can be a matrix of multiplication for convolution or average pooling, a spatial max for max pooling or an elementwise non-linearity for an activation function and so on for other types of layers.

Besides that, This functional form is maintained under composition with a kernel size and stride obeying the following transformation rule :

$$f_{ks} \circ g_{k's'} = (f \circ g)_{k'+(k-1)_{s'},ss'}$$

Most of general neural networks calculates general nonlinear function, this Fully convolutional neural network ( FCN ) uses one single layers to computes a nonlinear filter. This FCN are also referred to as deep filter. The FCNs operates on an input of any size and produces an output of corresponding, usually resampling the image, spatial dimensions.

A real valued loss function composed with an FCN defines a task. In the case of the loss function is a sum over the spatial dimensions of the final layer, the loss functios will be describe as :

$$l(x;\theta) = \sum_{ij} l^{'}(x_{ij};\theta)$$

The gradient parameter will be a sum over the parameter gradients of every spatial components in the same. Therefore, the stochastic gradient descent on l computed on whole images will be the same as stochastic gradient descent computed on $l^{'}$, taking every final layer receptive fields as a minibatch.

When these receptive fields overlap significantly, both feedforward computation and backpropagation are very more efficient in the case when computed layer-by-layer in a whole image instead of independently calculated on patch-by-patch.

## 3.5 Classifiers for Dense Prediction

fully connected layers of Fully convolutional neural networks have fixed dimensions and throw away the spatial coordinates. However, this fully connected layers can be also viewed as convolutions with kernels that cover their entire input regions by converting CNNs to fully convolutional neural networks to take input of any size and make spatial output maps.

Besides that, the resulting maps are equivalent to the CNNs evaluation in a particular input patches. The calculation used is highly amortized over overlapping regions in those patches. This spatial output maps of these convolutionalized models make them a natural choice for dense problems like semantic segmentation, that is to localizate which pixels of the object of interest, with a ground truth available at every output cell. The forward and backward passes are straightforward and both of them take advantage of the inherent computational efficiency and of convolution.

Moreover, neural convolutional neural networks uses fully convolutional yields output maps for inputs of any size. Thereby, output dimensions are typically reduced by subsampling. The CNNs classification method uses subsample to keep filters small and a reasonable computational requirements. This coarsens the output of a fully convolutional version of these nets, reducing it from the size of the input by a factor equal to the pixel stride of the receptive fields of the output units.

### 3.5.1 Shift-and-Stitch operations in Filter Dilation

Predictions in neural network can be obtained from coarse outputs by stitching together outputs from shifted versions of the input. In the case of the ouput downsampled by a f factor, the input x is shifted to the right and y pixels down. This operation is realized once for each ( x, y ) that respect the interval $0 \leq x, y < f$. Every $f^2$ input are processed and intertwined the output to make the predictions correspond the pixels at the centers of their receptive fields.

A convolution or pooling layer with a input stride s and a subsequent convolution layer with a $f_{ij}$ filter weights. The earlier layer input stride is setted earlier to produce one upsamples out put by s factor. But, intercalating the original filter with a upsampled output does not produce the same result as shift-and-stitch, due to the original filter that just looks a reduced portion of the input, which are upsampled at this point. Thus, to produce the same result, the filter is dilated using the expression :

$$f'_{ij} = \begin{cases} f_{i/s,j/s} & \text{if s divides i and j} \\ 0 & \text{otherwise} \end{cases}$$

Where i and j are zero-based. The equation bellow is use to representate a full neural network output of shift-and-stitch can be done by repeating this filter enlargement. This method uses a processed subsampled versions of the upsampled input.

Decreasing subsampling within a neural network filters see finer information, but in change have smaller receptive fields and take longer to compute. Besides that, the output is denser without decreasing the receptive field sizes of the filters, however all fillters can not acess information at a finer scale than their original design.

In pratice, the Shift-and-Stitch operations does not perform better than upsampling along with the skip layer fusion. Because of that, most of FCN use the combination of upsampling and skip layer. Those methods will be explained in the following sections.

### 3.5.2 Upsampling Operation

Upsampling is another method to connect coarse outputs to dense pixels is interpolation using a Binary Interpolation Function to computes a $y_{ij}$ output from the nearest four inputs by a linear map. Which depends only the relative positions of a input and output cells. Mathematically, this relation is described as follows

$$y_{ij} = \sum_{\alpha,\beta=0}^{1} |1 - \alpha - (i/f)||1 - \beta - (i/f)| x_{\lfloor i/f \rfloor + \alpha \lfloor j/f \rfloor + \beta}$$

Where f is the the upsampling factor and ( . ) is the fractional part. Is important to note that upsampling using a factor f convolution with the fractional input stride of 1/f. This only happens when f assumes a integer value. The upsampling is implement by reversing forward and backward passes of more typical input-strided convolution. Thus, upsampling is performed in all neural network for end-to-end learning using a backpropagation from the pixelwise loss.

The upsampling often used in deconvolution networks, which are CNNs with deconvolution. Where this deconvolution is the inverse operator of convolution. Besides that, the convolution filter in this kind of layer have no necessity to be fixed using a bilinear upsampling. However, this can be learned. Stacks of deconvolution layers and activation functions is also learned using a nonlinear upsampling.

#### 3.5.3 Loss Sampling and Patchwise

For the stochastic optimization, the gradient calculation is driven on the the training distribution. Besides that, patchwise training and FCN training can be used to make distribution of the all inputs. But the computational efficiency depends only the overlap and minibatch size. As for the training of FCN using whole images is practically the same to patchwise training, where every batch consists of all the receptive fields of the output units for an single or collection of images. This approach is more efficient compared to uniform sampling of patches, due to the fact that reduces the number of possible batches. However, the random sampling of patches method within an image, usually, is more easily recovered. This restrict the loss to a randomly sampled subset of its spatial terms that excludes patches from the gradient.

In the case of patches still having significant overlap, the fully convolutional computation will still speed up training. But, if gradients accumulated over multiple backward passes, the batches can include patches from several images. Another intresting case is when the inputs are shifted by values up to the output stride. Besides that, is possible to choose a random of all possible patches even even though the output units lie on a fixed strided grid.

The Sampling in patchwise training corrects the class imbalance problem. Class imbalance is a machine learning problem that occurs when the total number of a class of positive data is more less than the total number of the negative data. Also, sampling can make the spatial correlation of dense patches. During FCN training, the balance of class is achieved by weighting the loss and loss sampling is used to address spatial correlation.

#### 3.5.4 The Momentum and Batch Size

Using $g_t$ as the step taken by minibatch SGD with momentum at time t. Is possible to define and describe $g_t$ in the expression :

$$g_t = -\mu \sum_{i=0}^{k-1} \nabla l(x_{kt+i}; \theta_{t-1}) + p g_{t-1}$$

Where $l(x;\theta)$ is the loss for the parameters x and $\theta$, p ¡ 1 because of its definition, k is the batch size and $\mu$ is the learning rate. Making a expansion recurrence as an infinite sum with geometric coefficients, the expression above can be write as :

$$g_t = -\mu \sum_{s=0}^{\infty} \sum_{i=0}^{k-1} p^s \nabla_\theta l(x_{k(t-s)+i}; \theta_{t-s})$$

This examples includes in the sum the $p^{\lfloor j/k \rfloor}$ coefficient. Where the index j orders the examples from most to least recently considered. This Approximating for the expressions using the floor operation, makes the learning with momentum p and batch size of k have a similar behavior to learning with momentum $p^{'}$ and and batch size $k^{'}$ when equality $p^{(1/k)} \approx p^{(1/k^{'})}$ happens. Is important to note that smaller batch size in a frequent weight updates and have a chance to this neural network learning more for the same number of gradient computations.

The FCN used on this work, have momentum p = 0.9 and a batch size of k = 20 images. Besides that, the approximately equivalent training regime with momentum $0.9^{(1)/20} \approx 0.99$ and a batch size of one.

#### 3.5.5 The segmentation structure

Firstly, the segmentation in FCN is training using by fine-tuning. Afther that, is performed jumps between the layers to merge coarse, semantic and local, appearance information. Besides that, This skip architecture is learned end-to-end to refine the semantics and spatial precision of the output. Plus to this, FCN are trained using a per-pixel softmax loss and validate with the standard metric of mean pixel intersection over union. With the mean taken over all classes, including the background. A particularity of this network is that the same ignores pixels that are masked out, which are ambiguous or difficult, in the ground truth.

Before proceeding with the paper, it is important to emphasize some common operations in the area of semantic segmentation to quantize the pixel accuracy and region intersection over union ( UI ), such as:

pixel accuracy:

$$p_{acu} = \frac{\sum_i n_{ii}}{\sum_i t_i}$$

mean accuracy :

$$m_{acu} = \frac{1}{n_{cl}} \frac{\sum_i n_{ii}}{t_i}$$

mean UI :

$$m_{UI} = \frac{1}{n_{cl}} \frac{\sum_i n_{ii}}{t_i + \sum_j n_{ij} - n_{ii}}$$

frequency weighted IU :

$$freq_{WIU} = (\frac{1}{\sum_k t_k}) \frac{\sum_i t_i n_{ii}}{t_i + \sum_j n_{ij} - n_{ii}}$$

Where $n_{ij}$ is the number of pixels of the i class predicted that supposedly belongs to class j, $n_{cl}$ is the total of all the different classes and $t_i = \sum_j n_{ij}$ is the total number of pixels of class i.

#### 3.5.6 Behavior of the structures

In FCN, the loss function is not normalized. Consequently, all pixels will have the same weight regardless of the batch and image dimensions. With this, is possible a small learning rate since the loss is summed spatially over all pixels.

Besides that, is defined two regimes for batch size. In the first regime, the gradients are accumulated over 20 images. Thus, this acummulation will reduce the memory required and respect the different dimensions of every input by reshaping the network. The batch size will be chosen randomly

All the fully convolutional classifier are fine-tuned to semantic segmentation for recognition and location by adding skips to fuse layers fuse layer outputs and include shallower layers with finer strides in prediction. With that, the edges skip ahead from shallower to deeper layers, making more local predictions from shallower layers since their receptive fields are smaller and see fewer pixels.

After this skip implementacion is done, the network will make and fuse predictions from several streams learned jointly and end-to-end. The fine layers and coarse layers combined allows a flexible neural network with local predictions that respect global structure.

The Layer fusion is a key element in this operation. Still, the correspondence of elements across layers is complicated by resampling and padding. with that, the layers that will be fused must be aligned by scaling and cropping. To resolve this situation, is created two layers into scale agreement by upsampling the lower-resolution layer. This procediment is done by using the Upsampling Operation describe in section ( 3.5.2 ). The Cropping removes every portion of the upsampled layer that extends beyond the other layer due to padding. All

this layers results have equal dimensions in exact alignment. As the cropped region offset, this depends on the resampling and padding parameters of all intermediate layers. This determining the crop that results in exact correspondence can be intricate. However, crops follows automatically from the network definition.

Besides that, is defined fusion operations for treatment of the layers that are not patially aligned. This operations fuse features by concatenation and immediately follow with classification by a "score layer" consisting of a 1 x 1 convolution. Moreover, the concatenation and subsequent classification are communicated to each other, insted of storing concatenated features in memory. It is important to emphasize that both operation is linear.

In another words, the skips are implemented by first scoring each layer to be fused in a 1 x 1 convolution that carrying out any necessary interpolation and alignment and then summing the scores. The max fusion is also considered in this analyze. However, the network learning might to be difficult due to gradient switching.

All score layer parameters are initialized with zeros when a skip is added, in order to not interfere with the existing predictions of other streams. Finally, when every layers have been fused, the final prediction is upsampled back to image resolution.

## 3.6 Binary Interpolation Function

### 3.6.1 Localisation network

Firstly, a feature map $FM \in \mathbb{R}^{\mathrm{HxWxC}}$, where $W$ is the width, $H$ is the height and **C** is the number of channels, and outupts a $\alpha$ parameter associated with the transformation $T_\alpha$ to be aplie on the feature map $\alpha = f_{conv}(FM)$, where $f_{conv}()$ is a network function of locating the shape of a convolutional neural network with a final regression layer to produce the parameters $\alpha$ and this function is a 6-dimensional affine transformation. Besides that, $W$ = 512, $H$ = 512 and $C$ = 3.

### 3.6.2 Parameterizing the sampling grid

To make the a warping of the input feature map, is necessary to each output pixel is computed by a operation kernel centered in a particular location on this input feature map. Therefore, the output pixels are in a regular grid $G = G_i$ of pixels $G_i = (x_i^t, y_i^t)$ that form an output feature map of the form of $V \in \mathbb{R}^{H^{'}xW^{'}xC}$, where $H^{'}$ and $W^{'}$ are respectively height and width of the grid and $C$ is the number of channels colors. In this work, $W^{'} = 3$, $H^{'} = 3$ and $C = 3$. In other words, this transformation can be describe in a matrix function as follows :

$$\begin{bmatrix} x_i^s \\ \\ y_i^s \end{bmatrix} = \mathrm{T}_\alpha(G_i) = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \end{bmatrix} * \begin{bmatrix} x_i^t \\ \\ y_i^t \\ \\ 1 \end{bmatrix}$$

where $(x_i^t, y_i^t)$ are the target coordenates of the regular grid in the output feature feature map, $(x_i^s, y_i^s)$ are the sorce coordenates of the feature map with define the samples points and $A_\alpha$ is the affine transformation matrix of the form :

$$\mathrm{A}_\alpha = \begin{bmatrix} \alpha_{11} & \alpha_{12} & \alpha_{13} \\ \\ \alpha_{21} & \alpha_{22} & \alpha_{23} \end{bmatrix}$$

The width and height are normalized coordenades as follows : $-1 \leq (x_i^s, y_i^s) \leq 1$ when this coordenades are inside of the spatial bounds of the ouput and $-1 \leq (x_i^t, y_i^t) \leq 1$ when this coordenades are inside of the spatial bounds of the input.

To realize a spatial transformation on the input feature map, it is necessary to take a set of sampling points $T_\alpha(G)$ of a sample, alongside with the input feature maps FM and produce the output sampled feature map V. For each $(x_i^s, y_i^s)$ coordinate associade with $T_\alpha(G)$ defines one spatial location to get the ouput $V_i$, where $V_i$ is the input that a sampling kernel is apllied to obtain a value at a especific pixel. that is, $V_i$ can be describe as :

$$V_i^c = \sum_{n=0}^{H=511} \sum_{m=0}^{W=511} FM_{nm}^c k(x_i^s - m; \lambda_x) k(y_i^s - n; \lambda_y) \; \forall \; i \in [1 \ldots H^{'} W^{'}] \; \forall c \in [1, 3]$$

where $\lambda_x$ and $\lambda_y$ are parameters of a generic sampling kernel $k()$ whitch defines a bilinear image interpolation, $FM_{nm}^c$ is the value in the location $(n, m)$ on the respective channel color c of the input and $V_i^c$ is the ouput value of the pixel i at the coordenate $(x_i^t, y_i^t)$ at the respective channel of color c. For this work, as choosen to use the integer sample using a bilinear sample kernel expression. With that, the expression $V_i^c$ as lower in :

$V_i^c = \sum_{n=0}^{H=511} \sum_{m=0}^{W=511} FM_{nm}^c max(0, 1 - |x_i^t - m|) max(0, 1 - |y_i^t - n|)$(8)

From Eq3.6.2 is define the gradient referring to $FM_{nm}^c$, $x_i^s$ and $y_i^s$ for calculation of backpropagation and of the loss hrough the sampling mechanism. The parcial derivades of the bilinear sampling are defined as :

$$\frac{\partial V_i^c}{\partial FM_{nm}^c} = \sum_{n=0}^{H=511} \sum_{m=0}^{W=511} max(0, 1 - |x_i^t - m|) max(0, 1 - |y_i^t - n|) \quad (9)$$

$$\frac{\partial V_i^c}{\partial x_i^s} = \sum_{n=0}^{H=511} \sum_{m=0}^{W=511} FM_{nm}^c max(0, 1 - |y_i^t - n|) = \begin{cases} 1 \text{ if } |m - x_i^s| \geq 1 \\ 0 \text{ if m } \geq x_i^s \\ \text{-1 otherwise} \end{cases} \quad (10)$$

$$\frac{\partial V_i^c}{\partial y_i^s} = \sum_{n=0}^{H=511} \sum_{m=0}^{W=511} FM_{nm}^c max(0, 1 - |x_i^t - m|) = \begin{cases} 1 \text{ if } |m - y_i^s| \geq 1 \\ 0 \text{ if m } \geq y_i^s \\ \text{-1 otherwise} \end{cases} \quad (11)$$

For the continuous case in the sampling fuctions, , this sub-derivades allows to calculate the loss gradient and propagate from the end to begining not only on the feature map (9) but also in the sample grid coordenates (10) and (11)and consequently back for the transformation parameters $\alpha$ and localisation network since $\frac{\partial x_i^t}{\partial \alpha}$, $\frac{\partial x_i^s}{\partial \alpha}$, $\frac{\partial y_i^t}{\partial \alpha}$ and $\frac{\partial y_i^s}{\partial \alpha}$ can be trivially derived. The discontinuous case for sampling fuctions, the sub-gradient will be used on this case. This sampling mechanism is implemented efficiently on GPU, looking only to the kernel support region for each ouput pixel and ignore the sum over all of the input locations

### 3.6.3 Spatial Transformer Network

Combining the localisation network, grid generator and the sample form a spatial transformer. Besides that, this is a independent module which can drop in any point in the Mask-RCNN architecture, in this case is in RoIAlign layer, and in any amount. This module, in computational aspect is very fast and does not delay the speed of training consequently causes little time overhead even using in an exaggerated way.

The spatial transformers within a Mask-RCNN allows the Mask-RCNN learns how to transform the features maps actively to minimize the overall cost function of this network during training. Besides, the knowledge to make the transformation for each training is compressed and stored in cache memory in the weights of the localisation network during the Mask-RCNN training . this can also be done with the the weights of previus layers to a spatial transformer.

Is import to ressalt that can be possible to have multiple spatial transformers in a Mask-RCNN. The addition of several patial transformers along in the depths layers of the Mask-RCNN network enable transformations that increases the

generalization capacity and gives to the localisation networks more informative representations to serve as a basis to predict the parameters. With the use of multiple spatial transformers in parallel, in order to facilitate multiple objects, whether they are of the same class or not, or interested parts in a feature map should be focussed on individually.

## 3.7 Fast-RCNN Structure

Fast-RCNN are neural networks that takes a whole image and a set of object proposals as input. Firstly, this network realizes convolution operations and uses max pooling to produces a convolutional feature map. After this, for each object proposals, an Region of Interest ( RoI ) and pooling layers extracts a fixed-length feature vector from this feature map . Afther that, each feature vector is feeded in a sequence of fully connected layers which are combined with two sibling output layers. One of them estimates the probability of being a certain object class over K classes and a catch-all background class by using a softmax function. In this work, K = 1, because is only needed to classify if there a nucleus on a image.Already, the second layer outputs four real-valued numbers for each the K = 1 object classes and every set of 4 values encodes refined bounding-box positions for one of the K = 1 classes. In another words, for the nucleus class.

### 3.7.1 RoI pooling Layer

The Roi pooling is a layer that uses max pooling to convert features maps that are inside in any valid RoI into a small feature map with a fixed spatial extent of H = 512 x W = 512. Where H and W are layer hyper-parameters independent of any particular RoI. The RoI are defined as rectangular window into a convolutional feature map and uses a four-tuple ( r, c, h, w ), where ( r, c ) specifies its top-left corner and ( h, w ) the height and width, respectively.

Initially the RoI max pooling layer divide the h x w RoI window into a H = 512 x W = 512 of sub-windows of approximate size ( h/H ) x ( w/W ) and afther that uses max pooling on the values for each sub-window to the corresponding output grid cell. Finally, the pooling is applied independently in each feature map channel, like in the max pool.

### 3.7.2 Inicializatin Fast-RCNN

This network starts using the ImageNet pre-treined network and have five max pooling layers and around 30 to 35 convolution layers and undergoes with three changes of Fast-RCNN structure.

Firstly, the last max pooling layer is replaced by a RoI pooling layer configured with H and W to be compatible with the network first fully connected layer.

Secondly, the last fully connected layer and softmax function, which were trained with a 1000-way ImageNet classification, are switch to two sibling layers

described earlier. That is, a fully connected layer and a softmax over K + 1, where K = 1, categories and category-specific bounding-box regressors.

Lastly, the network is modified to take two data inputs. One is a list of images and the other is a list of RoIs in those images.

The following image shows a schematics how fast-RCNN works.

#### 3.7.3 Fine-tuning for Fast-RCNN

For Fast-RCNN, the stochastic gradient descent ( SGD ) minibatches are hierarchies by sampling N images and by sampling R/N RoIs for every image. Thus, the RoI on the same images e share computation and memory in the forward and backward passes doing N small decreases mini-batch computation.

An important addendum on strategy is that a delay in convergence can occur during the training, due to correlated RoIs from the same image.

Besisdes that, Fast-RCNN uses as treamlined training process with one fine-tuning stage to optimazes the softmax classifier and bouding-box regressor in three different stages.

### 3.8 Multi-Task loss

Fast-RCNN have to kinds of outputs layers. The first layer outputs a per RoI discrete probability distribution in the form $p = (p_0, p_1, \ldots, p_k)$ over the K + 1 categories. The value of p is computed using softmax over the K + 1 outputs of a fully connected layer. The second ouput layer ouputs a bounding-box regressor offsets with $t^k = (t^k_x, t^k_y, t^k_w, t^k_h)$ for every K object classes indexed by k. The parametrization $t^k$ is the same as Eq.3.9.4 describe on the topic of RPN. $t^k$ is used to specifie a scale-invariant translation and log-space height or width shift relative to an object proposal. In the training each RoI labeled with a ground-truth class u and a ground-truth bounding-box regression target v. In addition to this se a multi-task loss L is used each labele RoI jointly train for the classification and bounding-box regression which is of the form :

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v) \tag{12}$$

where $L_{cls}(p, u)$ = - log($p_u$) is the log loss for true class u, $L_{loc}$ is defined over a tuple of true bounding box regression target for class u, v = $(v_x, v_y, v_w, v_h)$ and a predicted tuple $t^u = (t^u_x, t^u_y, t^u_w, t^u_h)$ for class u. The Iverson bracket indicator function ( $[u \geq 1]$ ) is equal to 1 when u ≥ 1 and 0 otherwise. By convention the catch-all background class is labele as u = 0. The background RoIs does not take into account the bouding box ground truth, consequently $L_{loc}$ is ignored.

The bouding box loss function is in the form :

$$L_{loc}(t^u, v) = \sum_{i \in (x,y,h,w)} soomth_{L_1}(t^u_i - v_i) \tag{13}$$

and

$$soomth_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise} \end{cases}$$

Where $L_1$ is the loss. The hyper-parameter $\lambda$ in Eq.12 is defined as $\lambda = 1$ and used to balance this two task loss. Moreover, the $v_i$ ground-truth regression targets is normalized to have zero mean and a unit variance.

### 3.8.1 The Mini Batch Sampling

In the fase of fine tuning, every *SGD* mini batch is defined from N = 2 images which were chosen at random and uniformemente. The mini batches are defined of size R = 128 with a sampling of 64 RoI per image. In this samples, is taken 25% of RoI from the object proposals that have intersection over union IoU overlap with a ground truth bouding box with values greater than or equal to 0.5. This RoI includes labeled examples with a foreground object class.

The other 75% of RoI are samples from the object proposals with a maximum IoU ground truth between [0.1, 0.5). In this case, this samples are the background examples and consequently are labeled with u = 0. The lower threshold of 0.1 act as a heuristic for hard example mining. Besides that, during the training, all images have 0.5 probability of being horizontally flippep and no other data augmentation is used.

### 3.8.2 Using Back-Propagation along the RoI pooling layers

Back-Propagation performs derivatives operations along the RoI pooling layer. It is important to remember that is assumed a only one per mini-batch, that means N = 1. For the other extensions N > 1 is straightforwar, because the forward only pass treats of all images independently.

The $x_i \in \mathbb{R}$ is the i-th activation input into the RoI pooling layer and $y_{rj}$ is the j-th ouput from the r-th RoI. The RoI pooling layer calculates $y_{rj} = x_{i^{*(r,j)}}$, where $i^{*(r,j)} = argmax_{i' \in \mathbb{B}(r,j)} x_i$, $\mathbb{B}(r,j)$ is a set of inputs index on the sub-window over, where are the output unit $y_{rj}$ max pools. Besides that, one $x_i$ variable can be assigned to several distinct $y_{rj}$ outputs.

The backwards RoI pooling layers function calculates parcial derivative of the loss function in relation to each input variable $x_i$ by following the argmax switches :

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r,j)] \frac{\partial L}{\partial y_{rj}} \tag{14}$$

In other words, each mini batch RoI r and pooling ouput unit $y_{rj}$ is accumulated by the parcial derivative $\frac{\partial L}{\partial y_{rj}}$ when i the argmax selected by $y_{rj}$ using max pooling. During the Back-Propagation, the partial derivatives $\frac{\partial L}{\partial y_{rj}}$ are already been computeted by using Eq. 14 on top of the RoI pooling layer.

#### 3.8.3 The SGD hyper-parameters

All the Fully connected layers used for the softmax classification function and bouding-box regressor are inicialized by a zero-mean Gaussian distributions with standard deviations to 0.01 and 0.001, respectively, and the Biases are initialized to 0. Every layer use a per-layer learning rate of 1 for weights and 2 for biases, a global learning rate of 0.001 and a standard momentum of 0.9.

#### 3.8.4 The Scale invariance Property

Fast-RCNN use a brute force methods to obtain the scale invariant object detection. On the brute force method, every image is processed in a fixed pre-defined pixel size during both training and testing. With that, the network learning directly the scale-invariant object detection from the training data.

#### 3.8.5 Fast-RCNN detection method

After the fine-tunning of Fast-RCNN, detection turns a little more than running a forward pass, since the objects proposals are pre-computed. Firstly, the network take a image or a image pyramid encoded as a list of images as input and a list of R object proposals to score. In test time, R is arround 2000. For the pyramid images, each RoI is assigned in a scale such that the scaled RoI is around to $224^2$ pixels in the area.

Every tested RoI r the next pass a output of posterior probability distribution p and a set of predicted bounding-box offsets relative to r, where each the K ( in this work, K = 1 ) classes gets his unique refined bouding-box prediction. In this way, it is assumed a detection confidence to r for every object class c using the estimated probability $pr(class = c|r) =^{\delta}= p_k$

#### 3.8.6 Faster detection using SVD Truncate

The SVD truncate method is used on Fast-RCNN to acelerate the classification and detection on fully connected layers.

In this technique, a layer is parameterized by u x v and a weight matrix W is approximately factorized as :

$$W \approx U \Sigma_t V^T \tag{15}$$

In this SVD fatoration, U is u x t matrix comprising the first t left singular vectors of W, $\Sigma_t$ is a t x t matrix that contains the top t top singular values of

W and V is a v x t matrix comprising the first t right singular vectors of W. This method reduces the parameter count from uv to t( u + v ) parameters. This can be a significant gain when t is smaller than min ( u, v ). The network is compress by replacing the single fully connected layer corresponding to W by two fully connected layers without a non-linearity between them. thereby, the first of this layers uses a weighted matrix $\Sigma_t V^T$ without biases and the second layers uses the U matrix , in which it is associated with matrix W. This simple compression method gives good speedups when the number of RoIs is large.

## 3.9 *Region Neural Networks* (RPNs)

RPNs are fully-convolutional which simultaneously forecast object bounds and make the object score classification at each position. *Region Proposal Networks* (RPN) uses RGB m x l images as input datas and output a set of regular of object proposals, each with an objectness score delimited by a retangular area.

Firstly, is its applied convolutional layers on the images and is aplly another network on the feature map outuputed by the last of the convolutional layer which will serve as input to this other neural network. This process in the end generates the region proposals of the objects.

This another network receive as input $n$ x $n$ spatial window of the last convolutional feature map and every sling is mapped to a binay feature. In the algoritmh [1] is defined to use on this layer a 512-d with a *ReLu* [6] as the activate function

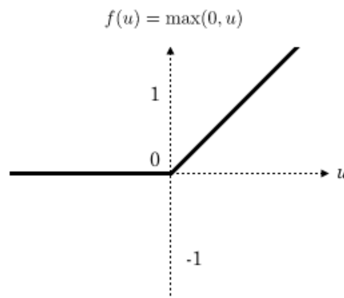



Figure 4: Relu Function

After that, two *fully-connected* layers will be aplly on this features, one to make the bounding box and the other to make the object classificaction .

It is important to note that this type of network is implemented with a $n$ x $n$ convolutional layer and two 1 x 1 convolutional layers in the end, respectively, one to make the *bounding-box* regression and the classification.

### 3.9.1 *Anchors* boxes on classification

*Anchors* boxes is the final *bounding box* along with reliability rate of the classification.

For each sliding-window location, happens in paralel a multiple predict of region proposals. Based on the article [8], the maximum possible of proposals for each location is denoted as $k$. Thus, the *bouding-box* regressor layer have *4*k ouputs for the coordinates of $k$ boxes and the *classification* layer ouputs a 2 $k$ scores that estimate the probability if is a object or not on each proposal. Note that this $k$ proposals are parameterized in relative way to $k$ refences boxes. The result of this process is if refered in leature as *Anchors*.

This *Anchor* is centered in the sliding window and is associate with a scale and 5 aspect ratios, ( *8* x *8*, *16* x *16*, *32* x *32*, *64* x *64* and *128* x *128* ).

Besides that, is used $k = 64$ on each sliding position. Therefore, a convolutional feature map of size $W$ x $H$ will have, $W$ x $H$ x $k$ anchors in total.

#### 3.9.2 Anchors by invariant-Translation

One important property is that the anchors terms and functions with compute relative proposals they are not influenced by any translation. If there is a object translation in a image, the proposal is capable translate the object in his original location on image and the same function is able to to predict the proposal in either location. In this case, for FCNs, the network is a translation invariant up to the network's total stride.

#### 3.9.3 Multi-Scale Anchors

this structuring of anchors created a differentiated method to address multiple scales. With this new model, is possible classify and make the regression bouding-boxes by reference of anchor boxes of multiple scales and aspect ratios.In another words, this method depends only on imagens of a single scale and features maps and filters , that sliding windows on the features maps, of a single size.

Because of this structure, it is possible to use convolutional features computed on a single-scale, the same way that Fast-RCNN detector does. This desing of multi-scale anchors is the main differential for sharing features without extra cost for addressing scales.

#### 3.9.4 Loss Function to RPNs

During the training phase of RPNs, a binary classification is carried out, defining whether there is an object or not in each anchor. Furthermore, a positive label is assigned to the two kinds of anchors. One of them is the anchor (or anchors) with the highest Intersection-over-Union (IoU) superimposed with a ground-truth box. The second is an anchor that has IoU overlap higher than 0.7 with any ground-truth box. It is important to notice that a ground-truth can associate several positive labels with several anchors.

Commonly, only the second case, which is the case of the second type of anchor, covers most cases determining positive samples. However, there are certain cases where only the second anchor does not find positive samples. To

solve this, we choose the anchor (or anchors) with the highest IoU superimposed with a ground-truth box. Besides that, a negative label is assigned to non-positive anchors when the IoU ratio is lower than 0.3 for every ground-truth boxes.

Once these characteristics are defined, the objective function following the multi-task loss in Fast-RCNN will be minimized. For the image, the following loss function is defined:

L( $p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \frac{\lambda}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*)$ (16),
where i is an index of each anchor in the mini-batch, $p_i$ is the probabillity of prediction of a i anchor to be an object, $p_i^*$ is the ground-truth label and has value 1 if the anchor is positive and 0for a negative anchor, $t_i$ is a vector that represents the 4 coordinates of the ground-truth box referring to a given anchor. For the classification function $L_{cls}$, a log loss over two classes is used, one of them is the Object and another one for what is not an object, in the form $L_{cls} = R(t_i - t_i^*)$ where $R$ is the same robust smooth function $L_1$ defined in the Fast-RCNN. It is important to notice that the term $p_i^* L_{reg}$ activates the regression loss only for positive anchors (that is, for $p_i^* = 1$) and turns it off when $p_i^* = 0$. Besides that, the classification and regression bouding-box are associated, respectively, with $p_i$ and $t_i$.

Both terms $N_{cls}$ and $N_{reg}$ are balanced by a parameter $\lambda$. For the implementation, the mini-batch size $N_{cls} = 256$ is chosen, the regression is chosen by the number of anchor locations, which revolves around $N_{reg} \approx 2400$ and $\lambda = 10$.

For the bouding box regression, the following parameterization was adopted, being the same used on Fully-Convolucinal-Neural-Network :

$$\begin{aligned}
t_x &= \tfrac{(x - x_a)}{w_a} & t_y &= \tfrac{(y - y_a)}{h_a} \\
t_w &= \log \tfrac{w}{w_a} & t_h &= \log \tfrac{h}{h_a} \\
t_x^* &= \tfrac{(x^* - x_a)}{w_a} & t_y &= \tfrac{(y - y_a)}{h_a} \\
t_w^* &= \log \tfrac{w^*}{w_a} & t_h^* &= \log \tfrac{h^*}{h_a}
\end{aligned} \quad (17)$$

,

where x and y are the box center coordinates, w is the width and h is the height. The variables x, $x_a$ and $x^*$ are, respectively, used for the predicted box, anchor box and truth box and the representation is similar to y, w and h.

This is a heavy task for a bouding-box regression to an anchor box for a nearby ground-truth box.

The bouding-box is performed on features pooled from arbitrarily sized RoIs and the regression weights are shared by all region sizes. Beyond that, the features used on regression have the same spatial size, which are (3 x 3), on the features maps. To accounting for the various sizes, the RPNs learn a set of k bouding-box regressors. Thereby, because of the design of the anchors, it is still possible to make prediction of various box sizes, even though the features are in a fixed size or scale.

### 3.9.5 RPNs Training

RPNs are trained end-to-end by backpropagation and using stochastic gradient descent (SGD) for optimization. Also for training, each mini-batch comes from a single image, which contains many positive and negative examplar anchors. It is possible optimize the loss functions for all anchors, consequently, the bias will be guided towards negative values, that is, it will have a high chance of overheating in negative images. To prevent this from happening, 256 anchor samples were randomized on an image to compute the loss function of a mini-batch. Besides that, these anchors have a ratio of up to 1:1 for sampled positives and negatives. In case of having fewer than 128 positive samples in the image, the mini-batches are shuffled with negative samples.

Before starting the training, all the weights of the layers are randomly initialized from a zero-mean Gaussian distribution with standard deviation 0.01 and the shared convolutional layers are initialized by the ImageNet weights. Besides that, a learning rate of 0.001 and a momentum of 0.9 are used.

## 3.10 Structure of a Faster-RCNN

Faster-RCNN uses end-to-end trained RPNs to generate high quality region proposals that Fast-RCNN will use to dection and with a alternating optimization is possible to RPNs and Fast-RCNN share convolutional features, making the classification and the bouding-box regressor more easly and fast.

### 3.10.1 Sharing convolutional features

As described in the training topic of Region Neural Networks, we train neural networks to generate region proposals without considering object based detection of a CNN. This procedure allows shared convolutional layers between RPNs and Fast-RCNN.

Both RPNs and Fast-RCNN are independently trained in parallel and modify the convolutional layers in different ways. The strategy can be divided into four steps. In the first one, the RPNs are trained like in the RPN training and are initialized with ImageNet pre-treined weights and fine-tuned end-to-end for the region proposal task. In the second, a detection network of type Fast-RCNN is trained separately by using the proposals generated by RPN in the previous

step and this detection network is also initialized by the ImageNet pre-trained model. It is important to notice that until this second step, both networks do not share the convolutional layers.

In the third step, the detection network is used to initialize the RPN training. Consequently, the convolutional and fine-tuning layers of RPN are ajusted and shared. In the last step, the fully-convolutional layers of Fast-RCNN are fine-tuned. Finally, both networks now share the same convolutional layers and form a unified network.

# 4 Mathematical-Computational Methodology

## 4.1 Neural networks of the type *Mask-RCNN*

*Mask R-CNN* have the same two-stages behavior of *Faster R-CNN*. The first stage is based on using a *Region Proposal Network* (RPN) [7] that uses as input the original images and as output a set of rectangular object proposals, each one with an objectness score. The second stage is where the prediction of the class and *bouding box* occur in parallel with the first stage.

Besides that, *Mask-RCNN* also outputs a binary mask for each *RoI*. That is, in a nutshell, *Mask-RCNN* have the same principles of *Faster-RCNN*, that simultaneously applies regression and *bounding-box* to the image.

Finally, we aplly a fully-connected network with 2 branchs in the output of the Region Neural Network. One of the branches perform the object classification using a sigmoid as the activation function. The other one applies a bounding-box regressor to carry out the semantic segmentation of the interested object in the image. In the literature, it is common to use an FFC to make the semantic segmentation of the interested object and obtain good results.

### 4.1.1 *Mask-RCNN* training

*Mask-RCNN* is defined as a multiple task loss on every *RoI* as $L = L_{class} + L_{box} + L_{mask}$, where $L_{class}$ is the classification loss, $L_{box}$ is the *bouding-box* loss and $L_{mask}$ is the mask loss. Those parameters are defined as the same as in [7].

The mask branch has $C\ n^2$ dimensional output for every *RoI*, which will generate $C$ binary masks with a resolution of $n$ x $n$, one for each one of the $C$ classes. Here, $C = 1$ as our segmentation goal are only nuclei in images.

Thus we apply a *sigmoid* function per pixel and define the average binary cross-entropy loss $L_{mask}$, with a *RoI* and a ground-truth class $k$ . The $L_{mask}$ is only determined in the $k$-true mask. This implies that other mask outputs do not contribue to the $L_{mask}$ loss.

The definition of $Lmask$ enables the *Mask-RCNN* to generate masks for the nucleus class. Besides that, we used a especial classification branch to predict the nucleus label and select the output mask, which leads to a *decoupled* mask and class prediction. For the *Mask-RCNN*, masks compete over the classes by a per-pixel *sigmoid* and a *binary* loss.

### 4.1.2 Binary masks on the features maps

The masks represent the object *spatial* layout. On the other hand, the class label or box offset that are converted into short output vectors by *fully-connected* (fc) layers by *Mask-RCNN* can extract the spatial structure of masks by a pixel-wise approach by convolutional operations with accurate results.

Thus an $n$ x $n$ mask can be predicted from each *RoI* using an FCN [8]. This prediction allows for each layer in the mask to maintain an $n$ x $n$ spatial layout of the object explicitly and without propagating errors in the space dimension.

This pixel-wise process requires the *RoI* features, which correspond to small *features maps* that need to be precisely aligned to preserve the explict per-pixel spatial correspondence. Therefore to maintain it precisely aligned, a layer called *RoIAlign* layer is created.

### 4.1.3 Function of *RoIPool*

The *RoIPool* is an operation that extracts a small *feature map* for each *RoI*. First, the *floating-point* values of the *RoI* are *quantized* to a discrete granularity of the feature maps. Second, a *max-pooling* operation is applied over the feature maps.

In the following, the quantized histogram of the pixel values is computed.

This previous computation introduces misalignments between the *RoI* and the extracted features. It is important to emphasize that the misaligments may not impact in the classification, which is supposed to be tolerant to small translations, but it negatively affects the large scale prediction of the pixel-wise masks.

### 4.1.4 Function of *RoiAlign*

*RoiAlign* is a branch layer that aligns properly the extracted features with the inputs, instead of making a massive harsh quantization of *RoIPool*

To avoid any quantization of *RoI* boundaries or bins, the *bilinear interpolation*[4] is employed to compute the exact values of the input features, which are delimited by four regularly sampled locations for each *RoI* bin and put together the result by using a maximum value.

It is worth to highlight that the results are not sensitive neither to the sample location nor to the number of sampled points, provided that there is no quantification operation.

## 4.2 Algorithmic structure of a Mask-RCNN

The Mask-RCCN implementation can be divided into two stages. In the first one, we define the convolucional backbone that will be used to make the feature extraction on the whole image. In the second stage, the network head is defined to make the bouding-box recognition, that is, for classification, regression, and mask prediction, which is applied separately to each RoI.

The backbone used here is a ResNet [3] network with a depth of 50 layers and a Faster-RCNN that extracts features from the final convolutional layer of the $4^{th}$ stage. The network head is the same presented in [3], but includes the $5^{th}$ stage of ResNet named as res5.

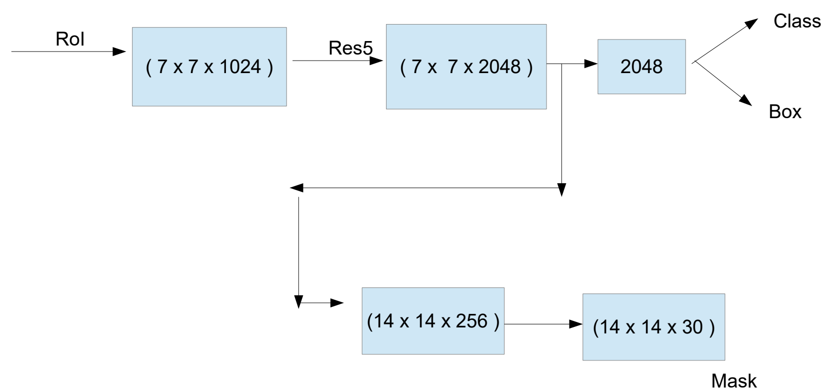



Figure 5: Representation of a mask-RCNN with resnet50 and fast-RCNN.

### 4.3 Medical image application

Biological images (such as those representing medical or plant structures) have been one of the most prominent areas to provide complex and large-scale appropriate data to be analyzed in deep learning. In this case, deep learning techniques are propitious to detect and discriminate hidden patterns that are not identifiable by classic image descriptors, much less by the human eye .

A very interesting aspect of deep learning in this scenery is that the attributes are presented in a hierarchical order: the frist layers detect lower level and more universal features while the last layers capture nuances of the presented object.

Curiously, this hierarchical paradigm is related with a multiscale analysis and with the diferent perspectives that professionals, such as pathologists, have used for years.

## 5 Dataset Description

The train, test and Validation datasets are composed of RGB images that contain cell nuclei, which will be used as input for the algorithm. Here we use the

nuclei segmentation dataset from Kaggle 2018 Data Science Bowl.

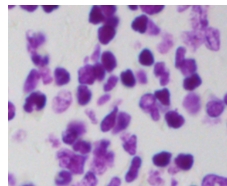
In addition, there are also binary images that contains only the image of a segmented nucleus. These binary masks will be used to compare the Mask-RCNN binary mask outputs and the original nucleus binary masks, in order to improve the accuracy of the classification.

The training dataset, as the name suggests, will be used to calibrate the weights of the neural network to locate the cell nucleus, i.e., to perform the segmentation of the cellular nuclei and provide the reliability rate of the object classification, which in this case is the cell nucleus.

Once the traing is over, the test dataset will be used as a mechanism to verify if the training was successful. In other worlds, we will verify if the algorithm generalization capacity of the algorithm is adequate to perform the classification of other images containing cell nuclei with a low associated error rate.

This type of approach is very common in the machine learning literature and is named *supervised learning*.

These datasets have samples that contain nucleus and the respective binary mask delimited by Mask-RCNN. The image below represents an example of some samples of the dataset.



(a) Mask-RCNN input image train

(b) nucleus binary mask (c) nucleus binary mask (d) nucleus binary mask (e) nucleus binary mask

Figure 6: Image and its respective binary masks for each nucleus.

# 6 Results

Based on the results presented by the Kaggle 2018 Data Science Bowl participants, the best models using the architecture described here achieves accuracy between 60.93% and 63.16% using “stage2-test-final” as the validation dataset. Besides, as for the best model presented in this competition, the average mask Intersection over Union (IoU) was between 66.98% and 95.00% for the threshold

precision. The presented results show the potential of Mask-RCNN to produce a high quality automatic segmentation of nuclei within varied microscopy images. It is also possible to apply this kind of neural network in promising models of prediction and segmentation of medical images that contain tumors in cellular tissues. Below, it is possible to see some images of the results of the automatic segmentation generated by the Mask-RCNN

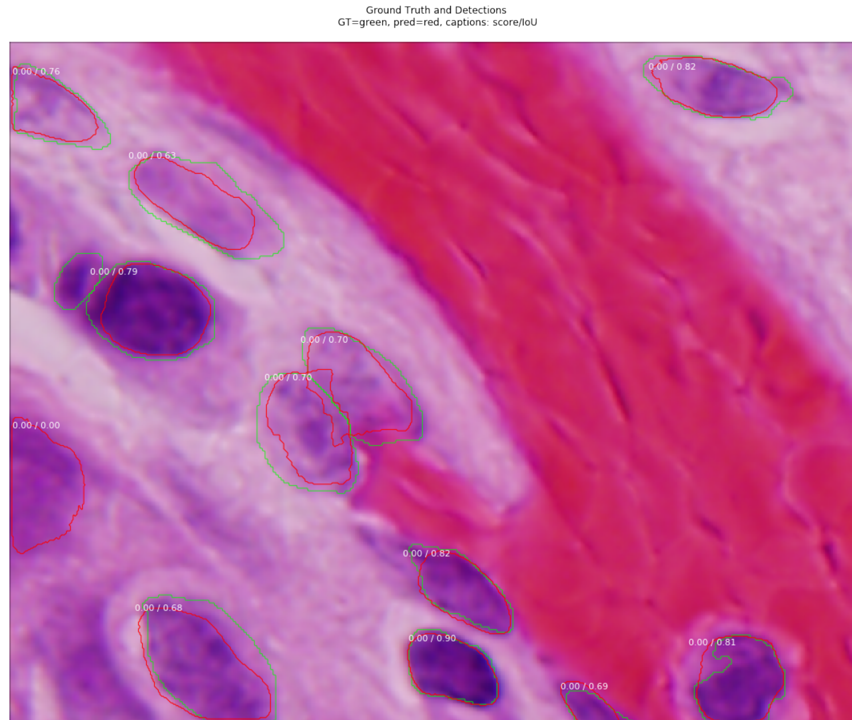



Figure 7: An outuput example of the network segmentation prediction, Where the green bounding box is the original nucleus and in red box we have the predicted bounding box.

# 7 Conclusions

The results presented demonstrate the potential of Mask-RCNN in the task of automatic classification of nuclei. Even with a reasonable performance of, on average, 62% accuracy, it is possible to improve this model by using more images to train the network or changing its architecture. Thus, with a better trained Mask-RCNN, it is becomes feasible deploy this model for professionals who perform the manually nuclei segmentation in images, making the analysis more faster and precisely using this Neural Network.

Detections

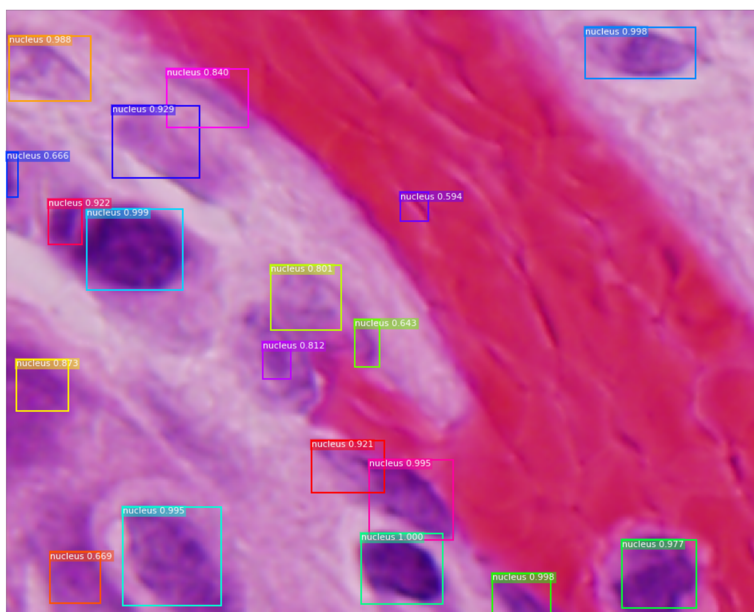



Figure 8: An outuput example of Mask-RCNN, with the class label score and the bounding box.

## References


[1] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. `https://github.com/matterport/Mask_RCNN`, 2017.

[2] B. Chen, C. Gong, and J. Yang. Importance-aware semantic segmentation for autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 20(1):137–148, 2019.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. pages 2017–2025, 2015.

[5] Yiming Liu, Pengcheng Zhang, Qingche Song, Andi Li, Peng Zhang, and Zhiguo Gui. Automatic segmentation of cervical nuclei based on deep learning and a conditional random field. *IEEE Access*, PP:1–1, 2018.

[6] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. pages 807–814, 2010.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. pages 91–99, 2015.

[8] Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):640–651, April 2017.

[9] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.