Premier Reference Source

Machine Learning and Deep Learning in Real-Time Applications

Mehul Mahrishi, Kamal Kant Hiran, Gaurav Meena, and Paawan Sharma

IGI Global

Chapter 1
Obtaining Deep Learning Models for Automatic Classification of Leukocytes

Pedro João Rodrigues
https://orcid.org/0000-0002-0555-2029
CeDRI, Research Centre in Digitalization and Intelligent Robotics, Instituto Politécnico de Bragança, Portugal

Getúlio Peixoto Igrejas
https://orcid.org/0000-0002-6820-8858
CeDRI, Research Centre in Digitalization and Intelligent Robotics, Instituto Politécnico de Bragança, Portugal

Romeu Ferreira Beato
Instituto Politécnico de Bragança, Portugal

ABSTRACT

In this work, the authors classify leukocyte images using the neural network architectures that won the annual ILSVRC competition. The classification of leukocytes is made using pretrained networks and the same networks trained from scratch in order to select the ones that achieve the best performance for the intended task. The categories used are eosinophils, lymphocytes, monocytes, and neutrophils. The analysis of the results takes into account the amount of training required, the regularization techniques used, the training time, and the accuracy in image classification. The best classification results, on the order of 98%, suggest that it is possible, considering a competent preprocessing, to train a network like the DenseNet with 169 or 201 layers, in about 100 epochs, to classify leukocytes in microscopy images.

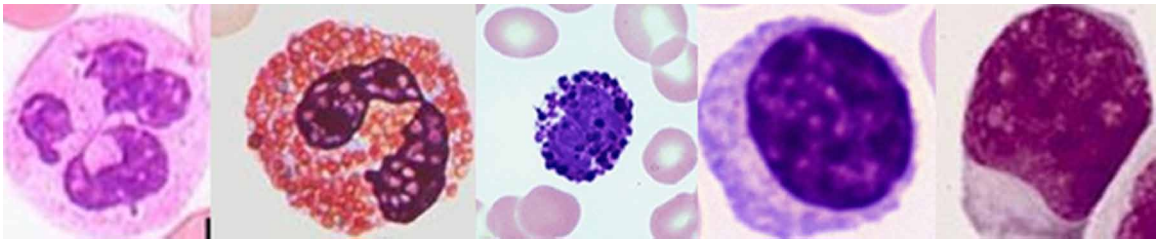
DOI: 10.4018/978-1-7998-3095-5.ch001

INTRODUCTION

The number of leukocytes presdataent in the blood, also known as white blood cells, provides important information regarding the state of the immune system, allowing to evaluate potential health risks. A significant change in the number of leukocytes, relative to reference values, is usually a sign that the body is affected by some type of antigen. Moreover, the variation in a particular white blood cell type is generally correlated with a specific type of antigen.

White blood cells are generally classified into 5 categories: lymphocytes, monocytes, neutrophils, eosinophils, and basophils. There is also the band designation for a specific form of the nucleus. Figure 1 shows examples of these categories.

Figure 1. Leukocyte types (from left to right): Neutrophil, Eosinophil, Basophil, Lymphocyte, and Monocyte (Noble, 2019)



The preparation and analysis of blood samples are usually affected by deviations naturally introduced by manual operations. These difficulties can be minimized when performed by highly trained technicians. However, these tasks are labor-intensive and time-consuming and always subject to error. For these reasons, there is interest in having systems that can automatically classify with high specificity and high sensitivity.

The evolution of techniques for counting and identifying leukocytes (and blood cells in general) began in the mid-nineteenth century with the use of capillaries and slides. Over the years, several types of devices designed to count blood cells have appeared, which would later enable their classification.

The classification of blood cells has always been done by human specialists until the 1960's, since when emerged the possibility of automating this task. Firstly, through the use of optical and impedance methods and later through algorithms developed specifically for this purpose from microscopy images within the scope of computer vision and, in the last two decades, using neural networks.

Several researchers have presented papers for leukocyte identification and counting. Techniques that use classic machine learning models, in opposite to deep learning models, are built on manually selected characteristics. This approach can use shallow neuronal networks (Rezatofighi & Soltanian-Zadeh, 2011; Nazlibilek et al., 2014), Support Vector machines (SVM) (Rezatofighi & Soltanian-Zadeh, 2011; Putzu et al., 2014; Agaian et al., 2014; Alférez et al., 2016; MoradiAmin et al., 2016), Bayes classifier (Stathonikos et al., 2013; Prinyakupt & Pluempitiwiriyawej, 2015), etc. The manipulation of characteristics, prior to the classification model, may involve segmentation (Chaira, 2014), extraction and selection of features that describe the leukocyte defining region (Alférez et al., 2016). Thus, we can divide these classic processes into three main stages: segmentation, feature extraction, and classification. These approaches

have the advantage of allowing the use of relatively small datasets, as the segmentation and the features used reduce the variability of the patterns delivered to the classification models. On the other hand, the segmentation performance and the lack of universality of the descriptors can limit the result achieved by the classification models. An example of this approach is found in Dan et al. (López-Puigdollers et al., 2020), where features such as SIFT (Lowe, 2004) are employed. However, the assertiveness of the classification is not very high.

The use of deep learning models tends to solve the problems presented in the approach described above, provided that the dataset is sufficiently representative of the pattern variability, associated with leukocyte optical visualization, or a transfer learning approach is employed, as is the case of the present work.

Recognition models based on convolutional neuronal networks (CNNs) have performed well in many applications. In this sense, studies appear in the literature where the use of CNNs is applied to leukocyte detection and recognition (Zhao et al., 2016). However, the accuracy presented in that paper for the identification of five base leukocytes is not high (eosinophil 70% and lymphocyte 75%). Other authors (Shahin et al., 2019), who also use CNNs to classify rather than detect the five base leukocytes, show results that exceed the classical approach (accuracy: 96%), so it can be deduced that the separation between detection and classification is beneficial for the problem-solving. Most methods using CNNs require leukocytes to be already segmented/detected (Shahin et al., 2019; Choi et al., 2017; Jiang et al., 2018; Qin et al., 2018; Rehman et al., 2018 and Tiwari et al., 2018). In this context, there are various approaches to CNNs: (e.g.) Regions with CNN features (R-CNN) (Girshick et al., 2020), Faster R-CNN (Ren et al., 2017), based on architectures that simultaneously integrate both tasks -detection and classification-. These architectures start from a preprocessing step where regions of interest are estimated for subsequent unification through a CNN-based training step and further classification.

Other architectures are one-stage and fully integrate detection and classification: e.g., You Only Look Once (YOLO) (Redmon et al., 2020), YOLOv2 (Redmon & Farhadi, 2020), YOLOv3 (Redmon & Farhadi, 2020) and Single Shot Multibox Detector (SSD) (Liu et al., 2020). Wang et al. used SSD and YOLOv3 models for the detection and classification of leukocytes, reaching a mean average precision of 93% (Wang et al., 2019). This value is noteworthy since this work shows the classification of eleven types of leukocytes, while most works use the five base leukocytes.

In the present work, the proposal is the classification of leukocyte images using the winning network architectures (with the exception of one) of the annual ILSVRC (ImageNet Large Scale Visual Recognition Challenge) software competition where programs compete to detect and classify objects and scenarios (Russakovsky, et al., 2015). The identification of leukocytes is made using pre-trained networks and these same network architectures trained from scratch in order to select the best network and the best approach for the intended task.

BACKGROUND

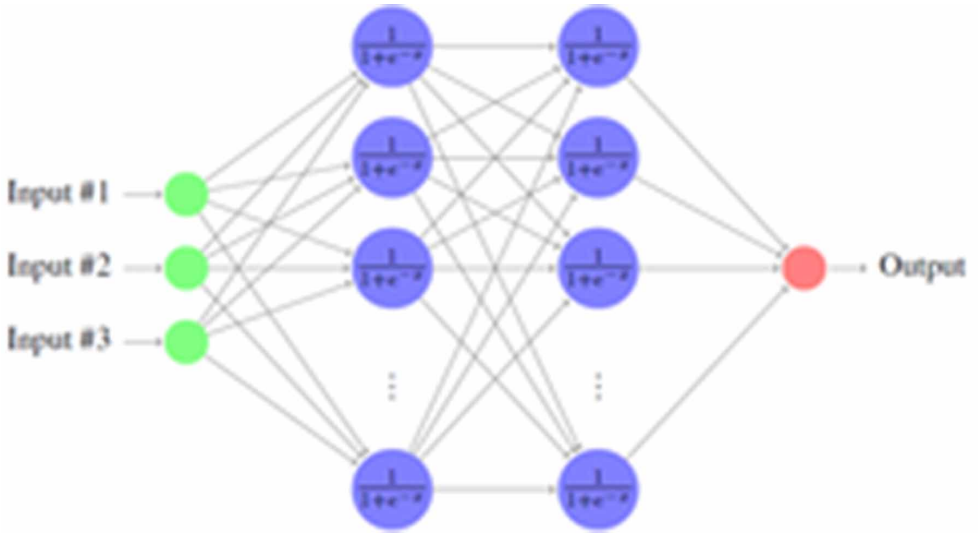
The difficulty of pattern recognition in images becomes apparent when we try to write a computer program that recognizes leukocytes in microscopy images. When we look for a set of precise rules for the recognition of forms, we quickly lose ourselves in an immensity of exceptions and special cases. Neural networks (NN) approach this type of situation differently (Nielsen, 2017). In the most common type of feedforward networks - MLP (multilayer perceptron) - neurons are organized in layers that have one-

way links between them. Different connectivities produce different behaviors. Generally speaking, the feedforward networks are static, and they produce only one type of output value instead of a sequence of values of a given input. So, they don't have memory in the sense that their response is independent of the previous state of the network. Recurrent networks, on the other hand, are dynamic systems. Due to the feedback paths, the inputs of each neuron are modified, which causes the neural network to enter a new state dependent on the past states.

Neural Network Architectures

Neural Networks architecture includes several layers organization. Usually, they have one input layer that receives the feature dataset, one or more hidden layers and one output layer. The number of neurons included in each layer can vary and is defined by a trial and error scheme. For example, the four-layer NN of Figure 2 has two hidden layers:

Figure 2. Neural network with an input layer, two hidden layers and output layer



Training

At the beginning of the neural network training, the weights or values of the filters must be set at random, using very small values (but different from each other). At this stage, the filters of the initial layers don't know how to look for edges and curves in images, nor the ones of the higher layers know how to look for legs or nozzles.

The idea of providing an image with a label is related to the training process through which convolutional neural networks (CNN) pass. So, we have a training set that has thousands of images of the classes that we want to identify, and each of the images has a label of the respective category (Deshpande, 2016). For example, a training image of a handwritten digit enters a CNN, a 32×32×3 matrix that passes through the entire network. In the first training example, with all the weights or filter values randomly

initialized, the output will probably be something like [.1 .1 .1 .1 .1 .1 .1 .1 .1 .1]. Basically, it gives an output that does not give preference to any particular number. Let's say the first training image entered was a "4". The image label would be [0000100000]. Here we have to calculate the loss function, which can be defined in many different ways, such as the MSE (mean square error). If the MSE is used, the loss is given by (1).

$$E_{total} = \sum 1/2\left(target - output\right)^2 \tag{1}$$

During the training, the value of each weight, w, is updated according to the negative direction of the E gradient, until E becomes sufficiently small. Here, the parameter η is the learning rate, and w is updated with (2) (Zhang & Devabhaktuni, 2003).

$$w = w - \cdot \frac{\partial E}{\partial w} \tag{2}$$

The learning rate is defined or chosen at random. A high learning rate means that larger steps are used in weight updates, and thus it may take less time for the model to converge to an ideal set of weights. However, a very high learning rate can result in jumps that are too large and not accurate enough to reach the ideal point.

All these steps constitute a training iteration. The program will repeat this process for a fixed number of iterations for each set of batch training images. After finishing the parameter update in the last training example, the network is expected to be trained, meaning that the weights of the layers are adjusted correctly (Deshpande, 2016).

Activation Functions

Neural networks can use various types of activation functions. The selection of the activation function is an important issue since it has a direct impact on the processing of input information. The activation functions used in this project were the ReLU and Softmax.

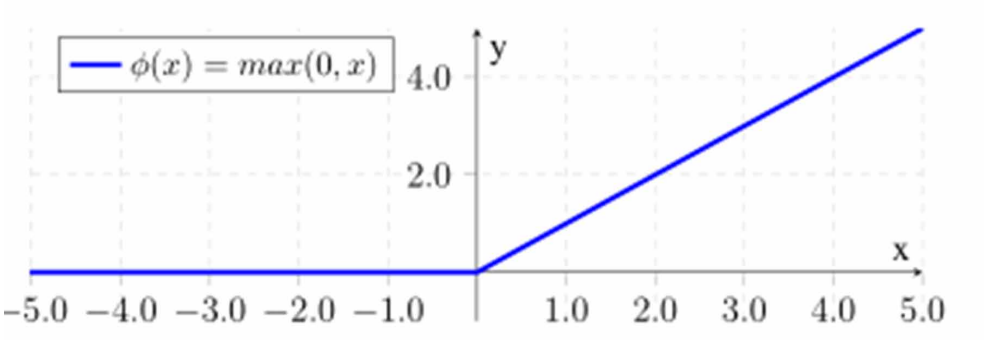
ReLU

In 2010 Teh and Hinton introduced the ReLU (Rectified Linear Unit), which, despite its simplicity, is a good choice for hidden layers. The advantage of ReLU (shown in Figure 3) lies, partly, in that it is linear and unsaturated. Unlike the sigmoid and hyperbolic tangent functions, the ReLU does not saturate in the positive domain, although it saturates in the negative domain. A saturation function tends to move to a certain value (Nascimento, 2016).

The neuroscience literature indicates that cortical neurons are rarely in their maximum saturation regime and suggest that their activation function can be approximated by a rectifier (Bush & Sejnowski, 1996) .

After uniform initialization of the weights, about 50% of the continuous output values of the hidden layers are real zeros, and this fraction can easily increase with the sparsity-induced regularization (relative to the number of connections between a neuron with other, which is to be minimized), being biologically more plausible than many other activation functions (Glorot, Bordes, & Bengio, 2011)

Figure 3. Plot of rectified linear unit (ReLU) function



Softmax

Along with the linear function, softmax is usually found in the output layer of a neural network. The activation function softmax forces the output of the neural network to represent the probability of the input is inserted into each of the classes. Without the softmax, the outputs are simply numerical values, being the largest indicative of the winning class. When we insert information into the neural network by applying the activation function softmax, we get the probability that that input can be categorized according to predefined classes. The formula is shown in (3).

$$\varnothing_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}} \tag{3}$$

where *i* represents the index of the output node, and *j* represents the indexes of all the nodes in the group or level. The variable *z* designates the vector of the output nodes. It is important to note that the softmax function is calculated differently from the other activation functions provided. When using softmax, the output of a single node depends on the other output nodes. In the previous equation, the output of the other output nodes is contained in the *z* variable, unlike the other activation functions (Kloss, 2015).

Feedforward

As seen before, the networks in which the output of a neuron is used as the input of the neuron of the next layer are called feedforward neural networks. This means that there are no loops or cycles in the network, information is always passed, and there is no feedback (Nielsen, 2017). Given the inputs $x = [x_1, x_2, \ldots, x_n]^T$ and the weights *w*, a feedforward network is used to calculate outputs $y = [y_1, y_2, \ldots, y_m]^T$ of a network MLP. In the feedforward process, external inputs feed the neurons of input (first layer). The outputs of the input neurons feed the hidden neurons of the second layer, and so on, and finally the outputs of the layer L^{-1} feed the neurons of the last layer (*L*) (Zhang & Devabhaktuni, 2003). The calculation is given by (4) and (5)

$$z_i^l = x_i,\ i = 1,2,\ldots,N_1, N_1 \tag{4}$$

$$z_i^l = \sigma\left(\sum_{j=0}^{N_{l-1}} w_{ij}^l z_j^{l-1}\right),\ i = 1,2,\ldots,N_l,\ l = 2,3,\ldots,L. \tag{5}$$

where σ is the activation function. The outputs of the NN are extracted from the output neurons, as in (6).

$$y_i = z_i^L,\ i = 1,2,\ldots,N_L,\ N_L = M \tag{6}$$

Backpropagation

The goal in developing neural models is to find an optimal set of weights w, so that $y=y(x,w)$ approaches the original behavior of a problem. This is achieved through the training process (that is, spatial optimization - w). A set of training data is presented to the neural network, consisting of pairs of (x_k,d_k), $k=1,2,\ldots,P$ where d_k is the desired output in the model for the inputs x_k and P is the total number of training examples.

During training, the performance of the neural network is evaluated by calculating the difference between your outputs and the desired output for all training examples. The difference, also known as error, could be quantified by (7).

$$E_{T_r}(w) = \frac{1}{2}\sum_{k\in T_r}\sum_{j=1}^{m}\left(y_j(x_k,w) - d_{jk}\right)^2 \tag{7}$$

where d_{jk} is the element j of d_k, $(y_j(x_k,w))$ is the output j of the neural network for the input x_k and T_r is an index of the training set. The weights w are adjusted during training so that this error is minimized. In 1986, Rumelhart, Hinton, and Williams proposed a systematic approach to NN training. One of the most significant contributions of their work is the backpropagation algorithm (Zhang & Devabhaktuni, 2003).

Optimization Methods

The term optimization refers to the search for minimum or maximum values for a given function, through a systematic choice of variable values within a viable set. It is intended, therefore, to find an optimal solution to a problem, which results in the best possible system performance.

Some of the most commonly used NN optimization methods are Stochastic Gradient Descent (SGD), Nesterov Accelerated Gradient (NAG), Adaptive Moment Estimation (Adam), and Root Mean Square Propagation (RMSprop). In the present study, the Stochastic Gradient Descent (SGD) was used because it leads to fast convergence and it doesn´t require memory storing of all the training data, which makes it appropriate for large datasets.

SGD

In the backpropagation algorithm, the gradient must be calculated in many iterations to adjust the weights of the neural network. When the training set is very large, in general, calculating the gradient for the whole set is impractical in terms of required computational resources. Considering this issue, the stochastic descending gradient is calculated with some examples iteratively (instead of the entire training base). Another advantage of using the SGD is the ability to reduce the occurrence of local minima in high dimensional error spaces. Batches are used because they reduce variance in learning and therefore have a more stable convergence, provided that the distribution of the examples is random. With the high computational power of GPU, mini-batches can also be processed quickly, since the operation is easily parallelized (Nascimento, 2016). Equation (8) shows the step of updating the SGD.

$$\theta = \theta - \alpha * \sum_{k=i}^{i+m} \nabla_{\theta} J\left(\theta; x^{(k)}, y^{(k)}\right) \tag{8}$$

where θ is the parameter to update, α is the learning rate, and *m* is the size of the batch.

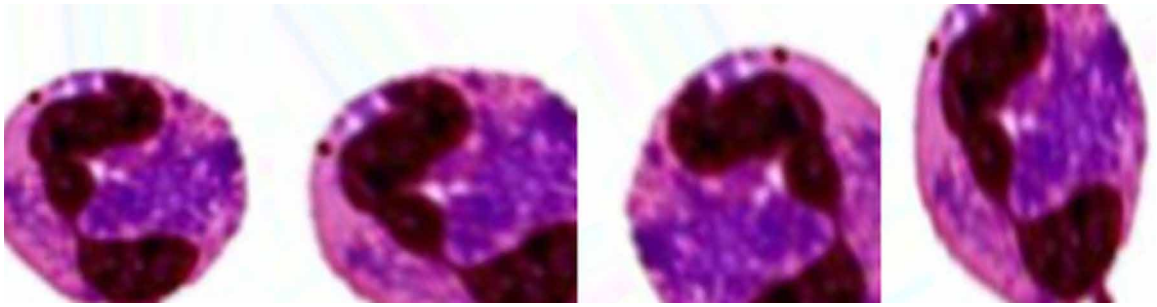
Data Augmentation

It is a fact that the larger and diverse the amount of data a machine learning algorithm has access to, the more effective it will tend to be.

Data augmentation is presented as a way to reduce overfitting in ML models, where data diversity is increased by using information that exists only in the available dataset.

A common practice to increase a set of images is to perform transformations at the level of their color and geometry, such as reflection, cropping, color palette changes, and image translation. Traditional transformation techniques (Figure 4) consist of using combinations of related transformations to manipulate training data. For each image, a "duplicate" image is generated that is shifted, enlarged/reduced, rotated, inverted, distorted or shaded with a tint. The image and its duplicate are then supplied to the neural network (Perez & Wang, 2017).

Figure 4. Four examples of morphological transformations performed on one cell image for the purpose of data augmentation



Fine-Tuning

Fine-tuning is a procedure based on the concept of transfer learning. It begins by training a network to learn characteristics for a broad domain with a classification function aimed at minimizing the error in that domain. Then, the classification function is replaced, and the network is optimized again to minimize the error in another more specific domain. Under this configuration, the characteristics and parameters of the broad domain network are transferred to a more specific one.

Assume a unique network in which the final classification function is the softmax (and a couple of dense layers), which calculates the probability in 1000 classes on the ImageNet dataset. To begin the fine-tuning procedure, this classifier component is removed, and another one is initialized with random values. The final classifier is then trained from scratch using the backpropagation algorithm with information relative to, for example, leukocyte classification.

In order to start the backpropagation for fine-tuning, it is necessary to define the learning rates of each layer appropriately. The classification layer requires a high learning rate due to its recent initialization. The rest of the layers require relatively small (or null) learning rates since it is intended to preserve the previous network parameters to transfer the knowledge to the new network (Reyes, Juan, & Camargo1, 2015).

One of the major difficulties in applying CNNs to cell images lies in the scarcity of labeled data. In this sense, the importance of the reuse of trained models in different tasks has already been demonstrated. However, the ability to transfer resources depends on the distance between the base task and the destination task. Booting a network with pre-trained resources improves its generalization capability (Kensert, Harrison, & Spjuth, 2018).

Convolutional Neural Networks

CNNs are useful in a large number of applications, especially those related to images, such as classification, segmentation, and object detection (Wu, 2017).

Let's say we have an image of a leucocyte in PNG format with size 480×480. The representative matrix will be 480×480×3. Each of these numbers receives a value from 0 to 255 that describes the pixel intensity at that point. The idea is to provide the computer with this array of numbers. With the output, one obtains the probability of the image belonging to a specific category (for example, 0.80 for neutrophil, 0.15 for monocyte, 0.05 for basophil). Similar to humans, the computer is able to perform image classification based on low-level features such as edges and curves, then constructing more abstract concepts through a series of convolutional layers. This is an overview of what a CNN does (Deshpande, 2016).

First Layer: Convolution

The first layer on CNN is always a convolutional layer. In this type of networks, a filter is used, which is in practice an array of numbers (weights or parameters) with the same depth of the input image, for example, 5×5×3 for 32×32×3 (RGB image).

The filter will slide, or convolve, around the input image, multiplying the values in the filter by the pixel values of the original image. A single representative number of each position of the filter is thus obtained throughout the convolution process, which gives rise to a new matrix 28×28, thus being smaller than the original matrix. If two 5×5×3 filters are used instead of one, the output volume will be

28×28×2. By using more filters, one can better preserve the spatial dimensions/directions. Each of these filters can be considered as identifying features such as borders, straight lines, simple colors and curves.

The output of the convolution layer is an activation map. The higher the value obtained, the greater the probability. The more filters you use, the greater the depth of the activation map, and consequently, more information about the input volume will exist.

For tensors of order 3, the convolution operation is done in a manner similar to that used for order 2 tensors. Assuming that the input in the i^{th} layer is a tensor of order 3 with size H^1*D, the convolution kernel will also be a 3-order tensor with size $H*W*D^l$. When the kernel is placed on top of the input tensor at the spatial location (0,0,0), the products of the corresponding elements are calculated on all the channels D^l, and summed to the products of $H*W*D^l$ to obtain the convolution result in this spatial location (Wu, 2017). Although there are other ways of treating tensors (channels), this is the most common.

Pooling

Pooling is a process of data reduction. In general, the maximum or mean of a small region of the input data is calculated. The main advantage of pooling is the speed increase in the network due to interlayer downsampling, reducing the amount of data to be processed and making "the curse of dimensionality" less intense (Nascimento, 2016).

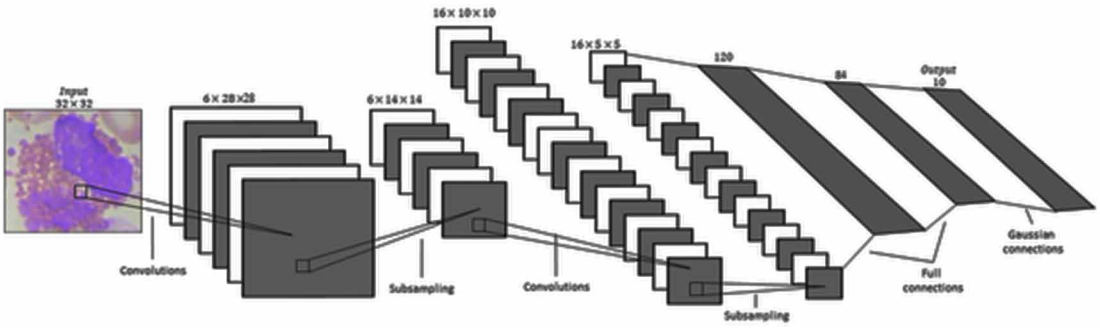
The purpose of pooling layers is also to achieve spatial invariance by reducing the resolution of feature maps. Each map corresponds to a characteristic map of the previous layer. Its units combine the input of small fragments(patches). These pooling windows can be of arbitrary size. The max-pooling consists in calculating the highest value present in a patch, normally, of four elements. Thus, those four elements are replaced by that value. The result is a lower-resolution feature map (Scherer, Muller, & Behnke, 2010).

Deeper Layers of the CNN

CNN are representatives of the multi-stage Hubel-Wiesel architecture, which extract local characteristics at high resolution and combine them successively into more complex features at lower resolutions. The loss of spatial information is offset by an increasing number of resource maps in the higher layers.

The general architecture is shown in Figure 5.

Figure 5. CNN architecture with convolution, pooling and fully-connected layers (Scherer, Muller, & Behnke, 2010)



Each entry layer describes locations in the original image where certain low-level features appear. When you apply a filter set to this information, the output will represent higher-level features. The types of these characteristics can be semicircles or squares. As you go through more layers of convolution, you get activation maps that represent increasing aggregations of complex resources (Deshpande, 2016).

Fully-Connected Layer

A fully-connected layer is useful at the end of a multi-layered CNN model. For example, if after multiple layers of convolution, ReLU, and pooling, the output of the current layer contains distributed representations for the input image; one wants to use all these features in the fully-connected layers to create the right responses at the final stage. A fully connected layer is useful for this purpose. Assuming that the input of a layer x^l has size $H^l*W^l*D^l$, if one uses convolution kernels with size $H^l*W^l*D^l$, then the D cores form a tensor of order 4 in $H^l*W^l*D^l*D$. The output is $y \in \mathbb{R}^D$. It is obvious that to calculate any element in y, one needs to use all elements in the input x^l. Therefore, this layer is a fully-connected layer but is implemented as the convolution layer it is, so it is not necessary to derive learning rules for a fully-connected layer separately (Wu, 2017).

IMPLEMENTATION

Frameworks, Programming Languages and Equipment Used

The Jupyter Notebook using Python and the Keras API (using the TensorFlow as backend) were used for the construction and implementation of the leukocyte classification project.

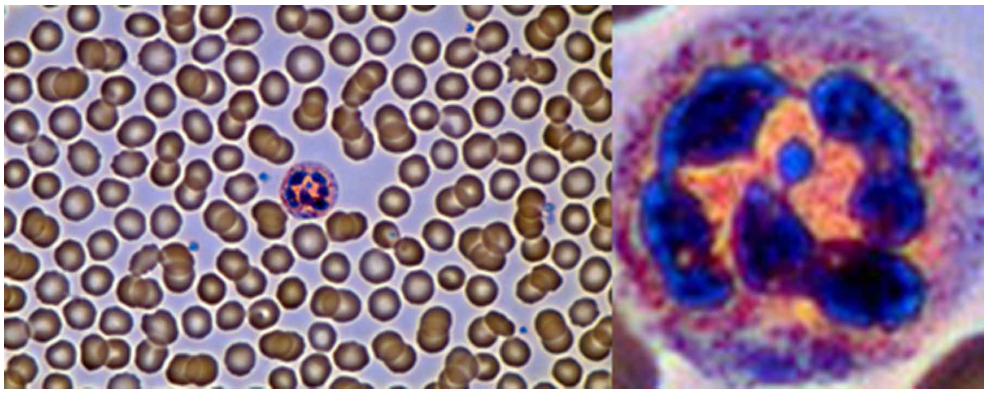
The machine used to implement the NNS has the following characteristics: Intel Core i7 5930K @ 3.7GHz with 32 GB RAM and 2 * GTX 1080 Ti 11 GB GPUs.

Data Pre-Processing

The dataset used initially in the development of the program contains 598 leukocytes (stained) images. Most of the images (approximately 352) come from a repository on GitHub (available at https://github.com/dhruvp/wbc-classification/tree/master/Original_Images). The remaining images were obtained through a search performed on Google according to specific terminology, namely *'monocyte', 'leucocyte'*, '*lymphocyte*' and *'neutrophil'*. Basophils were excluded because of the poor representativeness in the blood, which was reflected in the difficulty of finding images in amounts similar to those obtained for the remaining leukocytes. For similar reasons, band leukocytes were also excluded. The dimensions of the dataset images vary according to their provenance.

A big part of the focus of this work was the dataset preprocessing. It was necessary to label each of the collected images according to the 4 categories to be classified. Subsequently, each image was manually segmented in order to provide the NN with a register, as specific as possible, of the framing of the cell whose classification was intended. GIMP (GNU Image Manipulation Program), a popular image editor with several tools that facilitate graphics editing tasks was used to segment all images manually. Figure 6 exemplifies the manual segmentation process performed.

Figure 6. Eosinophil original image (900×588) and the same cell segmented (94×96)



Starting from this initial set of images, data augmentation was implemented, leading to a total of 10466 images distributed as follows:

- Eosinophils: 2610 training images, 215 validation and 215 test images;
- Lymphocytes: 2628 training images, 213 validation and 213 test images;
- Monocytes: 2604 training images, 211 validation and 211 test images;
- Neutrophils: 2624 training images, 218 validation and 218 test images;

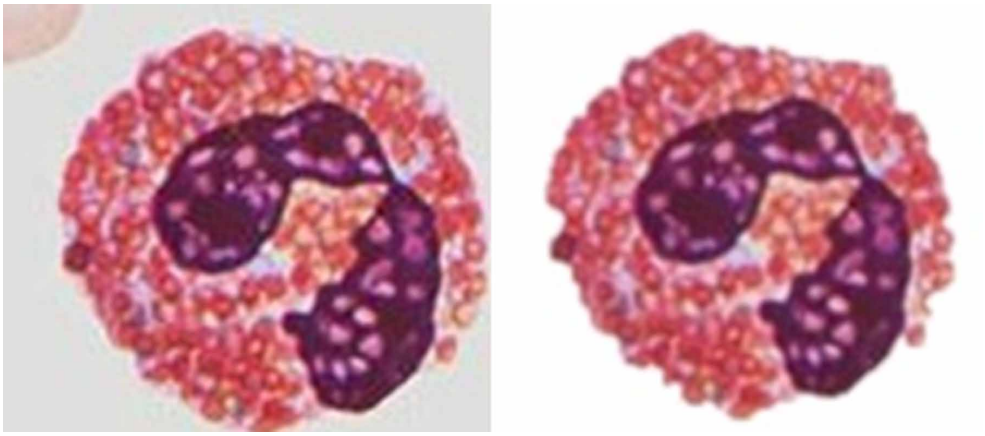
The data augmentation consisted essentially of morphological transformations such as:

- Random rotations up to a limit of 20°;
- Completion of points outside the input limits (image border padding), according to the selected mode (constant, near, reflect, and wrap). The selection of the mode in Keras translates into the extension of the limits of the morphology (color and shape) of the original image to the limits of the new image, that is, the maintenance of the color and shape of the boundary objects;
- Variations in the input area captured in the longitudinal and transverse axes by up to 10%, which means that the image may have been moved according to these axes;
- Inversions of the image in the horizontal and vertical axes;
- Changes in color.

The application of these transformations (exemplified in Figure 4) allowed the passage of the 598 initial images to the 10466 used in the training, validation, and test of the neural networks.

Due to the unsatisfactory results obtained after the training of the different networks, it was necessary to re-classify and segment all the leukocytes images as the leukocytes regions on the previous dataset were not segmented in a precise way. In this sense, the images were separated and categorized and all the images that raised doubts were eliminated and others were added, which allowed labeling with a higher level of confidence. A new segmentation was made using Paint 3D (the latest version of the popular paint editor usually found on Windows 10). This software allowed the creation of three-dimensional models and allowed the separation of the cell's foreground from the rest of the image background, as can be seen in Figure 7.

Figure 7. Original segmented eosinophil image (left) and the same image segmented with Paint3D, with the background eliminated (right)



From this second selection and segmentation procedure were obtained 100 training images, 10 validation images, and 10 test images, for each of the 4 categories. After a new process of data augmentation, 4394 training images, 440 validation images and 440 test images were distributed uniformly across the four categories.

NN Models Used

Keras offers some deep learning models under the topology of the application on its documentation, as well as some weights (pre-trained in the Imagenet database) capable of classifying up to 1000 categories. In the list of available models are: Xception, VGG16, VGG19, ResNet50, InceptionV3, Inception-ResNetV2, MobileNet, DenseNet, NASNet and MobileNetV2.

Next, it is presented a brief description of the models used in this project.

VGG16 and VGG19

The second place in the ILSVRC 2014 competition was given to the VGGNet, developed by Simonyan and Zisserman. The VGGNet consists of 16 convolutional layers and presents a very uniform architecture. Similar to the AlexNet, only with 3×3 convolutions, but many filters. It was trained in 4 GPUs for 2 to 3 weeks. It is currently one of the preferred choices for extracting features from images. The weight configuration of the VGGNet is publicly available and has been used in many applications and challenges as a base resource extractor (Das, 2018).

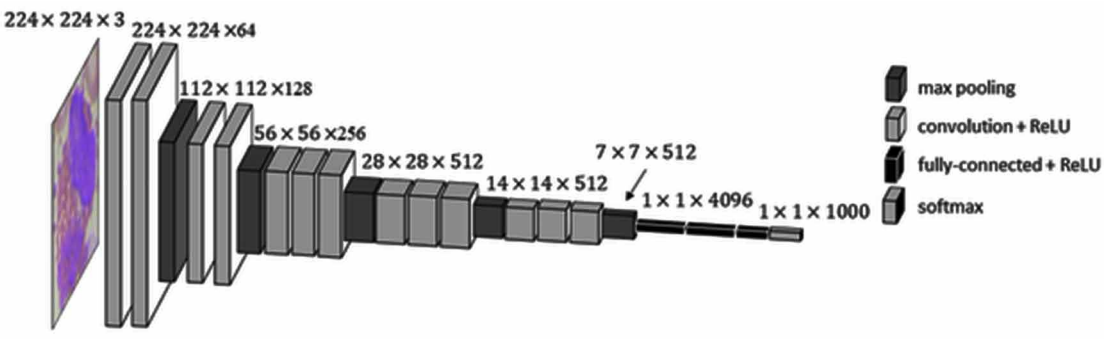
The input for the ConvNets VGG (Visual Geometry Group), shown in Figure 8, are RGB images with dimensions 224×224. The only pre-processing performed during its implementation in ILSVRC was the subtraction of the average of the RGB value calculated in the training dataset in each pixel. The image is then passed through a set of convolutional layers, where filters with a very small receiver field are used: 3×3 (which is the smallest that allows capturing notions left/right, up/down, and center). In one of the configurations, filters of 1×1 were also used, which can be seen as linear transformations of the input channels (followed by non-linearities). The convolution step is 1 pixel; the spatial fill of the convolution

layer is such that it allows the resolution to be preserved after the convolution, for example, the fill is 1 pixel for convolution layers with 3*3. Spatial grouping is accomplished with 5 layers of max-pooling, which follow some of the convolution layers. Max-pooling is done with windows of 2×2and step of 2.

The set of convolution layers (which have different depths depending on the architecture) is followed by three fully-connected layers: the first two contain 4096 channels each, the third classifies the 1000 classes (with one channel for each class). The final layer is a softmax layer. The configuration of the fully-connected layers was the same in all networks tested.

All concealed layers were equipped with the ReLU activation function (Simonyan & Zisserman, 2014).

Figure 8. Visualization of the VGG general architecture (Rosebrock, 2017)



The numbers 16 and 19 represent the number of layers with weights in the network. Some disadvantages of VGGNet are the slowness of training and the width of the weight architecture. Due to its depth and number of fully-connected nodes, VGG16 is more than 533MB, and VGG19 is more than 574MB, which makes deploying these networks complicated.

InceptionV3

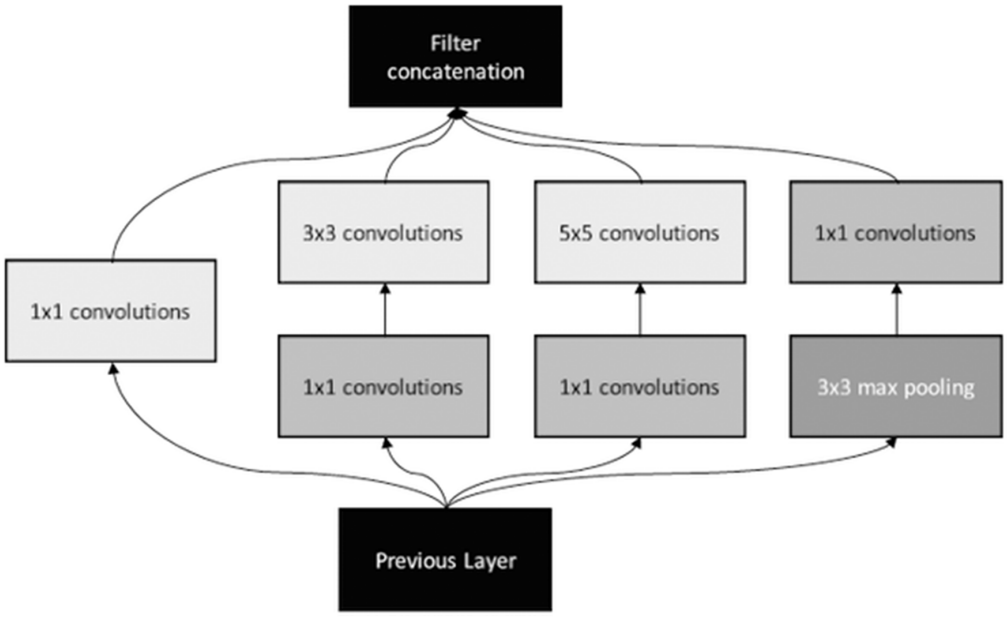
The winner of the ILSVRC 2014 contest was GoogLeNet (Inception V1) from Google. It achieved a top-5 error rate of 6.67%, very close to the human-level performance, also rated by the organizers.

The network used a LeNet-inspired CNN but implemented a new element called the Inception module (represented in Figure 9). It used batch normalization, image distortions and optimizer RMSprop. Its architecture consisted of a deep 22-layer CNN but reduced the number of parameters from 60 million (from AlexNet) to 4 million (Das, 2018).

The use of average pooling allows for easy adaptation and fine-tuning for other labels. The authors came to the conclusion that switching from fully-connected layers to average pooling layers improved the top-1 results by about 0.61%, with the use of dropout.

Given the large size of the network, the ability to propagate the gradient across all layers was a concern. Adding intermediate layered classifiers enabled discrimination at lower stages of the classifier, by increasing the propagated gradient signal back and providing additional regularization. These classifiers are in the form of small convolutional networks placed at the top of the Inception modules (additional pooling paths). During training, the loss is added to the total loss of the network with a weight discount. At the inference, these auxiliary networks are discarded (Szegedy, et al., 2014).

Figure 9. Original inception module used on GoogLeNet (Szegedy, et al., 2014)



The Inception V3 architecture included in Keras comes from the paper *Rethinking the Inception Architecture for Computer Vision* from 2015. The size (or memory space required) of Inception V3 is lower to VGG and ResNet, leaving it at 96MB (Rosebrock, 2017).

Xception

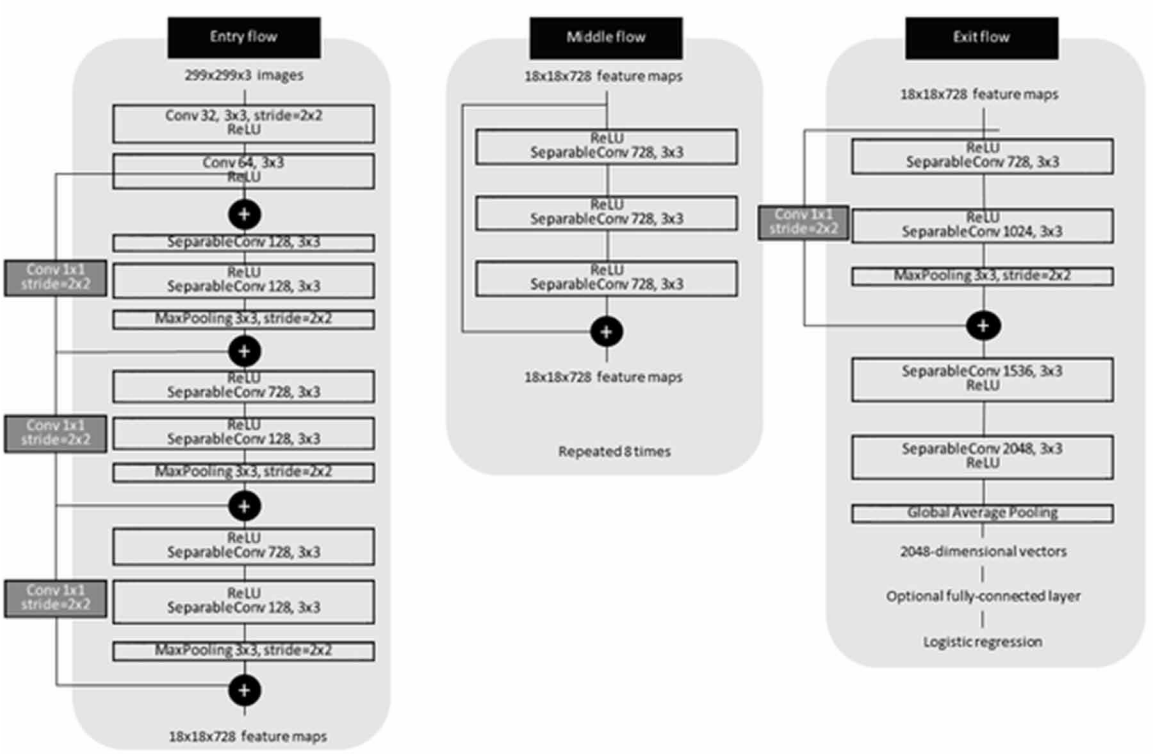
The Xception was proposed by the creator of Keras, François Chollet, and is an extension of the Inception architecture that replaces the Inception modules with separable convolutions by depth, Figure 10.

ResNet50

In ILSVRC 2015, the Residual Neural Network (ResNet), by Kaiming He et al., presented a new architecture with non-consecutive connections (skipped connections) and the strong use of batch normalization. Ignored connections are also known as blocked units or gated recurrent units and have a strong resemblance to recently applied success factors in RNNs. Thanks to this technique, it is possible to train a NN with 152 layers, although with complexity inferior to a VGGNet. ResNet achieves a 3.57% Top-5 error rate, surpassing the human performance level (He, et al., 2015).

The authors of the ResNet were inspired by VGG networks. The convolution layers used have mainly 3×3 filters and follow two simple rules: (i) for the same map size of features, the layers have the same number of filters and (ii) if the filter map has its size reduced by half, the number of filters doubles in order to preserve the temporal complexity per layer. Downsampling is performed directly by the 2-pixel step convolutional layers, and the network terminates with a global mean pooling fully-connected with softmax.

Figure 10. Xception architecture. The data enters the entry flow and goes by the middle flow which is repeated eight times, and lastly by the exit flow (Chollet, 2016)



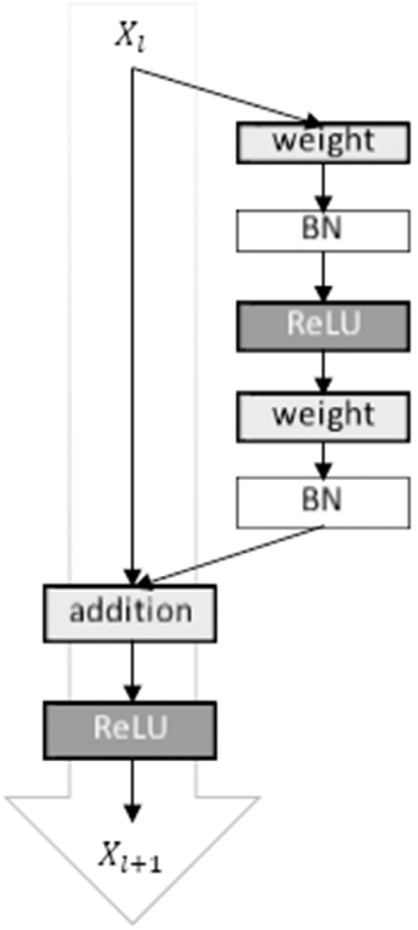
This model has fewer filters and complexity than VGG networks. A 34-layer network is equivalent to 3.6 billion FLOPs, which represents 18% VGG-19 (19.6 billion).

Based on the previous base network, the authors inserted shortcut links by transforming this network into its residual counterpart version. The identity shortcuts can be used directly when the input and output are of the same dimensions (represented with a solid trace in Figure 11).

When the dimensions increase, two options are considered: (A) The shortcut binding performs identity mapping, with zero fill entries to increase the dimensions. This option does not induce any extra parameters; (B) The projection shortcut link is used to match the dimensions (with convolutions 1×1). For both options, when shortcut links traverse feature-sized maps of two sizes, they are made with a step size of 2 (He et al., 2015).

So, the ResNet has an architecture based on micro-architecture modules (network-in-network architectures). This network has demonstrated that it is possible to train extremely deep networks using standard SGD through the use of residual modules. The implementation of ResNet50 in Keras is based on the original article of 2015. Although ResNet is much deeper than VGG, the model size is substantially lower (102MB) because of the greater abundance of pooling rather than layers fully-connected (Rosebrock, 2017).

Figure 11. ResNet residual module as originally proposed (Rosebrock, 2017)



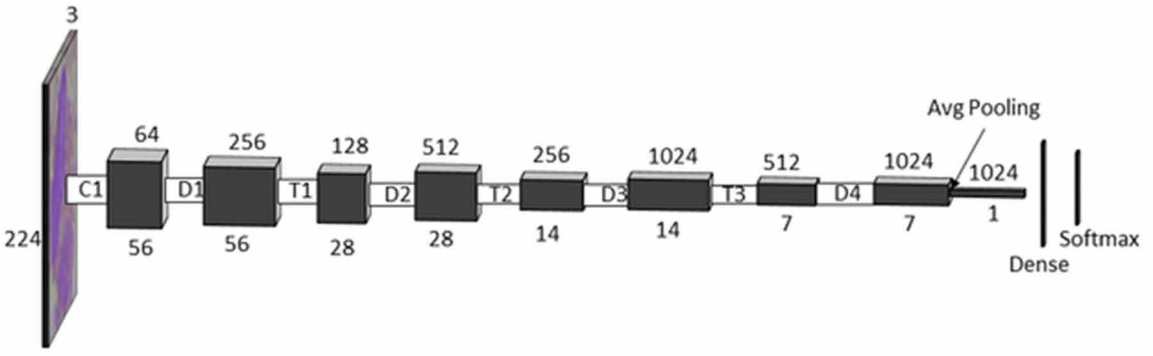
Densenet 121, 169 and 201

Generally, the ResNet architecture has a fundamental block structure, and in each of them, an anterior layer with a future layer is attached. In contrast, the DenseNet architecture proposes to concatenate the outputs of previous layers in a block.

The DenseNet architecture explicitly differentiates between information added to the network and preserved information. DenseNet layers are very narrow (for example, 12 feature-maps per layer), adding only a small set of resource maps to the "collective knowledge" of the network, keeping the resource maps unchanged and the final classifier makes a decision based on all network resource maps.

One of the great advantages of the DenseNet (Figure 12) is the improved flow of information and gradients across the network, which facilitates training. Each layer has direct access to the gradients of the loss function and the original input signal. This helps train deeper network architectures (Huang, Liu, Maaten, & Weinberger, 2016).

Figure 12. Over-simplified example of a DenseNet-121. Measures under each volume represent the width and depth and the numbers on the top of the feature map dimension. D represents dense blocks that perform operations with dense layers, T represents transition blocks performing 1x1 and 2x2 (with a stride of 2) convolutions (Ruiz, 2018).



RESULTS

Implementing a model in a new set of data is a highly complex task whose evaluation depends on the innumerable factors inherent in the type, data source (as well as the level of pre-processing of data), the network architecture used, the hardware and frameworks used, among others. Some metrics not evaluated in this work are: transfer rate, utilization of GPU computing, use of FP32, CPU usage and memory consumption (Zhu, et al., 2016).

In the classification of leucocytes intended in this work are used open-source implementations available on the Internet and provided by its developers and, in this case, mediated by the Keras library.

The analysis of the project results mainly focuses on the accuracy and loss of training and validation of a set of models subsequently tested in the classification of four categories of leukocytes based on microscopic images. In the test dataset, which has different training and validation images, the percentage of the correctness of each of the models used (with and without transfer learning) is given, taking into account the time required to perform the training.

For the value of the loss, the lower it is, the better the model in question (unless it is overfitting on the training dataset). The loss is calculated during the training and validation processes, and its interpretation tells us how good the performance in these datasets is. Unlike accuracy, a loss is not a percentage value, but the sum of the errors for each training example (using the term cost for a set of examples or batch) in the training and validation sets.

In the case of the neural networks, the loss is usually given by the negative logarithm of the probability (softmax) of obtaining the number of categories. The main purpose of the model implementation is to reduce or minimize the loss function by changing the network weights using different optimization methods. The value of the loss gives an idea of the behavior of the model after each optimization iteration. Ideally, a reduction in this value should be expected after one or several iterations.

Although accuracy can be verified after each iteration, in the case of training, this value is only a reference for the correctness percentage of the model. The final precision value, obtained after learning all the parameters with the conclusion of the training, is the one that should be taken as correct. After

the training, the test images are supplied to the network and the network forecasts are compared with the actual image labels, and the error percentage is usually calculated.

In this work, the comparative study was carried out between some models while training using transfer learning and without using this technique, that is, training or not the networks from scratch in the classification of leukocytes.

Models Trained with Transfer Learning

In the case of the training with transfer learning, the original top of the networks was excluded. That way it is possible to use the weights from previous training with Imagenet, that is, the knowledge of some characteristics of objects applicable to the identification of leukocytes. A small network with three layers was then created (the new top), a dense layer with the ReLU activation, a dropout layer and an output layer with the function softmax for 4 categories. The code needed to program this network in Keras is as follows:

```
model = models.Sequential()
model.add(layers.Dense(256, activation='relu', input_dim=7*7*512))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(4, activation='softmax'))
```

The network received the weights related to pre-training features and was then trained to classify leukocytes in 4 categories. The initial training lasted 100 epochs, and a batch size of 16 examples was used. The average training time of the 8 NN (*'resnet50', 'vgg16', 'vgg19', 'inceptionV3', 'xception', 'densenet121', 'densenet169', 'densenet201'*) for 100 epochs was 9 minutes and 59 seconds on the computational architecture referred previously. The fastest network, VGG19, completed the training in 5 minutes and 47 seconds, and the slowest, Xception, took 16 minutes and 57 seconds. The results obtained in terms of loss are presented in the graph of Figure 13.

The networks that achieved the best results with lower loss values were the Xception, the Densenet201, the Densenet169, the Densenet121 and InceptionV3. The network that converged faster to (approximately) 0 was the Xception. The worst results were obtained by the VGG and the Resnet50 networks.

Although the networks were initially presented by their authors to operate with certain optimizers, in this work, all the training used the SGD optimizer, for comparing purposes (with the conscient possibility of penalizing the performance of some of the networks used), with a learning rate of 1^{-5} and decay of 0.001.

Figure 14 shows that the accuracy is closely related to loss, and the models that obtained the best accuracies were the same ones that had obtained the losses with lower values, logically. Thus, the Xception network was the one that converged faster to 1 (with possible overfitting), and the ResNet50 was the one that obtained the worst results with a training accuracy of about 40%. The poor ResNet results may be related to the use of SGD.

The results for the validation process are close to the ones obtained during the training, with the losses being slightly lower (and the accuracies slightly higher) on the training accuracies as expected.

In the case of transfer learning, the ResNet50 would be expected to perform better, because it is one of the newer networks, with an error of only 3.57% in ILSVRC'15. However, under the established training conditions, it was not able to track the networks with better results.

Figure 13. Loss registered on the training dataset by the set of models trained with transfer learning for 100 epochs

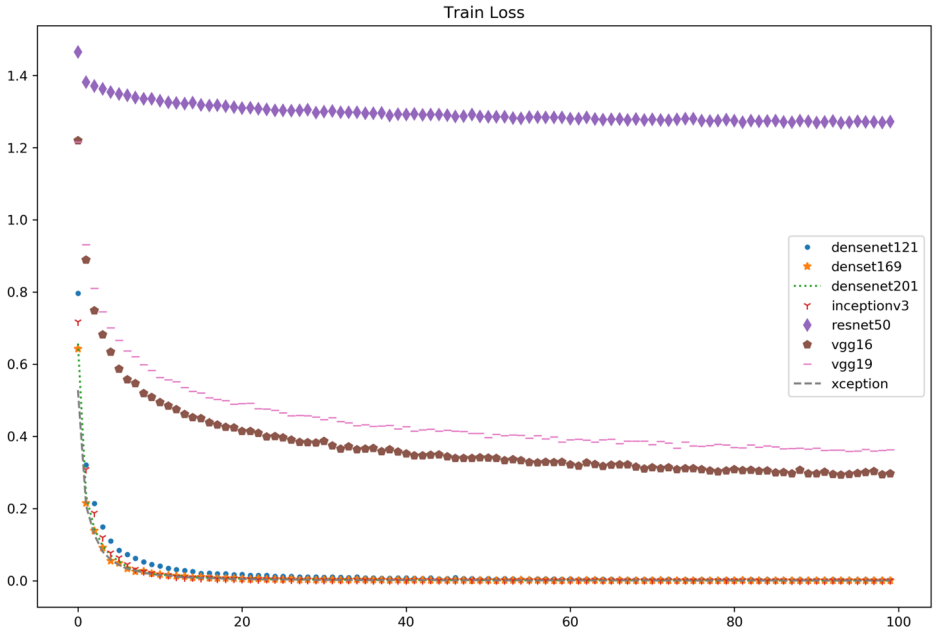
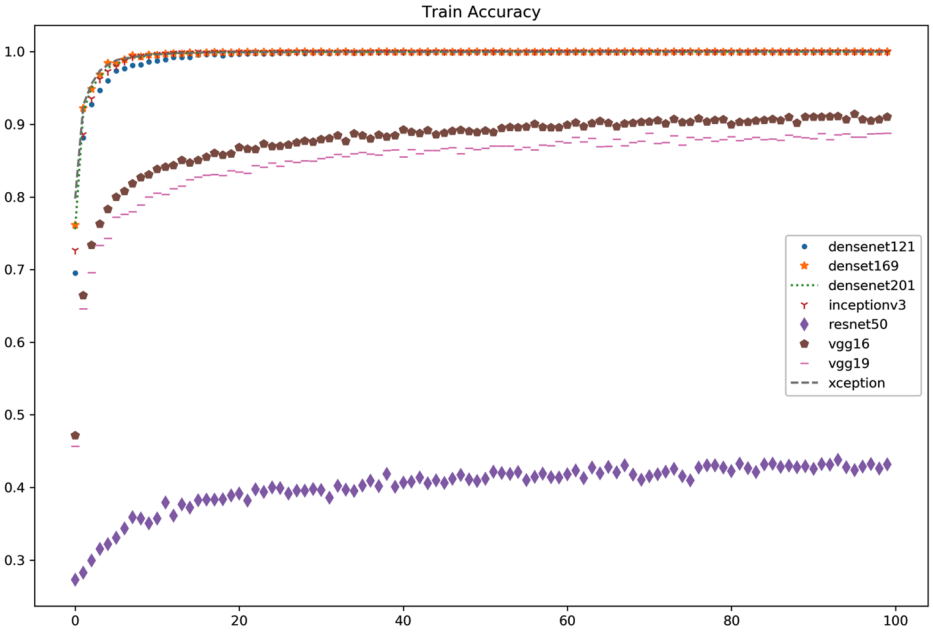


Figure 14. Accuracy registered on the training dataset by the set of models trained with transfer learning for 100 epochs



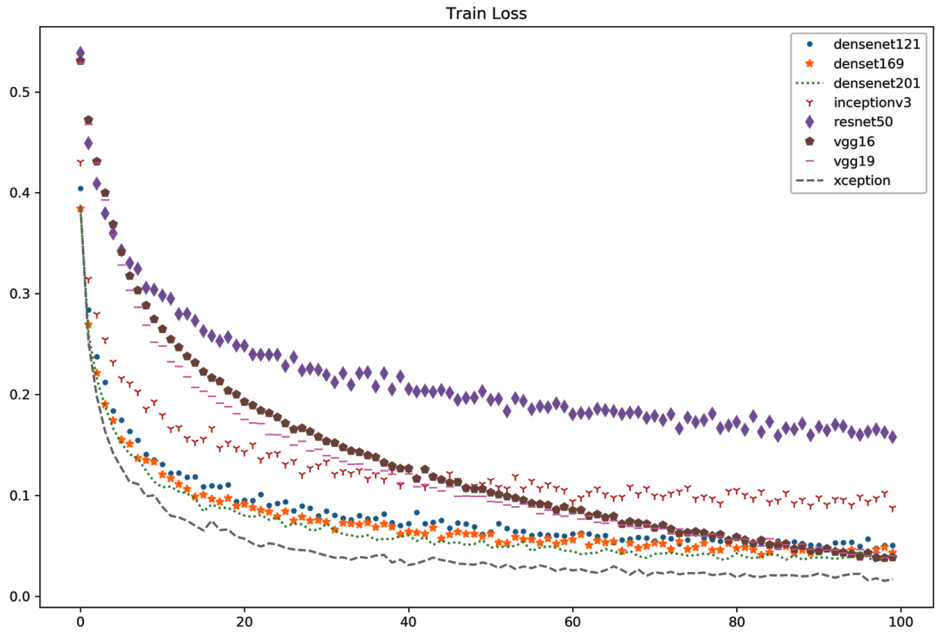
In general, the percentage of correctness for each of the categories has similar values, with the exception of the ResNet50, where some discrepancy in the classification process can be verified.

Models Trained From Scratch

In the models trained from scratch, it was necessary to train the entire network, being indispensable the specification of the number of classes or categories. The fact that it is necessary to train the whole network makes this training much more time consuming than training with transfer learning. Thus, the training of the eight networks took about 18 hours on the computational architecture used (referred previously). Each network took an average of 2 hours and 14 minutes to train for 100 epochs. The fastest network was VGG16 with 1 hour and 15 minutes and the slowest was the Xception with 3 hours and 24 minutes. Figure 15 shows the behavior of the models in terms of losses.

As shown in Figure 15, the models that obtained lower loss values were the Xception and the Densenet201, with ResNet50 having the highest loss.

Figure 15. Loss registered on the training dataset by the set of models trained from scratch for 100 epochs



The training accuracies reflect the losses previously presented with results, similar to the losses and accuracies in the validation dataset.

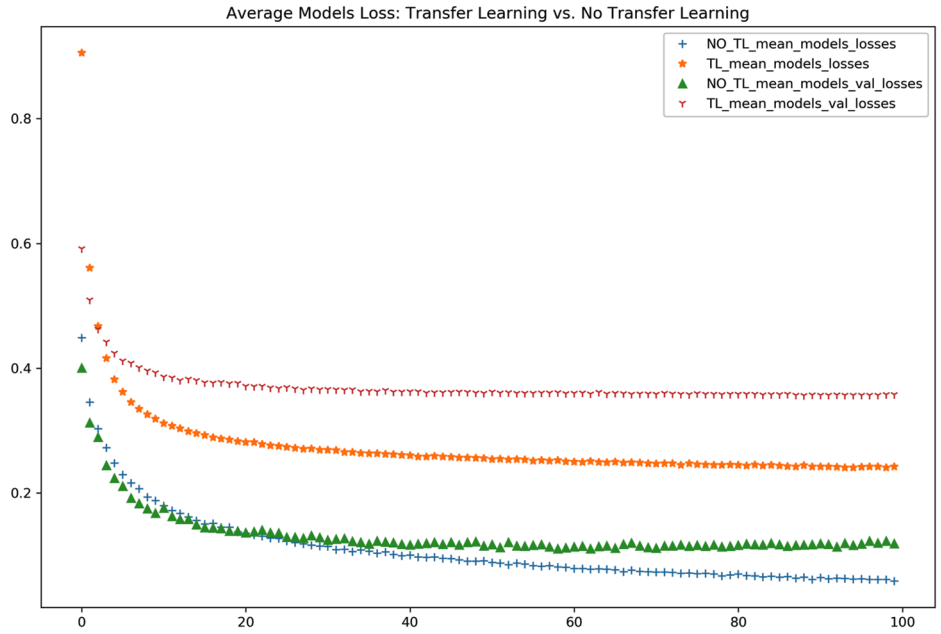
Some variation in the hit percentages in each of the categories indicating that the models are not properly trained, given the number of training examples for each category being very similar.

Models Trained With Transfer Learning vs. Models Trained From Scratch

In order to get a general idea of the performance of the set of trained models with transfer learning and models trained from scratch, the graph of Figure 16 shows the mean of the values of loss obtained by the set of models in training and validation datasets.

This graph reveals lower values of the loss values of the models trained from scratch when compared to the models with transfer learning, implying a better performance in both the training and validation dataset. The values of loss for models trained without transfer learning are, after 100 epochs, below 0.15, converging to 0 in the case of the training dataset. The models with transfer learning have mean values of loss of about 0.4 for the validation dataset and about 0.3 for the training dataset.

Figure 16. Average training and validation losses obtained with the set of models trained for 100 epochs from scratch and with transfer learning



Higher accuracies were also obtained with the models trained from scratch, with values above 95% in both training and validation sets. The models trained with transfer learning got results below 90% in the two datasets referred previously.

The reduction of the value of the loss is associated with several factors. The fact that the results show very low values for this metric may be indicative of the existence of overfitting, meaning that the model "memorized" the training examples, with the possibility of having become inefficient in the test dataset. Overfitting also happens when regularization is not applied in very complex models (where the number of parameters W, the set of weights is large) or the number of data N (number of examples) is too low. In the case of models trained from scratch, no regularization method was used because there were no

modifications made to the models supplied by Keras API. In the models with transfer learning, a layer of dropout (dropout = 0.5) was used at the network's top.

Table 1 shows the results obtained by the models trained from scratch. At the end of the training, these models were tested in a dataset with images different from those images used during the training and validation processes. From its observation, it is possible to verify that the percentage of correctness is very low. On average, the set of models trained from scratch achieve an accuracy of 26.31% for the 4 categories of leukocytes, which means that the process has results similar to a random classification.

Table 1. Classification accuracy percentages on the test dataset for each of the neural networks trained from scratch for 100 epochs, on four types of leukocytes

	Eosinophils	**Lymphocytes**	**Monocytes**	**Neutrophils**
Densenet121	19,09	18,18	29,09	27,27
Densenet121	20,91	29,09	20,00	25,45
Densenet121	24,55	24,55	22,73	28,18
InceptionV3	25,45	19,09	22,73	19,09
Resnet50	20,91	24,55	26,36	20,91
VGG16	26,36	23,64	21,82	27,27
VGG19	26,36	24,55	21,82	28,18
Xception	25,45	26,00	30,00	30,91

From this, it is possible to deduce the high probability of overfitting during training since the training and validation results are very good, but the models cannot perform with high precision classifications (or accuracy) in images different from those that were used in training.

Table 2 shows the results obtained by the models with transfer learning in the test dataset. Some models, such as the DenseNet, achieved average hit percentages (in all 4 categories) above 97%. The worst-performing model was the ResNet50 with only 46.4% hit. This value is due to the choice of the optimizer, the need for regularization, or the need for a higher number of training emphases. The networks

Table 2. Classification accuracy percentages on the test dataset for each of the neural networks trained with transfer learning for 100 epochs, on four types of leukocytes

	Eosinophils	**Lymphocytes**	**Monocytes**	**Neutrophils**
Densenet121	96,36	98,18	99,09	98,18
Densenet121	96,36	99,09	98,18	99,09
Densenet121	97,27	99,09	98,18	98,18
InceptionV3	91,82	97,27	93,64	97,27
Resnet50	34,18	82,73	40,91	23,64
VGG16	78,18	86,36	86,36	90,00
VGG19	81,82	90,00	80,00	88,18
Xception	95,45	97,27	96,36	97,27

VGG16 and VGG19 obtained results of approximately 85% on 100 epochs of training. InceptionV3 and Xception achieved 95% and 96.6% respectively.

From the observation of the obtained results, it is possible to realize that in spite of the initial indication that the models trained from scratch would obtain better performance (based on the best values of loss obtained during the training) that didn't happen in the test dataset. This can be explained by the large complexity of the models compared to the dataset images diversity. In addition to much less time-consuming workouts, the models trained with transfer learning were more successful being able to perform more correct ranks in the test dataset, with an overall hit rate of 87.8%, compared to 26.3% of the models trained from scratch.

Training of Models With Transfer Learning for 5000 Epochs

In an attempt to improve the performances of the models with transfer learning in the test dataset, new training of the same models was done for 5000 epochs. Given that the percentage of correctness was already quite high in some models, it was verified that the training increased the percentage of the correctness of these networks slightly. The training of all models for 5000 epochs took 69 hours and 2 minutes (while for the 100 epochs, all models took 1 hour and 20 minutes). The network that took less time to train was the VGG16 with 4 hours and 56 minutes and the one that took the most time to train was the Xception with 14 hours and 32 minutes.

Overall, the mean increase in the percentage of network set-up between the set of trained models for 100 and 5000 epochs was 1.68%, going from 87.81% in 100 training epochs to 89.49% in 5000 epochs of training.

The networks that had already achieved hit percentages above 96% obtained minor improvements, although they should not be neglected. The DenseNet121 achieved slightly lower results than those obtained after training for 100 epochs.

The networks that most benefited from the increase in the number of training epochs were the VGG16, the VGG19, and the ResNet50 with increases between 3% and 5%, although the ResNet50 maintained very low hit percentages. There was a tendency to increase the validation loss in most of the models, which suggests that the models are performing overfitting, remarkable also for the high accuracy of training (very close to 1). One could use a higher dropout value, increase decay and decrease the learning rate as a way to prevent overfitting.

Comparably to what verified in training by 100 epochs, that in general, the hit percentage for each of the categories has similar values, with the exception of the ResNet50 where one can see some discrepancy in the classification process.

In general, given the simplicity of the methodology used, the results are quite encouraging, even when compared with similar recent works. In 2016 Sajjad *et al.* created a mobile framework in the cloud to segment and classify leukocytes and achieved a 98.6% hit percentage using k-means algorithms to segment images and morphological operations to remove unwanted components. For the classification, they used an EMC-SVM (Sajjad, Khan, & Jan, 2016).

In 2018 Habidzadeh *et al.* used the ResNet50 V1, ResNet152 V1, and Resnet101 networks, trained by 3000 epochs, with 11200 training examples and 1244 test samples, having obtained results above 99% for 5 categories of leukocytes (Habibzadeh, Jannesari, Rezaei, & Totonchi, 2018).

Also, in 2018, Bhadauria *et al.* classified four categories of leukocytes, with an accuracy of 95% (Bhadauria, Devgun, Virmani, & Rawat, 2018).

However, it is not possible to make reliable comparisons between this and other works because the datasets used in this are not the same as the ones used in the other authors referred works, and some may have more visual quality than others.

FUTURE RESEARCH DIRECTIONS

Among some interesting proposals in terms of neural networks, in the last couple of years, the following articles are highlighted:

- *Identity mappings in deep Residual Networks* (He, et al., 2016) proposes an improvement of ResNet's block design. A more direct path has been created to propagate information across the network (it moves the activation to the residual path) performance.
- *Aggregated Residual Transformations for Deep Neural Networks* (RestNExt) (Xie, et al., 2017), from the creators of ResNet, presents an increase in the width of the residual blocks through of multiple parallel pathways ("cardinality") similar to the Inception modules.
- *Deep Networks with Stochastic Depth* (Huang, et al., 2016) aims to reduce the vanishing gradients and the training time through the use of short networks during the train. They do not use sets of layers, chosen at random, during a training pass. They bypass the identity function but use the network complete at the time of testing.
- *FractalNet: Ultra-Deep Neural networks without Residuals* (Larsson, et al., 2017) args that the key to good performance 'and the ability to make the transition between different network formats. They use fractal architectures that have little and a lot of depths up to the output. They are trained to omit some paths. The complete network is used during the test.
- *SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size* (Iandola, et al., 2016) presents modules that consist in a 'squeeze' layer with 1×1 and 3×3 filters. The authors are able to accuracy at the level of an AlexNet in ImageNet with 50× fewer parameters. The network is compressed to a size 510× smaller than Alexnet (0.5 MB).

In the domain of cell identification and classification and microorganisms in general, it is expected to see an increase of AI techniques applied in hospitals and laboratories. In this specific case, the use of technology is dependent on the way the microscopy images are obtained. So, one hopes that this side of the equation sees an equal technological improvement, as the quality of the results depends directly on the quality of the images acquired, the speed of acquisition (as well as the efficiency of the pre-processing techniques).

CONCLUSION

The primary purpose of this document is the description of the implementation project of a set of neural networks in the classification of white blood cells or leukocytes. Open-source implementations were used with the Keras framework, with most implementations being available from ILSVRC winning networks. The Keras models used were the ResNet50, the VGG16, the VGG19, the InceptionV3, the Xception, the Densenet121, the Densenet169 and the Densenet201.

All the models were trained in two different ways: using transfer learning and training from scratch. Transfer learning allows transferring the values of the weights acquired in a previous training (in this case, with 1000 categories in the ImageNet) being necessary to train only the top of the network to recognize specific characteristics and set the number of categories to sort.

Training from scratch involves training the entire network, which is usually initialized at random and is a significantly more extended process.

The dataset is usually preprocessed and, in the case of this work, consisted in the selection, classification and segmentation of images containing eosinophils, lymphocytes, monocytes and neutrophils - basophils and band leucocytes were excluded due to the difficulty in finding images in equivalent number to the other categories. This process of preprocessing is also subject to human error, but it is the starting point for the automated classification performed by neural networks. The data augmentation was done to move from 110 images in each category to a total of 4394 training images, 440 validation, and 440 test images.

The training was performed for 100 epochs for both training approaches. In spite of the high accuracy and low loss of the models trained from scratch, it was found that these networks were overfitting, not achieving test scores with an accuracy above the random result to the number of categories concerned.

The networks with transfer learning have, on the whole, achieved classification accuracies of 87.8% versus 26.3% of the models trained from scratch. The best performing networks were the DenseNet169 and the DenseNet201, both with 98.2% accuracy. In addition, the average training time of the models with transfer learning was 9 minutes and 59 seconds, while, on average, it was needed 2 hours and 14 minutes to train a model from scratch.

The solution for the improvement of the results in the models trained from scratch should pass by the use of regularization, which introduces slight modifications in the learning algorithm in order to allow a better capacity of generalization, which translates into an improvement of the performance of the network. Some common regularization techniques are L2 and L1 regularization, dropout and early stopping. In spite of this issue, another approach would be formed by the usage of a larger and more diverse dataset.

Considering that it was intended to compare the performances with and without transfer learning, no changes were made to networks trained from scratch.

In the attempt to improve the performance of the networks and at the same time, perceive the effect given by the number of epochs on the functional performance, the training of the models with transfer learning was carried out for 5000 epochs. The results show an increase of 1.68%, on average, in the percentage of the correctness of these networks, when tested. There has also been a growing tendency for overfitting due to the scarce number of training examples used for a significant increase in the number of training epochs. Also, in this case, it would be advisable to use regularization techniques or a more suited dataset.

The good results obtained with some deep learning models are directly related to the pre-processing work done in the images. The absence of background in the images allowed the networks to extract more easily the pertinent features for the classification process. During the first preprocessing or segmentation process, there was only an approximate area containing the cell to be classified, and the test results in the test dataset were significantly lower than those obtained after leukocyte 'isolation' of the background. In this sense, it is understood the importance of data processing before providing it to neural networks for training (as well as for the test phase).

The analysis of the results must be made taking into account the limitations, mainly in the number of images used. The use of a dataset composed of more cell images (with an increase of two or three orders of magnitude) would be extremely important as a way of corroborating with greater confidence

the results obtained and would certainly allow the development with more practical applicability (because only 4 categories were used).

Although there is some variability in the morphology of the images, the segmentation carried out caused that the different lighting conditions, and microscopy did not weigh too much on the final result. The existence of images from different sources was a way to increase the generalization power of the networks.

With this work, it was possible to implement a set of networks in the classification of 4 categories of leukocytes with percentages above 98% in the available dataset. It is highly probable that with a significantly larger set of images and the coupling of a computational mechanism for identification and segmentation of images (and this could also be a neural network) it could be possible to automate the identification and counting of all types of leukocytes existing from microscopy blood cell images.

Compared to state of the art, this work has higher levels of accuracy (98%). It should, however, be emphasized that the dataset used is not the same as that used in works leading to state of the art, so it is risky to make comparisons. In addition, the leukocytes used belong to the set of five base leukocytes. On the other hand, in some studies, leukocyte detection and classification are performed simultaneously, which may reduce the accuracy of classification. In the present work, we tested the performance of CNNs in relation to the classification process, but the detection was performed manually. Future work will analyze the performance of dedicated leukocyte detectors. With the use of these two methods, detection and classification, as they are separately trained methods, it is expected to achieve higher performances than achieved with hybrid methods.

REFERENCES

Agaian, S., Madhukar, M., & Chronopoulos, A. (2014). Automated Screening System for Acute Myelogenous Leukemia Detection in Blood Microscopic Images. *IEEE Systems Journal*, *8*(3), 995–1004. doi:10.1109/JSYST.2014.2308452

Alférez, S., Merino, A., Bigorra, L., & Rodellar, J. (2016). Characterization and automatic screening of reactive and abnormal neoplastic B lymphoid cells from peripheral blood. *International Journal of Laboratory Hematology*, *38*(2), 209–219. doi:10.1111/ijlh.12473 PMID:26995648

Bhadauria, H. S., Devgun, J. S., Virmani, J., & Rawat, J. (2018, January). Application of ensemble artificial neural network for the classification of white blood cells using microscopic blood images. *International Journal of Computational Systems Engineering*, 202–216.

Bush, P., & Sejnowski, T. (1996). Inhibition synchronizes sparsely connected cortical neurons within and between columns in realistic network models. *Journal of Computational Neuroscience*, *3*(2), 91–110. doi:10.1007/BF00160806 PMID:8840227

Chaira, T. (2014). Accurate segmentation of leukocyte in blood cell images using Atanassov's intuitionistic fuzzy and interval Type II fuzzy set theory. *Micron (Oxford, England)*, *61*, 1–8. doi:10.1016/j.micron.2014.01.004 PMID:24792441

Choi, J., Ku, Y., Yoo, B., Kim, J., Lee, D., Chai, Y., ... Kim, H. C. (2017). White blood cell differential count of maturation stages in bone marrow smear using dual-stage convolutional neural networks. *PLoS One*, *12*(12), e0189259. doi:10.1371/journal.pone.0189259 PMID:29228051

Chollet, F. (2016, October 7). Xception: Deep Learning with Depthwise Separable Convolutions. *CVPR*, *2017*, 5–6.

Das, S. (2018, July 23). *CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more….* Retrieved from Medium: https://medium.com/@sidereal/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

Deshpande, A. (2016). *A Beginner's Guide To Understanding Convolutional Neural Networks*. Retrieved from A Beginner's Guide To Understanding Convolutional Neural Networks: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2020). *Rich feature hierarchies for accurate object detection and semantic segmentation*. ArXiv.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, 315-323.

Habibzadeh, M., Jannesari, M., Rezaei, Z., & Totonchi, M. (2018, April). Automatic white blood cell classification using pre-trained deep learning models. *Tenth International Conference on Machine Vision (ICMV 2017)*. 10.1117/12.2311282

He, K., Zhang, X., Ren, S., & Sun, J. (2015, December). 2015). Deep Residual Learning for Image Recognition. *CVPR*, *2015*, 6–8.

He, K., Zhang, X., Ren, S., & Sun, J. (2016, March 16). Identity Mappings in Deep Residual Networks. *CVPR*.

Huang, G., Liu, Z., Maaten, L. v., & Weinberger, K. Q. (2016, August 25). Densely Connected Convolutional Networks. *CVPR*, 4.

Huang, G., Sun, Y., Liu, Z., Sedra, D., & Weinberger, K. (2016, July). Deep Networks with Stochastic Depth. *Machine Learning*.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016, November 4). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *CVPR*.

Jiang, M., Cheng, L., Qin, F., Du, L., & Zhang, M. (2018). White Blood Cells Classification with Deep Convolutional Neural Networks. *International Journal of Pattern Recognition and Artificial Intelligence*, *32*(09), 1857006. doi:10.1142/S0218001418570069

Kensert, A., Harrison, P. J., & Spjuth, P. (2018, June 14). Transfer learning with deep convolutional neural network for classifying cellular morphological changes. *SLAS Discovery: Advancing Life Sciences R&D*, 8.

Kloss, A. (2015). *Object Detection Using Deep Learning - Learning where to search using visual attention*. Tübingen: Eberhard Karls Universitat Tubingen.

Larsson, G., Maire, M., & Shakhnarovich, G. (2017, May 24). FractalNet: Ultra-Deep Neural Networks without Residuals. *CVPR*.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C., & Berg, A. (2020). *SSD: Single Shot MultiBox Detector*. arXiv.

López-Puigdollers, D., Javier Traver, V., & Pla, F. (2020). Recognizing white blood cells with local image descriptors. *Expert Systems with Applications*, *115*, 695–708. doi:10.1016/j.eswa.2018.08.029

Lowe, D. (2004). Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, *60*(2), 91–110. doi:10.1023/B:VISI.0000029664.99615.94

MoradiAmin, M., Memari, A., Samadzadehaghdam, N., Kermani, S., & Talebi, A.MoradiAmin. (2016). Computer aided detection and classification of acute lymphoblastic leukemia cell subtypes based on microscopic image analysis. *Microscopy Research and Technique*, *79*(10), 908–916. doi:10.1002/jemt.22718 PMID:27406956

Nascimento, P. P. (2016). *Applications of Deep Learning Techniques on NILM*. Rio de Janeiro: Academic Press.

Nazlibilek, S., Karacor, D., Ercan, T., Sazli, M., Kalender, O., & Ege, Y. (2014). Automatic segmentation, counting, size determination and classification of white blood cells. *Measurement*, *55*, 58–65. doi:10.1016/j.measurement.2014.04.008

Nielsen, M. (2017). *Neural Networks and Deep Learning*. Retrieved from neuralnetworksanddeeplearning.com: http://neuralnetworksanddeeplearning.com/

NobleR. (2019). *Leucocytes*. Retrieved from Pinterest: https://www.pinterest.pt/pin/83457399321177163

Perez, L., & Wang, J. (2017, December). *The Effectiveness of Data Augmentation in Image Classification using Deep Learning*. ArXiv.

Prinyakupt, J., & Pluempitiwiriyawej, C. (2015). Segmentation of white blood cells and comparison of cell morphology by linear and naïve Bayes classifiers. *Biomedical Engineering Online*, *14*(1), 63. doi:10.118612938-015-0037-1 PMID:26123131

Putzu, L., Caocci, G., & Di Ruberto, C. (2014). Leucocyte classification for leukaemia detection using image processing techniques. *Artificial Intelligence in Medicine*, *62*(3), 179–191. doi:10.1016/j.artmed.2014.09.002 PMID:25241903

Qin, F., Gao, N., Peng, Y., Wu, Z., Shen, S., & Grudtsin, A. (2018). Fine-grained leukocyte classification with deep residual learning for microscopic images. *Computer Methods and Programs in Biomedicine*, *162*, 243–252. doi:10.1016/j.cmpb.2018.05.024 PMID:29903491

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2020). *You Only Look Once: Unified, Real-Time Object Detection*. arXiv.

Redmon, J., & Farhadi, A. (2020). *YOLO9000: Better, Faster, Stronger*. arXiv.

Redmon, J., & Farhadi, A. (2020). *YOLOv3: An Incremental Improvement*. arXiv

Rehman, A., Abbas, N., Saba, T., Rahman, S., Mehmood, Z., & Kolivand, H. (2018). Classification of acute lymphoblastic leukemia using deep learning. *Microscopy Research and Technique*, *81*(11), 1310–1317. doi:10.1002/jemt.23139 PMID:30351463

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149. doi:10.1109/TPAMI.2016.2577031 PMID:27295650

Reyes, A. K., J. C., & Camargo1, J. E. (2015). Fine-tuning Deep Convolutional Networks for plant recognition. *CLEF 2015*.

Rezatofighi, S., & Soltanian-Zadeh, H. (2011). Automatic recognition of five types of white blood cells in peripheral blood. *Computerized Medical Imaging and Graphics*, *35*(4), 333–343. doi:10.1016/j.compmedimag.2011.01.003 PMID:21300521

Rosebrock, A. (2017, March 20). *ImageNet: VGGNet, ResNet, Inception, and Xception with Keras*. Retrieved from pyimagesearch: https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

Ruiz, P. (2018, October 10). *Understanding and visualizing DenseNets*. Retrieved from Medium: https://towardsdatascience.com/understanding-and-visualizing-densenets-7f688092391a

Russakovsky, O., Jia Deng, H. S., Krause, J., Satheesh, S., Ma, S., Huang, Z., ... Fei-Fei, L. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, *115*(3), 211–252. doi:10.100711263-015-0816-y

Sajjad, M., Khan, S., & Jan, Z. (2016, December). Leukocytes Classification and Segmentation in Microscopic Blood Smear: A Resource-Aware Healthcare Service in Smart Cities. *IEEE Access: Practical Innovations, Open Solutions*, 3475–3489.

Scherer, D., Muller, A., & Behnke, S. (2010, September). Evaluation of Pooling Operations. *20th International Conference on Artificial Neural Networks (ICANN)*, 4.

Shahin, A., Guo, Y., Amin, K., & Sharawi, A. (2019). White blood cells identification system based on convolutional deep neural learning networks. *Computer Methods and Programs in Biomedicine*, *168*, 69–80. doi:10.1016/j.cmpb.2017.11.015 PMID:29173802

Simonyan, K., & Zisserman, A. (2014, September 4). Very Deep Convolutional Networks for Large-Scale Image Recognition. *ILCR*, *2015*, 2–4.

Stathonikos, N., Veta, M., Huisman, A., & van Diest, P. (2013). Going fully digital: Perspective of a Dutch academic pathology lab. *Journal of Pathology Informatics*, *4*(1), 15. doi:10.4103/2153-3539.114206 PMID:23858390

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2014, September 17). Going Deeper with Convolutions. *CVPR*, *2015*, 4.

Tiwari, P., Qian, J., Li, Q., Wang, B., Gupta, D., Khanna, A., ... de Albuquerque, V. H. C. (2018). Detection of subtype blood cells using deep learning. *Cognitive Systems Research*, *52*, 1036–1044. doi:10.1016/j.cogsys.2018.08.022

Wang, Q., Bi, S., Sun, M., Wang, Y., Wang, D., & Yang, S. (2019). Deep learning approach to peripheral leukocyte recognition. *PLoS One*, *14*(6), e0218808. doi:10.1371/journal.pone.0218808 PMID:31237896

Wu, J. (2017, May 1). Introduction to Convolutional Neural Networks. *National Key Lab for Novel Software Technology,* 5-8.

Xie, S., Girshick, R., Dollár, P., Tu, Z., & He, K. (2017, April 11). Aggregated Residual Transformations for Deep Neural Networks. *CVPR*.

Zhang, Q.-J., & Devabhaktuni, V. K. (2003, APRIL). *Artificial Neural Networks for RF and Microwave.* IEEE.

Zhao, J., Zhang, M., Zhou, Z., Chu, J., & Cao, F. (2016). Automatic detection and classification of leukocytes using convolutional neural networks. *Medical & Biological Engineering & Computing*, *55*(8), 1287–1301. doi:10.100711517-016-1590-x PMID:27822698

Zhu, H., Zheng, M. A., Pelegris, A., Jayarajan, A., Phanishayee, A., Shroeder, B., & Pekhimenko, G. (2016, April 14). Benchmarking and Analyzing Deep Neural Network. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, (pp. 13-15). IEEE.

ADDITIONAL READING

Bayramoglu, N., & Heikkila, J. (2016). Transfer learning for cell nuclei classification in histopathology images. *European Conference on Computer Vision*, pp. 532–539. Springer.

Bhagavathi, S., & Thomas Niba, S. (2016). An automatic system for detecting and counting rbc and wbc using fuzzy logic. *Journal of Engineering and Applied Sciences (Asian Research Publishing Network)*, *11*(11), 6891–6894.

Lawrence, S., Giles, C. L., & Tsoi, A. C. (1997). Lessons in neural network training: Overfitting may be harder than expected. AAAI/IAAI, pp. 540–545.

Othman, M. Z., Mohammed, T. S., & Ali, A. B. (2017). Neural network classification of white blood cell using microscopic images. *International Journal of Advanced Computer Science and Applications*, *8*(5), 99–104.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv*:1609.04747.

Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

KEY TERMS AND DEFINITIONS

Data Augmentation: Allows to increase the diversity on the dataset creating new patterns from small variations on the original dataset patterns.

Eosinophil: Type of leukocyte responsible for combating parasites and infection in vertebrates.

Features Maps: Are the output activations for a given filter produced by the convolution operator between the filter weights and the input signals.

Leukocyte: Also known as white blood cell, are involved in the protection of the body against foreign invaders and deseases.

Lymphocyte: Type of leukocyte found on the lymph. They include T and B cells as well as natural killer cells.

Monocyte: They are the largest type of leukocyte, being responsible for phagocytosis, antigen presentation and cytokine production.

Neutrophil: The most prevalent type of leukocyte, is a type of phagocyte normally found on the bloodstream.

Overfitting: Analysis that is too close to a specific dataset, that tends to fail to predict future observations or to fit additional data.

Pre-Processing: Includes processes like selection, cleaning, normalization, transformation feature extraction and transformation in order to obtain data that is more easily treatable.

Training: Determination of the best set of weights for maximizing a neural network's accuracy.