White blood cell image classification with convolutional networks

Martin Martinez, Alex

Abstract

This project tries to find a CNN microscope image classifier for white blood cells as the actual methods for classifying this cells are expensive. The original dataset is extracted from the website Kaggle and the images are previously processed with a segmentation method before feeding them into the models to optimize the learning. Three different models were tested, and every different model showed its advantages and disadvantages, although the Zhimin Gao, Lei Wang et al. model performed very well in one of the classification tasks that the models were tested. Some learning has been obtained from this experience in the cell classification matter.

***Index Terms*—White blood cells, CNN, deep learning, image classification, Tensorflow.**

I. Introduction

THE significant importance of blood cells in the human body makes them target of numerous studies. One of many lines of research dedicated to the blood tries to classify blood cells with hematology analyzers that use spectrophotometry or the electric impedance of the blood to determine the number of each different blood cell type.
In the last few years there's been an increase in the usage of neural networks to classify different cell types to achieve a cheaper and better alternative to the hematology analyzers but there isn't yet a definitive model that ensures a perfect classification for blood cells only with the use of a microscope image of the cell .
The goal of this project is to compare two successful models, as well as a third one created by myself , analyze how they change their behavior varying the data that it's feed into the convolutional neural networks using RGB and phase images as input and proving how well each model classify white blood cells(WBC).

II. Data and preprocessing

The data(obtained from the website Kaggle[2]) that's been used to train the different models is formed by 400 different microscope images of white blood cells with a resolution of 320x240 pixels.

As 400 microscope images is not enough to train a convolutional network these images were rotated in different angles to augment the number of images to a total of 12.500 images. This is a common process used in the training of machine learning models known as data augmentation and it ensures that the model learns more generally as well as ensures enough data to train.

The images of this dataset contained the white blood cell (in purple) as well as red blood cells in the background (pink). This red blood cells act as noise in the background of the images and the images had to be segmented in order to extract only the parts where the white blood cells were.
In order to accomplish these, I used a KNN method to classify the pixels in the image in 4 different types: white blood cells, red blood cells, white background and black background. After this the pixels detected as white blood cells were used to calculate the weighted mean value across each dimension to extract where the white blood cell was centered. Lastly the image was cut to a 100x100 pixel image around the calculated position which had to advantages, first the red blood cells where eliminated and second as the dimensions of the image were reduced the computational cost of training the model was reduced. This segmentation was crucial, as trying to train with the red blood cell background had very bad results with around a 50% validation accuracy in almost every model trained.

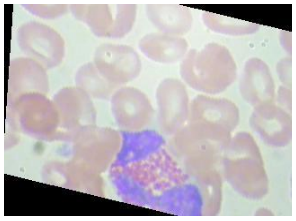
Figure 1: image from the dataset before applying

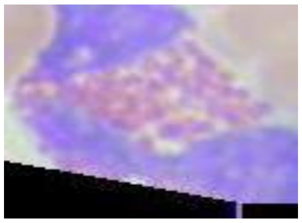
Figure 2: image from the dataset after applying segmentation.

This project was made in summer by Alex Martin Martinez as a part of the subject Image processing and artificial vision in the Physics degree of the University of Barcelona

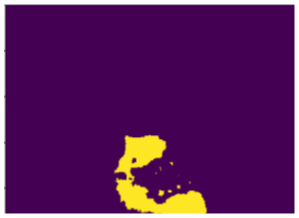

Figure 3: binary mask obtained with the KNN segmentation method.

The formula used to calculate the weighted mean value in the rows was this one, where p_i is the sum of all the pixels value in the columns that intersect the row i in the binary mask and P_{total} is the sum of the intensity of all the pixels in the image.

$$\bar{x} = \frac{\sum_{i=0}^{i=320} p_i \cdot i}{P_{total}}$$

The same formula is applied to calculate the columns mean value.

III. CONVOLUTIONAL NEURAL NETWORK

A. HOW DOES IT WORK

The final objective of a convolutional neural network is to classify the images that conform the dataset and that is accomplished extracting the features of the different types of images through the convolution operation.

Like in a standard neural network the different neurons receive an input, make some sort of non-linear operation and sends the result to the next layer of neurons and so on, but in the case of a convolutional neural network the input is usually image and before making the other operations the image is passed through a convolution layer .

The convolution layer first convolves the image with a filter which transforms it into a feature map. The feature map then gets an activation function applied, that transforms every pixel in the feature map to another value depending on the function used. This function is typically a non-linear function as they work better than linear function.
This is applied with many different filters in each layer, which are called neurons, and the values of each filter will vary as the model is trained to optimize the feature extraction.

After that, the images are flattened and feed into a neural network which finally returns the number of values you specified the model to have in the last layer.

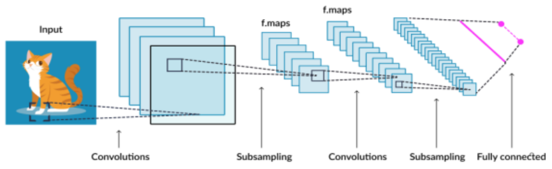

Figure 4: sketch of a CNN

Besides the convolutional layer there are other operations than can be made before feeding the feature map into the neural network or fully connected layer to help extract the features and make a better classification.

Hereafter, there's an explanation of other operations used in the project.

- **MaxPooling**

It works applying a max filter to the image. It replaces the values that are inside the filter with the maximum value among them reducing the size of the feature map.

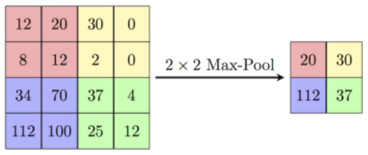

Figure 5: example of a 2x2 filter maxpool.

- **Dropout**

It randomly selects some of the neurons from previous layers and ignores them in the training process which may take out from the classifying process some filters that aren't giving any useful information.

- **Batch Normalization**

You can input the training data in batches (smaller than the training size) so that the model updates the weights used to make predictions more frequently. A problem in this is that the values can go very quickly to the limits of the activation function leading to a bad classification.

The batch normalization layer prevents this by normalizing the values in each batch subtracting the mean value in the batch to each value and then dividing them by the batch standard deviation.

B. TRAINING THE MODEL

Each neuron in each layer has parameters to optimize such as the weights of the filters in the convolutional layers or the weights and the bias in the fully connected layer which make the final predictions. These values are updated as it follows.

Firstly, the CNN gets an image and the label of the image as the input. The CNN with the randomized initialized weights makes a prediction after applying all the explained operation and it outputs values, in our case it will output four different values which represent the probability of the image to be part of each different type of cell.

The model checks how close it was to make the right prediction using the loss function which in the case of classification tasks it's the cross-entropy loss function.

Keeping in mind that to make this prediction the values in the image had to go through different mathematical operations which depend on multiple weights the loss function is a function of multiple variables that reflects how well those weights were and all the variables in the process can be updated so as to achieve a better classification through backpropagation.

Backpropagation is the method used to update the different weights to try to minimize the value of the loss function updating the weights. To know if the weights must be increased or decreased the gradient of the loss function must be calculated to update the weights to advance in the opposite direction of the gradient. This way the weights are updated making small steps towards a minimum of the loss function that will subsequently lead to an optimal classification.

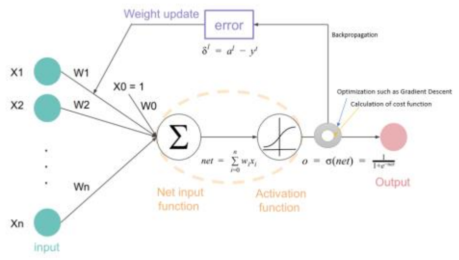
Weight update
error
$\delta^l = a^l - y^l$
Backpropagation
X1
X2
Xn
W1
W2
Wn
X0 = 1
W0
Optimization such as Gradient Descent
Calculation of cost function
$net = \sum_{i=0}^{n} w_i x_i$
$o = \sigma(net) = \frac{1}{1+e^{-net}}$
Net input function
Activation function
Output
input

Figure 6: sketch of the backpropagation process in a singles neuron of a neural network.

After applying this process multiple times, the CNN will be trained and then tested to a new dataset, that wasn't used to train it, to validate how well it had learned to classify the images. This is done to ensure the model had learned general patterns and features from the training dataset and not ones that are exclusive from the training data.
When that occurs, the model is said to be overfitted, it had learned too well to classify the training data in detriment of other images by the model.
This is avoided with regularization techniques such as dropout or batch normalization.

IV. MODELS USED IN THIS PROJECT

Three different CNN models were tested in this project to classify white blood cells. The first model was designed by Zhimin Gao, Lei Wang et al. [1] to classify epithelial human cells. The second model was designed by Paul Mooney [2] and was initially designed to work with the dataset used in this project but without making the segmentation of the images.

Both of this models were already successful and they were used as a guide to create the third model and to prove how the processing of the data could affect the performance of the model as the Zhimin Gao, Lei Wang et al. model worked with the segmented phase image of epithelial cells and the Paul Mooney model worked with the whole images of the dataset. Moreover, the activation functions of the previous models were interchanged to look which one was more suitable for cell image classification and how well did they work in other CNN architectures.

Layer type	Number of filters	Size of feature map	Size of kernel
Input image		100(height)x100(width)x3(channel)	
Convolution	6	92x92x3	9x9
Activation function:Hyperbolic Tan			
BatchNorm			
MaxPool		46x46x3	2x2
Convolution	16	42x42x3	5x5
Activation function:Hyperbolic Tan			
BatchNorm			
MaxPool		14x14x3	3x3
Convolution	32	12x12x3	3x3
Activation function:Hyperbolic Tan			
BatchNorm			
MaxPool		4x4x3	3x3
Flatten		48	
Dense(HyperTan)	128		
Dense (Softmax)	4		

Figure 7: architecture table of the CNN of Zhimin Gao, Lei Wang et al

Layer type	Number of filters	Size of feature map	Size of kernel
Input image		100(height)x100(width)x3(channel)	
Convolution	32	98x98x3	3x3
Activation function:ReLU			
Convolution	64	96x96x3	3x3
Activation function:ReLU			
MaxPool		48x48x3	2x2
Dropout 0,25			
Flatten		6912	
Dense(ReLU)	128		
Dropout 0,5			
Dense (Softmax)	4		

Figure 8: architecture table of the CNN created by Paul Mooney

Layer type	Number of filters	Size of feature map	Size of kernel
Input image		100(height)x100(width)x3(channel)	
Convolution	6	92x92x3	9x9
Activation function:ReLU			
MaxPool		46x46x3	2x2
Convolution	16	42x42x3	5x5
Activation function:ReLU			
MaxPool		21x21x3	2x2
Convolution	32	19x19x3	3x3
Activation function:ReLU			
MaxPool		10x10x3	2x2
Dropout 0,25			
Flatten		300	
Dense(ReLU)	128		
Dropout 0,5			
Dense (Softmax)	4		

Figure 9: architecture table of the CNN created combining both previous CNN

The new proposed model combines the activation function and regularization method from the Paul Mooney model and the number of features, layers and filter sizes from the Zhimin Gao, Lei Wang et al. model.

All the models were trained in four different ways varying the kind of input they were fed (RGB images or phase images) and the classification that had to be done (four types of white blood cells or mononuclear vs polynuclear WBC).

V. RESULTS

After ten epochs of training the results obtained were the following.

	RGB input results			
	Accuracy	**Loss**	**Validation accuracy**	**Validation loss**
Zhimin Gao et al.	0.9489	0.1252	0.8668	0.5017
Zhimin Gao et al. with ReLu	0.9598	0.118	0.7591	0.9093
Paul Mooney	0.6221	0.9067	0.7764	0.6944
Paul Mooney with tanh	<0.25	<0.25	<0.25	<0.25
Combined model	0.8216	0.4170	0.7551	0.7560
	Phase input results			
	Accuracy	**Loss**	**Validation accuracy**	**Validation loss**
Zhimin Gao et al.	0.8491	0.3685	0.5809	1,3255
Zhimin Gao et al. with ReLu	0.8591	0.3489	0.6778	0.9477
Paul Mooney	0.6274	0.8379	0.5671	1,0731
Paul Mooney with tanh	0.5226	1,1007	0.4373	1,991
Combined model	0.7178	0.6570	0.6840	0.8310

Figure 9: results of the trained models after ten epochs for the classification of the 4 types of white blood cells.

	RGB input results			
	Accuracy	**Loss**	**Validation accuracy**	**Validation loss**
Zhimin Gao et al.	0.9829	0.0484	0.9236	0.4108
Zhimin Gao et al. with ReLu	0.9946	0.0164	0.9578	0.1450
Paul Mooney	0.5542	0.6856	0.5561	0.6730
Paul Mooney with tanh	0.4896	0.7027	0.5014	0.7155
Combined model	0.5173	0.6928	0.5464	0.6874
	Phase input results			
	Accuracy	**Loss**	**Validation accuracy**	**Validation loss**
Zhimin Gao et al.	0.9370	0.1400	0.7555	0.6190
Zhimin Gao et al. with ReLu	0.9191	0.1896	0.8039	0.5803
Paul Mooney	0.8237	0.3844	0.7591	0.6334
Paul Mooney with tanh	0.6603	0.6389	0.6908	0.6225
Combined model	0.9048	0.2323	0.8187	0.4097

Figure 10:results of the trained models after ten epochs for the classification between polynuclear and mononuclear cells.

Many different outcomes from the training can be observed in the results tables.

In the first table the Zhimin Gao et al. model presents overfitting with both inputs. The Paul Mooney model is underfitted when the input is an RGB image and ReLu the activation function while all the other combinations reflects very bad results after the training. The combined model seems to avoid these huge overfitting problems as well as the lack of learning from the Paul Mooney model.

In the second table the Zhimin Gao et al. model presents very good results with the RGB input with a 0.9236 validation accuracy and a 0.9578 when the activation function is changed from hyperbolic tangent to ReLU. With the phase input it keeps showing overfitting as in the first table. The Paul Mooney model doesn't reach that good results and keeps showing underfitting in both inputs when the activation function is changed to the hyperbolic tangent. The combined model presents overfitting with the input phase and non-optimal learning with the RGB input.

VI. CONCLUSIONS

Looking up the results when the RGB input is used the models have in average better results and when the phase is used there's more overfitting and the learning seems to slow down. This may confront with the fact that the phase image contains the information from the image, but it could be that the features needed to recognize the cells from the dataset are extracted from the intensity values.

The models that used the ReLU activation function had overall better results than the ones which used the hyperbolic tangent. Better results were expected from the phase input trained models that had the hyperbolic tangent, as this function with negative inputs has different responses while ReLU outputs a zero for any negative number. But the faster learning and non-saturation from the gradient overcomes this.

Both regularization models had its advantages and disadvantages. Batch normalization seems to avoid a better learning in detriment of overfitting the model while dropout prevents this but leading to a slower learning.

From all the models the best one in the mononuclear vs polynuclear classification is the Zhimin Gao et al. with ReLU while in the four-cell classification the Paul Mooney and the combined model had the best result.

The models need to be trained to more than ten epochs to confirm the results I obtained as there's still room for improvement for some models, but with the time and computational power I had this is the most I could train them, even so, some learning had been obtained in the matter of white blood cell classification with CNN and a model was founded to be successful in on of the classifications.

REFERENCES AND BIBLIOGRAPHY

[1] Z. Gao, L. Wang, L. Zhou and J. Zhang, "HEp-2 Cell Image Classification With Deep Convolutional Neural Networks," in IEEE Journal of Biomedical and Health Informatics.

[2] https://www.kaggle.com/paultimothymooney/blood-cellsB

[3] https://www.kaggle.com/paultimothymooney/identify-blood-cell-subtypes-from-images

- N. Meng, H. K. H. So and E. Y. Lam, "Computational single-cell classification using deep learning on bright-field and phase images,"