Comparing the performance of different optimizers in convolutional neural networks for leukocyte classification

Krushelnytskyi Illia

Abstract. This paper investigates the effectiveness of different optimizers for leukocyte classification in convolutional neural networks. We compare the performance of optimization functions supported by TensorFlow. Our experiment shows that "Adam" optimizer achieves the highest accuracy of 0.9844, followed by "RMS Prop" with 0.9776. The lowest accuracy of 0.4158 was achieved by "Gradient Descent". Our study demonstrates the importance of selecting optimal optimizer for best performance in leukocyte classification tasks using CNNs on blood images.

Keywords:
- leukocyte
- machine learning
- optimizer
- convolutional neural networks
INTRODUCTION

A white blood cell, also known as a leukocyte or white corpuscle, is a cellular component of the blood that contains a nucleus, is capable of movement, and lacks hemoglobin. The main function of leukocytes is to protect the body against infection and disease. White blood cells can produce antibodies, destroy infectious agents and cancer cells, as well as ingest foreign materials and cellular debris to carry out their defensive functions. Leukocytes are typically found in circulation, but can also be encountered in tissues where they fight infections [1].

There are several types of leukocytes, each responsible for different aspects. Lymphocytes produce antibodies that help the body fight against bacteria, viruses, and other threats. Neutrophils aid in destroying bacteria and fungi. This type is the most abundant white blood cell in circulation. Using cytokines, neutrophils are attracted to bacteria and migrate through tissues to infection sites. These cells engulf and destroy their targets. Their granules break down cellular macromolecules and then destroy the neutrophil itself [2]. Basophils secrete chemicals into the bloodstream to alert the body to infections, mostly to combat allergies. Basophils are the least common type of white blood cell. Their granules contain immune-boosting compounds such as histamine and heparin. Histamine dilates blood vessels, increasing blood flow, so that leukocytes can be transported to infected areas. Heparin thins the blood and prevents blood clot formation. Eosinophils are responsible for destroying parasites and cancer cells. The nucleus of these cells on blood smears resembles the letter "U". Eosinophils are most active during allergic reactions and parasitic infections. They mainly bind to antigens and signal that they need to be destroyed. The most common location for eosinophils is in the tissue of the stomach and intestines [2].

Monocytes are the largest cells among leukocytes in terms of size. They have a single nucleus, which in most cases resembles a kidney shape. These cells migrate from the blood into tissues and develop into macrophages or dendritic cells. Macrophages are most necessary during pregnancy. With
their help, a network of blood vessels develops in the ovary, which in turn is important for the production of the hormone progesterone.

Dendritic cells are found in the skin, digestive system, and lungs. The function of these cells mainly lies in the development of antigenic immunity [2].

Today, the process of leukocyte recognition and classification is performed by an expert. Since this procedure is done manually, it is time-consuming and complex, therefore automating the procedure would help overcome these problems.

CONVOLUTIONAL NEURAL NETWORK

Convolutional Neural Networks, also known as ConvNets or simply CNNs, are a deep learning algorithm that takes images as input, determines weights and biases for different objects in the image, and distinguishes them from each other.

There are many algorithms for image recognition and classification [3], but the Convolutional Neural Networks (CNNs) are considered the most effective among them.

An important feature of convolutional networks is the fact that feature extraction is not required. The core concept of CNN involves generating invariant features that are passed on to the next layer using convolution of the image and filters. Using different filters to generate more invariant and abstract features, the already obtained features are convoluted and passed on to the next layer, and this process continues until we obtain the final feature that is invariant to occlusions [4].
There is no specific fixed architecture for convolutional neural networks, as the architecture of the neural network will vary depending on the number of layers. However, the general structure looks like Fig. 1.

One of the primary tasks of ConvNet is to reduce the image to a form that is easier to process, without losing features that may be critical for obtaining the correct answer. This is important when we need to create not just good at recognizing features, but an architecture that can also easily scale to process large datasets.

The objective of the Convolution Operation is to extract the high-level features from the input image. ConvNets may contain more than one Convolutional Layer. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as gradient orientation, color, etc. With added layers, the architecture adapts to the High-Level features as well, giving us a network that has a wholesome understanding of images in the dataset.

In case the convolution operation results in a reduction of the convolved feature dimensionality compared to the input image, this is achieved by using Valid Padding. If the dimensionality remains the same or increases, Same Padding is used [5].

The Pooling layer performs the task of reducing the spatial size of the convolved feature, thereby reducing the computational power required to process the data. Additionally, dominant features that are invariant to rotation and position are highlighted by the Pooling layer, enabling the model to learn more efficiently. There are two types of Pooling: Max Pooling and Average Pooling. Max Pooling serves as a noise suppressor by discarding noisy activations and reducing dimensionality, returning the maximum value from the portion of the image covered by the Kernel. On the other hand, Average Pooling returns the average of all the values from the portion of the image covered by the Kernel while also reducing dimensionality to suppress noise. Overall, it can be concluded that Max Pooling performs better than Average Pooling [5].

The Convolutional Layer and the Pooling Layer merge into one layer of a Convolutional Neural Network. The number of such layers may increase depending on the complexity of the
images, thereby capturing even more features. However, increasing the number of layers comes at the cost of computational power.

After converting the image into a suitable format for the network, the image is flattened into a column vector. Then, the resulting output is passed to a feedforward neural network where backpropagation is applied at every iteration of training. Over the course of training, the model is able to distinguish between dominant and low-level features in images and classify them using Softmax classification technique [5].

**DATASET AND PREPROCESSING**

To train the model, a dataset available in the public domain on the free Kaggle platform was used [6]. This dataset contains 12,500 augmented images of blood cells (JPEG format) with accompanying cell type labels (CSV format). Unfortunately, the dataset does not include the coordinates of leukocytes, so it was necessary to use mask filters to isolate them in the images. As shown in the example image from the dataset (Fig. 2), the background (i.e. the area that is neither an erythrocyte nor a leukocyte) is gray, the color of erythrocytes tends towards red hues, and the color of leukocytes has a strong blue component.

![Example of an image from the training dataset](image_url)
After applying the mask filter, the image remains not very "clean" (Fig. 3).

![Image with many scraps after using mask filter](image)

**Figure 3**

*Image with many scraps after using mask filter*

To supplement the mask filter, we performed the following steps:

1. Get rid of the little scraps and make the masks more round.
2. Find the bounding boxes of the leukocytes.
3. Finally, crop the leukocytes according to their bounding boxes.

As a result, we obtained an isolated image of a leukocyte, which was used as an example for training the model (Fig. 4).

![Isolated leukocyte](image)

**Figure 4**

*Isolated leukocyte*
INFORMATION AND WEB TECHNOLOGIES

OPTIMIZERS
Optimization algorithms improve the performance of deep learning models by significantly enhancing the speed and accuracy of model training. During model training, minimizing the loss function and modifying the weights of each epoch are important.

An optimizer is an algorithm or function that changes the weights and learning rate, resulting in reducing overall loss and improving accuracy.

Gradient Descent Optimizer
This optimization algorithm consistently modifies the values and achieves the local minimum using calculus. The equation (1) describes what the gradient descent algorithm does [7].

\[ b = a - \gamma \nabla f(a) \]  

\( b \) is the next position, while \( a \) represents the current position. The minus sign refers to the minimization part of the gradient descent algorithm. The gamma in the middle is a waiting factor and the gradient term is simply the direction of the steepest descent. So, this formula tells us the next position, which is the direction of the steepest descent.

Gradient descent starts with some coefficients, sees their cost, and searches for cost value lesser than what it is now. Then it moves towards the lower weight and updates the value of the coefficients. The process repeats until the local minimum is reached. A local minimum is a point beyond which it cannot proceed [8].

Gradient descent is a good approach for most tasks, but it also has some drawbacks. If the data size is large, calculating the gradient can be expensive. For non-convex functions, this method does not know how far to move along the gradient, while for convex functions it works very well.

Stochastic Gradient Descent Optimizer
As mentioned above, gradient descent on large datasets might not be the best option. Therefore, stochastic gradient descent was created. This algorithm is based on randomness, which is what the term "stochastic" means. Instead of selecting all the data, we randomly choose a portion of the
data, known as a batch.

\[ w = w - \eta \nabla Q_i(w) \]  

(2)

The equation (2) shows that first we select the initial parameters \( w \) and learning rate \( \eta \). Then randomly shuffle the data at each iteration to reach an approximate minimum [8].

Considering that we are not using the entire dataset but only a portion of it in each iteration, the path taken by the algorithm is full of noise compared to the gradient descent algorithm. Thus, SGD requires more iterations to reach the local minimum. With an increase in the number of iterations, the overall computation time also increases. However, even after increasing the number of iterations, the computational cost is still lower than that of the gradient descent optimizer.

**Adagrad (Adaptive Gradient Descent) Optimizer**

Adaptive gradient descent algorithm differs from other gradient descent algorithms in that it uses different learning rates for each iteration. The change in the learning rate depends on the difference in parameters during training. This modification is highly beneficial since real-world datasets contain both sparse and dense features, and it is unfair to have the same learning rate value for all the features.

\[ W_t = W_{t-1} - \eta_t \frac{\partial l}{\partial w(t-1)} \]  

(3)

\[ \eta_t = \frac{\eta}{\sqrt{\alpha_t + \epsilon}} \]  

(4)

Formulas (3) and (4) show how the Adagrad algorithm updates weights. Here the \( \alpha_t \) denotes the different learning rates at each iteration, \( \eta \) is a constant, and \( \epsilon \) is a small positive to avoid division by 0 [8].

A crucial aspect of using Adagrad is that there is no need to manually adjust the learning rate. It converges to the minimum faster and is more reliable than gradient descent algorithms and their variations.

The main downside of the Adagrad optimizer is that there may come a point where the learning rate becomes extremely
small, which compromises the accuracy of the model as it becomes unable to acquire more knowledge. This can happen because the squared gradients in the denominator keep accumulating, and thus the denominator part keeps increasing [8].

RMS Prop (Root Mean Square) Optimizer

RMS prop is an extended version of RPROP which resolves the problem of varying gradients. The issue with gradients is that some may be large while others small. Therefore, defining a single learning rate may not be ideal. In the RPROP algorithm, two gradients are first compared for their signs. If they have the same sign, the step size is increased by a small fraction as we are going in the right direction. If they have opposite signs, the step size is decreased. The step size is then limited and the weights are updated accordingly.

However, the problem with RPROP is that it struggles to work with large datasets. Hence, RMS prop was introduced, which is sometimes considered as an improvement over the AdaGrad optimizer as it reduces the monotonically decreasing learning rate [8].

The main focus of the RMS prop algorithm is to accelerate the optimization process by reducing the number of function evaluations needed to reach the local minimum. The algorithm keeps the moving average of squared gradients for every weight and divides the gradient by the square root of the mean square.

The algorithm primarily focuses on speeding up the optimization process by reducing the number of function evaluations required to reach the local minimum. To achieve this, the algorithm maintains the moving average of squared gradients for each weight and divides the gradient by the square root of the mean square. Formula (5) mathematically describes the process.

\[
    v(w, t) = \gamma v(w, t - 1) + (1 - \gamma)(\nabla Q_i(w))^2
\]  \hspace{1cm} (5)

Where gamma is the forgetting factor. Weights are updated by the formula (6).

\[
    w = w - \frac{\eta}{\sqrt{v(w, t)}} \nabla Q_i(w)
\]  \hspace{1cm} (6)
If there exists a parameter due to which the cost function oscillates a lot, we want to penalize the update of this parameter. This algorithm has several benefits as compared to earlier versions of gradient descent algorithms. The algorithm converges quickly and requires lesser tuning than gradient descent algorithms and their variants. The problem with RMS Prop is that the learning rate has to be defined manually, and the suggested value doesn’t work for every application [8].

**Adam Optimizer**

The name Adam comes from "adaptive moment estimation". This optimization algorithm is a modification of stochastic gradient descent used to update the weights of a network during training. Unlike stochastic gradient descent, Adam updates the learning rate for each weight separately. Adam also incorporates the advantages of the AdaGrad and RMSProp algorithms. RMSProp adapts the learning rates based on the mean squared gradient, while Adam also uses the mathematical expectation of the squared deviations from the mean (which also called the uncorrected variance).

\[
m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\delta L}{\delta w_t} \\

v_t = \beta_2 v_{t-1} + (1 - \beta_2) \left( \frac{\delta L}{\delta w_t} \right)^2
\]

Formulas (7) and (8) represents the working of adam optimizer. Here \( \beta_1 \) and \( \beta_2 \) represent the decay rate of the average of the gradients [8].

The Adam optimizer has several benefits, making it widely used and recommended as a default optimization algorithm for deep learning. It is straightforward to implement, has a faster running time, requires low memory, and needs less tuning than any other optimization algorithm.

This is because even Adam has some downsides. It tends to focus on faster computation time, whereas algorithms like stochastic gradient descent focus on data points. That’s why algorithms like SGD generalize the data in a better manner at the cost of low computation speed [8].

METRICS
The primary performance metric used for monitoring the model during training and selecting the best model is accuracy. This metric represents the percentage of images from the test set that are correctly classified by the model. All models will be presented with additional traditional classification metrics, including precision, recall, f1-score and support.

It should be noted that comparing the absolute errors in this study is valid since the separation of the sample into training and testing sets was performed only once, before the training of any of the models, ensuring that all models used the same training and testing data.

RESULTS

Five models were trained: model_gradient_descent, model_sgd, model_adagrad, model_rms_prop, model_adam, using the optimizers: Gradient Descent, SGD, AdaGrad, RMS Prop, Adam, respectively.

The results are presented in the form of a table (Table 1), where the first column contains information about the model and the second and third columns show the values for training_accuracy and training_loss, respectively. The graphs of the changes in training_accuracy and validation_accuracy, as well as training_loss and validation_loss (Fig. 5, Fig. 6, Fig. 7, Fig. 8, Fig. 9, Fig. 10, Fig. 11, Fig. 12, Fig. 13, Fig. 14), will be provided for each model separately, along with additional classification metrics such as precision, recall, f1-score, and support (Table 2, Table 3, Table 4, Table 5, Table 6).

### Table 1

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Loss</th>
</tr>
</thead>
<tbody>
<tr>
<td>model_gradient_descent</td>
<td>0.4158</td>
<td>1.2264</td>
</tr>
<tr>
<td>model_sgd</td>
<td>0.9461</td>
<td>0.1442</td>
</tr>
<tr>
<td>model_adagrad</td>
<td>0.6959</td>
<td>0.7019</td>
</tr>
<tr>
<td>model_rms_prop</td>
<td>0.9776</td>
<td>0.0721</td>
</tr>
<tr>
<td>model_adam</td>
<td>0.9844</td>
<td>0.0424</td>
</tr>
</tbody>
</table>

As can be seen from Table 1, the model that used the "Adam" optimizer performed the best, while "Gradient Descent" performed the worst. "RMS Prop" and "SGD" have fairly good

---

This work is distributed under the terms of the Creative Commons Attribution-ShareAlike 4.0 International License (https://creativecommons.org/licenses/by-sa/4.0/).
values, while "AdaGrad" performs better than "Gradient Descent", but still not as well as the others.

The following graphs show the changes in training/validation accuracy/loss, as well as the tables with other classification metrics for each model separately.

Figure 5
training & validation accuracy of the model_gradient_descent

Figure 6
training & validation loss of the model_gradient_descent
Figure 7
training & validation accuracy of the model_sgd

Figure 8
training & validation loss of the model_sgd
This work is distributed under the terms of the Creative Commons Attribution-ShareAlike 4.0 International License (https://creativecommons.org/licenses/by-sa/4.0/).
INFORMATION AND WEB TECHNOLOGIES

Figure 11
training & validation accuracy of the model_rms_prop

Figure 12
training & validation loss of the model_rms_prop
INFORMATION AND WEB TECHNOLOGIES

Figure 13
training & validation accuracy of the model_adam

Figure 14
training & validation loss of the model_adam
INFORMATION AND WEB TECHNOLOGIES

Table 2

<table>
<thead>
<tr>
<th>Classification metrics of the model gradient descent</th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOSINOPHIL</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>312</td>
</tr>
<tr>
<td>LYMPHOCYTE</td>
<td>0.40</td>
<td>0.87</td>
<td>0.54</td>
<td>303</td>
</tr>
<tr>
<td>MONOCYTE</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>301</td>
</tr>
<tr>
<td>NEUTROPHIL</td>
<td>0.33</td>
<td>0.61</td>
<td>0.43</td>
<td>299</td>
</tr>
</tbody>
</table>

Table 3

<table>
<thead>
<tr>
<th>Classification metrics of the model sgd</th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOSINOPHIL</td>
<td>0.92</td>
<td>0.93</td>
<td>0.92</td>
<td>312</td>
</tr>
<tr>
<td>LYMPHOCYTE</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>303</td>
</tr>
<tr>
<td>MONOCYTE</td>
<td>0.99</td>
<td>1.00</td>
<td>0.99</td>
<td>301</td>
</tr>
<tr>
<td>NEUTROPHIL</td>
<td>0.94</td>
<td>0.93</td>
<td>0.94</td>
<td>299</td>
</tr>
</tbody>
</table>

Table 4

<table>
<thead>
<tr>
<th>Classification metrics of the model adagrad</th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOSINOPHIL</td>
<td>0.64</td>
<td>0.34</td>
<td>0.44</td>
<td>312</td>
</tr>
<tr>
<td>LYMPHOCYTE</td>
<td>0.92</td>
<td>0.96</td>
<td>0.94</td>
<td>303</td>
</tr>
<tr>
<td>MONOCYTE</td>
<td>0.75</td>
<td>0.84</td>
<td>0.79</td>
<td>301</td>
</tr>
<tr>
<td>NEUTROPHIL</td>
<td>0.64</td>
<td>0.78</td>
<td>0.70</td>
<td>299</td>
</tr>
</tbody>
</table>

Table 5

<table>
<thead>
<tr>
<th>Classification metrics of the model rms_prop</th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOSINOPHIL</td>
<td>0.96</td>
<td>0.99</td>
<td>0.97</td>
<td>312</td>
</tr>
<tr>
<td>LYMPHOCYTE</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>303</td>
</tr>
<tr>
<td>MONOCYTE</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>301</td>
</tr>
<tr>
<td>NEUTROPHIL</td>
<td>0.98</td>
<td>0.97</td>
<td>0.97</td>
<td>299</td>
</tr>
</tbody>
</table>

Table 6

<table>
<thead>
<tr>
<th>Classification metrics of the model adam</th>
<th>precision</th>
<th>recall</th>
<th>f1-score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>EOSINOPHIL</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
<td>312</td>
</tr>
<tr>
<td>LYMPHOCYTE</td>
<td>1.00</td>
<td>1.00</td>
<td>1.00</td>
<td>303</td>
</tr>
<tr>
<td>MONOCYTE</td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>301</td>
</tr>
<tr>
<td>NEUTROPHIL</td>
<td>0.98</td>
<td>0.98</td>
<td>0.98</td>
<td>299</td>
</tr>
</tbody>
</table>

CONCLUSION

In conclusion, leukocyte recognition and classification are important tasks, as white blood cells can contain information about harmful, and sometimes deadly, viruses,
fungi, bacteria, and diseases in general.

We have discussed and compared the results of several existing optimizers offered by TensorFlow. Our experiments have shown that Adam performs the best, followed by RMS Prop, then SGD and AdaGrad. Gradient Descent performed the worst. Although testing all existing optimizers would be needed for a complete investigation, we can confidently say that Adam is the best for this task, although it will depend on the problem at hand.

By continuing to explore leukocyte recognition algorithms, we can help improve the accuracy of diagnoses and potentially save lives.

References:


