Final project in an introductory course to Artificial Intelligence

Presenters: Gleb Tcivie | Lecturers in the study: Dr. Tamar Shurat

Table of Contents

Introduction

In this project, I was required to choose a problem that requires a solution using AI, solve it, explain why I chose these methods, and review the results. The problem I decided comes from the field of medicine, where I had to classify pictures of bone marrow samples and classify them into the types of cells found in the samples. [1] [2] [3] In the pictures, the cells appear alone and are sometimes accompanied by other cells not in the center of the picture, making it difficult to classify.

While working on this project, I encountered many problems in different areas of artificial intelligence. One of the central and most significant problems I had was dealing with the large number of images that needed to be worked with. This required great attention and care to not destroy the local copy on the computer (since each reconstruction of the database required a lot of time and resources).

Another and no less important part that I will focus on will be the distinct differences between different CNN architectures and their impact on the performance of learning and prediction.

Finally, I will measure the performance of the SVN algorithm and compare all the features I have collected to see on which levels which algorithm functions better. In addition, it is essential to note that all algorithms are supervised Learning algorithms.

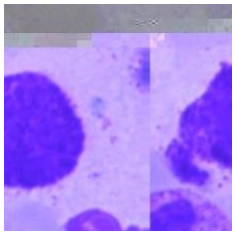
The things I will measure throughout the runs will be:

- Hardware requirements
- Memory usage
- Algorithm runtimes (especially the learning time that takes most of the time)
- Model size at the end of the run
- Efficiency of models

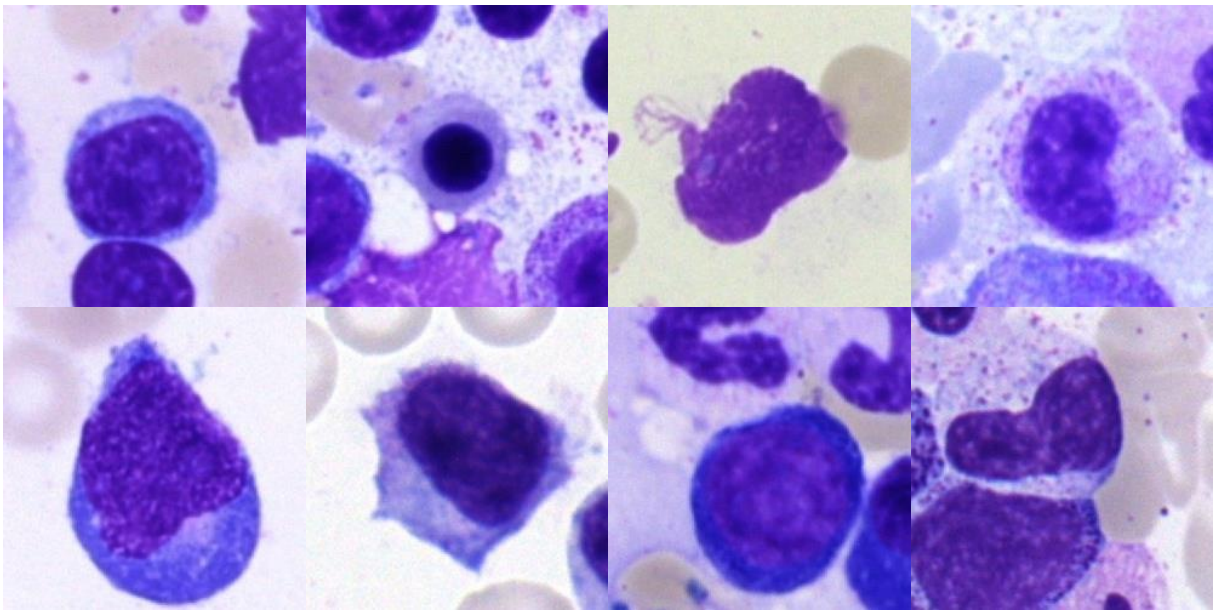
Technical details about the database

Number of subjects	945
Number of images	171,375
Total size of the images (GB)	6.8
Size each image	250x250

After downloading the data to a computer and trying to open all the images through Python, I encountered a severe problem; some of the pictures became corrupted during download/extraction from the zipped file. Which made it very difficult for me and required me to filter the damaged images without removing the good ones I needed for training.
The main problem was finding these images. Because in terms of the correctness of the information, the file was intact, and the beginning and extension of the files were correct. So I had to use heavy libraries that scan all the images in their entirety and look for those flaws.
Example of a damaged image:



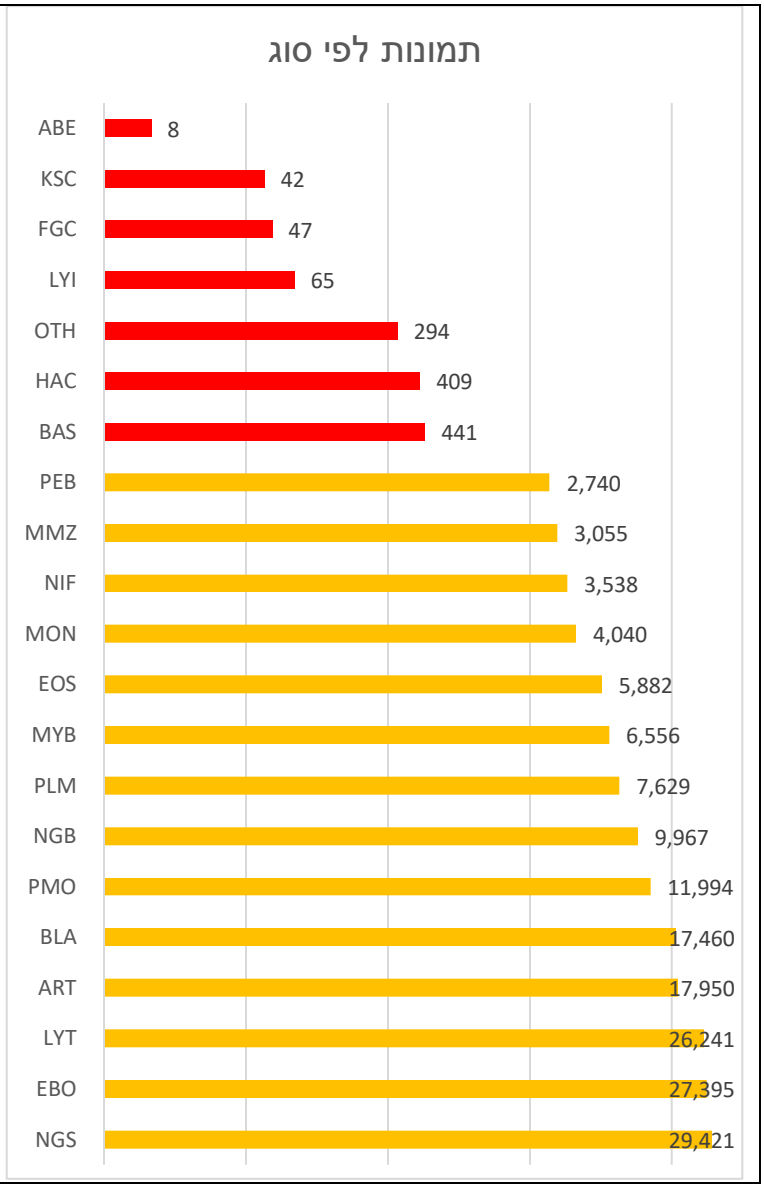
An example of valid images in the repository:



The database has 21 different classifications, and the amount of samples of each type is uneven, making it difficult to correctly classify the images with various AI algorithms. In addition, from the table below, you can see that there was not enough information about some classifications.

Detailed information on all Classifications with the number of files in each class

Abbreviation of the name	Cell type name	Number of images
ABE	Abnormal eosinophil	8
KSC	Smudge cell	42
FGC	Faggott cell	47
LYI	Immature lymphocyte	65
OTH	Other cell	294
HAC	Hairy cell	409
BAS	Basophil	441
PEB	Proerythroblast	2,740
MMZ	Metamyelocyte	3,055
NIF	Not identifiable	3,538
MON	Monocyte	4,040
EOS	Eosinophil	5,882
MYB	Myelocyte	6,556
PLM	Plasma cell	7,629
NGB	Band neutrophil	9,967
PMO	Promyelocyte	11,994
BLA	Blast	17,460
ART	Artefact	17,950
LYT	Lymphocyte	26,241
EBO	Erythroblast	27,395
NGS	Segmented neutrophil	29,421



It is important to note that the data is after the cleaning of the corrupted files

From the table, you can see several groups (marked in red) in that we have a relatively minimal number of samples compared to the rest of the data. If we work with these data, it is doubtful that we will be able to distinguish these types of cells from the rest of the cells.

To overcome this, we will first have **to produce more information with what we have** to help the AI learn the characteristics of all the images. I will use a standard method of Data Augmentation to generate new pieces of information from existing data to produce new learning images.

I will apply the method only to the existing classes to increase the number of images in those classes without changing the quantitative ratio between the other departments. And the number of pictures I will limit to the median of all values (4,040).

On each image in each of the folders with the minimum amount of pictures, I will run an algorithm using a Keras library that will perform on each photo one (or more) of the following augmentations:

- Rotate the image clockwise by 15°
- Collapse the image by $\frac{1}{255}$
- 20% image zoom
- Mirror with the X-axis
- Mirror the Y-axis

Results of an attempt to overcome a lack of information in the system

Abbreviation of the name	Number of images
ABE	72
KSC	1649
FGC	2043
PEB	2740
MMZ	3055
LYI	3327
HAC	3419
OTH	3427
BAS	3433
NIF	3538
MON	4040
EOS	5882
MYB	6556
PLM	7629
NGB	9967
PMO	11994
BLA	17460
ART	17950
LYT	26241
EBO	27400
NGS	29426

Now you can see that the difference is tiny except for the ABE classification; in this case, I had nothing more to do as, from the beginning, the amount of data was meager.
After all of the above processing, I shrunk the images into TensorFlow objects called TFRecords[1], which allowed me to bind multiple pictures in a single binary file that works with the TensorflowAPI and will enable me to reduce the number of files I need to work with. I uploaded the information to the Google Storage Bucket cloud[2], which later allowed me to work with Google tools, such as Colab[3] , and access computers called Tensor Processing Unit (TPU) [4].

[1] https://www.tensorflow.org/tutorials/load_data/tfrecord
[2] https://cloud.google.com/storage/docs/buckets
[3] https://colab.research.google.com/
[4] https://cloud.google.com/tpu/docs/tpus

Total After shrinking, I was left with 841 tfrec files whose total size equals 6.6 Gib, with each file containing 230 images *(or less, because each file contains only photos from one class and some classes do not exactly divide by* 230*).*

Algorithms and methods

In my work, I chose to try to solve the problem with two AI "Algorithms", one using several Convolution Neural Network (CNN) architectures, which I will detail below. A second method will involve an attempt to solve the same problem, only this time using Simple Vector Machine (SVM).

Convolution Neural Network (CNN)

CNN is a neural network (ANN) that works well with grid-like topologies, such as photos or video. The grid usually consists of many layers, including convolutional, pooling, and well-connected layers, which I will detail immediately.

General explanation of the layers

All different architectures most often use the same types of layers. The 3 most common layers in CNN architectures are the following layers:

Convolutional layers

Responsible for extracting attributes from the images, they do this by running "filters" (matrices with predefined values) that we "slide" on the data (multiplying the matrix on the part it hides and the resulting value we classify in a new smaller matrix). In other words, we extract characteristics from the image in different locations.

Pooling layer

The layer is responsible for reducing the maximum resolution of the data, which helps us lower the computational cost of the network and improve its performance. We often take a data window of a specific size (say 3x3) and find its maximum value or perform a particular operation of calculation that brings us a result, such as an average or a median.

Well-connected layers

These layers are layers that, as the name says, are well connected to the following network and allow the network to perform the classification (similar to a standard neural network).

The selected algorithms

The algorithm has many ready-made networks/architectures that have been tested and trained and have reached a state-of-the-art state. Therefore I will use some of them to achieve an optimal solution and save time on running/optimization. It is important to note that the networks already come in a pre-trained state (with values and weights), meaning that the

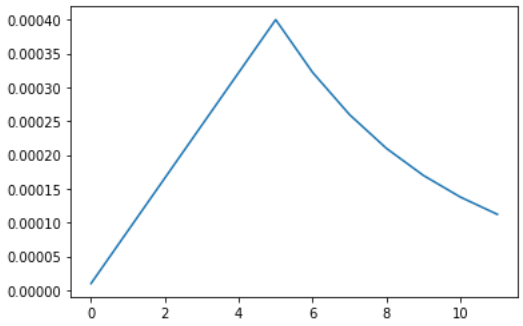
training I perform is performed on a network with existing weights, contributing to the network's learning time.
In this work, I will use the following architectures:

1. ResNet (Residual Network) [4]
2. Inception [5]
3. DenseNet [6]

I chose to use the following architectures because of their popularity, ability to learn from data with many characteristics (like images) and being modern.

The selected learning function

The learning function I chose is called the Cyclical Learning Rate, and in my case, it is the Single Cyclical Learning Rate. The process changes during learning and helps to reach convergence faster. The function first increases linearly and increases the learning rate each time, and at the extreme point of learning, it begins to decrease in a logarithmic way with each EPOCH. I took the method from the following article. [7]



ResNet (Residual Network)

The architecture was first introduced in 2015 and uses "Residual connections or skip connections" that do not connect neurons as usual. These connections can bypass neuronal layers and thus preserve specific characteristics in the network.

Network architecture

Model: "sequential"

```
_________________________________________________________________
 Layer (type) Output Shape Param #
=================================================================
 resnet50 (Functional) (None, 8, 8, 2048) 23587712

 global_average_pooling2d (G (None, 2048) 0
 lobalAveragePooling2D)

 dense (Dense) (None, 21) 43029

=================================================================
Total params: 23,630,741
Trainable params: 23,577,621
Non-trainable params: 53,120
_________________________________________________________________
```

The total number of weights in the entire system is 23,630,741, of which 23,577,621 are trainable.

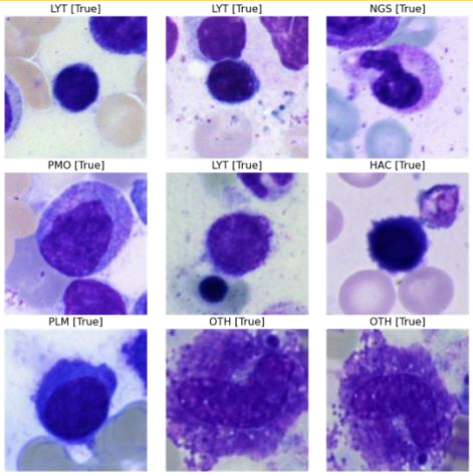
Runtimes and other features

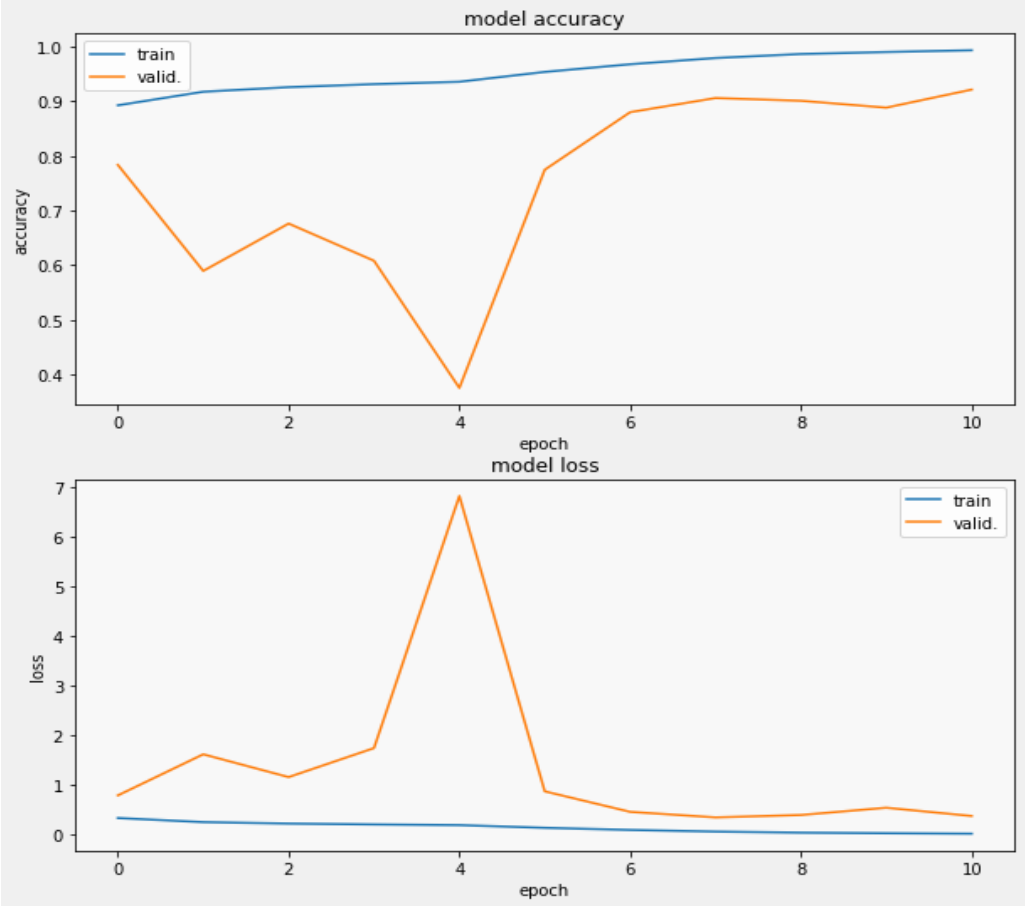
It took the system 3898.8 seconds to learn 191,248 images. (hours: 1 | Minutes: 4)
At the end of the run, we received the following parameters:

loss	0.0193
accuracy	0.9935
val_loss	0.3752
val_accuracy	0.9215

Testing on 160 random images from the Validation set, we got the following results:

loss	0.6100
Accuracy	0.9125





In a few words

From the graphs, it can be seen that the network had a significant jump in the fourth iteration. This could have been due to a class ABE that could not be effectively studied and therefore caused a sharp increase in the loss function and a sharp decrease in the model's accuracy.
Some crucial points that can indicate us overfitting:

- The difference in the results in the loss function of the test data set and the learning data set is enormous$|loss - val_{loss}| = |0.0193 - 0.3752| = 0.3559$ which is 35% difference!
- Another characteristic is accuracy, especially in the Training Set, which reaches 99.3%.

Inception

The architecture is designed especially for image classification tasks. It was introduced in 2014 and is characterized by the fact that it uses several filters of different sizes in the convolution layers, which allows it to learn attributes in different scales.

Network architecture

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type) Output Shape Param #
=================================================================
 inception_v3 (Functional) (None, 6, 6, 2048) 21802784

 global_average_pooling2d_1 (None, 2048) 0
 (GlobalAveragePooling2D)

 dense_1 (Dense) (None, 21) 43029

=================================================================
Total params: 21,845,813
Trainable params: 21,811,381
Non-trainable params: 34,432
_________________________________________________________________
```

The total number of weights in the entire system is 21,845,813, of which 21,811,381 are trainable.

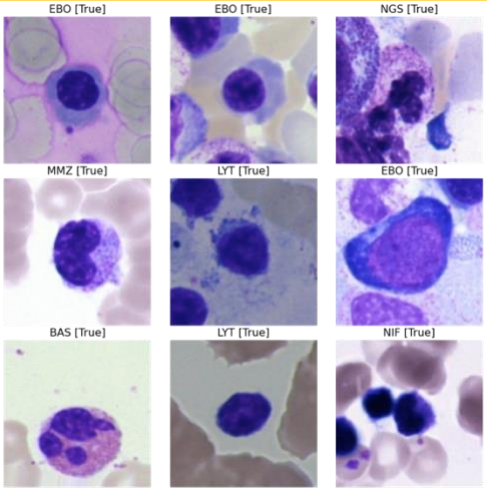
Runtimes and other features

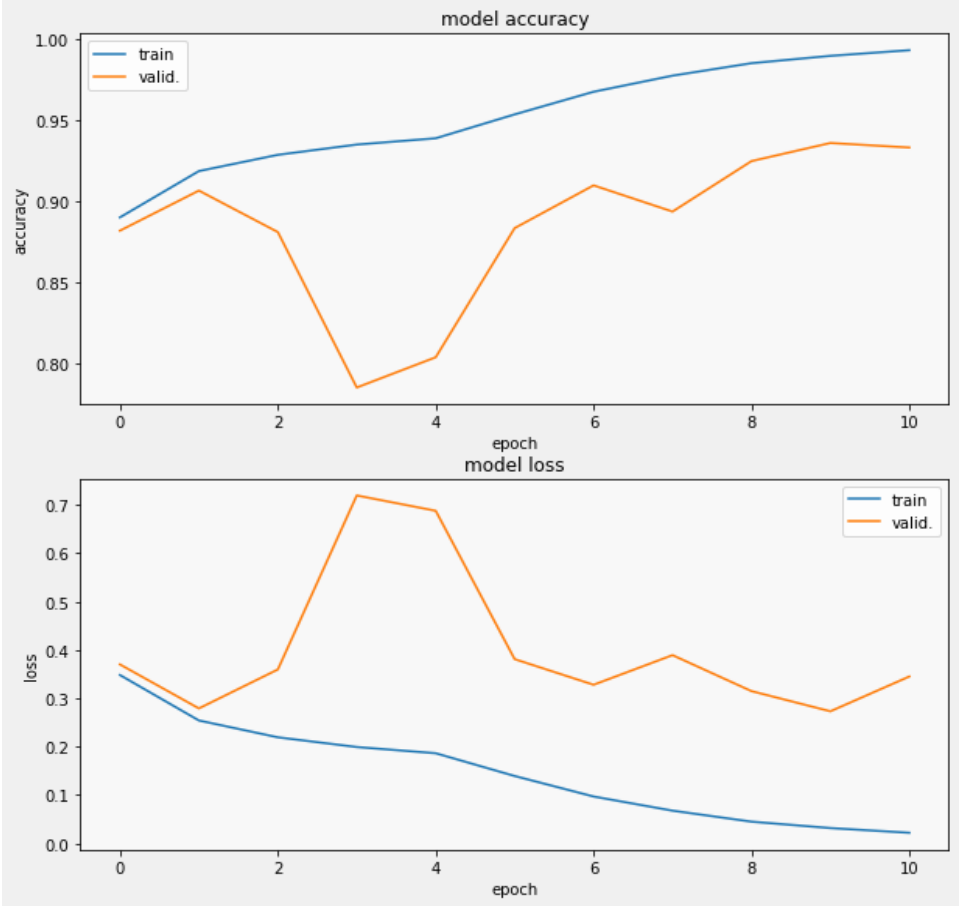
It took the system 3998.1 seconds to learn 191,248 images. (Hours: 1 | Minutes:6)
At the end of the run, we received the following parameters:

loss	0.0216
accuracy	0.9928
val_loss	0.3450
val_accuracy	0.9329

Testing on 160 random images from the Validation set, we got the following results:

loss	0.5756
accuracy	0.9375





In a few words

In this architecture, unlike the previous one, you can see that here, too, there is a chance of overfitting. Still, the results look better on examination than in the last architecture.
In terms of runtimes, there is also not much difference.

DenseNet

The architecture is designed for image classification tasks. It was introduced in 2016 and uses dense connections. This allows the network to use all attribute maps from all previous layers as input for the current layer. This makes the network more efficient in terms of the number of parameters and calculations and allows it to learn more complex features.

Network architecture

```
Model: "sequential"
_________________________________________________________________
 Layer (type) Output Shape Param #
=================================================================
 densenet201 (Functional) (None, 7, 7, 1920) 18321984

 global_average_pooling2d (G (None, 1920) 0
 lobalAveragePooling2D)

 dense (Dense) (None, 21) 40341

=================================================================
Total params: 18,362,325
Trainable params: 18,133,269
Non-trainable params: 229,056
_________________________________________________________________
```

The total number of weights in the entire system is 18,362,325, of which 18,133,269 are trainable.

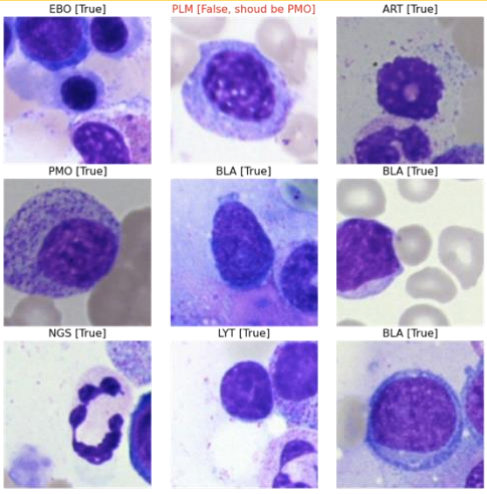
Runtimes and other features

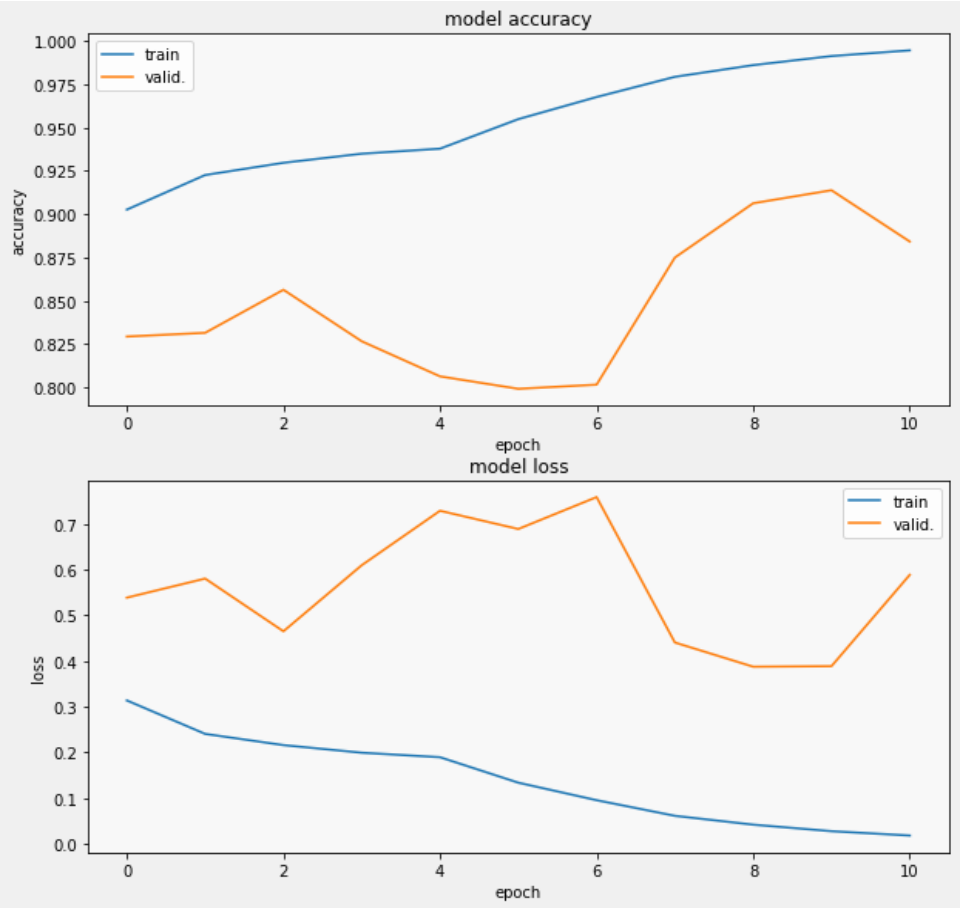
It took the system 7046.9 seconds to learn 191,248 images. (Hours: 1 | Minutes:57)
At the end of the run, we received the following parameters:

loss	0.0177
accuracy	0.9944
val_loss	0.5882
val_accuracy	0.8842

Testing on 160 random images from the Validation set, we got the following results:

loss	0.3082
accuracy	0.9125





In a few words

In this architecture, you can see unequivocally that overfitting has indeed occurred. It may be because the network is built of well-connected layers that require a lot of images to learn well. But here, we have several cell groups with a small number of images (especially the ABE class, with 27 photos in total).

Simple Vector Machine (SVM)

Is a Supervised algorithm that separates data into different classes using planes. It works very well with classification and regression tasks. The algorithm works fine with small amounts of information, which fits my conditions because of the ABE class. In addition, it gets along with details with lots of properties and a high amount of dimensions, which is also a critical element because the data I work with are images (and images contain lots of dimensions).

The first problem I immediately encountered is, in contrast to CNN, the **need for the algorithm to load all images into memory (RAM) to work with them**. Therefore, to solve this problem, I thought of three main ways to deal with this problem:

1. Selecting representative samples of the information from all Cell groups - the
   The idea is to randomly take from each class a predefined amount of images to arrive at a representative sample of the class and thus lower the total quantity of photos.
2. Lowering the resolution of the images -
   This option will necessarily degrade the quality but will allow us to run the algorithm on many images.
3. Another option is to extract attributes—instead of using the pixel values of the images, we extract features such as image edges, texture, or color histograms.

Unfortunately, because of the time limit of this project, I chose to go only with the first approach and not check the other techniques because processing all the information required a lot of time and allocating more resources to keep the data. The third way requires quite a bit of processing time as well.

Selection of representative samples of information from all cells

I decided to use the image repository that contains the produced images so that I could classify the ABE images better (this is the classes with the least images, whereas before the creation of the pictures, I had a total of 8 photos in the class, and now there are 72).

After copying all the pictures to the new repository, I was left with $72 * 21 = 1{,}512$ images.

Kernel functions

The SVM algorithm uses different kernel functions to divide the information efficiently. I chose to test 3 familiar kernel functions to compare them.

1. Linear Kernel
2. Polynomial Kernel
3. Radial Basis Function (RBF)

Linear Kernel

Linear Kernel is a simple and efficient kernel that can be used for problems when data can be linearly separated. It calculates the Dot Product between two input vectors, representing them as a scalar, and maps the input data to a higher dimension property space where they can be linearly separated.

Runtimes and other features

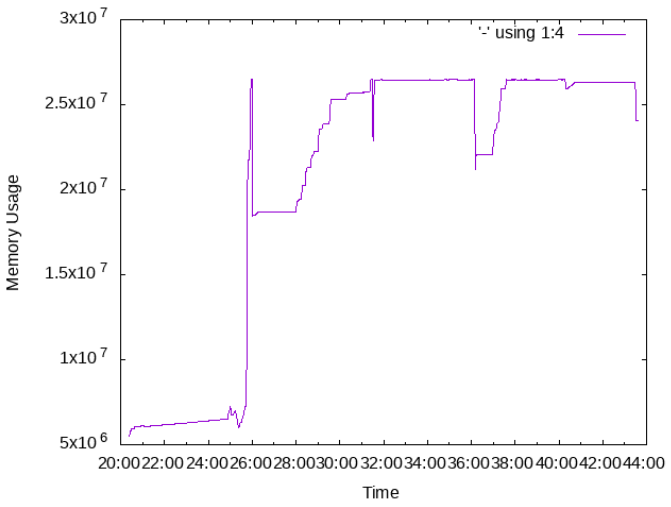
It took the system 14.6 minutes to learn 1,512 images with an accuracy of 24.4%

	precision	recall	f1-score	support
ABE	0.54	0.44	0.48	16
ART	0.13	0.17	0.15	12
BAS	0.20	0.11	0.14	9
BLA	0.11	0.12	0.11	17
EBO	0.35	0.35	0.35	17
EOS	0.20	0.13	0.16	15
FGC	0.38	0.18	0.24	17
HAC	0.55	0.67	0.60	9
KSC	0.71	0.88	0.79	17
LYI	0.14	0.08	0.10	13
LYT	0.11	0.18	0.13	11
MMZ	0.11	0.08	0.09	13
MON	0.21	0.21	0.21	14
MYB	0.12	0.10	0.11	21
NGB	0.25	0.25	0.25	16
NGS	0.07	0.09	0.08	11
NIF	0.18	0.15	0.16	20
OTH	0.43	0.45	0.44	20
PEB	0.27	0.25	0.26	12
PLM	0.04	0.08	0.05	12
PMO	0.00	0.00	0.00	11
accuracy			0.24	303
macro avg	0.24	0.24	0.23	303
weighted avg	0.25	0.24	0.24	303

Legend:
Precision-The rate of true positive predictions among all positive forecasts.
Recall – The rate of true positive predictions among all actual positive observations.
F1-score-the harmonic mean of accuracy and recall- a balance between the two.
Support-The number of observations in each group.

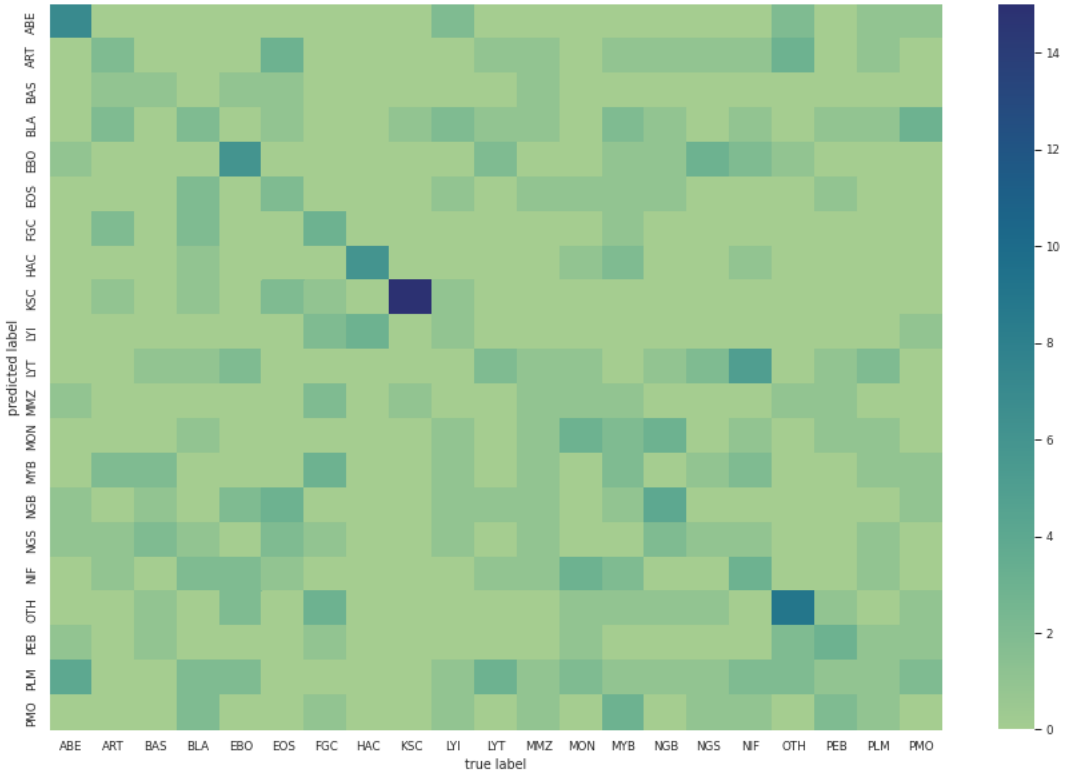
Use of memory during training

(Time is in seconds| The use is in KB)



From the graph, you can see that after the training (in the 26th second), the memory reached its peak at 23 GB and remained stable.

Heat map of test results



According to the map, you can see that the algorithm classified the images in a not-so-good shape and does confuse different types of images. Another thing that I was able to identify is that you can see that for HAC and KSC, the identification was very good.

Polynomial Kernel

A polynomial kernel allows the input to be manipulated to a higher dimensional space where the data can be linearly separated or separated using a Polynomial function. This allows us to divide more complex parts. In this work, I used a level 3 polynomial kernel function because it was the limit of the hardware I had.

Runtimes and other features

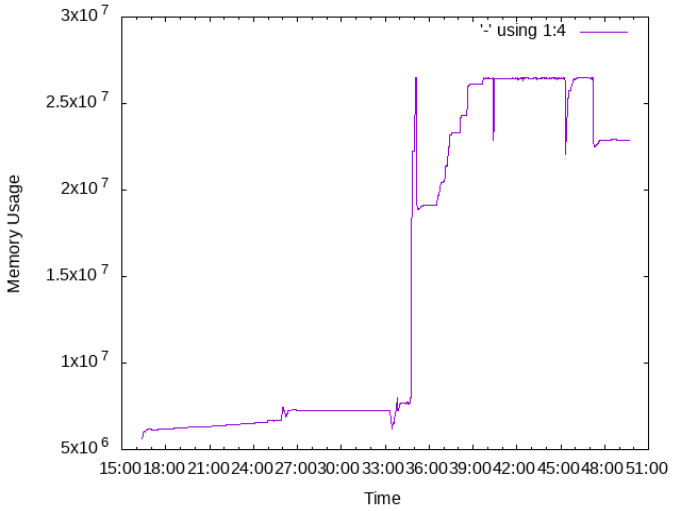
It took the system 12.5 minutes to learn 1,512 images with an accuracy of 24. 7%

	precision	recall	f1-score	support
ABE	0.61	0.69	0.65	16
ART	0.07	0.08	0.08	12
BAS	0.20	0.11	0.14	9
BLA	0.08	0.12	0.10	17
EBO	0.33	0.35	0.34	17
EOS	0.20	0.20	0.20	15
FGC	0.20	0.18	0.19	17
HAC	0.50	0.78	0.61	9
KSC	0.77	0.59	0.67	17
LYI	0.25	0.08	0.12	13
LYT	0.08	0.09	0.09	11
MMZ	0.09	0.08	0.08	13
MON	0.17	0.14	0.15	14
MYB	0.13	0.10	0.11	21
NGB	0.14	0.12	0.13	16
NGS	0.14	0.18	0.16	11
NIF	0.18	0.10	0.13	20
OTH	0.67	0.60	0.63	20
PEB	0.21	0.25	0.23	12
PLM	0.08	0.17	0.11	12
PMO	0.06	0.09	0.07	11
accuracy			0.25	303
macro avg	0.25	0.24	0.24	303
weighted avg	0.26	0.25	0.25	303

Legend:
Precision-The rate of true positive predictions among all positive forecasts.
Recall – The rate of true positive predictions among all actual positive observations.
F1-score-the harmonic mean of accuracy and recall, which is a balance between the two.
Support-The number of observations in each group.

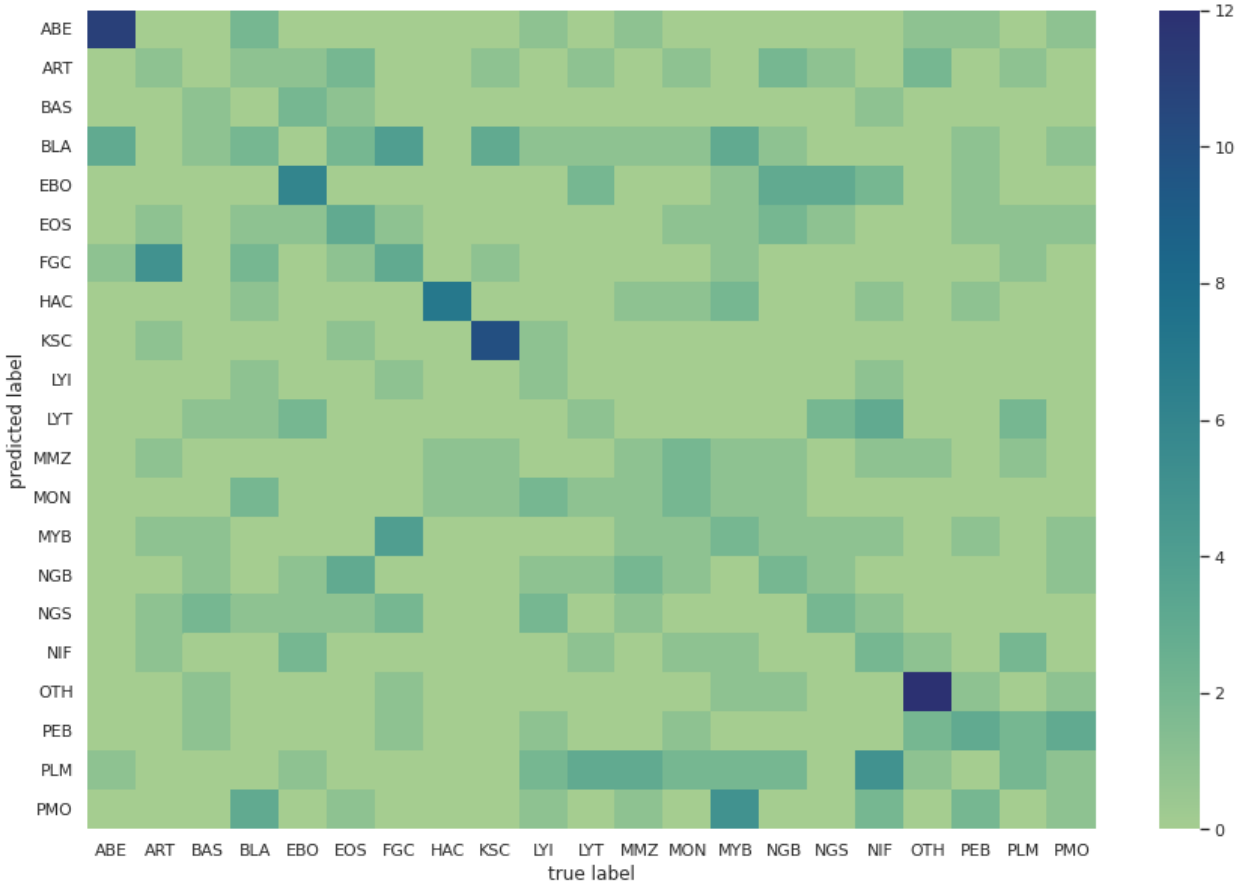
Use of memory during training

(Time is in seconds | The use is in KB)



From the graph, it can be seen that after the training (in the 36th second), the memory reached its peak at 23Gb and remained stable.

Heat map of test results



From the map, you can see that the polynomial method is indeed better but still not good enough. There are several categories of images that have reached impressive levels of accuracy, such as KSC, which reached an accuracy of 67%.

Radial Basis Function (RBF)

A radial kernel transforms the input data into a higher dimensional space where it can be linearly separated. The function creates a non-linear decision boundary (as opposed to the polynomial kernel). In addition to the function, there is a value called a gamma value that controls the complexity of the decision boundary.

Runtimes and other features

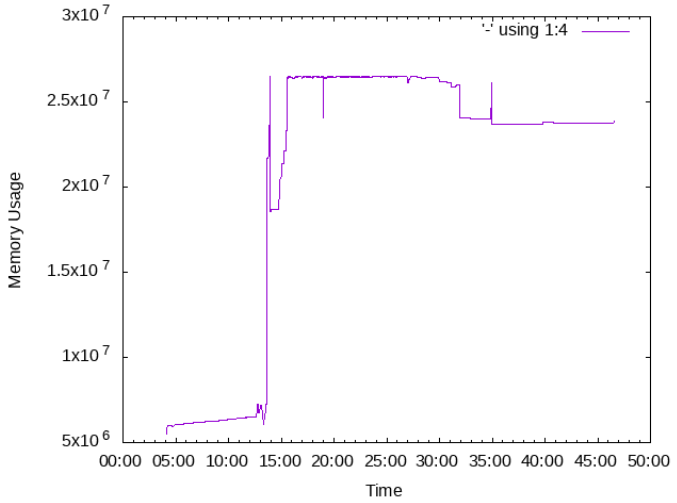
It took the system 13.3 minutes to learn 1,512 images with 33.3% accuracy

	precision	recall	f1-score	support
ABE	0.57	0.81	0.67	16
ART	0.14	0.17	0.15	12
BAS	0.50	0.22	0.31	9
BLA	0.33	0.29	0.31	17
EBO	0.50	0.47	0.48	17
EOS	0.09	0.07	0.08	15
FGC	0.42	0.47	0.44	17
HAC	0.47	1.00	0.64	9
KSC	0.62	0.94	0.74	17
LYI	0.44	0.31	0.36	13
LYT	0.00	0.00	0.00	11
MMZ	0.00	0.00	0.00	13
MON	0.00	0.00	0.00	14
MYB	0.17	0.14	0.15	21
NGB	0.17	0.12	0.14	16
NGS	0.12	0.18	0.15	11
NIF	0.29	0.20	0.24	20
OTH	0.83	0.75	0.79	20
PEB	0.29	0.33	0.31	12
PLM	0.17	0.25	0.20	12
PMO	0.00	0.00	0.00	11
accuracy			0.33	303
macro avg	0.29	0.32	0.29	303
weighted avg	0.31	0.33	0.31	303

Legend:
Precision-The rate of true positive predictions among all positive forecasts.
Recall – The rate of true positive predictions among all actual positive observations.
F1-score-the harmonic mean of accuracy and recall, which is a balance between the two.
Support-The number of observations in each group.

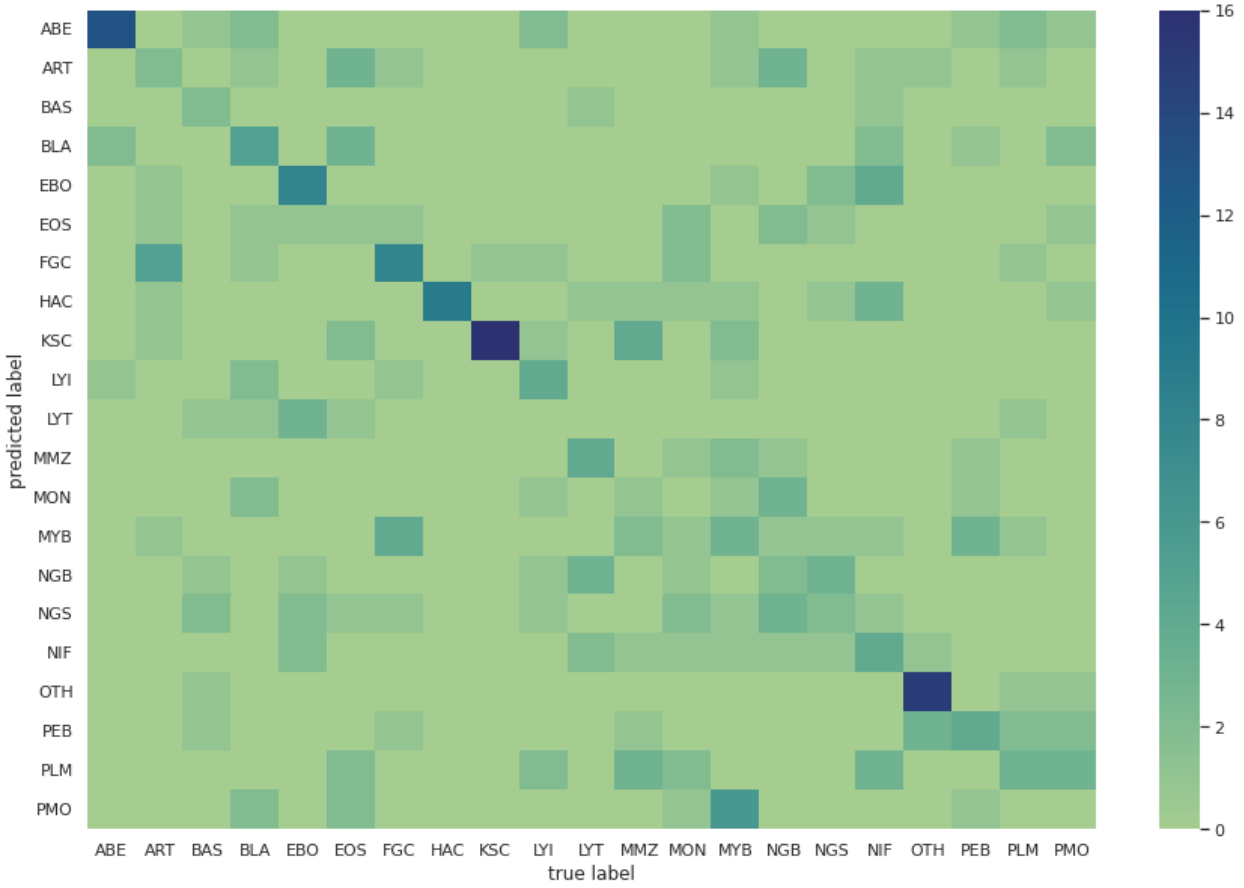
Use of memory during training

(Time is in seconds | The use is in KB)



From the graph, it can be seen that after the training (at 15 seconds), the memory reached its peak at 23 GB and remained stable.

Heat map of test results



From the map, you can also see that this method is indeed more effective, and you can actually notice that we have created a diagonal, which is our main goal. This diagonal will indicate that all the tests have been successful.

A few words about the different methods

Compared to the three different kernels, it can be clearly seen that the amount of memory consumed by all of them is the same and does not change much (but still, for a really small amount of images, we consume a lot of memory).
On the other hand, in terms of system performance, the SVM algorithm ran on a computer with 2 cores and optimized memory, managed to cope well with the task, and reached a level of accuracy of 33% at best.

I'm not sure that if I brought more images and access to more memory to the computer, we would see more accurate results here, but it's surely something that I will have to look into at some point.

Conclusion and reasoning

After running 3 different types of Convolutional Neural Networks architectures and also running SVM with 3 different Kernel types, I came to some important conclusions regarding the different algorithms and methods of working with them.

hardware

First of all, regarding hardware, the two methods have completely different hardware requirements, which can be the most important factor when selecting one of them because we don't have a way to change the hardware (like we can change the shape of the information).
After the runs, the two main components that came up were that for CNN algorithms, the main requirement in the system was the processing power (CPU/GPU)—in my tests, I worked with a dedicated Google machine for training networks, in order to lower the runtimes and get results faster. TPU computers are not always available to everyone, and access to them can be chargeable.
Upon an initial running attempt (on a basic architecture that included only one single convolution layer), My PC with an Apple M1 processor worked continuously for 4 whole days until the result was received. So running such a large amount of information with a large number of dimensions will also not be possible for everyone.

On the other hand, the SVM algorithm managed to run superbly on a processor with only 2 cores and access to a very large amount of memory 32GB managed to cope with the task (on a smaller amount of information) in a few minutes.

I will also note that taking into account the cost of hardware, it is cheaper to buy memory than a quality graphics card. That is why in the choice of algorithms, the main question should be related to hardware first of all.

Runtimes

Another important feature that is very dependent on hardware (as I explained earlier) is runtime. With access to better and dedicated hardware for the task, we can have larger runtimes, but if you ignore the hardware and observe how the algorithms work, you can notice that training a neural network takes much longer than using the SVM algorithm (assuming that we teach large and complex networks and not networks with several single layers). Each image that passes through the network must update a vast amount of weights in the network, Something that takes a lot of time. Compared to SVM, which doesn't need to update values and doesn't have an entire "network."

Memory

As indicated in the section on the hardware, the SVM algorithm puts much more pressure on the memory and requires a large amount of RAM. On the other hand, CNN does not require reading the data directly from memory and can allow itself to read data straight from a hard drive for training (memory that costs us much less than RAM).

Model size

From the model experiments I was able to save for CNN, the size of the models was between 211MB and 270MB, which is a negligible weight compared to the size of the models I got for running SVM. The SVM is ~6GB in size, which is 24 times more memory than CNN! And it is important to note that the number of pictures with which I trained the SVM model was 113 times smaller! (~1,500 images in the SVM model and ~170,000 images for CNN).

Accuracy

The no less important part is the accuracy. According to the tests (and the access to the hardware I had), I came to the conclusion that the preferred method and algorithm for the experiment would be a CNN algorithm with a ResNet architecture because, according to the data, it seems that this is the least overfitted model (the graphs of the training results and the results of the loss function are very close to each other and the function of accuracy aims up and at the same time the function of the loss aims down, which shows us that indeed the results are good and also the chances of overfitting are small).

In addition, if I had the opportunity to test the SVM algorithm with the same amount of information, I could have known more clearly if it was better for me to choose SVM instead. Taking into account the element of the cost of hardware, if the results were the same, it would be better to choose SVM – Due to the costs of Hardware and the computing time.

Bibliography and sources

As I mentioned at the beginning of the work, I used a lot of Google resources to work with the data; if I had not used the help of Google resources, I would not be able to complete this task.

Bibliography

[1] C. K. S. M. C. H. T. &. M. C. Matek, "An Expert-Annotated Dataset of Bone Marrow Cytology in Hematologic Malignancies [Data set]. The Cancer Imaging Archive.," 2021. [Online]. Available: https://doi.org/10.7937/TCIA.AXH3-T579.

[2] C. K. S. M. C. H. T. a. M. C. Matek, "Highly accurate differentiation of bone marrow cell morphologies using deep neural networks on a large image dataset.," [online]. Available: https://doi.org/10.1182/blood.2020010568.

[3] V. B. S. K. F. J. K. J. K. P. M. S. P. S. M. D. P. M. T. L. P. F. Clark K, "The Cancer Imaging Archive (TCIA): Maintaining and Operating a Public Information Repository," *Journal of Digital Imaging, Volume* 26, No. Number 6, pp. 1045-1057, 2013.

[4] X. Z. S. R. J. S. Kaiming He, "Deep Residual Learning for Image Recognition," [Online]. Available: https://arxiv.org/abs/1512.03385.

[5] V. V. S. I. J. S. Z. W. Christian Szegedy, "Rethinking the Inception Architecture for Computer Vision," [online]. Available: https://arxiv.org/abs/1512.00567.

[6] Z. L. L. v. d. M. K. Q. W. Gao Huang, "Densely Connected Convolutional Networks," [online]. Available: https://arxiv.org/abs/1608.06993.

[7] L. N. Smith, "Cyclical Learning Rates for Training Neural Networks," *2017 IEEE Winter Conference on Applications of Computer Vision,* 2017.

NotebookforColab CNN - https://colab.research.google.com/drive/13OPTpX1CQ_1jvgRoOKa-iVAE_UlKwIeN?usp=sharing

NotebookforColabSVM - https://colab.research.google.com/drive/1rpeG-CLLsboep4LihOFDXacaOazCZNtc?usp=sharing

Access to the Google Storage Bucket processed data will be granted upon request.