Università di Pisa

Master Degree in Artificial Intelligence and Data Engineering

Computational Intelligence and Deep Learning

Identify immature leukemic blasts from normal cells

Simone Landi

1. Introduction

Acute Lymphoblastic Leukemia (ALL) represents a form of blood and bone marrow cancer that is particularly significant, especially in the pediatric context. This condition is characterized by the rapid proliferation of immature lymphoid cells called lymphoblasts, which tend to replace normal blood and bone marrow cells. ALL is, in fact, the most common type of leukemia in children, with a higher incidence between the ages of two and five, and is sadly known to be the leading cause of death among pediatric cancers.

Early and accurate diagnosis of ALL is essential to significantly increase the chances of a cure and to ensure timely and effective treatment. However, identifying these malignant cells presents a considerable challenge for medical professionals. Morphologically, lymphoblasts often appear very similar to normal blood cells, making visual distinction under the microscope difficult. This morphological similarity complicates the diagnosis and requires a thorough and accurate analysis by experienced pathologists.

In response to this complexity, the use of computer-assisted tools and advanced technologies has become crucial. Automating the process of identifying malignant cells through deep learning architectures can play a key role in making the work of pathologists and oncologists more efficient and precise. These tools not only allow for faster diagnosis but also data-driven analysis, which can lead to more informed therapeutic decisions, thus improving the prospects of patients with ALL.

With this goal in sight and taking inspiration from the ISBI 2019 C-NMC Challenge, this project is dedicated to the development of a deep learning architecture specifically designed to tackle the challenge of classifying cancer cell images.

2. Related works

In recent years, the emergence of convolutional neural networks has paved the way for impressive achievements in various image classification tasks, including those within the realm of medical imaging. However, when it comes to the classification of leukemic B-lymphoblast cells, the challenge remains formidable even when harnessing these cutting-edge technologies.

The intrinsic visual resemblance between abnormal and normal cells, combined with the substantial diversity among cells from different subjects, adds an additional layer of complexity to this task. This is precisely why the ISBI Challenge 2019 was initiated, and even upon its conclusion, many researchers continue to delve into this intricate issue.

The use of machine learning techniques for Acute Lymphoblastic Leukemia (ALL) classification has witnessed significant research endeavors in recent years. The influence of artificial intelligence and deep learning technologies on the landscape of medical image analysis, particularly in the sphere of disease diagnosis, cannot be overstated. Simultaneously, multiple investigations have been undertaken, focusing on the detection and multi-classification of ALL, while taking into consideration the distinct stages and subtypes of the disease.

Here are a few outcomes achieved by the participants of the challenge. It's important to bear in mind that the ultimate test set, which is used to determine the challenge winners and for which we lack access to the labels, restricts the extent to which we can directly compare my results with the current state of the art.

- F. Xiao, R. Kuang, Z. Ou & B Xiong (2019) “DeepMEN: Multi-model Ensemble Network for B-Lymphoblast Cell Classification” propose Deep Multi-model Ensemble Network (DeepMEN), an ensemble of six deep learning models. Their model obtains an accuracy of 0.8856 on the final test set.
- C. Marzahl, M. Aubreville, J. Voigt & A. Maier (2019) “Classification of Leukemic B-Lymphoblast Cells from Blood Smear Microscopic Images with an Attention-Based Deep Learning Method and Advanced Augmentation Techniques” propose a solution based upon advanced augmentation techniques and transfer learning. Additionally, they incorporate a basic attention mechanism based on a region proposal subnetwork. Their ensemble of ResNet18 networks achieves an accuracy of 0.8284 on the final test set.
- E. Verma & V. Singh (2019) “ISBI Challenge 2019: Convolution Neural Networks for B-ALL Cell Classification” propose an ensemble solution that combines variants of classifiers

designed with pretrained MobileNetV2 architecture. Their model achieves an accuracy of 0.8947 on the final test set.

There are also many users on the Kaggle platform who have attempted to perform the cell classification task and still obtained similar results to those reported above.

3. Dataset

The dataset contains 15114 images of of white blood cells which are classified into 2 classes: normal (hem) vs cancer (all). The distribution of the images per classes is shown below:

	ALL cells	HEM cells	Total
Training set	7272	3389	10661
Validation set	1219	648	1867
Test set	Unknown	Unknown	2586
Total	Unknown	Unknown	15144

Original data distribution

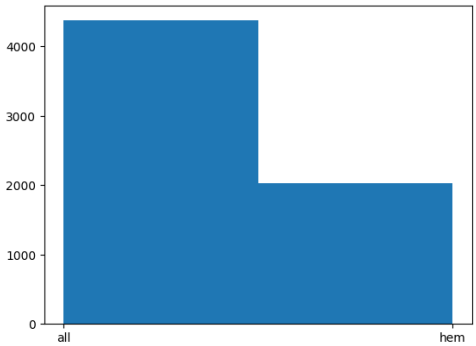
Regarding the ground truth labels for the test set, they were not provided. In addition, I found limitations with Google Colaboratory in terms of GPU utilization, making it impractical to use the entire dataset for training due to excessively long training times with CPU resources alone. To address this issue, I made the decision to work exclusively with the training set, which I subsequently divided into three subsets: the training set, the validation set, and the test set. I allocated 60% of the data to the training set, 20% to the validation set, and another 20% to the test set. When selecting images for each of these sets, I aimed to do so randomly while ensuring that the distribution of data closely matched that of the original dataset.

The distribution of the images per classes of the training set is shown below:

	ALL cells	HEM cells	Total
Training set	4373	2023	6396
Validation set	1436	696	2132
Test set	1463	670	2133
Total	7272	3389	10661

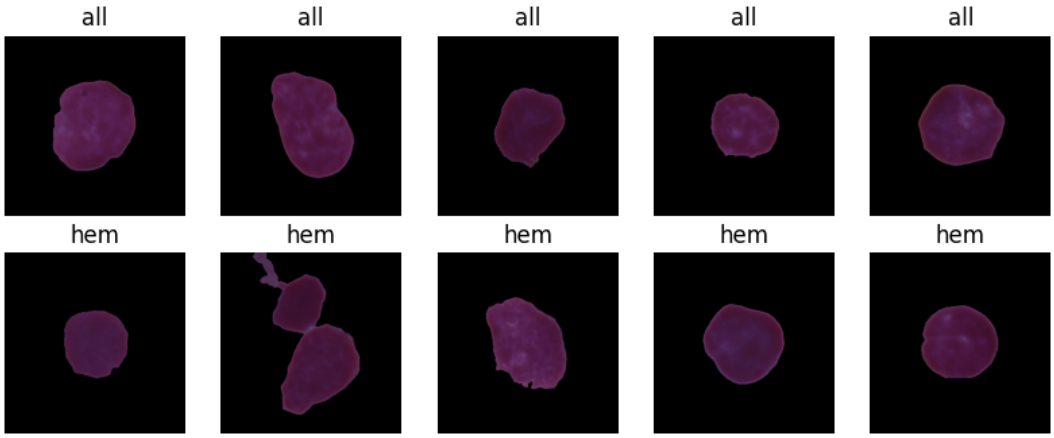
New data distribution

I also report the diagram related to the new training set:



As can be seen, the training set is not balanced, so it was necessary to apply subsampling and increase the number of images in the "hem" class by performing data augmentation.

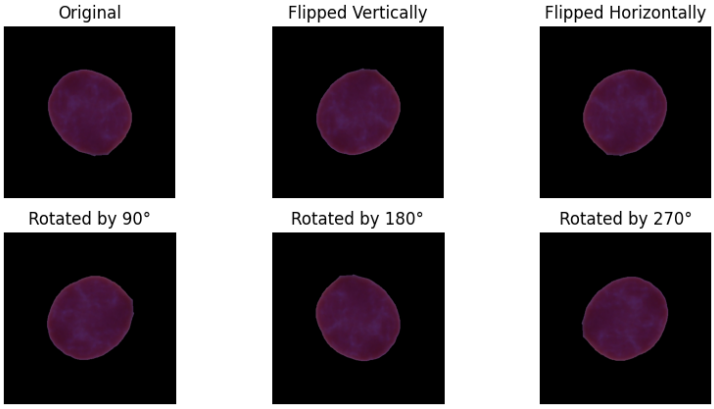
Examples of the images in the dataset can be seen in the following figure:



The individual microscopic images in my dataset are three-channel images with dimensions of 450×450 pixels. These images closely resemble real-world microscopy images, as they exhibit some staining noise and illumination errors. However, it's important to note that many of these errors have been mitigated or corrected through a process known as stain color normalization. Stain color normalization is a technique used to enhance the consistency and accuracy of color in stained images, often employed in the field of digital pathology to improve image quality and reduce variability caused by staining artifacts and lighting inconsistencies.

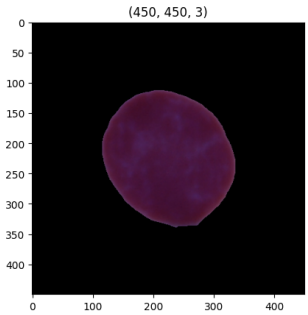
4. Data Preprocessing

As we saw in the last chapter, the number of images for each class is unbalanced, so I decided to employ subsampling and data augmentation to adjust the imbalance in class distribution. To resolve this inequality, my goal is to equalize the image count for both classes. In order to achieve balance, I'll apply a random sub-sampling on the larger class, which corresponds to cancer cells, to select the target number of images (3198). For the smaller class, I'll implement data augmentation. New images were generated from the original ones by randomly executing either a vertical or horizontal flip, or by introducing a rotation with a random angle within the range of 90°, 180°, and 270°, as illustrated below.



Example of flips and rotations

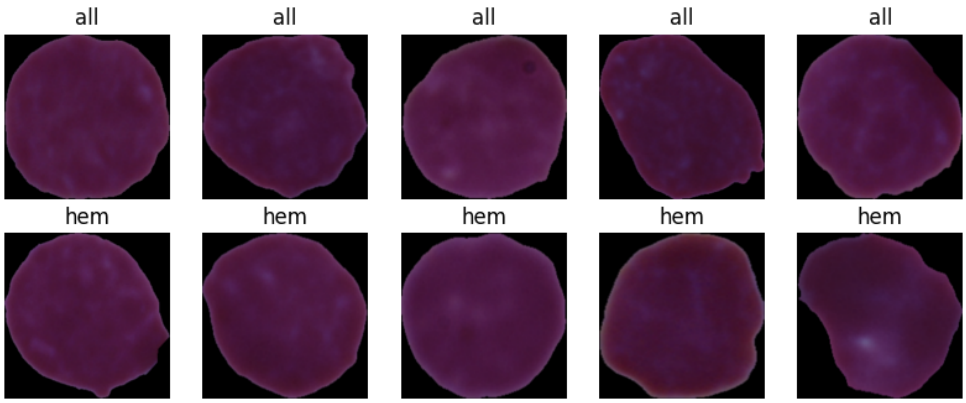
The dataset's images feature a black background, and all of them include black margins that do not encompass any portion of the cell. These margins could potentially scatter the classification results, diverting attention away from the focal point of interest.



Image's shape example

It's evident that each image comprises three channels. What I've done is to trim the black borders from the images, retaining only the segments containing the cell parts, and then resized the images. The new dimensions for the images were set to 224x224, which is

the standard size compatible with all pre-trained CNNs commonly used. This choice eliminates the need for an additional resizing step.



Images after cropping and resizing

5. Experiments

In order to identify the best-performing model for our dataset, I experimented with various solutions involving different hyper-parameters. I explored custom-built CNNs, pre-trained CNNs, and even Vision Transformers. The method employed to determine the optimal configurations for each model type involved a trial-and-error approach.

5.1. Evaluations techniques

Each trained model will be evaluated using our test set, and we will record all evaluation statistics, apart from the confusion matrix. We will also pay close attention to other valuable information, such as accuracy on the training and validation set history after each epoch. This will provide us with insights into the training phase. In addition to accuracy and loss, we will give special focus to precision on the "hem" class.

The significance of this last metric lies in the potential consequences of misclassification by our model. If the cell pertains to a healthy patient but our model misclassifies them as ill, the patient will undergo medical supervision and additional tests, eventually confirming their good health and allowing them to resume their normal life promptly. Conversely, if an unhealthy patient is wrongly classified as healthy, they might be discharged without further examination, potentially missing necessary treatment.

Precision on the hem class is crucial for evaluating a model's capability to prevent these types of issues.

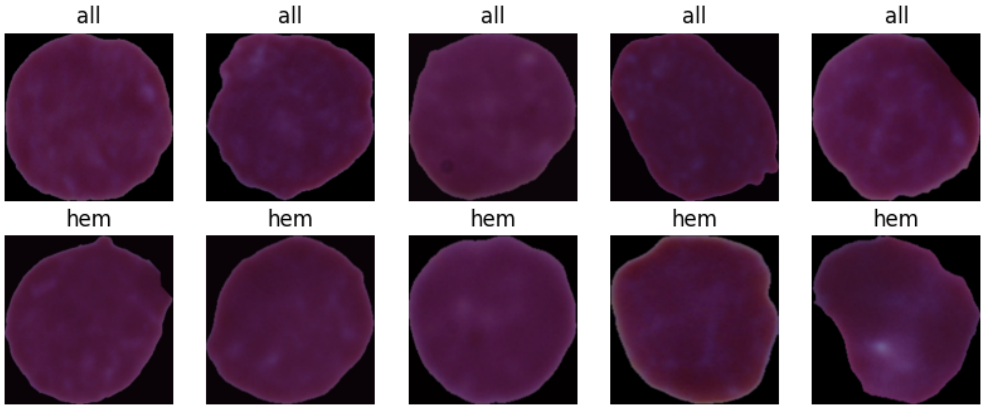
5.2. Handling overfitting

To address the issue of overfitting, I've incorporated three techniques that will be applied to each model: Early Stopping, Data Augmentation and Dropout (if necessary).

The Early Stopping technique assesses the loss on the validation set after each training epoch. If there's no improvement for a specified number of epochs, the training is halted because it indicates that the model is overfitting the training data. The model retained is the one achieved after the epoch with the best loss on the validation set. The parameter that determines the number of non-improving epochs before stopping training is referred to as "patience." In our models, we will use a patience value ranging from 3 to 10, depending on the specific model.

The Data Augmentation technique will be employed to ensure that our model does not encounter the same image in exactly the same way during training. For this purpose, I have introduced a Data Augmentation input layer into each model. This layer will perform random horizontal and vertical flips on each image and apply random contrast adjustments within a

range of [-20%, +20%]. An illustration of the application of this transformation can be observed in the image below, where Data Augmentation has been applied to the images featured in the previous chapter.



Images after cropping, resizing and random transformation

Another technique employed to mitigate overfitting is the use of a Dropout layer. If it is deemed that the other two techniques are insufficient to address the overfitting issue, a dropout layer is introduced to the model.

5.3. CNN from scratch

The first experiment involved constructing a CNN from scratch. The CNN's structure consists of 2D Convolutional layers with a kernel size of 3, followed by a MaxPooling layer with a pool_size of 2. The initial Convolutional layer employs 32 filters, and as the network progresses toward the output, the number of filters doubles. Thus, after the Data Augmentation layer, we start with a Conv2D layer featuring 32 filters and a kernel size of 3, followed by MaxPooling with a pool_size of 2. Subsequently, there is a Conv2D layer with 64 filters, and so on, until we reach a Conv2D layer with 256 filters, which is followed by the last MaxPooling (with a pool size adjusted to 5). This serves as our foundational network, and in the upcoming chapters, we will conduct various experiments by introducing different networks based on this foundation.

5.3.1. One dense layer and a dropout

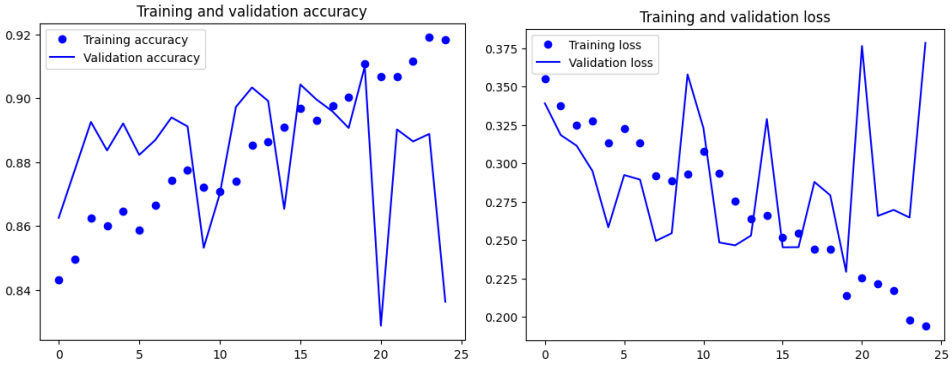
The first trial was building on top of the base CNN a Dense layer with 256 neurons.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 224, 224, 3)]	0
sequential (Sequential)	(None, 224, 224, 3)	0
rescaling (Rescaling)	(None, 224, 224, 3)	0
conv2d (Conv2D)	(None, 224, 224, 32)	896
max_pooling2d (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_1 (Conv2D)	(None, 112, 112, 64)	18496
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_2 (Conv2D)	(None, 56, 56, 128)	73856
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 128)	0
conv2d_3 (Conv2D)	(None, 28, 28, 256)	295168
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 256)	0
flatten (Flatten)	(None, 6400)	0
hidden_classifier (Dense)	(None, 256)	1638656
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 1)	257

Total params: 2027329 (7.73 MB)
Trainable params: 2027329 (7.73 MB)
Non-trainable params: 0 (0.00 Byte)

Summary of the model

The results of the training are shown below.

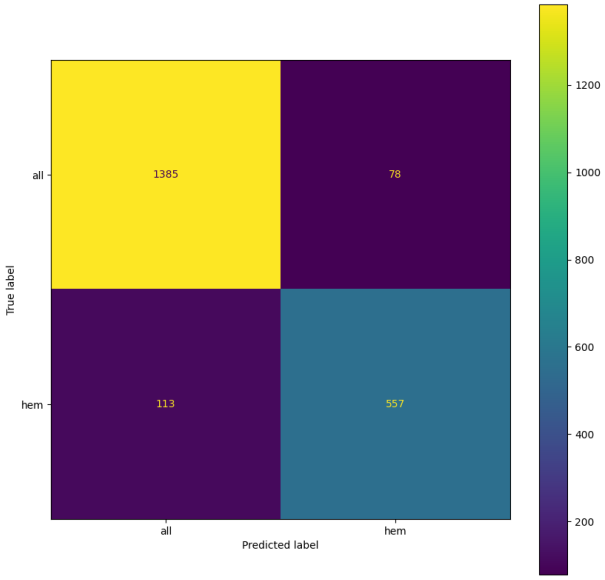


The evaluation of the model against the test set is shown below.

Accuracy on test set	Loss on test set
0.9105	0.2284

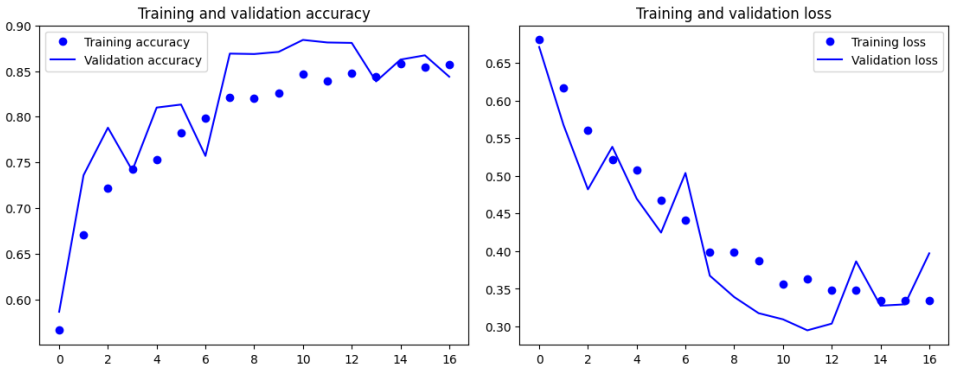
Classification report:

	precision	recall	f1-score	support
all	0.9246	0.9467	0.9355	1463
hem	0.8772	0.8313	0.8536	670
accuracy			0.9105	2133
macro avg	0.9009	0.8890	0.8946	2133
weighted avg	0.9097	0.9105	0.9098	2133



We achieved a test loss of 0.2284 and an accuracy of 0.9105 in our initial trial. Since this was our first attempt, we were uncertain about whether these results were considered good or not. To explore further, we initially reduced the network's capacity by adjusting the number of neurons in the dense hidden layer to 128. Subsequently, we experimented with the opposite approach, doubling the number of neurons to 512. The outcomes of these two trials are presented below.

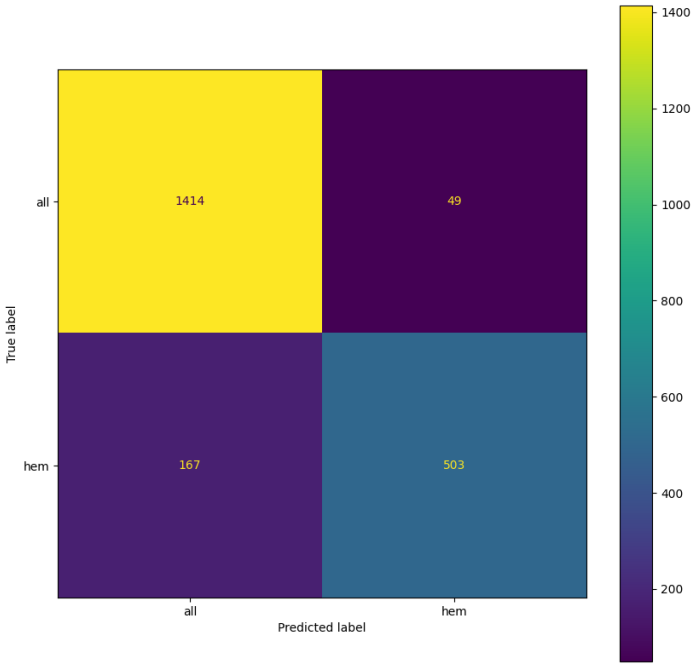
Training of the 128 dense neurons model.



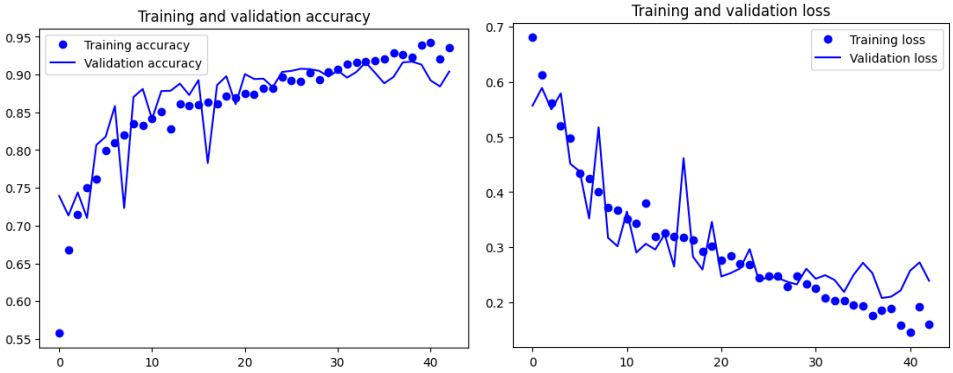
The evaluation of the model against the test set is shown below.

Accuracy on test set	Loss on test set
0.8987	0.2710

```
Classification report:
              precision    recall  f1-score   support

         all     0.8944    0.9665    0.9290      1463
         hem     0.9112    0.7507    0.8232       670

    accuracy                         0.8987      2133
   macro avg     0.9028    0.8586    0.8761      2133
weighted avg     0.8997    0.8987    0.8958      2133
```



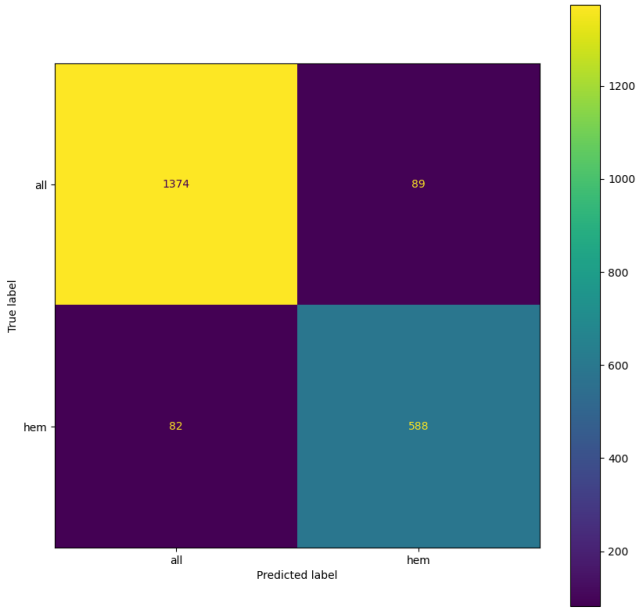
Training of the 512 dense neurons model.



The evaluation of the model against the test set is shown below.

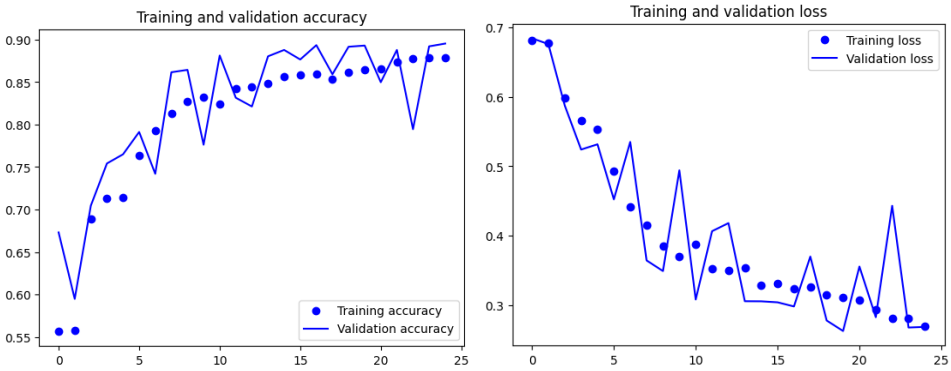
Accuracy on test set	Loss on test set
0.9198	0.2147

```
Classification report:
              precision    recall  f1-score   support

         all     0.9437    0.9392    0.9414      1463
         hem     0.8685    0.8776    0.8731       670

    accuracy                         0.9198      2133
   macro avg     0.9061    0.9084    0.9072      2133
weighted avg     0.9201    0.9198    0.9199      2133
```



It's evident that both models exhibit similar performance. The more complex model achieves a higher accuracy score, and from the training history, we observe a slight indication of overfitting towards the end of the training process. This is reflected in the accuracy and loss curves on the validation set diverging from those of the training set. In an attempt to address this issue, we experimented with increasing the dropout value to 0.5, which did result in a modest reduction in this phenomenon; however, it came at the cost of a decrease in the model's overall performance.

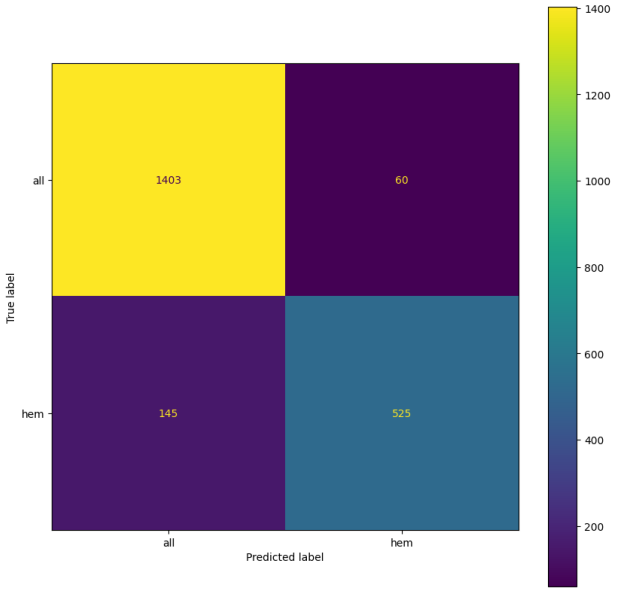
Training of the 512 dense neurons model with 0.5 Dropout.



The evaluation of the model against the test set is shown below.

Accuracy on test set	Loss on test set
0.9039	0.2508

```
Classification report:
              precision    recall  f1-score   support

         all     0.9063    0.9590    0.9319      1463
         hem     0.8974    0.7836    0.8367       670

    accuracy                         0.9039      2133
   macro avg     0.9019    0.8713    0.8843      2133
weighted avg     0.9035    0.9039    0.9020      2133
```



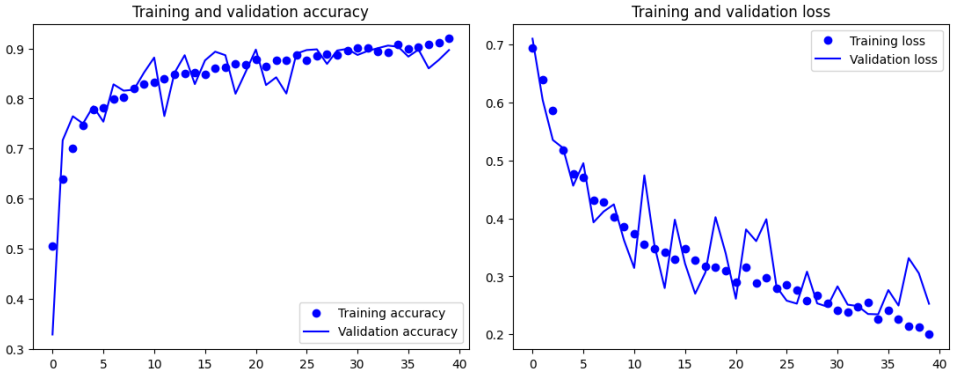
5.3.2. One convolutional layer

In the next trial, we introduced a Convolutional layer with 256 filters.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_5 (InputLayer)         [(None, 224, 224, 3)]     0

sequential (Sequential)      (None, 224, 224, 3)       0

rescaling_4 (Rescaling)      (None, 224, 224, 3)       0

conv2d_16 (Conv2D)           (None, 224, 224, 32)      896

max_pooling2d_16 (MaxPooli   (None, 112, 112, 32)      0
ng2D)

conv2d_17 (Conv2D)           (None, 112, 112, 64)      18496

max_pooling2d_17 (MaxPooli   (None, 56, 56, 64)        0
ng2D)

conv2d_18 (Conv2D)           (None, 56, 56, 128)       73856

max_pooling2d_18 (MaxPooli   (None, 28, 28, 128)       0
ng2D)

conv2d_19 (Conv2D)           (None, 28, 28, 256)       295168

max_pooling2d_19 (MaxPooli   (None, 9, 9, 256)         0
ng2D)

dropout_4 (Dropout)          (None, 9, 9, 256)         0

conv2d_20 (Conv2D)           (None, 9, 9, 256)         590080

dropout_5 (Dropout)          (None, 9, 9, 256)         0

flatten_4 (Flatten)          (None, 20736)             0

dense_4 (Dense)              (None, 1)                 20737

=================================================================
Total params: 999233 (3.81 MB)
Trainable params: 999233 (3.81 MB)
Non-trainable params: 0 (0.00 Byte)
```

Summary of the model

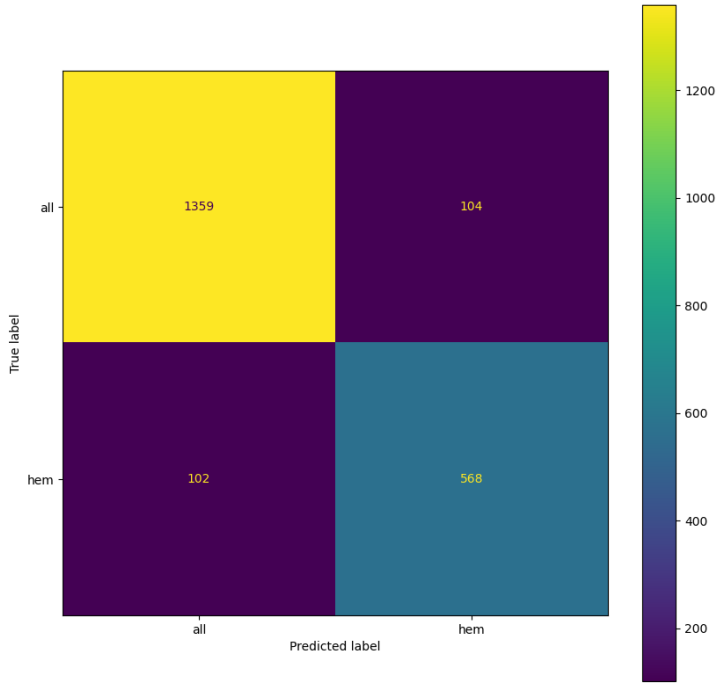
Training of the One Conv2D layer with 256 filters model.



The evaluation of the model against the test set is shown below.

Accuracy on test set	Loss on test set
0.9034	0.2308

```
Classification report:
              precision    recall  f1-score   support

         all     0.9302    0.9289    0.9295      1463
         hem     0.8452    0.8478    0.8465       670

    accuracy                         0.9034      2133
   macro avg     0.8877    0.8883    0.8880      2133
weighted avg     0.9035    0.9034    0.9035      2133
```



The results obtained were in line with the previous ones, finding no particular improvement in term of accuracy.

5.3.3. More complex Models

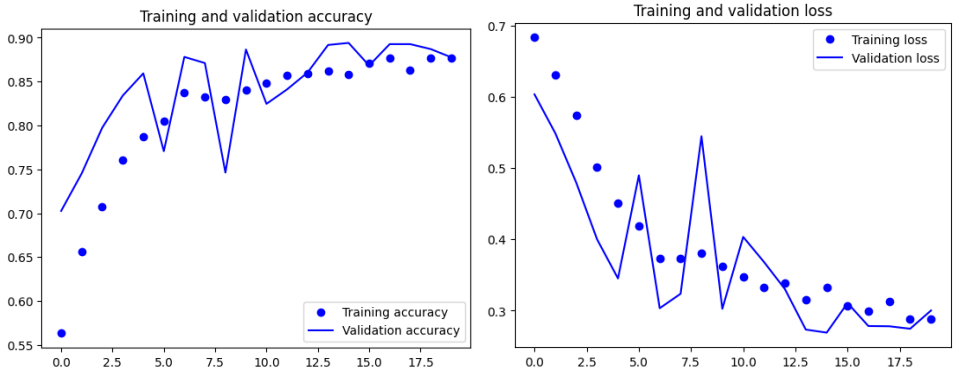
Following these trials, we aimed to further enhance the model's complexity by increasing the depth of the base model with a Conv2D layer containing 512 filters and two dropout layers.

Layer (type)	Output Shape	Param #
input_6 (InputLayer)	[(None, 224, 224, 3)]	0
sequential (Sequential)	(None, 224, 224, 3)	0
rescaling_5 (Rescaling)	(None, 224, 224, 3)	0
conv2d_21 (Conv2D)	(None, 224, 224, 32)	896
max_pooling2d_20 (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_22 (Conv2D)	(None, 112, 112, 64)	18496
max_pooling2d_21 (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_23 (Conv2D)	(None, 56, 56, 128)	73856
max_pooling2d_22 (MaxPooling2D)	(None, 28, 28, 128)	0
conv2d_24 (Conv2D)	(None, 28, 28, 256)	295168
max_pooling2d_23 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_25 (Conv2D)	(None, 9, 9, 512)	1180160
max_pooling2d_24 (MaxPooling2D)	(None, 3, 3, 512)	0
dropout_6 (Dropout)	(None, 3, 3, 512)	0
conv2d_26 (Conv2D)	(None, 3, 3, 512)	2359808
dropout_7 (Dropout)	(None, 3, 3, 512)	0
flatten_5 (Flatten)	(None, 4608)	0
dense_5 (Dense)	(None, 1)	4609

Total params: 3932993 (15.00 MB)
Trainable params: 3932993 (15.00 MB)
Non-trainable params: 0 (0.00 Byte)

Summary of the model

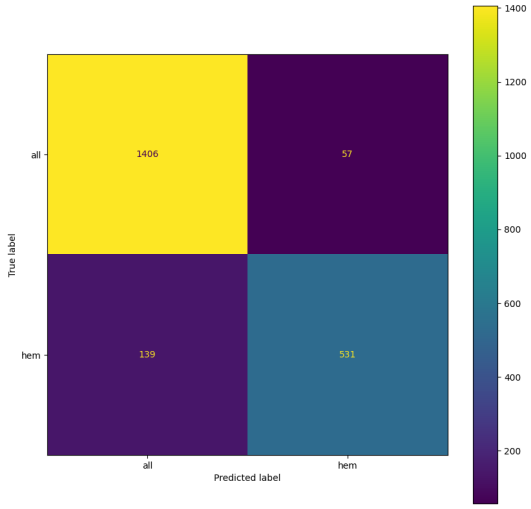
Training of the One Conv2D layer with 512 filters and an additional layer model.



The evaluation of the model against the test set is shown below.

Accuracy on test set	Loss on test set
0.9081	0.2463

```
Classification report:
              precision    recall  f1-score   support

         all     0.9100    0.9610    0.9348      1463
         hem     0.9031    0.7925    0.8442       670

    accuracy                         0.9081      2133
   macro avg     0.9065    0.8768    0.8895      2133
weighted avg     0.9078    0.9081    0.9064      2133
```



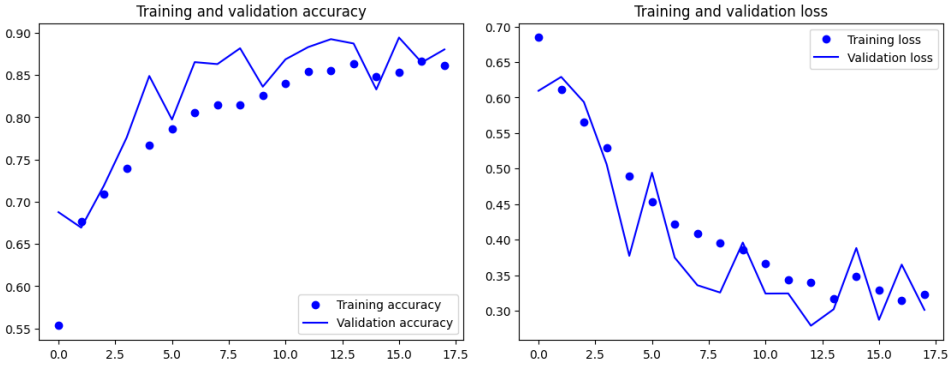
Following these trials, we aimed to further enhance the model's complexity by increasing the depth of the base model with a Conv2D layer containing 512 filters. On top of this, we experimented with various structures, including a Dense layer with 512 neurons, and even a configuration with two Dense layers, each featuring 512 neurons.

Layer (type)	Output Shape	Param #
input_8 (InputLayer)	[(None, 224, 224, 3)]	0
sequential (Sequential)	(None, 224, 224, 3)	0
rescaling_7 (Rescaling)	(None, 224, 224, 3)	0
conv2d_32 (Conv2D)	(None, 224, 224, 32)	896
max_pooling2d_30 (MaxPooling2D)	(None, 112, 112, 32)	0
conv2d_33 (Conv2D)	(None, 112, 112, 64)	18496
max_pooling2d_31 (MaxPooling2D)	(None, 56, 56, 64)	0
conv2d_34 (Conv2D)	(None, 56, 56, 128)	73856
max_pooling2d_32 (MaxPooling2D)	(None, 28, 28, 128)	0
conv2d_35 (Conv2D)	(None, 28, 28, 256)	295168
max_pooling2d_33 (MaxPooling2D)	(None, 9, 9, 256)	0
conv2d_36 (Conv2D)	(None, 9, 9, 512)	1180160
max_pooling2d_34 (MaxPooling2D)	(None, 3, 3, 512)	0
flatten_7 (Flatten)	(None, 4608)	0
hidden_classifier0 (Dense)	(None, 512)	2359808
dropout_10 (Dropout)	(None, 512)	0
hidden_classifier1 (Dense)	(None, 512)	262656
dropout_11 (Dropout)	(None, 512)	0
dense_7 (Dense)	(None, 1)	513

```
Total params: 4191553 (15.99 MB)
Trainable params: 4191553 (15.99 MB)
Non-trainable params: 0 (0.00 Byte)
```

Summary of the model

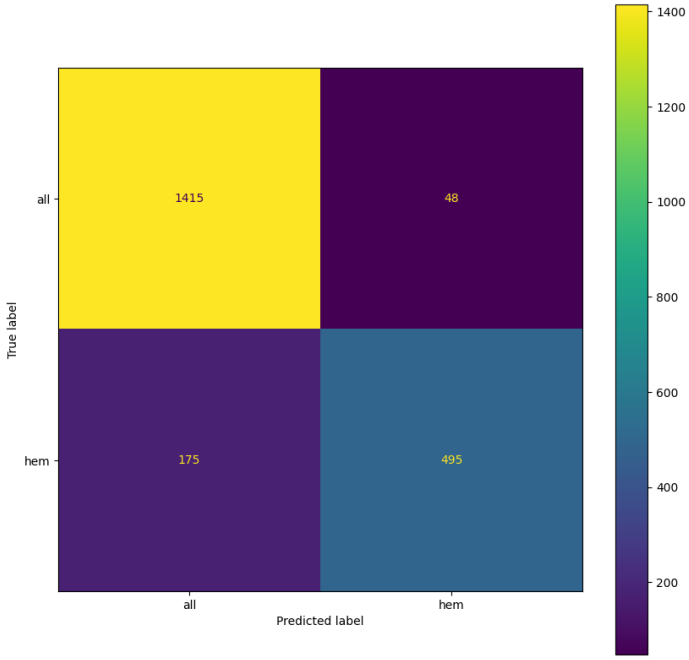
Training of the Two dense layer with 512 neurons and an additional layer model.



The evaluation of the model against the test set is shown below.

Accuracy on test set	Loss on test set
0.9039	0.2662

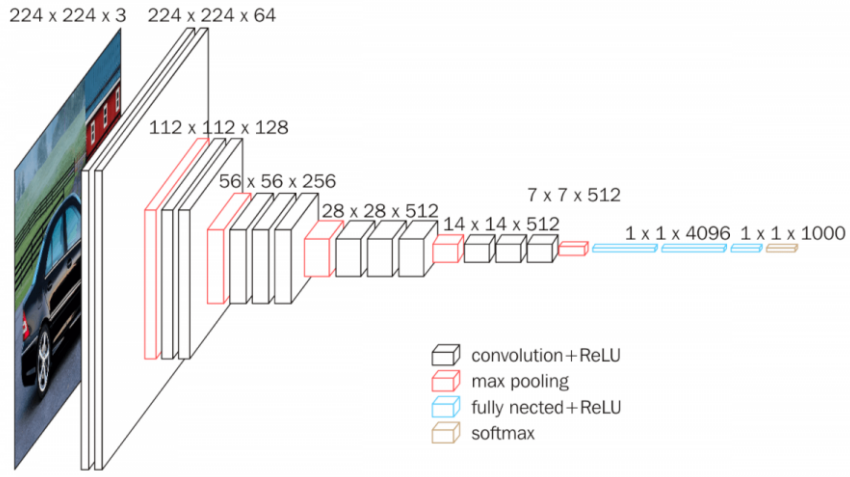
```
Classification report:
              precision    recall  f1-score   support

         all     0.9063    0.9590    0.9319      1463
         hem     0.8974    0.7836    0.8367       670

    accuracy                         0.9039      2133
   macro avg     0.9019    0.8713    0.8843      2133
weighted avg     0.9035    0.9039    0.9020      2133
```



All of these network variations exhibited good performances, but the most successful model we discovered was the one with one dense layer with 512 dense neurons and one dropout with 0.3, with an accuracy of 0.9198.

5.4. VGG16

VGG16 is the initial CNN that was employed, and its architecture is shown below.



VGG16 structure

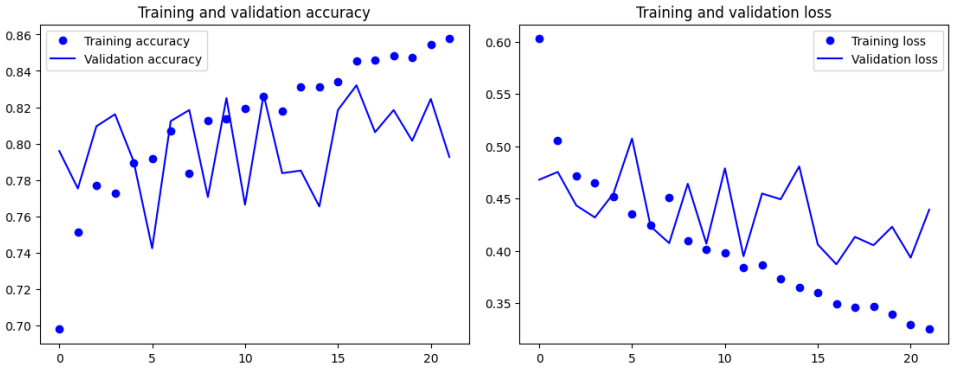
The image undergoes a series of convolutional layers, utilizing filters with a small receptive field of 3×3. These convolutional layers maintain a fixed stride of 1 pixel and include spatial padding to ensure the preservation of spatial resolution after convolution, with a 1-pixel padding for 3×3 convolutional layers. Spatial pooling involves the application of five max-pooling layers, which are placed after certain convolutional layers (not all convolutional layers are followed by max-pooling). Max-pooling is executed using a 2×2 pixel window and a stride of 2. Following the convolutional layers, three Fully-Connected (FC) layers are employed, each incorporating rectification (ReLU) non-linearity in their hidden layers.

5.4.1. MLP built on top training

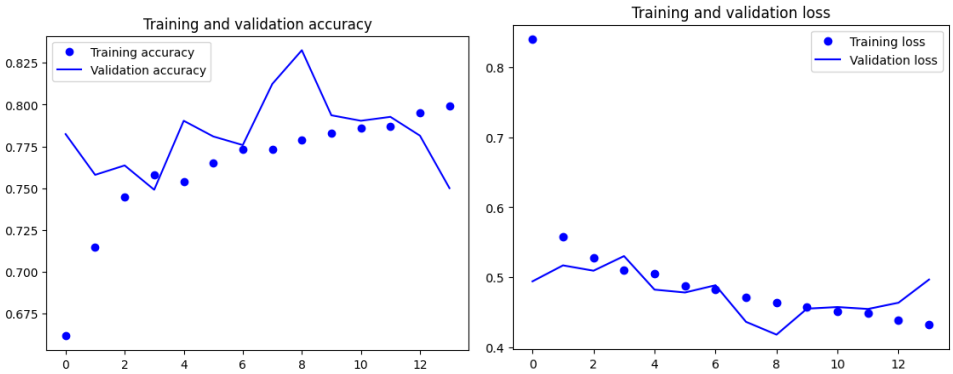
After the VGG16 base model, we constructed a Multi-Layer Perceptron (MLP) network and began with various configurations. We found it beneficial to consistently include a Global Average Pooling layer after each pre-trained CNN. Moving forward, this layer will be a standard component in all trials involving these network types. The Data Augmentation layer remains a constant fixture in all subsequent models.

The initial experiment aimed to assess the significance of the Dropout layer. Specifically, we added a Dense layer with 256 neurons on top of VGG. Initially, we implemented this layer without Dropout, and subsequently, we experimented with the inclusion of a Dropout layer set to a dropout rate of 0.5.

The accuracy and loss without dropout are shown below.



The accuracy and loss with 0.5 dropout are shown below.



The training curves of the model without Dropout show considerable deviation, with the validation curves differing significantly from the training curves. This phenomenon is slightly mitigated when using the Dropout model.

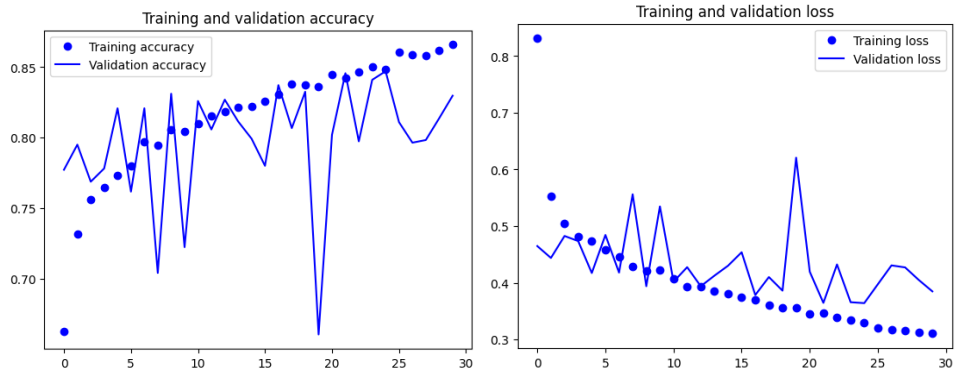
The best results in terms of accuracy and loss on the test dataset were achieved when we increased the number of neurons in the dense layer from 256 to 512 while maintaining the same dropout rate.

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 224, 224, 3)]	0
sequential (Sequential)	(None, 224, 224, 3)	0
tf.__operators__.getitem_2 (SlicingOpLambda)	(None, 224, 224, 3)	0
tf.nn.bias_add_2 (TFOpLambda)	(None, 224, 224, 3)	0
vgg16 (Functional)	(None, 7, 7, 512)	14714688
gap (GlobalAveragePooling2D)	(None, 512)	0
flatten (Flatten)	(None, 512)	0
classifier_hidden (Dense)	(None, 512)	262656
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 1)	513

Total params: 14977857 (57.14 MB)
Trainable params: 263169 (1.00 MB)
Non-trainable params: 14714688 (56.13 MB)

Summary of the model

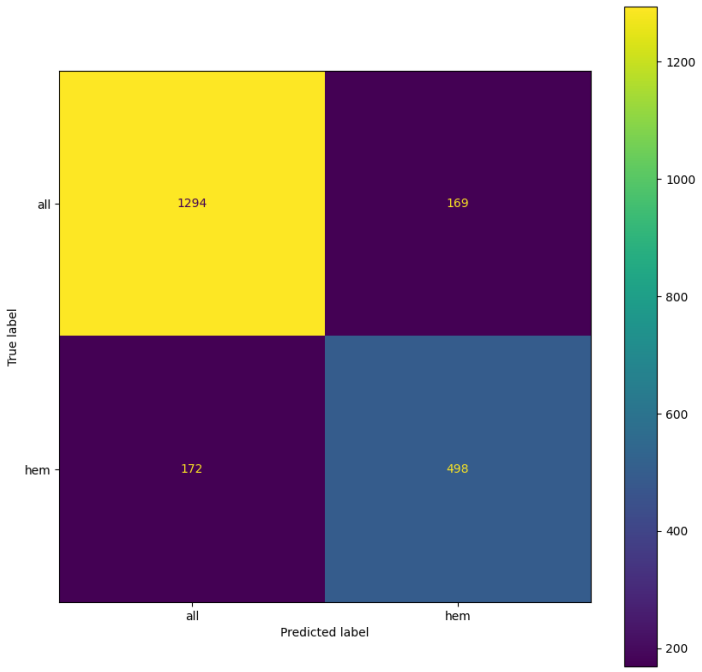
The results of the training are shown below.



The evaluation of the model against the test set is shown below.

Accuracy on test set	Loss on test set
0.8401	0.3930

```
Classification report:
              precision    recall  f1-score   support

         all     0.8827    0.8845    0.8836      1463
         hem     0.7466    0.7433    0.7450       670

    accuracy                         0.8401      2133
   macro avg     0.8147    0.8139    0.8143      2133
weighted avg     0.8399    0.8401    0.8400      2133
```

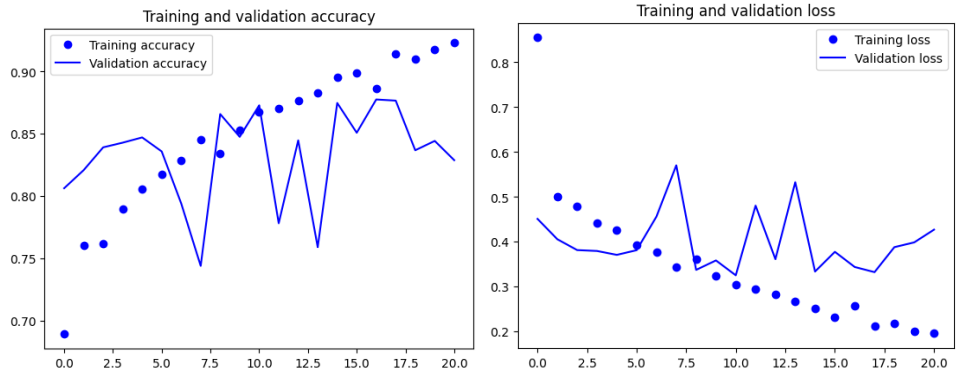


5.4.2. Model finetuning

Our goal in this phase was to fine-tune the best model obtained in the previous chapter. We began by unfreezing only the last layer of VGG16 and then reinitiated the training

process, making adjustments to the weights of this layer. The results of this training were promising, with the model showing improved performance.

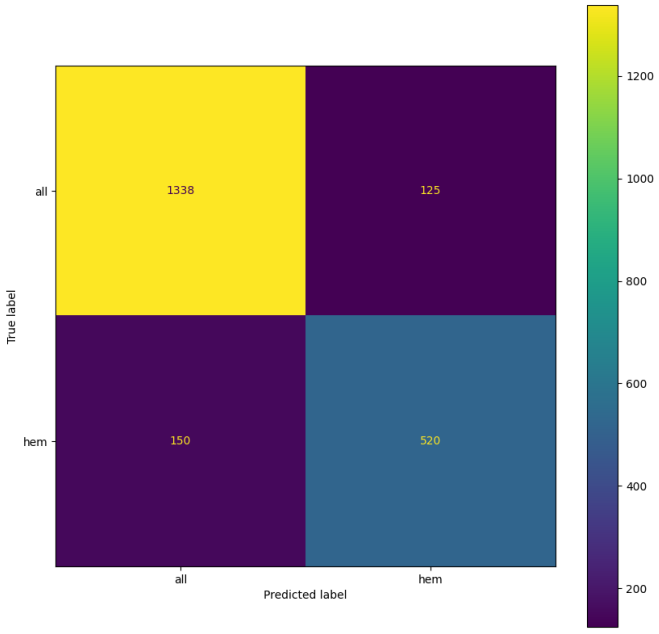
The results of the training are shown below.



The evaluation of the model against the test set is shown below.

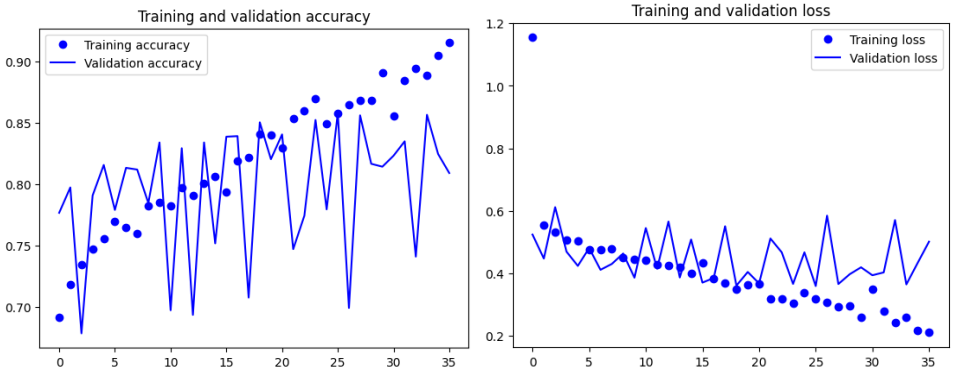
Accuracy on test set	Loss on test set
0.8711	0.3480

```
Classification report:
              precision    recall  f1-score   support

         all     0.8992    0.9146    0.9068      1463
         hem     0.8062    0.7761    0.7909       670

    accuracy                         0.8711      2133
   macro avg     0.8527    0.8453    0.8488      2133
weighted avg     0.8700    0.8711    0.8704      2133
```



Given the better results, we decided to proceed by unfreezing the second last layer, with the intention of gradually fine-tuning the entire last block.

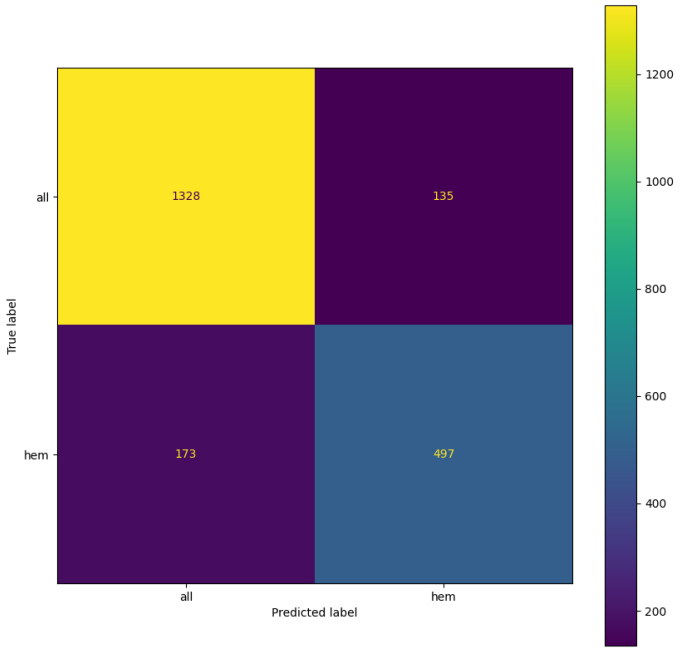
The results of the training are shown below.



The evaluation of the model against the test set is shown below.

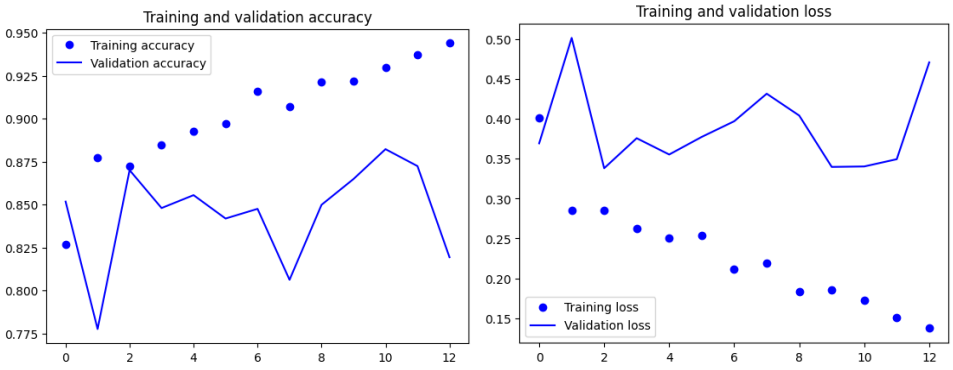
Accuracy on test set	Loss on test set
0.8556	0.3528

```
Classification report:
              precision    recall  f1-score   support

         all     0.8847    0.9077    0.8961      1463
         hem     0.7864    0.7418    0.7634       670

    accuracy                         0.8556      2133
   macro avg     0.8356    0.8248    0.8298      2133
weighted avg     0.8539    0.8556    0.8544      2133
```



A peculiar pattern emerged from the training history, characterized by erratic fluctuations in accuracy on validation sets. Usually, the learning rate is reduced to solve these problems. In this case, however, repeating the training with a learning rate of 0.0001 did not give much better results.

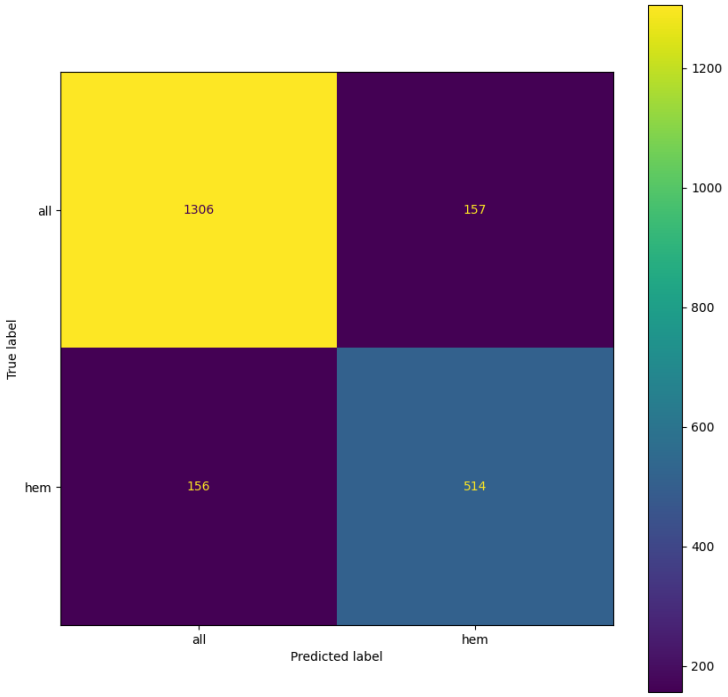
The results of the training are shown below.



The evaluation of the model against the test set is shown below.

Accuracy on test set	Loss on test set
0.8533	0.3672

```
Classification report:
              precision    recall  f1-score   support

         all     0.8933    0.8927    0.8930      1463
         hem     0.7660    0.7672    0.7666       670

    accuracy                         0.8533      2133
   macro avg     0.8297    0.8299    0.8298      2133
weighted avg     0.8533    0.8533    0.8533      2133
```



Finally, we unfroze the entire last block of VGG16 and conducted the training.

Layer (type)	Output Shape	Param #
input_4 (InputLayer)	[(None, 224, 224, 3)]	0
sequential (Sequential)	(None, 224, 224, 3)	0
tf.__operators__.getitem_2 (SlicingOpLambda)	(None, 224, 224, 3)	0
tf.nn.bias_add_2 (TFOpLambda)	(None, 224, 224, 3)	0
vgg16 (Functional)	(None, 7, 7, 512)	14714688
gap (GlobalAveragePooling2D)	(None, 512)	0
flatten (Flatten)	(None, 512)	0
classifier_hidden (Dense)	(None, 512)	262656
dropout_1 (Dropout)	(None, 512)	0
dense_2 (Dense)	(None, 1)	513

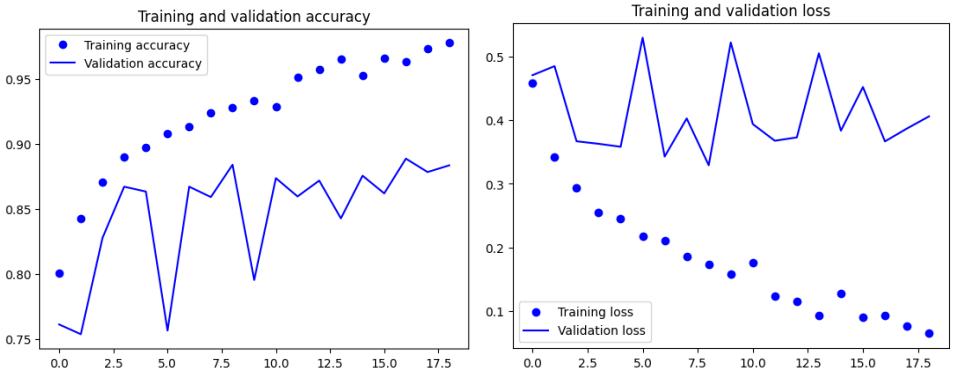
Total params: 14977857 (57.14 MB)
Trainable params: 7342593 (28.01 MB)
Non-trainable params: 7635264 (29.13 MB)

Summary of the model

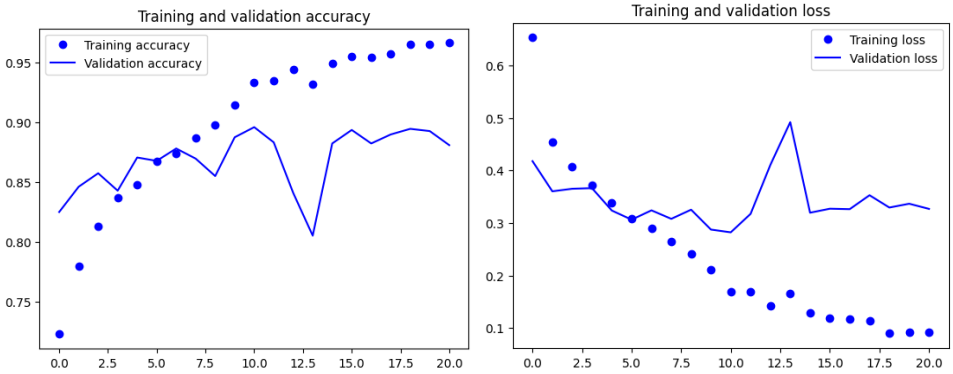
As we can see the trainable parameters passed from 263,169 to 7,342,593

We also reloaded the model trained with the entire VGG16 and unfroze the last block to facilitate a comparison between the performance of the progressively fine-tuned model and the non-progressively fine-tuned one. The results of these two training approaches are compared below.

The results of the training of the progressively fine-tuned model are shown below.



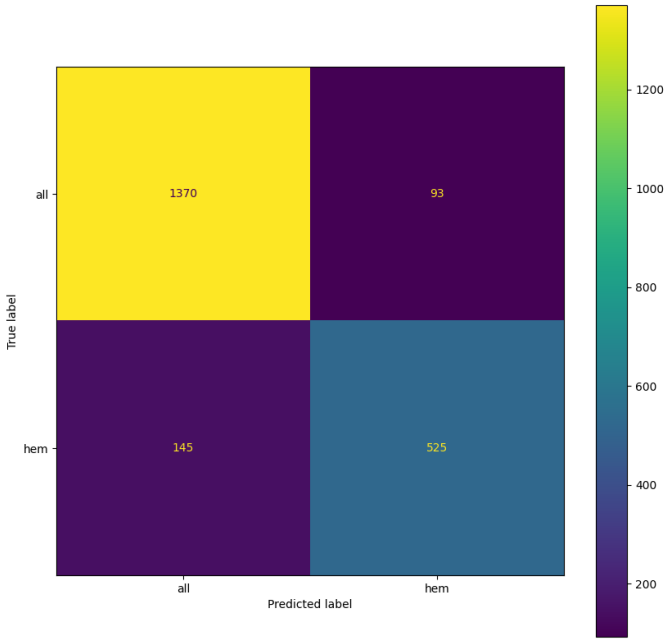
The results of the training of the non-progressively fine-tuned model are shown below.



The evaluation of the progressively fine-tuned model against the test set is shown below.

Accuracy on test set	Loss on test set
0.8884	0.3277

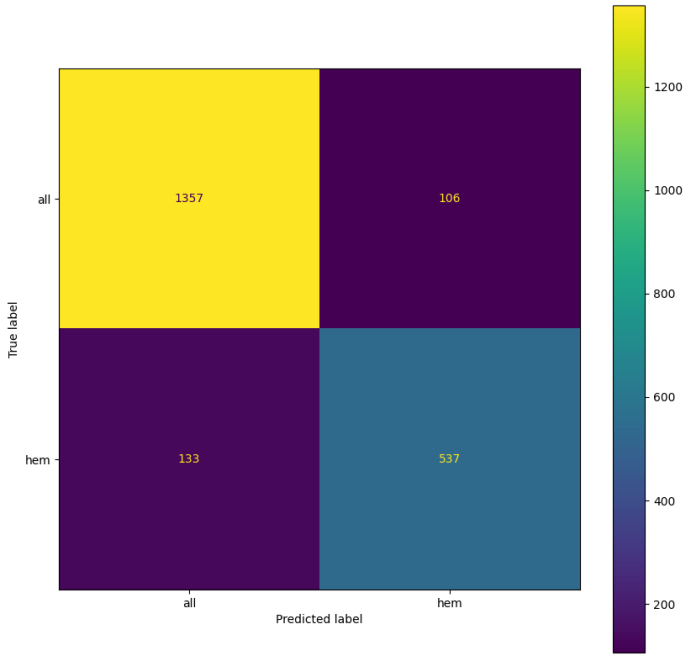
```
Classification report:
              precision    recall  f1-score   support

         all     0.9043    0.9364    0.9201      1463
         hem     0.8495    0.7836    0.8152       670

    accuracy                         0.8884      2133
   macro avg     0.8769    0.8600    0.8676      2133
weighted avg     0.8871    0.8884    0.8871      2133
```



The evaluation of the non-progressively fine-tuned model against the test set is shown below.

Accuracy on test set	Loss on test set
0.8880	0.3163

```
Classification report:
              precision    recall  f1-score   support

         all     0.9107    0.9275    0.9191      1463
         hem     0.8351    0.8015    0.8180       670

    accuracy                         0.8880      2133
   macro avg     0.8729    0.8645    0.8685      2133
weighted avg     0.8870    0.8880    0.8873      2133
```



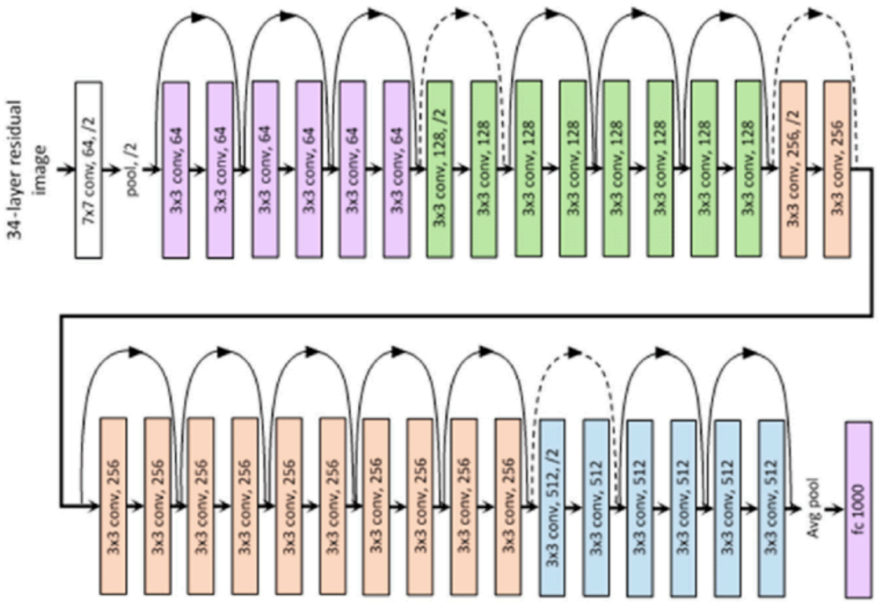
The results obtained from both models are very similar to each other, both in terms of overfitting and accuracy. However, the first model still obtains slightly higher accuracy than the second, even that related to the "hem" class.

Also by analyzing the confusion matrices it is possible to see that the two models have similar predictions.

5.5. Resnet-50

ResNet-50 is a convolutional neural network composed of 50 layers. It is part of the ResNet family, which stands for Residual Networks, and it serves as a foundational architecture for numerous computer vision tasks. The key innovation introduced by ResNet was its ability to facilitate the training of exceptionally deep neural networks, often exceeding 150 layers, which was a significant breakthrough in the field of deep learning.

ResNet makes effective use of Skip Connections, also known as Shortcut Connections. As the name implies, these connections skip some of the intermediate layers in the neural network and directly feed the output of one layer as the input to subsequent layers. Skip Connections address the Vanishing Gradient Problem, which can hinder the training of very deep neural networks. Thanks to this innovation, ResNet has become one of the most widely employed pre-trained convolutional neural networks in various applications.



ResNet structure

5.5.1. MLP built on top training

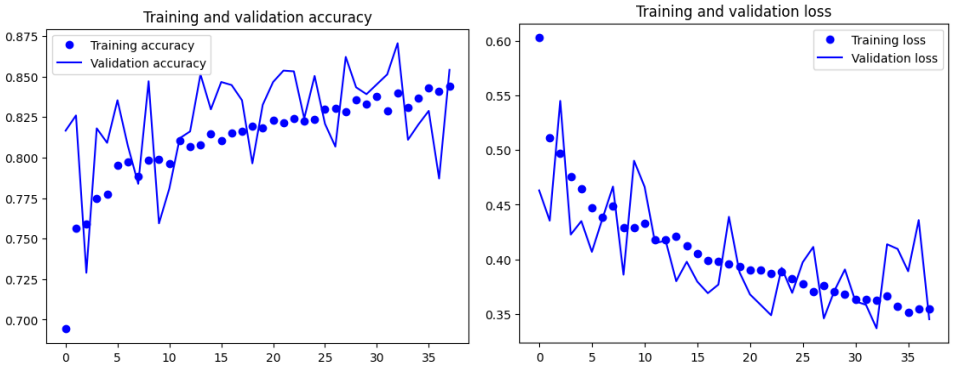
We chose to augment ResNet-50 with a MLP network consisting of a Dense layer with 256 neurons, accompanied by a Dropout layer with a dropout rate of 0.5.

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	[(None, 224, 224, 3)]	0
sequential (Sequential)	(None, 224, 224, 3)	0
tf.__operators__.getitem (SlicingOpLambda)	(None, 224, 224, 3)	0
tf.nn.bias_add (TFOpLambda)	(None, 224, 224, 3)	0
resnet50 (Functional)	(None, 7, 7, 2048)	23587712
gap (GlobalAveragePooling2 D)	(None, 2048)	0
flatten (Flatten)	(None, 2048)	0
classifier_hidden (Dense)	(None, 256)	524544
dropout (Dropout)	(None, 256)	0
dense (Dense)	(None, 1)	257

Total params: 24112513 (91.98 MB)
Trainable params: 524801 (2.00 MB)
Non-trainable params: 23587712 (89.98 MB)

Summary of the model

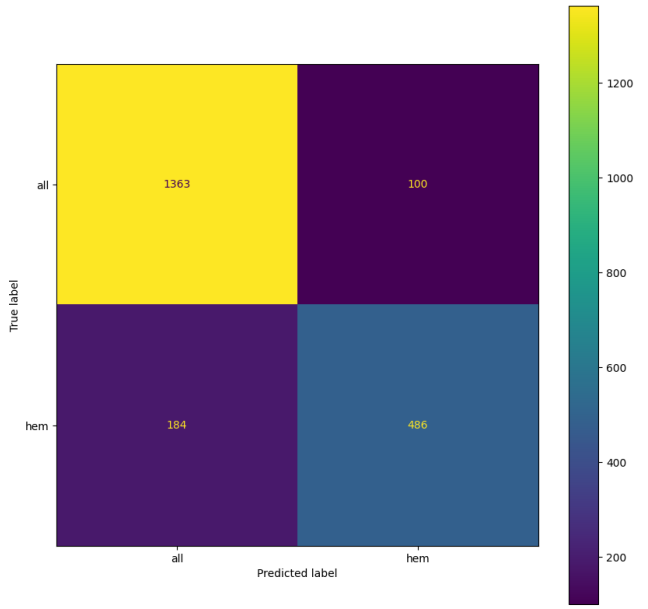
The results of the training are shown below.



The evaluation of the model against the test set is shown below.

Accuracy on test set	Loss on test set
0.8669	0.3305

```
Classification report:
              precision    recall  f1-score   support

         all     0.8811    0.9316    0.9056      1463
         hem     0.8294    0.7254    0.7739       670

    accuracy                         0.8669      2133
   macro avg     0.8552    0.8285    0.8398      2133
weighted avg     0.8648    0.8669    0.8643      2133
```



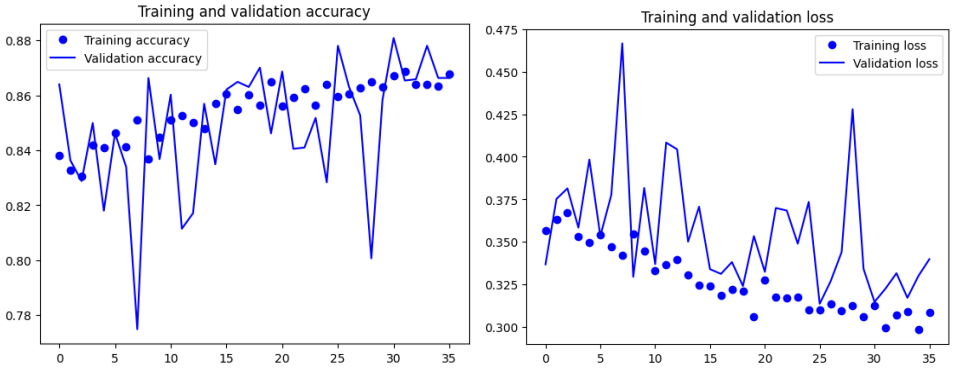
5.5.2. Model finetuning

The model described above was subsequently fine-tuned with the last block unfrozen, and the obtained results are shown below.

```
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 224, 224, 3)]     0

sequential (Sequential)      (None, 224, 224, 3)       0

tf.__operators__.getitem (   (None, 224, 224, 3)       0
SlicingOpLambda)

tf.nn.bias_add (TFOpLambda   (None, 224, 224, 3)       0
)

resnet50 (Functional)        (None, 7, 7, 2048)        23587712

gap (GlobalAveragePooling2   (None, 2048)              0
D)

flatten (Flatten)            (None, 2048)              0

classifier_hidden (Dense)    (None, 256)               524544

dropout (Dropout)            (None, 256)               0

dense (Dense)                (None, 1)                 257

=================================================================
Total params: 24112513 (91.98 MB)
Trainable params: 4984321 (19.01 MB)
Non-trainable params: 19128192 (72.97 MB)
```

As we can see the trainable parameters passed from 524,801 to 4,984,321.

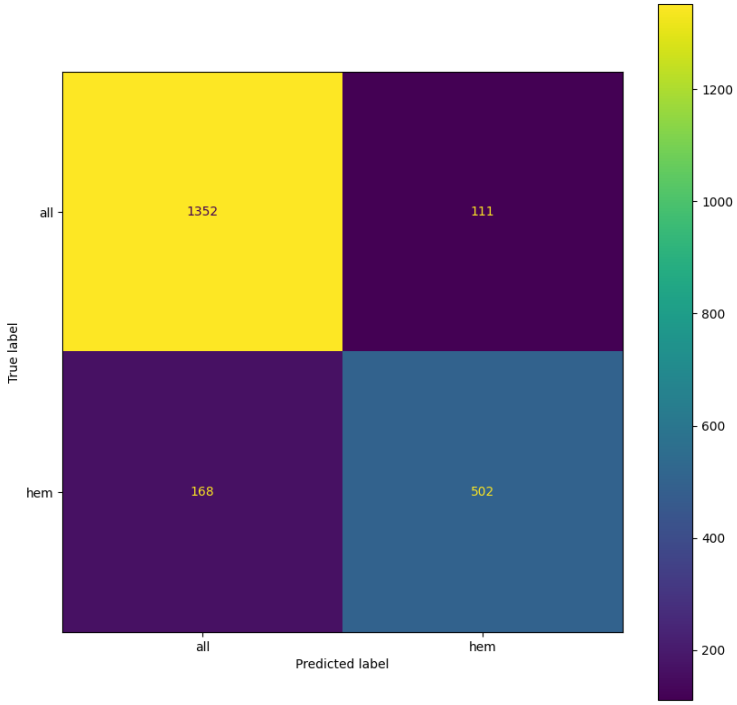
The results of the training are shown below.



The evaluation of the model against the test set is shown below.

Accuracy on test set	Loss on test set
0.8692	0.3123

```
Classification report:
              precision    recall  f1-score   support

         all     0.8895    0.9241    0.9065      1463
         hem     0.8189    0.7493    0.7825       670

    accuracy                         0.8692      2133
   macro avg     0.8542    0.8367    0.8445      2133
weighted avg     0.8673    0.8692    0.8675      2133
```

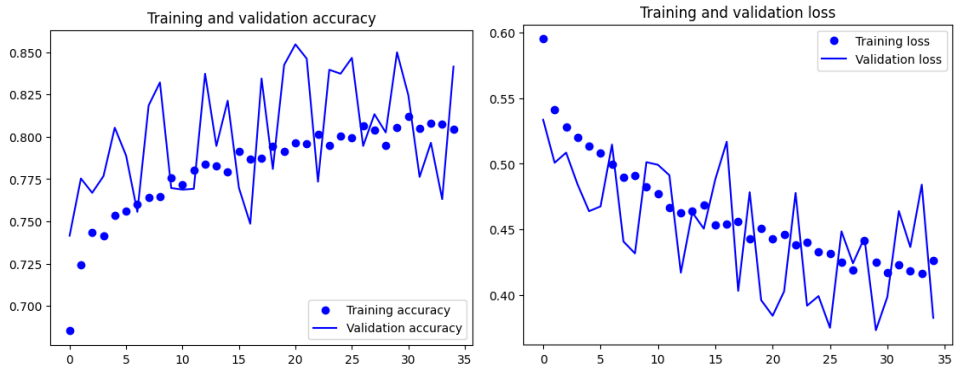


5.5.3. Chopping the last block of Resnet-50

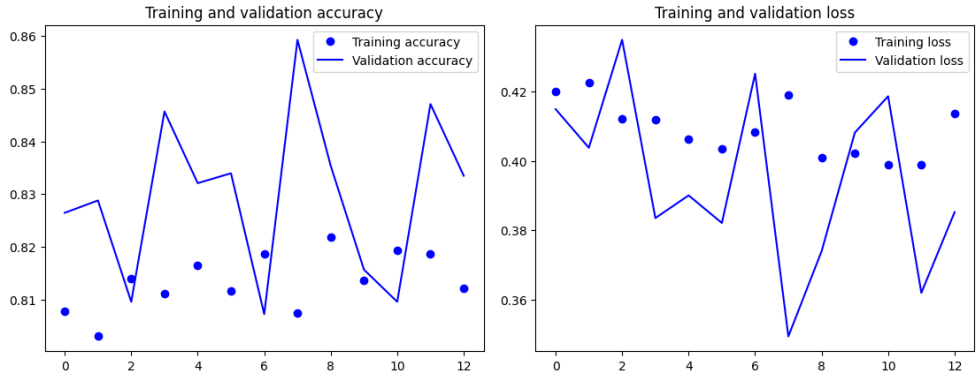
All the pre-trained CNNs used were initially trained on ImageNet, which is a dataset significantly distinct from ours, composed of microscopic cell images. It is plausible to hypothesize that the convolutional filters in the last layers of these CNNs are specialized in recognizing very specific features within the images, while the earlier filters are designed to capture more general, low-level features that might be more relevant to our dataset.

To adapt the model to our data, we experimented with removing the last block of ResNet-50 before connecting the baseline CNN to our MLP network. We then trained this MLP network on our dataset with the frozen baseline. Subsequently, we performed fine-tuning by unfreezing the last block of the network.

The results of the training of the freezed chopped model are shown below.



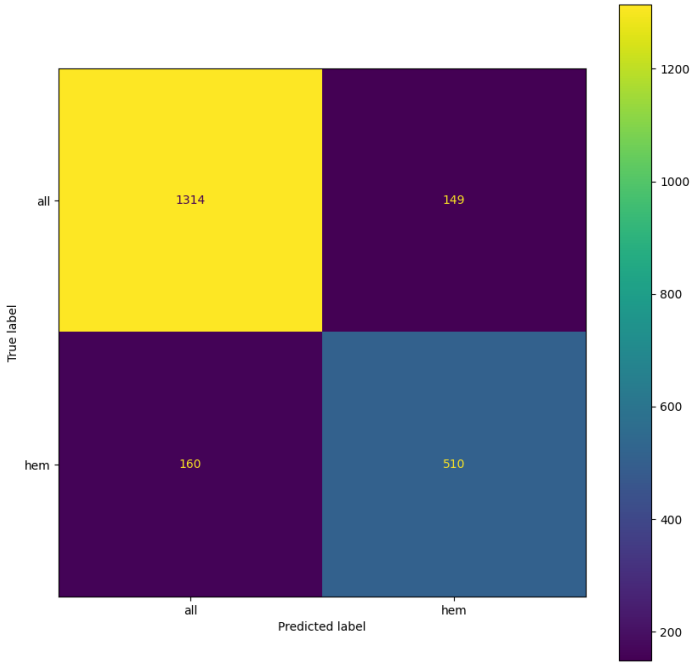
The results of the training of the finetuning of the model are shown below.



The evaluation of the freezed chopped model against the test set is shown below.

Accuracy on test set	Loss on test set
0.8551	0.3755

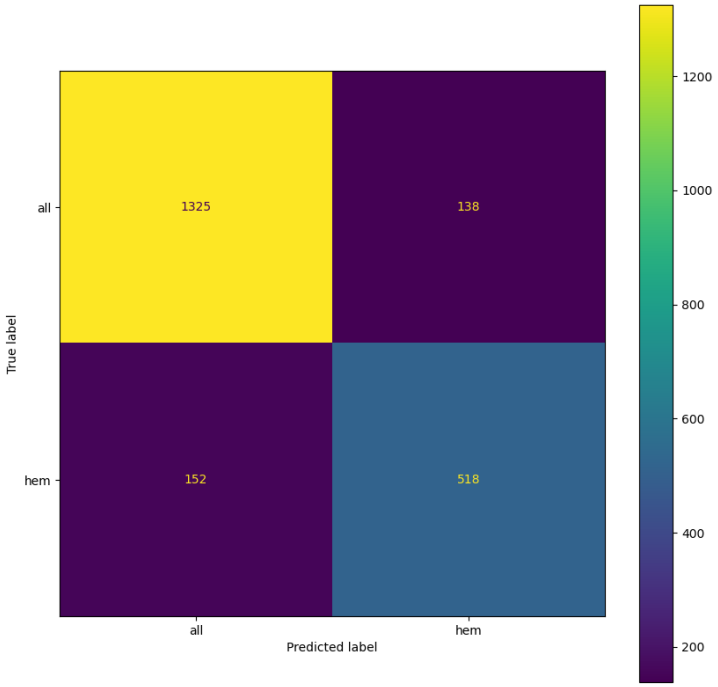
```
Classification report:
              precision    recall  f1-score   support

         all     0.8915    0.8982    0.8948      1463
         hem     0.7739    0.7612    0.7675       670

    accuracy                         0.8551      2133
   macro avg     0.8327    0.8297    0.8311      2133
weighted avg     0.8545    0.8551    0.8548      2133
```



The evaluation of the training of the freezed chopped model against the test set is shown below.

Accuracy on test set	Loss on test set
0.8640	0.3533

```
Classification report:
              precision    recall  f1-score   support

         all     0.8971    0.9057    0.9014      1463
         hem     0.7896    0.7731    0.7813       670

    accuracy                         0.8640      2133
   macro avg     0.8434    0.8394    0.8413      2133
weighted avg     0.8633    0.8640    0.8636      2133
```



As we can see from these latest results, the best model obtained was the one seen in the previous chapter.

6. Explainability

In this chapter, our focus is on understanding the functioning of the constructed model. We will specifically investigate the intermediate activations of the filters and analyze the heat-maps of class activations.

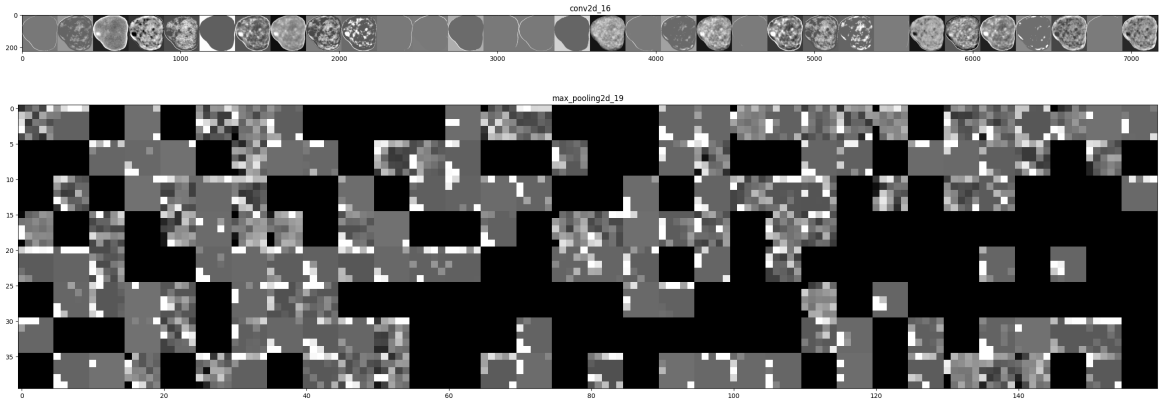
We will analyze the best model for each type of network trained, including the CNN from scratch, VGG16, and ResNet. To evaluate a model's performance, we took into account its accuracy on the test set.

6.1 Intermediate activation

In this chapter, our focus is on understanding the functionality of the best CNN-based model identified in the previous chapters. We will achieve this by showcasing intermediate activations derived from an ALL-type image as input.

6.1.1 CNN from scratch

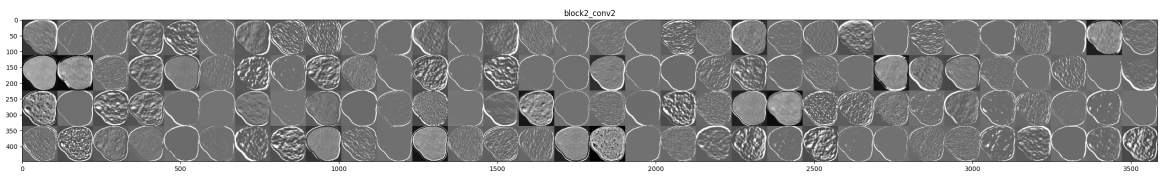
The following images display the feature maps of the first convolutional layer and the last layer.



In the initial layer, the network exhibits a concentration on the edges of the image, while in the final layer, abstract information is prevalent, making it challenging for a human observer to attribute it directly to the initial images. Additionally, it is noteworthy that the last layer contains various inactive filters.

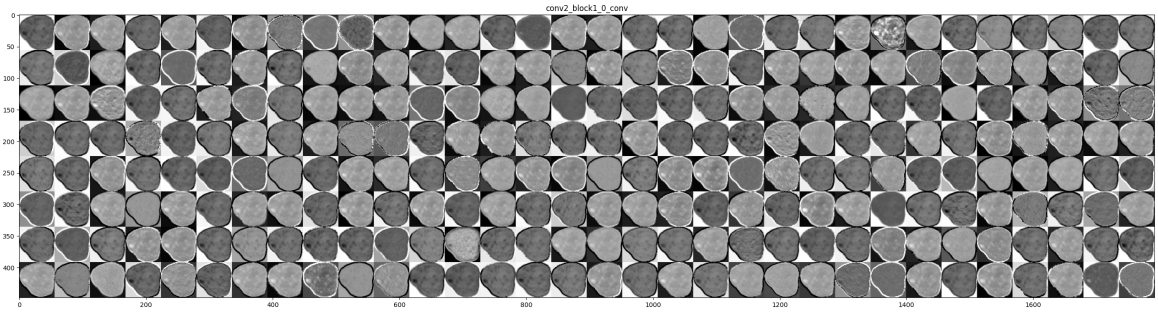
6.1.2 VGG16

In this case we show intermediate activation on an inner layer of the model.



The feature maps of VGG16 showcase a consistent contrast, with all edges highlighted in white.

6.1.3 Resnet-50

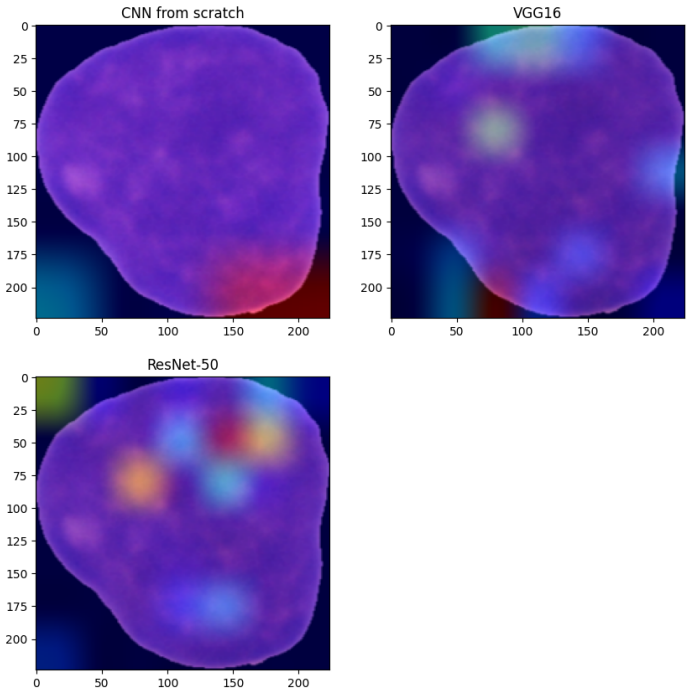


This is not the case in ResNet-50, where there is a significantly high variance in the activation constants of filters within the same layers.

6.2 Heatmaps

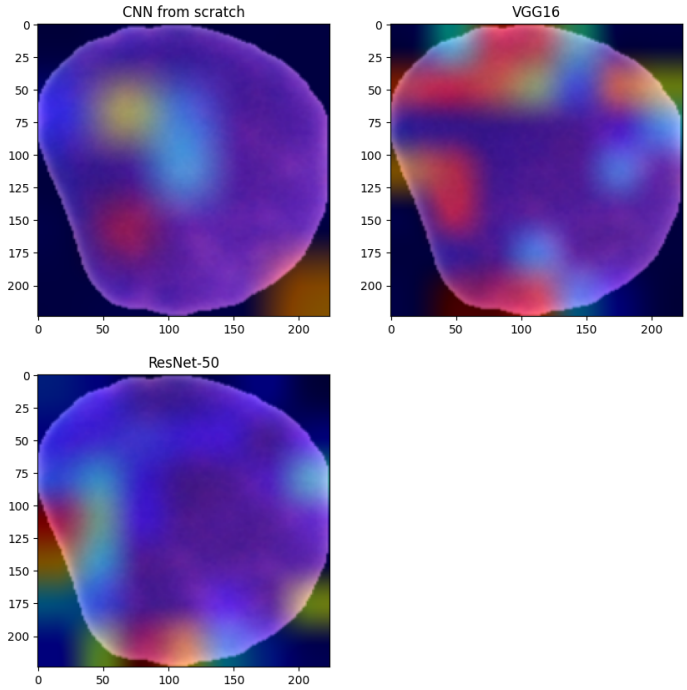
In this chapter, our emphasis shifts to visualizing the heatmaps for both the ALL and HEM classes across all the analyzed models. The objective is to bring attention to the distinct reasoning employed by each model in the image classification process.

6.2.1 ALL class



We can see that ResNet-50 is the one that can capture more information of the inner part of the cell, unlike the other two that tend to focus more on the edges. In addition, all three networks tend to focus on different parts of the same image.

6.2.2 HEM class

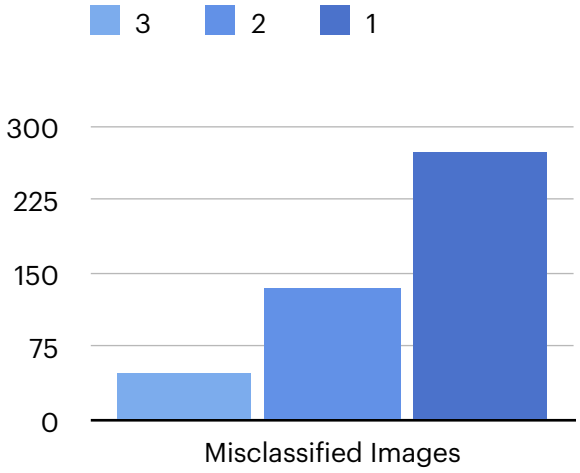


In this case, it is CNN from scratch that is the network that focuses more on information within the cell. The other two networks tend to focus more on the edges. Again we can see that all networks tend to focus on different aspects of the image.

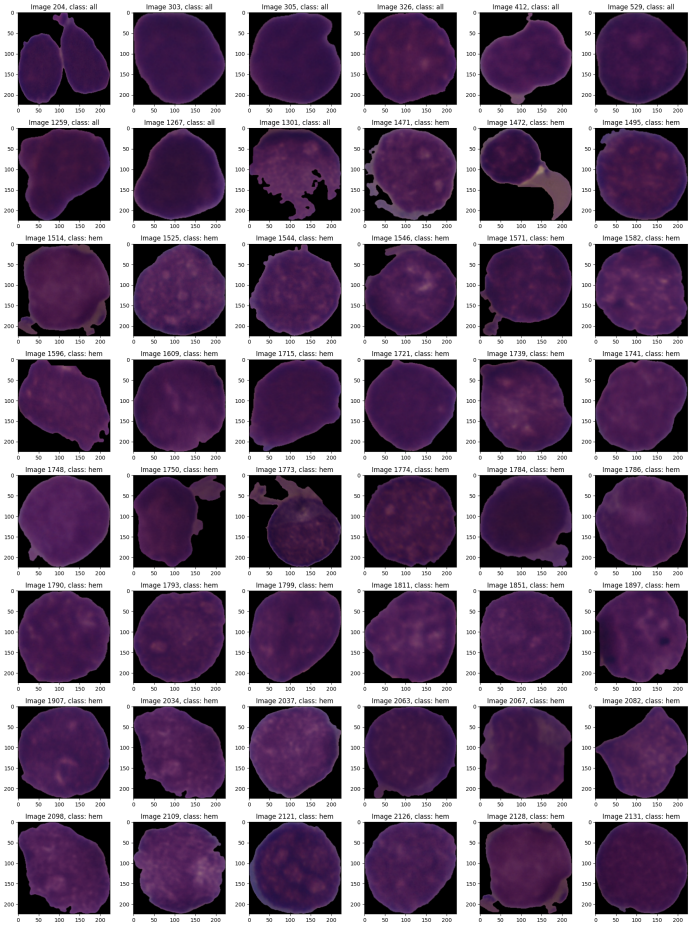
Having networks that focus on different aspects will allow us to get very good results in the ensemble case.

7 Error Analysis

In this chapter, our objective is to analyze the errors made by our top networks. Specifically, we aim to discern whether these errors are consistent across the networks or if the networks exhibit distinct predictive capabilities.



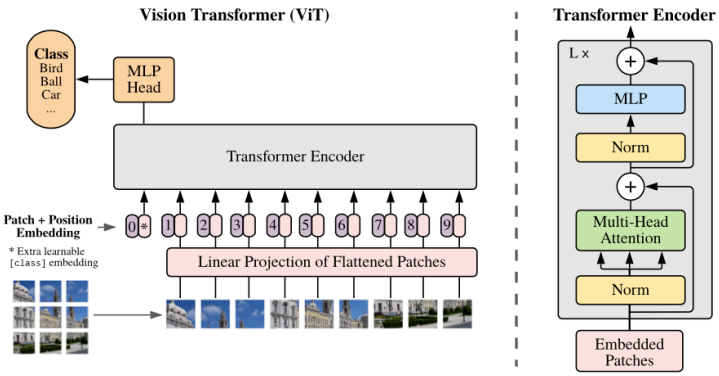
First, we can see how the three models tend to make misclassification errors on different images. In fact, there are 48 images misclassified by all three models. This aspect will prove useful for the ensemble model.



We can see that most of the errors are for the HEM class, so fewer errors regarding the ALL class of our interest.

Due to the challenging nature of distinguishing between the two cell types, the recurrent errors in the HEM class might stem from its inadequate representation in the training set. The available number of images may not be adequate for generalizing features effectively, making further considerations challenging, even for experts in the field. Nevertheless, it appears that the misclassified images share the presence of relatively more irregular shapes compared to the rest of the dataset.

8 Vision Transformer



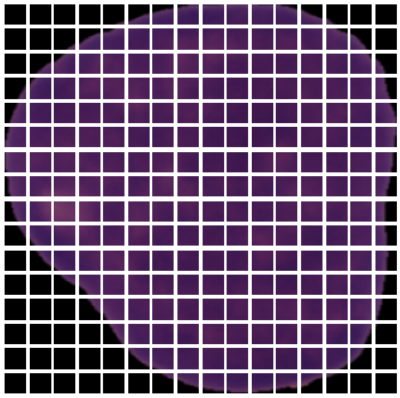
The transformer stands out as the most successful deep learning model in Natural Language Processing (NLP). Introduced in 2017, it utilizes the self-attention mechanism, assigning distinct weights to each part of the input data. This innovation has led to the widespread replacement of RNN models, such as LSTMs, in various applications.

A significant development in 2021 showcased the adaptability of transformer architectures to image classification. The Vision Transformer, or ViT, revolutionizes image classification by applying a transformer-like structure to image patches, akin to word tokens in NLP tasks. The process involves dividing an image into fixed-size patches, linearly embedding each patch, adding positional embeddings, and feeding the resulting vector sequence into a standard Transformer encoder. For image classification objectives, an MLP (Multi-Layer Perceptron) is constructed on top of the transformer to classify the image.

Considering their ability to outperform CNNs in image classification accuracy, Vision Transformers have proven to be state-of-the-art, particularly when pre-trained on substantial datasets. Hence, we are eager to experiment with these models on our dataset.

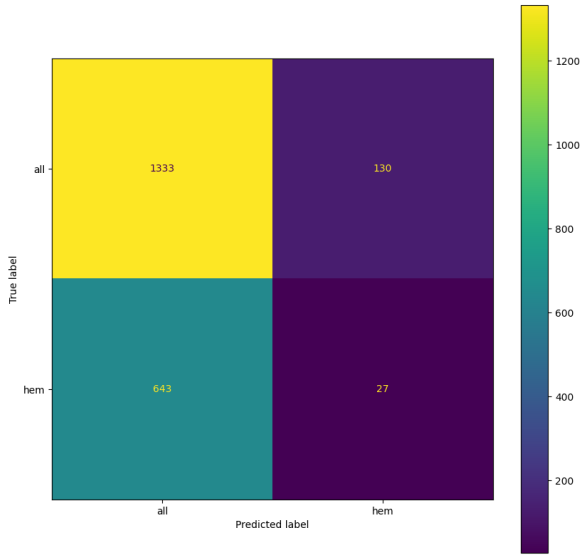
8.1 From scratch Vision Transformer

Our first experiment revolves around creating a Vision Transformer from the ground up and then applying this model to our dataset. The model construction was inspired by an example found on the Keras website. We defined the patch class and its encoding, choosing to use 14x14 pixel patches. This decision leads us to segment our images into 16x16 patches. Below is reported an example of patch extraction for a sample from our dataset (ALL class).



With 8 transformer layers and 4 multi-head attention layer heads, the model yields results significantly lower than CNNs. However, there is an expectation that the performance will enhance with pre-trained Vision Transformers.

```
Classification report:
              precision    recall  f1-score   support

         all     0.6746    0.9111    0.7752      1463
         hem     0.1720    0.0403    0.0653       670

    accuracy                         0.6376      2133
   macro avg     0.4233    0.4757    0.4203      2133
weighted avg     0.5167    0.6376    0.5522      2133
```

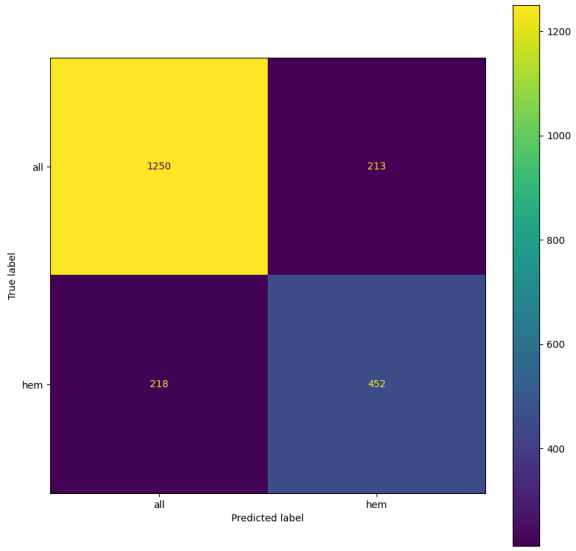


8.2 Pre-trained Vision Transformers

We are going to conduct some experiments involving a MLP built on top of a pre-trained Vision Transformer. The model used is Vit-B32, provided by Keras.

Constructed on top of ViT-B32, we developed a dense neural network with 256 neurons. We conducted two training experiments: one without a dropout layer and another with a dropout layer featuring a dropout rate of 0.3. Despite the results being quite similar, the superior performance was achieved without dropout. The outcomes are as follows:

```
Classification report:
              precision    recall  f1-score   support

         all     0.8515    0.8544    0.8530      1463
         hem     0.6797    0.6746    0.6772       670

    accuracy                         0.7979      2133
   macro avg     0.7656    0.7645    0.7651      2133
weighted avg     0.7975    0.7979    0.7977      2133
```

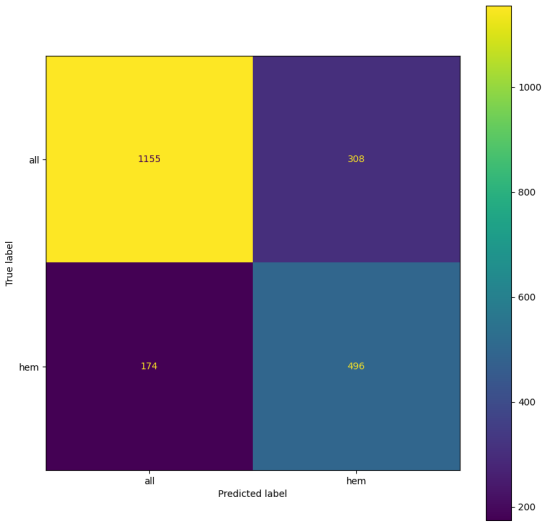


The results obtained are lower than those obtained with CNNs. Fine-tuning was necessary to obtain comparable results.

8.3 Vision Transformer Fine Tuning

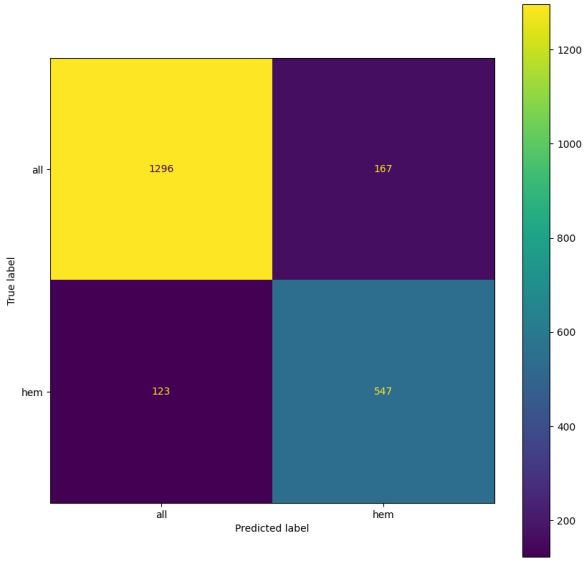
During the training of the CNNs, we initially employed pre-training of the MLP network, keeping the base CNN frozen. Subsequently, we unfrozen some blocks of the model and conducted fine-tuning. We followed a similar approach with the Vision Transformer to ensure comparable results. In this case, during fine-tuning, we unfrozen the entire ViT model. We opted to construct the same model as seen in the last chapter without Dropout, given its better results. The initial training phase produced the following results:

```
Classification report:
              precision    recall  f1-score   support

         all     0.8691    0.7895    0.8274      1463
         hem     0.6169    0.7403    0.6730       670

    accuracy                         0.7740      2133
   macro avg     0.7430    0.7649    0.7502      2133
weighted avg     0.7899    0.7740    0.7789      2133
```



However, the results are not comparable to those obtained with CNNs. A final attempt to try to get better results was to unfreeze ViT and redo the training:

```
Classification report:
              precision    recall  f1-score   support

         all     0.9133    0.8859    0.8994      1463
         hem     0.7661    0.8164    0.7905       670

    accuracy                         0.8640      2133
   macro avg     0.8397    0.8511    0.8449      2133
weighted avg     0.8671    0.8640    0.8652      2133
```



Only in this case we were able to obtain results comparable with those obtained with CNNs. However, this does not represent the best result obtained so far. It may be that the dataset used is not optimal for training a Vision Transformer. These models tend to be more sensitive to the size of the dataset and its variety than CNNs.

9 Ensemble

In this chapter, we will build ensemble models using the best models developed so far. Our specific objectives include creating a model with the highest accuracy on the test set and another model optimized for the best precision on non-leukemia images. This is important as it helps us assess the model's ability to avoid misclassifying cells with leukemia as “hem."

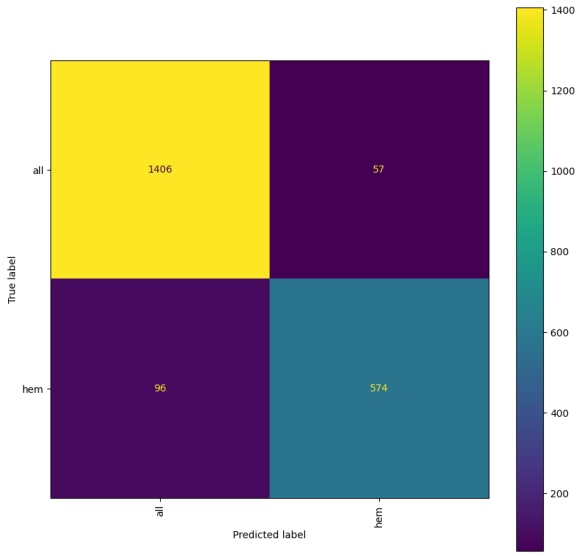
Below is reported the loading of the best models:

```
from_scratch_model = models.load_model(MODELS_PATH + '/Dense512Neurons_OneDropout0-3/Dense512Neurons_OneDropout0-3.h5')
vgg_model = models.load_model(MODELS_PATH + '/VGG16_LastBlockFineTuned/VGG16_LastBlockFineTuned.h5')
resnet_model = models.load_model(MODELS_PATH + '/ResNet50_Finetuned/ResNet50_Finetuned.h5')
vit_model = models.load_model(MODELS_PATH + "/VisionTransformerb32_Dense256_Finetuning/"+"VisionTransformerb32_Dense256_Finetuning.h5")
```

9.1 Average model

The simplest method to combine the predictions of a group of classifiers is to calculate the average of their predictions and then determine the ensemble-classified label based on this aggregated prediction.

	precision	recall	f1-score	support
all	0.9361	0.9610	0.9484	1463
hem	0.9097	0.8567	0.8824	670
accuracy			0.9283	2133
macro avg	0.9229	0.9089	0.9154	2133
weighted avg	0.9278	0.9283	0.9277	2133



As expected, given also the considerations made in the previous chapters, we get the best result seen so far.

9.2 Weighted Average Model

To optimize the effectiveness of the ensemble technique, it is advisable to calculate a weighted average of the predictions, even though all classifiers demonstrate relatively

similar performance. This involves assigning higher weights to the best classifiers and lower weights to the less effective ones. This ensures that the most reliable models contribute more significantly to the final prediction.

In our approach, two distinct techniques will be employed to determine the best weights for the ensemble model. For both experiments, we will build two models: one focused on maximizing accuracy on the validation set and another dedicated to maximizing precision for "hem" images.

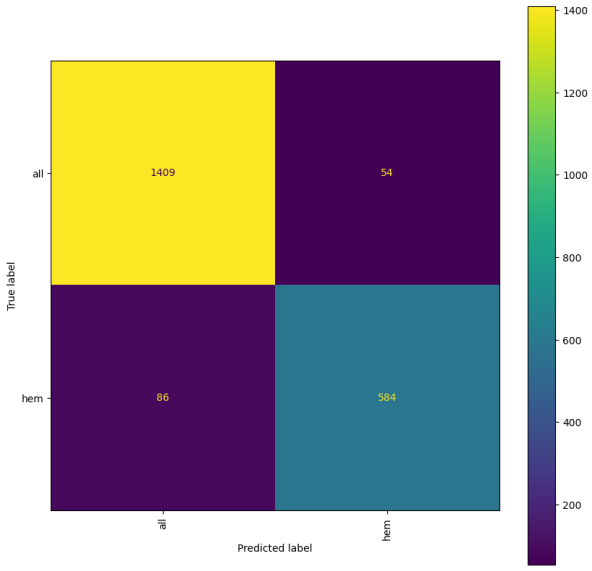
9.2.1 Brute force

To systematically discover a suitable set of weights, one approach involves exploring all weight combinations within a specific step size while ensuring that the sum of all weights equals 1. In this method, all possible combinations are tested, and only the models yielding the best results on the validation set are retained. This approach is manageable within a moderate timeframe with four models and a reasonably sized step (e.g., 0.05).

It's crucial to note that the algorithmic complexity of this solution is O(step^n), where n represents the number of models. Consequently, if the number of models significantly increases or if a higher level of precision is required by reducing the step size, this approach can become impractical due to its exponential time complexity.

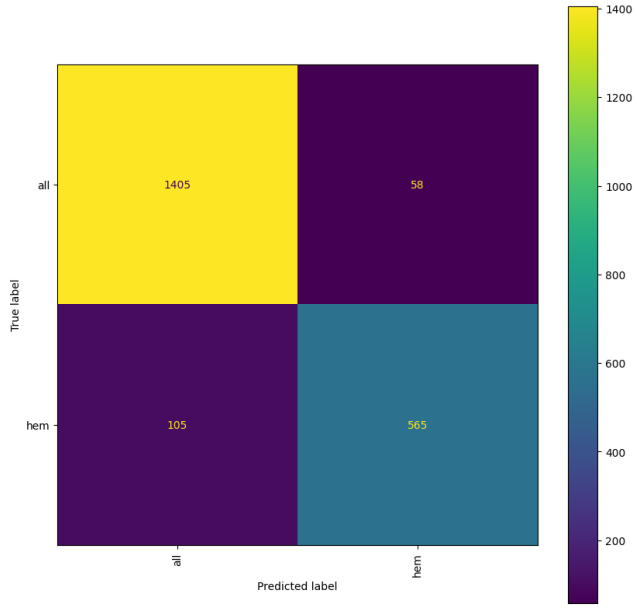
Below are the results for the model with the best accuracy on validation set:

```
Model with best accuracy:
Weights: [0.55, 0.05, 0.4, 0.0]
Accuracy on the set: 0.9343647444913268
              precision    recall  f1-score   support

         all     0.9425    0.9631    0.9527      1463
         hem     0.9154    0.8716    0.8930       670

    accuracy                         0.9344      2133
   macro avg     0.9289    0.9174    0.9228      2133
weighted avg     0.9340    0.9344    0.9339      2133
```



Below are the results for the model with the best precision:

```
Model with best precision on nocancer images:
Weights: [0.35000000000000003, 0.15000000000000002, 0.5, 0.0]
Accuracy on the set: 0.923581809657759
              precision    recall  f1-score   support

         all     0.9305    0.9604    0.9452      1463
         hem     0.9069    0.8433    0.8739       670

    accuracy                         0.9236      2133
   macro avg     0.9187    0.9018    0.9096      2133
weighted avg     0.9231    0.9236    0.9228      2133
```



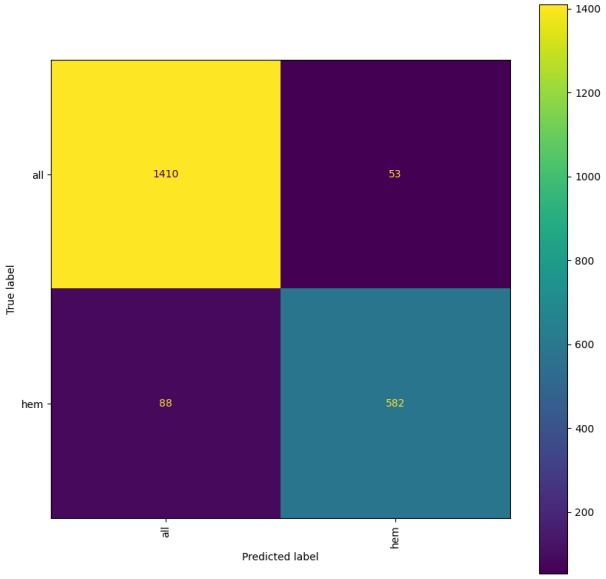
9.2.2 Genetic Algorithm

Using a Genetic Algorithm, model weights can be determined by employing chromosomes with four genes, each representing the weight of a specific model. Throughout transformation steps like crossover and mutation, the constraint that the sum of the genes in each chromosome equals one is maintained. The algorithm will be executed twice, each time with a different fitness function: one based on accuracy on the validation set and the other focused on precision for "hem" images.

In both runs, 8 iterations will be carried out with a population size of 100, 50 generations per iteration, 0.5 for crossover probability and 0.01 for mutation probability (parameters values found experimentally).

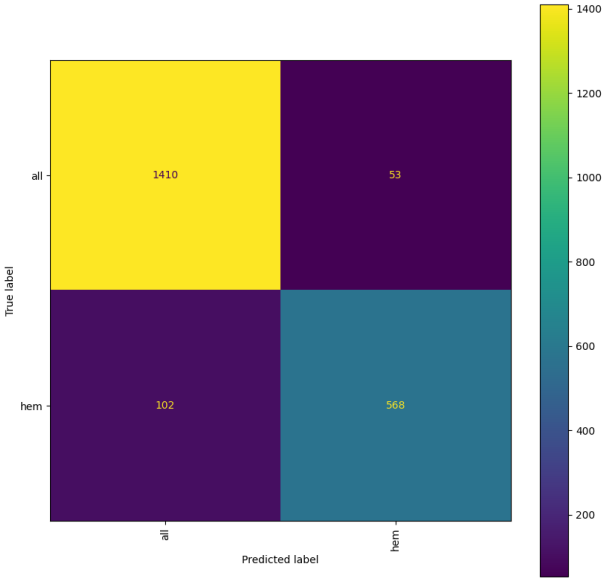
For the model with the highest accuracy, the obtained significance is 43% for CNN, 15% for VGG16, 34% for ResNet-50, and 8% for Vision Transformer.

```
Accuracy on the set: 0.9338959212376934
              precision    recall  f1-score   support

         all     0.9413    0.9638    0.9524      1463
         hem     0.9165    0.8687    0.8920       670

    accuracy                         0.9339      2133
   macro avg     0.9289    0.9162    0.9222      2133
weighted avg     0.9335    0.9339    0.9334      2133
```



For the model with the highest precision, the obtained significance is 37% for CNN, 33% for VGG16, 29% for ResNet-50, and 1% for Vision Transformer.

```
Accuracy on the set: 0.927332395686826
              precision    recall  f1-score   support

         all     0.9325    0.9638    0.9479      1463
         hem     0.9147    0.8478    0.8799       670

    accuracy                         0.9273      2133
   macro avg     0.9236    0.9058    0.9139      2133
weighted avg     0.9269    0.9273    0.9266      2133
```



Although the models seen report discordant results regarding precision for the "hem" class, considering the methodology applied, we have to treat it as a probabilistic event based on the particularity of the subdivision of our data set. This is because we have a small difference that we can attribute to the probability.

As can be seen from the various results obtained with ensemble techniques, we do not have one model that prevails over the others. However as was to be expected these represent the best results obtained so far.

10 Conclusion

In this study, we evaluated CNNs built from scratch, pre-trained CNNs, and Vision Transformers for cell classification. We discussed the distinctions among these models and subsequently combined them to enhance overall performance. The significance of various weight-finding techniques in maximizing model results was also explored.

Our achieved accuracy stands at approximately 93.4%, a figure that aligns well with the results detailed in the Related Work chapter. Similar outcomes were obtained for precision in classifying healthy cells.

A potential refinement for this work involves hyper-parameter optimization to fine-tune our networks. While we adopted a trial-and-error strategy for finding optimal configurations, a more advanced and non-heuristic approach, such as grid search, could likely yield superior results.

It's crucial to reiterate the limitation imposed by computational resources, preventing the utilization of the entire dataset or pursuing further enhancements.

References

- Website, Acute lymphoblastic leukemia, https://en.wikipedia.org/wiki/Acute_lymphoblastic_leukemia
- Website, C_NMC_2019 Dataset: ALL Challenge dataset of ISBI 2019 (C-NMC 2019), https://wiki.cancerimagingarchive.net/pages/viewpage.action?pageId=52758223
- C. Marzahl, M. Aubreville, J. Voigt and A. Maier, "Classification of Leukemic B-Lymphoblast Cells from Blood Smear Microscopic Images with an Attention-Based Deep Learning Method and Advanced Augmentation Techniques," in ISBI 2019 C-NMC Challenge: Classification in Cancer Cell Imaging, Springer, 2019.
- E. Verma and V. Singh, "ISBI Challenge 2019: Convolution Neural," in ISBI 2019 C-NMC Challenge: Classification in Cancer Cell Imaging, Springer, 2019.
- A. Gupta, R. Duggal, S. Gehlot, R. Gupta, A. Mangal, L. Kumar, N. Thakkar and D. Satpathy, "GCTI-SN: Geometry-inspired chemical and tissue invariant stain normalization of microscopic medical images," Medical Image Analysis, vol. 65, 2020.
- K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," 2014.
- K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," 2015.
- Website, Image classification with Vision Transformer, https://keras.io/examples/vision/image–classification–with– vision–transformer/
- Website, Attention Is All You Need, https://arxiv.org/abs/1706.03762
- Website, An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, https://arxiv.org/abs/2010.11929
- Website, Vision Transformers are Robust Learners, https://arxiv.org/abs/2105.07581
- Website, Image classification with Vision Transformer, https://keras.io/examples/vision/image_classification_with_vision_transformer/