

Deep Learning methods applied to medical images recognition



Table of contents

[**Introduction**](#_decr49nm31sa) **4**

[**Deep learning and machine learning methods**](#_v9abedv9my8q) **5**

[Convolutional Neural Networks - CNN](#_khj8dyylnme6) 6

[**Previous work done**](#_l7qbu5snuzj7) **7**

[**Datasets used**](#_rpmafwjs8t1n) **8**

[**Our experiences and works**](#_bhza0ujbabo7) **9**

[Fixed Feature Extraction](#_s24zgklp55sa) 9

[PCA and VLAD](#_s24zgklp55sa) 12

[Pretrained models](#_z1s3ap6eeixs) 13

[Fine-tuning](#_9kiga3vjsmb8) 15

[Some custom CNNs](#_wdutf47b19ln) 16

[Histogram equalization - CLAHE](#_3sxdk7v8qbz3) 16

[Data Augmentation](#_io5fr9vabyhg) 17

[**Final results**](#_wpt67z73zm8t) **17**

[**Conclusions and future improvements**](#_8i2et6xpj436) **18**

# Introduction

This report is about the **medical images classification problem**. Image classification consists of knowing the category of an input image, based on the output of a system trained in that purpose, thanks to an annotated by human image set. That process can be applied to many fields, such as the very famous [ImageNet](http://www.image-net.org/) designed to recognize an input image between a tremendous 21.841 categories set based on the learning of millions of images.

Since Alexey Ivakhnenko and Lapa introduced the multilayer perceptrons in 1965, methods based on deep learning has been significantly improved, and especially within the field we are interested in, image classification problem.

We will focus on the use of **Convolutional Neural Networks** to diagnosis medical images. Our experiments were conducted on the field of **Transfer Learning**.  
 Firstly, our work covers the extraction of the features produced by the processing of an image inside a pre-trained CNN to feed a Support Vector Machine (SVM), use of Principal Component Analysis (PCA) and Vector of Locally Aggregated Descriptors (VLAD) to improve performance, which is called **fixed features extractor**.

In a second time, we will explore **pretrained models** and **fine-tuning** various neural networks. We finally use some image processing methods such as pixel regularization and data augmentation in order to see the evolution of previously listed methods with increased amount of data.

# Deep learning and machine learning methods

**Deep learning is one of the machine learning technique that learns features directly from data.** When the amount of data is increased, deep learning gives better performance.

It’s usually hard to answer on the ‘What is amount of big’ question, and we’ll see later on various dataset sizes.

Today, deep learning is mainly used in speech recognition, image classification, natural language processing or recommendation systems.

The main differences of deep learning from machine learning are :

* machine learning covers deep learning,
* features are given to machine learning manually,
* deep learning learns features directly from data.

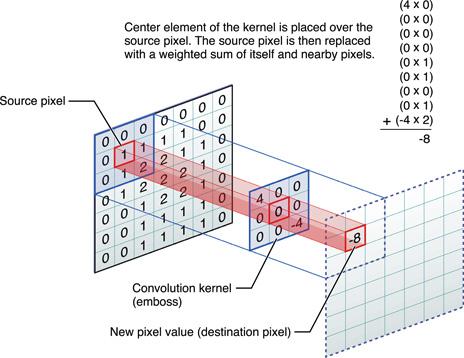
Due to the inherence of deep learning, it’s sometimes hard to get a feedback on what your network learns. We will try to check what we predict as long as our experiments go on.

# 

# Convolutional Neural Networks

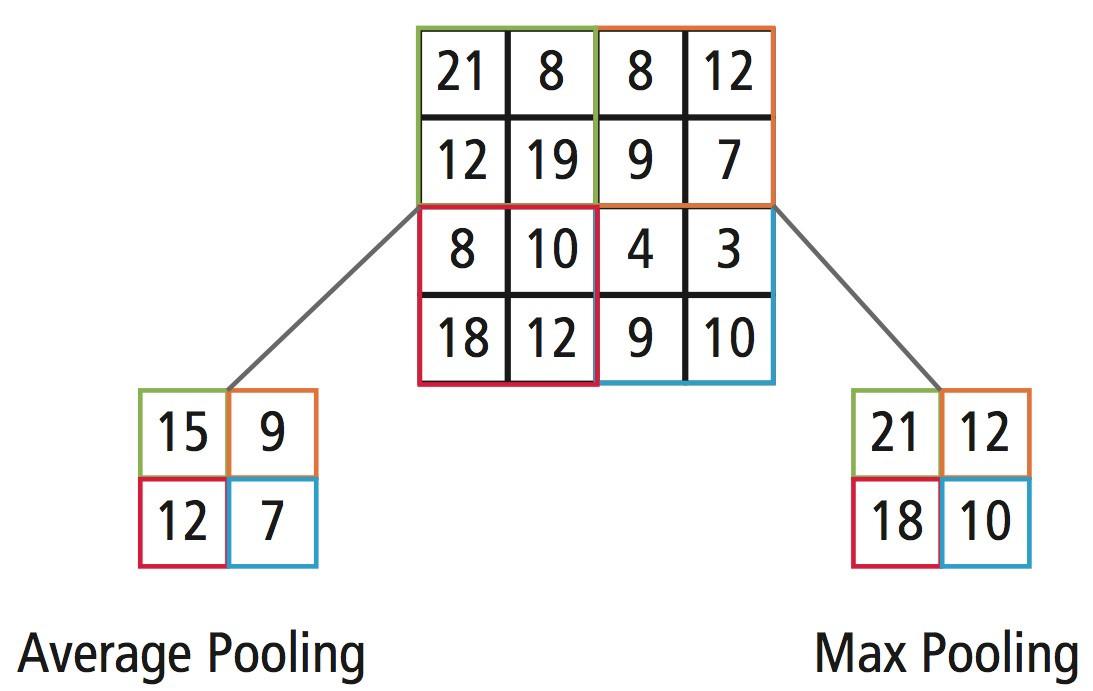
A Convolutional Neural Network - CNN - is an **artificial neural network with multiple layers of so-called convolutions**. To have a global view on how it works, we can see it as a big machine that takes an image as input, which is then processed by the differents layers of the network, and finally outputs probabilities. These probabilities could be anything, so you can predict various things, from facial to cancer recognition.

The network contains three major parts : convolutions, pooling and fully connected layers.



**Convolutions**

A convolution consists as a local analyze : it does a convolution product between the input image (h x w x d) and a filter (fh x fw x d) to return a map of size ((h - fh +1) x (w - fw +1) x 1) if there is no padding.

**Pooling**

A pooling take a convolution map (h x w) and apply it a mathematical function (min, max, average) to get a new map.

**Fully connected layers**

A fully connected layer take the output array of a previous layer and perform a pooling on the different elements by performing operations with all of them.

# 

# 

# Previous work done

This work is a follow-up to the works of two other students : Hichma KARI and Houssem FARHAT.

The work of H. KARI shows that the features extracted from convolutional neural network can be used to classify medical images with a linear SVM. A VGG-19 convolutional neural network pre-trained on ImageNet was used. Features are generated from three last layers of network: block5, fc1 and fc2. The impact of different scales of images and different pooling methods are also studied. The result showed a good prediction on the datasets. His work also mentioned the possibility of using aggregation to extract image features and other networks, which may improve the performance of the classification.

H. FARHAT’s work studied the combination of CNN and Vector of locally aggregated descriptors (VLAD) aggregation used for medical image classification. A VGG-19 CNN pre-trained on ImageNet is also used to extract features from images and the output layer is block5\_pool. The features generated from the same image of three different scales are combined and then reduced to a dimension of 128 with principal component analysis(PCA). Afterwards, VLAD is applied and a linear SVM using the result produced is employed to make the classification. This method didn’t have a remarkable improvement for all the datasets. This work also proposed to use different layer of VGG-19 to extract features or different CNN such as Resnet.

# Datasets used

## Chest X-ray

This data set present chest x-ray images selected from pediatric patients of one to five years old from Guangzhou Women and Children Medical Center, at Guangzhou. The data set contains 5,863 chest x-ray images which can be classify into two categories : those of pneumonia patients and those of healthy patients. The data is split into three folders, a train set, a test set and a validation set, and each of them contains two folders for the two categories. However, we didn’t use the validation set for this classification since it only contains 16 images.

The state-of-art for this dataset is presented in that article : <https://arxiv.org/abs/1711.05225>

You can find more informations about the generalisation to other pathologies here : [https://www.cell.com/cell/fhttps://www.cell.com/cell/fulltext/S0092-8674(18)30154-5ulltext/S0092-8674(18)30154-5#fig6](https://www.cell.com/cell/fulltext/S0092-8674(18)30154-5#fig6)

## Kvasir (version 2)

The data is collected using endoscopic equipment at 4 hospitals in Norway and are carefully annotated by one or more medical experts. The ***kvasir-dataset-v2*** archive contains 8,000 images, 8 classes, 1,000 images for each class. The classes are Z-line, pylorus, cecum, esophagitis, polyps, ulcerative colitis, dyed and lifted polyps and dyed resection margins. We split the dataset into three folders: a train set, a test set and a validation set, containing

Mini MIT

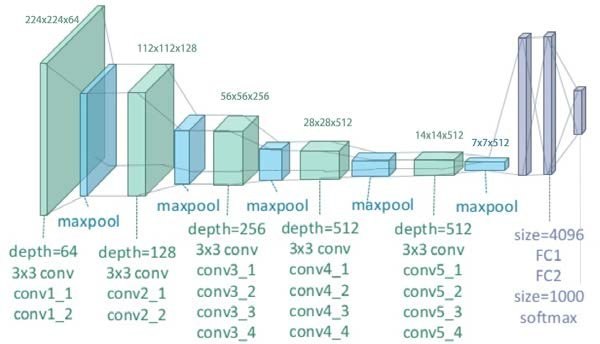
TODO

## Cancer cells

TODO

# Our experiences and works

In practice, it’s rare to train an entire Convolutional Network with random initialization because it requires a very large dataset (probably millions of training images for a network as deep as VGG-19). Instead, it’s more common to use a **pretrained network** either as an initialization or a **fixed feature extractor** for the task of interest. Such an approach is called **Transfer Learning**. We use three of transfer learning scenarios based mainly on the VGG-19 network which detailed structure is shown below :

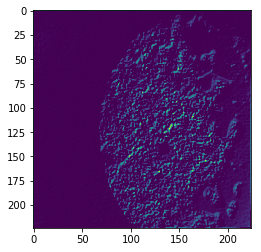
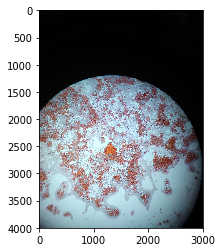


## Fixed Feature Extraction

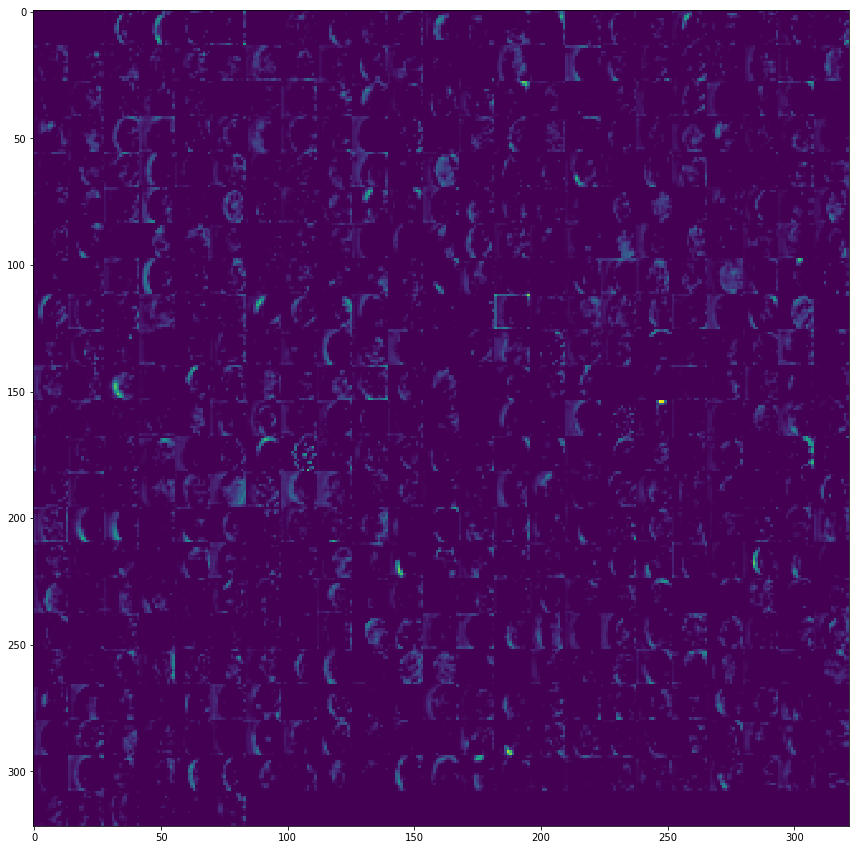
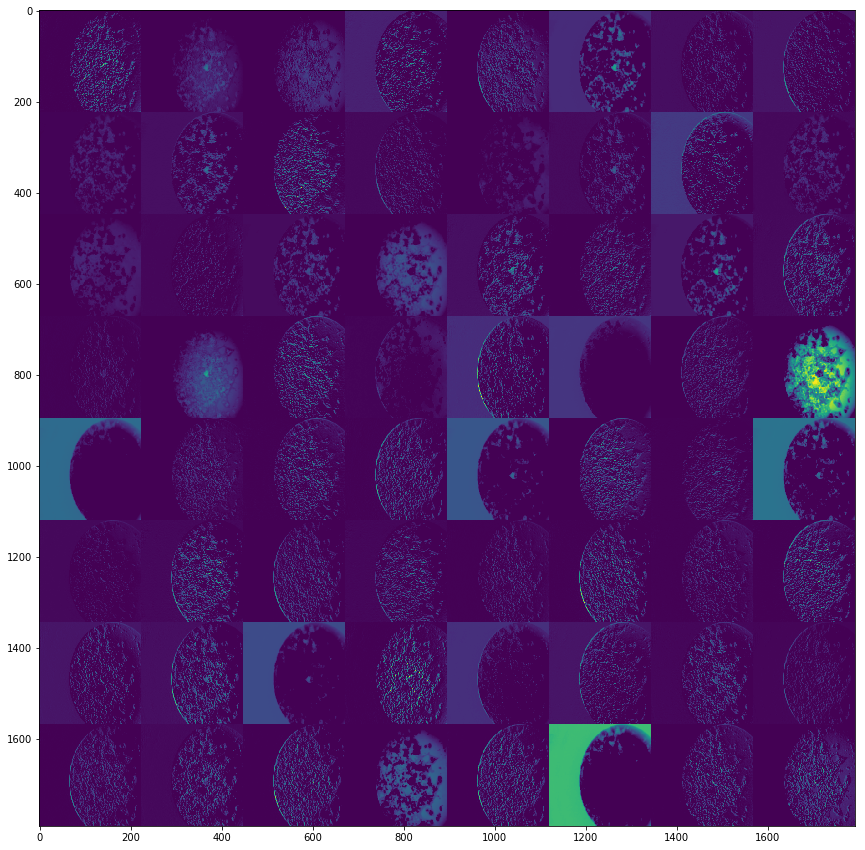
In this step we use a pre-trained neural network called VGG-19 as a feature extractor. This network is pre-trained on the ImageNet, who is composed with five blocks of convolutional layers and three fully connected layers.

Each layer of the convolutional layers represent a certain level of graphic feature. In general, the more convolutional layers we have, the more complicated features can be represented. For example, the first convolutional layer may detect borders of different directions, and the next layer may detect all the combinations of borders. In this way, the last convolutional layer may be able to represent very complicated image features. These features are send to several fully connected layers to produce more complicated features and finally make the prediction.

For example, the image of a cancer cell and the result correspondent of one of the filter in the first layer block1\_conv1 of VGG-19 are shown as below:



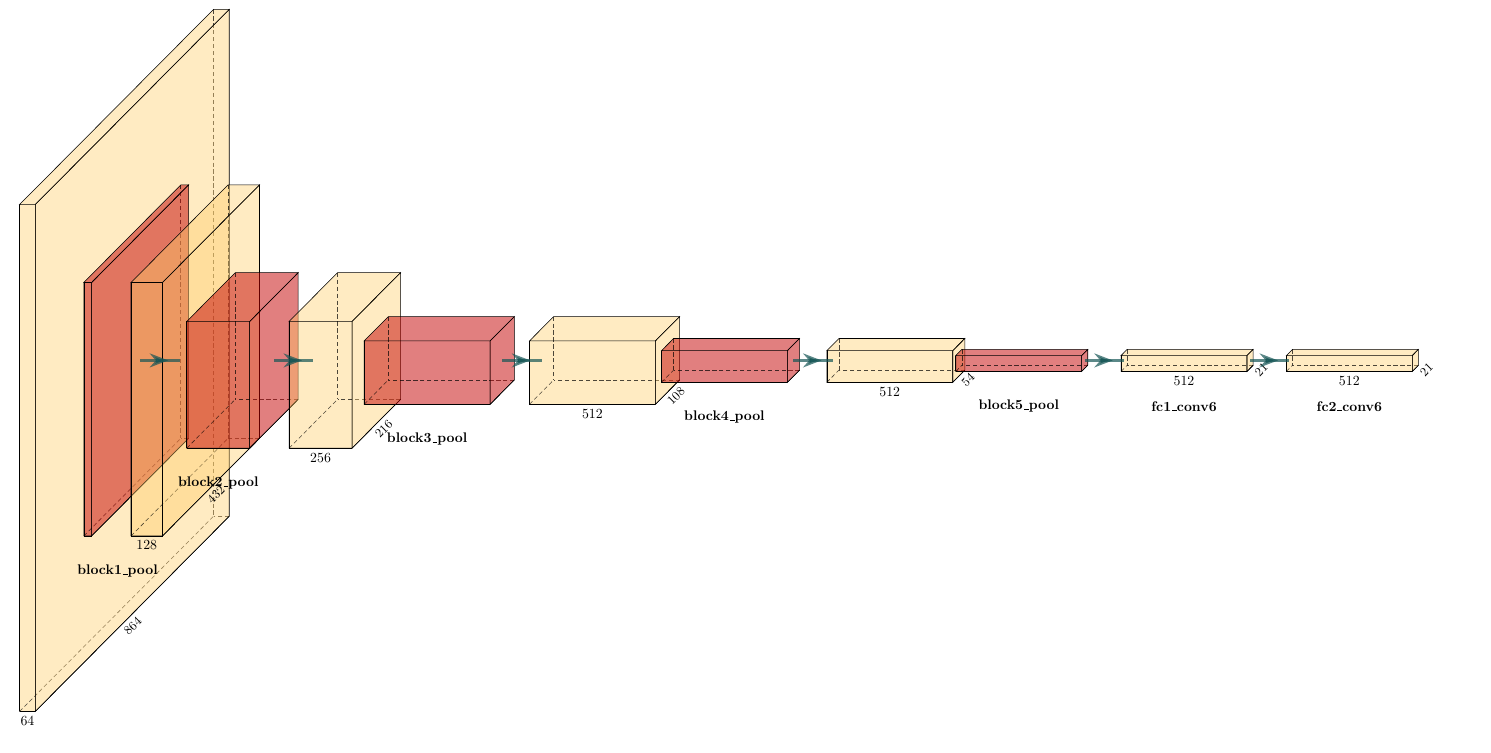
The features of different layers extracted from this image are shown as follows:



block1\_conv1 block5\_conv1

Previous work has shown that features of different scales images extracted from block5\_pool layer of VGG-19 can have good result with linear kernel SVM. The sizes of images are (224+320n, 224+320n,3), where n is the scale, which is among 0, 1 and 2. An average pooling and the L2 normalization are applied to the features. An SVM are then used for the classification. In our work, we want to know the impact of changing layer of feature extraction and different pooling methods. We tried max pooling and average pooling on the features extracted from last three layers of VGG-19: block5\_pool, fc1 and fc2.

VGG-19 is designed for the images of size (224,224,3). To get features correctly of different image scales, the last two layers of fully connected must be changed to convolutional layers. The weights of fc1 and fc2 trained from ImageNet are then used as a filter to be applied to the features. This is therefore a fully convolutional neural network. For example, the neural network structure when image scale is 2 is shown as follow:



The datasets we used in this step is as before: miniMIT, chest\_xray, kvasir\_v2 as well as cancer\_cells. The results are in the table below:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | mean\_pooling | | | max\_pooling | | |
|  | block5\_pool | fc1 | fc2 | block5\_pool | fc1 | fc2 |
| N=0 | 0.78 | 0.83 | 0.82 | 0.78 | 0.83 | 0.82 |
| N=1 | 0.82 | 0.74 | 0.82 | 0.78 | 0.82 | 0.87 |
| N=2 | 0.78 | 0.75 | 0.79 | 0.75 | 0.82 | 0.81 |

miniMIT

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | mean\_pooling | | | max\_pooling | | |
|  | block5\_pool | fc1 | fc2 | block5\_pool | fc1 | fc2 |
| N=0 | 0.80 | 0.80 | 0.81 | 0.78 | 0.80 | 0.81 |
| N=1 | 0.81 | 0.78 | 0.82 | 0.80 | 0.78 | 0.81 |
| N=2 | 0.75 | 0.78 | 0.78 | 0.78 | 0.77 | 0.80 |

chest\_xray

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | mean\_pooling | | | max\_pooling | | |
|  | block5\_pool | fc1 | fc2 | block5\_pool | fc1 | fc2 |
| N=0 | 0.87 | 0.88 | 0.86 | 0.86 | 0.86 | 0.86 |
| N=1 | 0.88 | 0.88 | 0.87 | 0.85 | 0.88 | 0.87 |
| N=2 | 0.86 | 0.87 | 0.86 | 0.81 | 0.86 | 0.87 |

kvasir\_v2

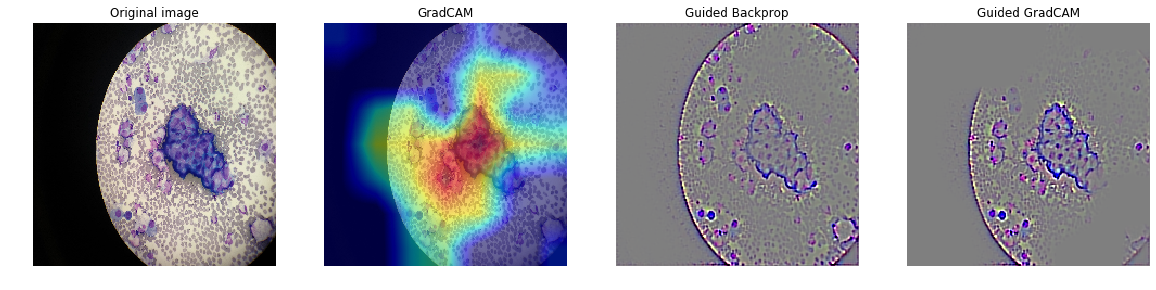
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | mean\_pooling | | | max\_pooling | | |
|  | block5\_pool | fc1 | fc2 | block5\_pool | fc1 | fc2 |
| N=0 | 0.67 | 0.62 | 0.59 | 0.69 | 0.63 | 0.59 |
| N=1 | 0.88 | 0.78 | 0.78 | 0.59 | 0.88 | 0.78 |
| N=2 | 0.61 | 0.51 | 0.51 | 0.71 | 0.78 | 0.76 |

cancer\_cells

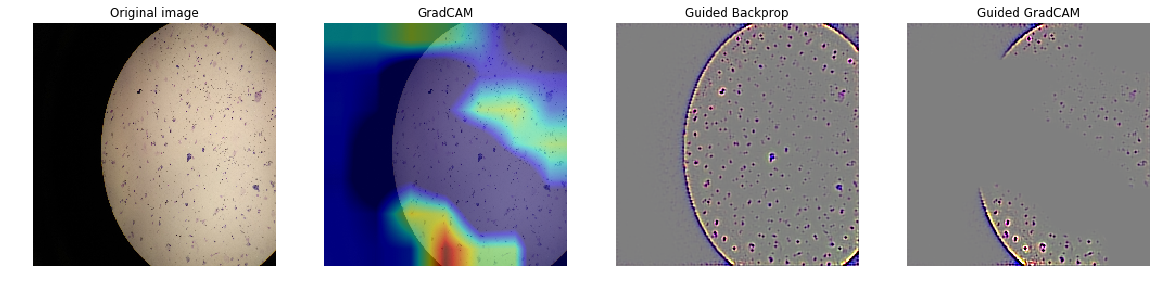
The result shows that when we extract features from fc1 layer or fc2 layer, max pooling almost always gives higher accuracy score than mean pooling. While extracting features from block5\_pool, mean pooling is a better choice.

We have also notice that in some small data sets, such as miniMIT\_Etus and cancer\_cells, extracting features from two last layers can have significant improvement to the accuracy score. However when the size of data set is relatively large, the influence of extracting features from different layers become not so obvious.

To give an intuitive representation of the prediction, we used GradCAM to visualize what we have extracted from the layer block5\_pool. GradCAM, which means Gradient-weighted Class Activation Mapping, can produce a coarse localization map highlighting the important regions in the image for predicting. It uses the gradients of a image, flowing into certain convolutional layer. The advantage is that it is applicable to a wide variety of CNNs. Here is the result of GradCAM applied to the images in cancer\_cells dataset. The red zone stands for the highlighting region that gives important features for predicting.



Cancer cells



Normal cells

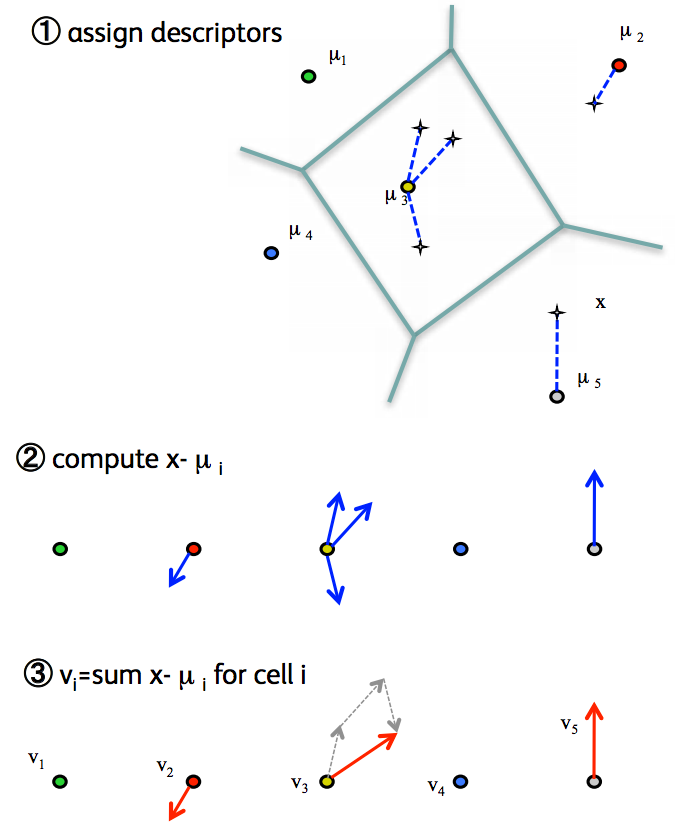
## PCA and VLAD

To improve the classification of images, some methods based on image vector representation are also often used. According to the Identity document classification as an image classification problem, features extracted from pre-trained CNN can be successfully transferred as image descriptors. Among different image descriptors, the more refined encoding results in the longer the global descriptor, and therefore the better performance.

Before applying these methods, Principal Component Analysis need to be employed to reduce the dimension of vectors. The number of dimensions is reduced to 128. Then we use VLAD as image descriptor, which is a simplified version of Fisher Vectors. It only takes the difference between descriptors and their closest centre of K-means clustering. Supposing a set of features I=(x1, x2, .., xn) extracted from an image, and μ\_k are the cluster means which are the same dimension as features x\_i. Features are encoded by VLAD by considering the differences between x\_i and their closest centre μ\_k:

v\_k=∑q\_{ik}(xi−μk).

And the vectors are then stacked together as a global descriptor: (v\_1, v\_2, ..., v\_k)^T. Its principle can be expressed as the following figure:



Aan L2 normalisation are also performed after VLAD. Because of the requirement of huge memory and storage when processing big data sets, we are only able to test on the relatively small datasets. We have also managed to reduce the size of descriptors of big dataset by choosing them randomly, but the result was not very reliable. The result of small datasets is shown below:

|  |  |  |  |
| --- | --- | --- | --- |
|  | number of descriptors | 64 | 256 |
| block5\_pool | miniMIT\_Etus | 0.82 | 0.84 |
| cancer\_cells | 0.77 | 0.74 |
| fc1 | miniMIT\_Etus | 0.71 | 0.84 |
| cancer\_cells | 0.77 | 0.71 |
| fc2 | miniMIT\_Etus | 0.82 | 0.82 |
| cancer\_cells | 0.79 | 0.74 |

We noticed that changing layers didn’t have significant improvement on the accuracy score. The augmentation of number of k-means center also does not give better results on all data sets.

## Pretrained models

The third strategy is to take a pretrained model - again the VGG19 - and try to find the best layers to fine-tune. To start, we remove the last layer which is a dense layer with 1000 neurons (the number of classes for the imagenet dataset) and replace it by a dense layer with the number of classes of our dataset. The idea behind that is that such a network has been trained on a big amount of data (millions of images) and will be able to catch a lot of features useful for our task.

It gives us a nice baseline to start. Then, we try to go further and remove the weights of more layers : fc2, fc1 and then all the convolutional ones.

[ … convolutional blocks of the VGG-19 model … ]

layer name size params

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (25088) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fc1 (Dense) (4096) 102764544

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

fc2 (Dense) (4096) 16781312

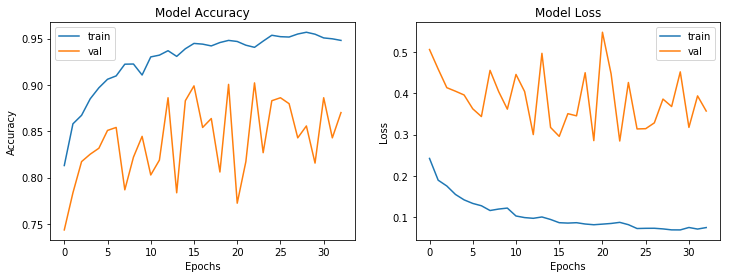
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

predictions (Dense) (2) 8194

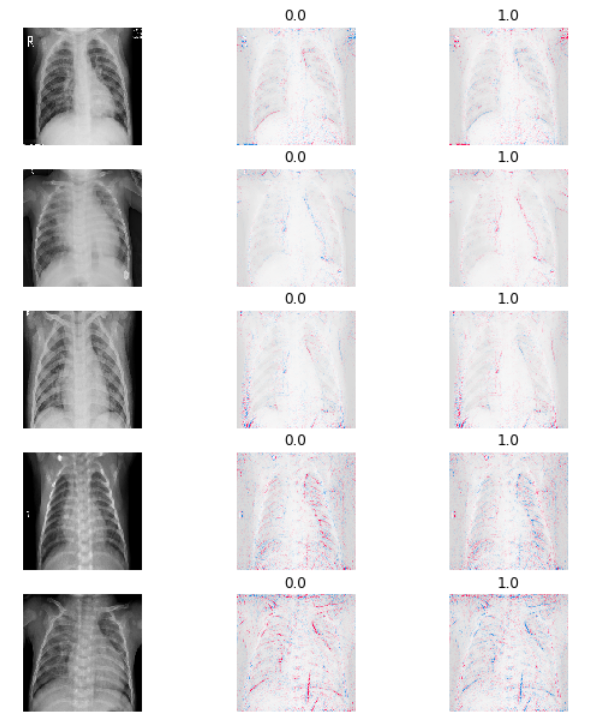
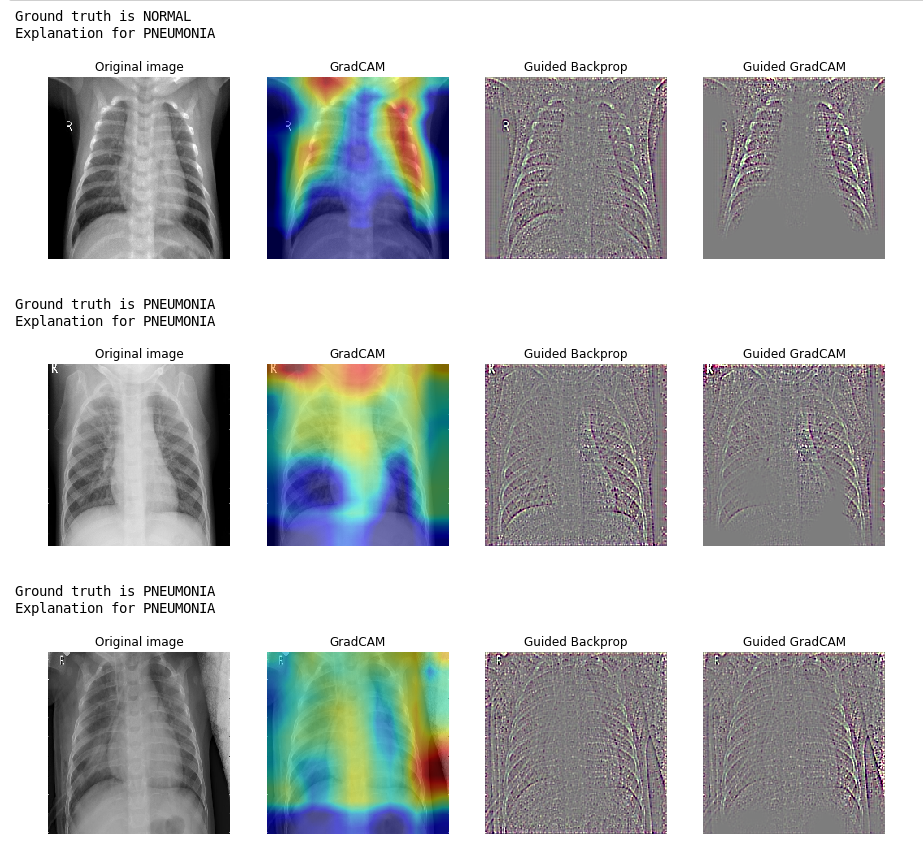
Results are presented on the tableau below for the chest\_xray dataset. Training was done for 300 epochs (we defined an early stopping callback at 10 epochs without any improvements). The optimizer chosen was Adam with a learning rate of 1e-6.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Frozen layers | Accuracy | Recall | Epoch | Trainable params |
| \_ | 0.892 | 0.953 | 1 | 139.578.434 |
| blocks | 0.878 | 0.979 | 2 | 119.545.856 |
| fc1 | 0.829 | 0.992 | 2 | 36.813.544 |
| fc2 | 0.816 | **0.995** | 1 | 122.797.122 |
| blocks, fc1 | **0.902** | 0.964 | **19** | 16.781.312 |
| blocks, fc2 | 0.851 | 0.969 | 2 | 102.764.544 |

Results above are clearly better if the top layers are trainable and the bottom ones frozen. Training only the deep layers was expectable to give bad results. It shows also that we obtain our best results on the first epochs of the training. Thus we can predict our model to hugely overfit. A notable exception is the training where only the last layer was trainable. We got the best accuracy and a very good recall and our training history seems to be less overfitting.



To check if our intuition is correct, we can visualize what our model is predicting. We will use two tools for that purpose : [GradCAM](https://arxiv.org/abs/1610.02391) and [Shap](https://github.com/slundberg/shap) (seems that [CNN Fixations](https://arxiv.org/pdf/1708.06670.pdf) is also very good but we didn’t try it).



Even if we have our best results in terms of metrics (accuracy, precision, recall), we cannot be happy about what our model is predicting. This is mainly motivated by the observation that the features of the pretrained model contains features that should be useful for the tasks it was originally designed. We seems to **mainly predict ‘chest with pneumonia/normal’ rather than ‘pneumonia/normal’**.

For the sake of comparisons on other datasets, we got these results using the same architecture as above with the best method (only fc2 layer is trainable) :

|  |  |
| --- | --- |
| Dataset | Accuracy, recall |
| chest\_xray | 0.902, 0.964 |
| cancer\_cells | 0.78, |
| kvasir\_v2 | 0.82, |
| miniMIT | 0.77, |

Up to that point, we can observe two major points :

* pretrained models are better on larger datasets,
* pretrained models, even on larger datasets, are overfitted.

The first point was predictable and we asserted it on our introduction. The second point is a bit more tricky, we can see that the last two FC contains 4096 neurons each. In case the input image had shape (224, 224, 3), the last convolutional layer will have shape (7, 7, 512). So we got 7x7x512x4096 = 102.764.544 params to train only on the fc1 layer. Thus, since our datasets don’t exceed 5000 train images, our model will certainly assign one neuron to each image… which is really really bad. We can move around that by adding one [Dropout](http://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf) layer between each FC and decrease the number of neurons in our FC. We tried the following classification block :

x = Flatten(name='flatten')(x)

x = Dense(1024, activation='relu', name='fc1')(x) # 4096

x = Dropout(0.7, name='dropout1')(x) # not present

x = Dense(512, activation='relu', name='fc2')(x) # 4096

x = Dropout(0.5, name='dropout2')(x) # not present

x = Dense(num\_classes, activation='softmax', name='predictions')(x)

Results, as shown as below, are not better.

|  |  |  |
| --- | --- | --- |
| Dataset | VGG-19 changed classification block | VGG-19 classification block |
| chest\_xray | 0.908, 0.971 | 0.902, 0.964 |
| cancer\_cells | 0.78 | 0.78 |
| kvasir\_v2 | 0.82 | 0.83 |
| miniMIT | 0.77 | 0.78 |

## Fine-tuning

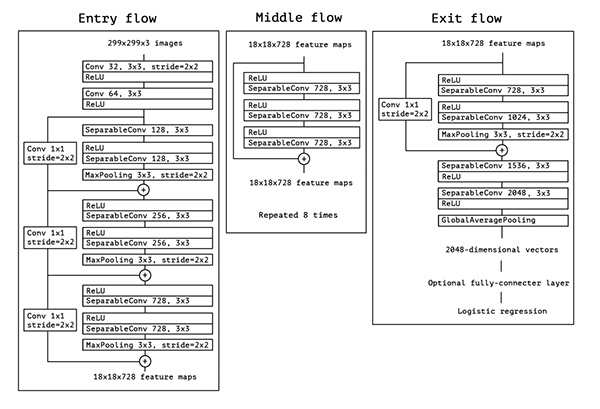
The strategy here is not only replace and retrain the classification block on top of the ConvNet, but also fine-tune the weights of the pretrained network. Since the datasets used are very different from the original dataset, we could predict our training to not get profit from the weights of the full network. It’s mainly motivated by the observation that the earlier features of a ConvNet contains more generic features that could be useful for many tasks, but the late layers becomes progressively more specific to the details of the classes contained in the original dataset. Thanks to Class Activation Mapping, we can check which convolutional layers are useful for our task.

[TODO ADD CAMS FOR LAYERS]

[TODO add final conclusion of VGG-19 network]

## Some custom CNNs

The [Xception](https://arxiv.org/abs/1610.02357) network proposed by François Chollet is detailed on the next figure.



That network is build on top of **SeparableConv** which are **Depthwise Separable Convolutions**. The main difference between both convolutions is that in the normal convolution, we first transform the image *Z* times. If we had a kernel of size 5x5 with 3 channels (RGB) applied on a 12x12 image, we would have *Z* 5x5x3 kernels that moves (12-5)x(12-5) times. Thus it does 4800 x *Z* multiplications. In the case of separable convolution, since we separate in 2 steps, we will have 3 kernes of shape 5x5x1, each moving 8x8 times. That gives us 4800 multiplications. In the second step, the pointwise convolution, we have Z 1x1x3 kernels that move 8x8 times, leading to 1x1x3x8x8 = 192 *Z* multiplications.

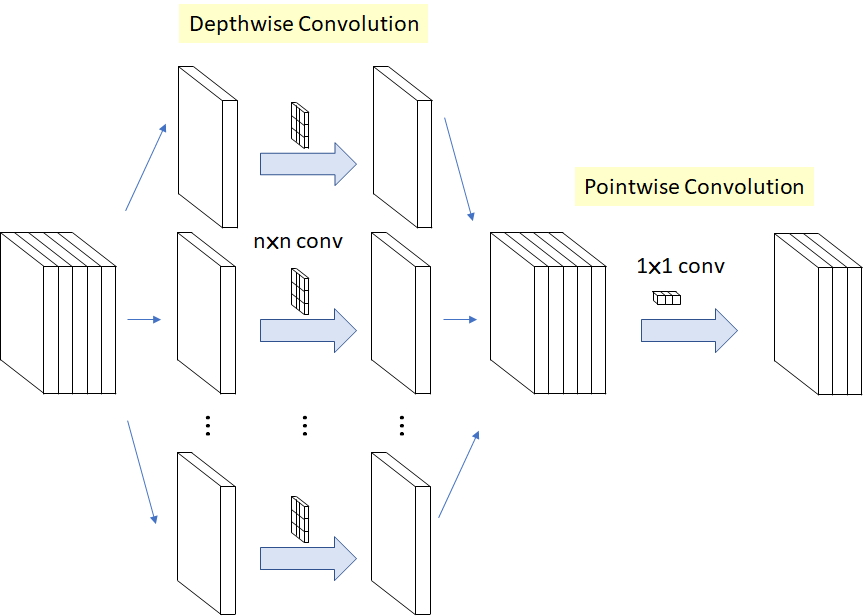
We end up with two formulas for an input image of 12x12x3 pixels and a 5x5x3 convolution.

*Z* denotes the number of channels, usually 256 for RGB standard.

Normal convolution : **4800 *Z* multiplications**

Depthwise convolution : **4800 + 192 *Z* multiplications**

You can see difference like this :in the normal convolution, we transform the image as many times as we got channels. On the opposite, in a separable convolution, we transform the image once (depthwise convolution) then we elongate it to the number of channels desired (pointwise convolution).



## 

As a network point of view of convolutions, using depthwise ones drastically reduces the number of parameters for a block of convolution but it doesn’t reduce significantly performance. The idea of the François Chollet was to add a lot more layers with fewer parameters. He obtained better results than VGG-19 on ImageNet dataset.

It would be interesting to explore that network exactly as we did with the VGG-19, and keep the architecture with the exception of the last fully connected layers replaced in the same way we explained earlier.

We explore a different approach and try to combine the VGG-19 architecture with Xception by replacing the VGG-19 Conv2D layers to SeparableConv2D followed by BatchNormalization. We use that classification block :

x = Flatten(name='flatten')(x)

x = Dense(512, activation='relu', name='fc1')(x)

x = Dropout(0.7, name='dropout1')(x)

x = Dense(216, activation='relu', name='fc2')(x)

x = Dropout(0.5, name='dropout2')(x)

x = Dense(num\_classes, activation='softmax', name='predictions')(x)

Previous convolutional blocks become :

[ … ]

x = SeparableConv2D(256, (3, 3), activation=’relu’))

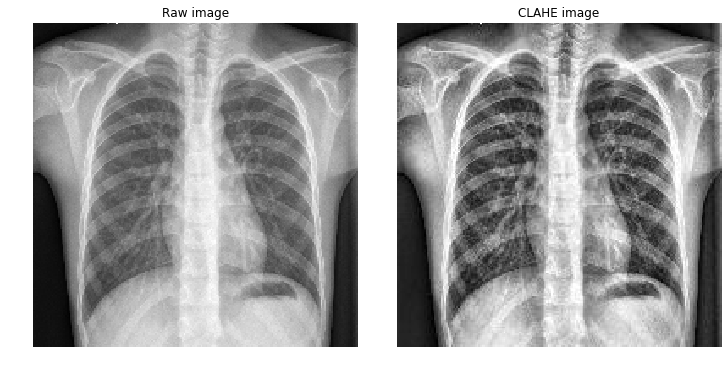
x = BatchNormalization(x)

[ … ]

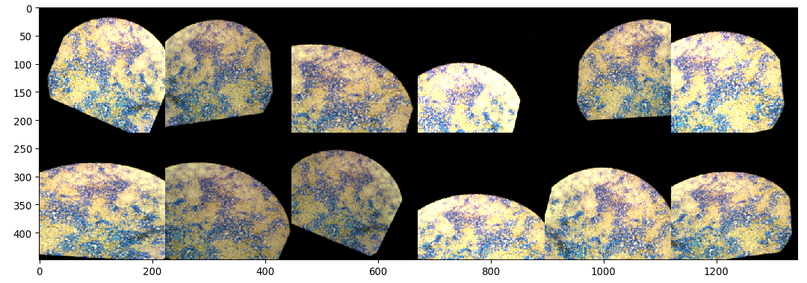
|  |  |  |
| --- | --- | --- |
| Dataset | VGG-19 with depthwise convolutions | VGG-19 previous best |
| chest\_xray | 0.93 | 0.908 |
| cancer\_cells |  |  |
| kvasir\_v2 |  |  |
| miniMIT |  |  |

## Histogram equalization - CLAHE

We try to use the [Contrast Limited Adaptive Histogram Equalization](https://en.wikipedia.org/wiki/Adaptive_histogram_equalization#Contrast_Limited_AHE) for enhancing the local contract of images and see if we can get better results. Even if the images are way more readable for humans, we didn’t get better results. An interesting feature that we didn’t compute would be to combine both images to see if we can improve our predictions.



## Data Augmentation

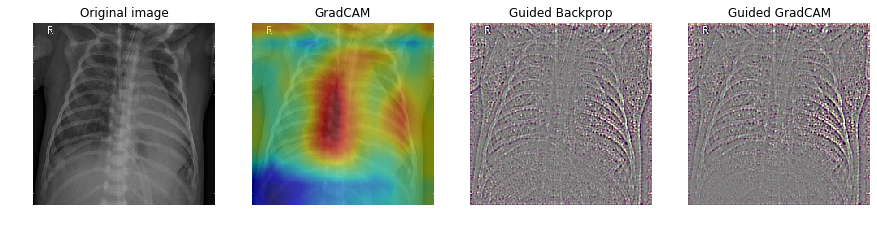
We lately tried data augmentation for the small datasets in order to see the impact of increasing data. We tried [Albumentations](https://arxiv.org/pdf/1809.06839.pdf) and [Imgaug](https://github.com/aleju/imgaug). Data augmentation prevents the network to see the same image twice during the training process. You can see 16 differents views of a same image on the following figure.

Data augmentation offered excellent improvements on the two small datasets.

|  |  |  |
| --- | --- | --- |
| Dataset | VGG-19 with data augmentation | VGG-19 without |
| miniMIT\_Etus | 0.83 | 0.78 |
| cancer\_cells | 0.84 | 0.78 |

# 

# Final results



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Dataset | Dataset shape | Best method name | Accuracy, Recall | Previous best |
| Mini MIT Etus | train (120, 3)  test (120, 3) | Feature extraction with max pooling on layer fc2 with image scale (864, 864) + linear SVM | 0.87 | 0.84 |
| Cancer cells | train (72, 2)  test (26, 2)  val (26, 2) | Feature extraction with max pooling on layer fc1 with image scale (544, 544) + linear SVM | 0.88, 0.8 | - |
| Kvasir v2 | train (4800, 8) test (1600, 8) val (1600, 8) | fine-tuning the vgg-19 with weights kept from the first 3 layers | 0.92, 0.94 | 0.88 |
| Chest Xray Pneumonia | train (5221, 2) test (624, 2)  val (16, 2) | fine-tuning the vgg-19 + depthwise convolutions | 0.94, 0.97 | 0.78 |

# Conclusions and future improvements

When, how and which Transfer Learning scenarios should you use for your next image classification project ?

With our experiments on 4 datasets of various properties, we can conclude that using one or another type of transfer learning you should perform depends of several factors. The most important ones are the size of the dataset and its similarity to the original dataset.

If the dataset you use is quite large (at least thousands of images), it can be a good move to try to fine-tune through the full network. In case the data is similar to the original dataset (as in Kvasir v2 in a certain measure), you can be enough confident to keep the same architecture the original network had. If the data is very different from the original dataset (as in Chest X-ray Pneumonia), it can be a good idea to explore a network and keep weights and architecture of the first layers, which are very generic, then train again the network by running backpropagation.

If the dataset you use is small, and in our case it was *really* small with under 250 images, it might not be a good idea to fine-tune the whole network due to overfitting concerns.

In case the data is similar to the original data (as in mini MIT Etus), the best idea might be to train a linear classifier (SVM, PCA, VLAD) on the fully connected layers extracted directly from the network.

In case the data is very different from the original dataset (as in Cancer Cells), you should explore a given network to find a good activation layer somewhere earlier in the network.