Towards a plant pathologies detection solution

Abou SANOU[1], Jean Serge Dimitri OUATTARA[2], Didier BASSOLE[3], Abdoulaye SERE[4], and Yaya TRAORE[5]
{aboudra1996@gmail.com[1], jean.ouattara@ujkz.bf[2], dbassole@gmail.com[3] }

Université Nazi BONI [1], Université Joseph KI-ZERBO [2], Université Joseph KI-ZERBO [3].

Abstract. Agriculture is very important in Africa. But farmers are dealing with many plants pathologies that keep agriculture from developing and boosting the economy. Current pathologies diagnosis based on human screening is time consuming and expensive, while computer vision-based models are efficiently promising.That is why we address the problem of identification of the pathologies affecting agricultural crops with computer vision tools. Though the high variability of symptoms due to the age of infected tissues, genetic differences, and light conditions in trees reduces the precision of detection, we propose a method to achieve our goal.

Keywords: Plant pathologies, Computer vision, EfficientNet, ResNet, Deep Learning

1 Introduction

The problem we are dealing with is the lag time in the diagnosis of plants pathologies. These pathologies can jeopardise crops and soil health if they are not detected early. Our challenge is to create a model that could detect pathologies from images of plants leaves. We use an existing dataset containing images of healthy and well-labelled infected leaves. The dataset consider 3 categories of leaves : two with diseases (rust, scab) and one healthy. Our goal is to classify in an accurate way the images of the leaves in one or multiple categories of diseases. Indeed one leave can have multiple diseases and our model must be able to detect this kind of situation.

2 Preliminaries

Before any formal analysis of the data, we have to know the number of items and the variables in the data set, the number of missing observations and the general assumptions the data suggest. To answer these questions, an initial exploration of the dataset will allows us to become familiar with the data that will handle. Data exploration or data description is a very important phase in any machine learning project. This step allows us to get more details about the dataset which we will be used to train our model.

We got the dataset from kaggle [1]. The database folder consists of **3642** images in jpg format. Each image is in **2048** pixels * **1365** pixels with **3** channels(RGB). Added to these images, there are three others files.

JRI 2021, November 11-13, Bobo-Dioulasso, Burkina Faso
Copyright © 2022 EAI
DOI 10.4108/eai.11-11-2021.2317975

- sample_submission.csv: this is the sample for the submission. We will start by simply looking at a few random samples.
- train.csv: this file contains the training set with its labels. We will use it during the training of the model.
- test.csv: this file will contains the test set.

After describing the files which constituted the dataset, we will explore it through statistical metrics.

Let's start with the distribution of the dataset row (image) over the classes. To sum up this information let's draw a pie chart 1

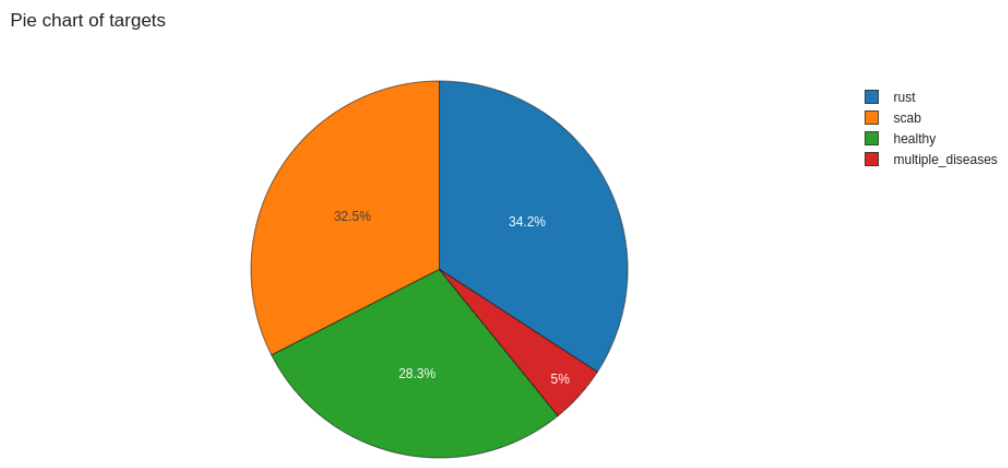


Fig. 1. Distribution of items in categories

The number of photos is quite similar for each target, unless for multiple diseases class.

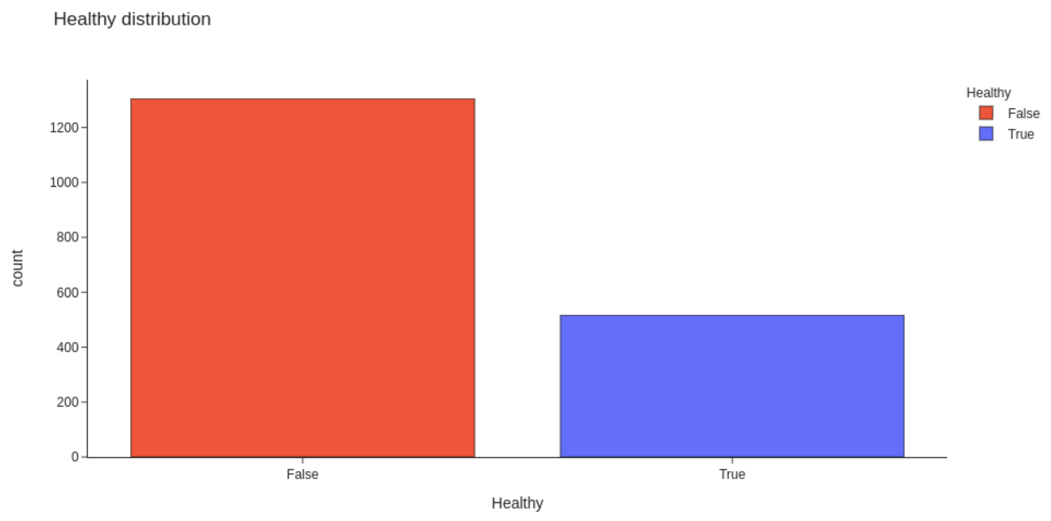


Fig. 2. Distribution of items in: Diseases or healthy

According to the plot 2, this dataset is pretty balanced.

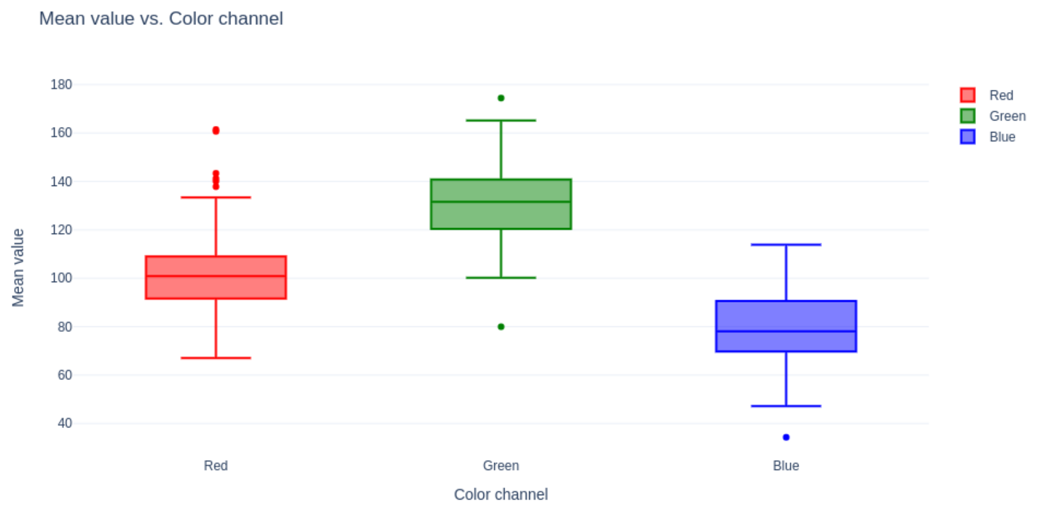


Fig. 3. Distribution of item in RGB

Let's explore the color distribution of images refers to figure 3:
Now, let see the distribution on the histogram, the distribution of color channel 4 :

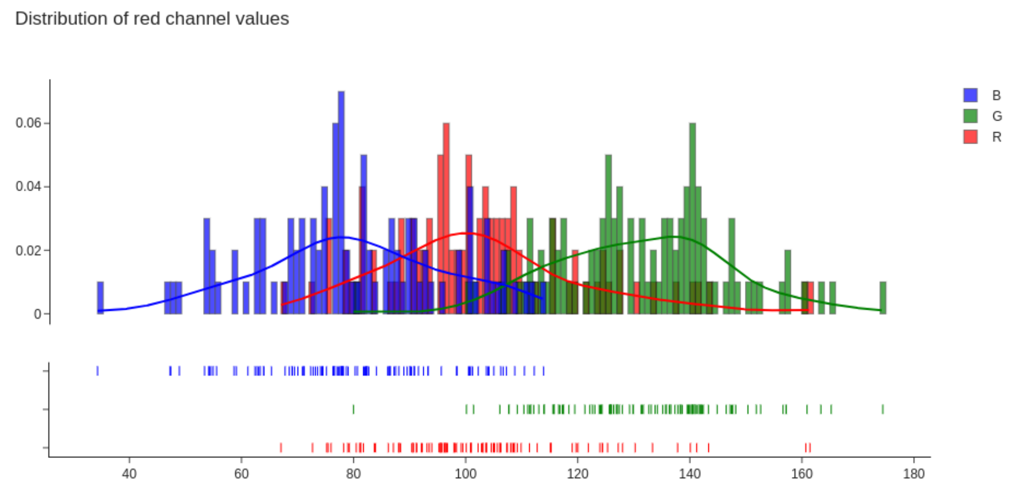


Fig. 4. Distribution of color:

We can notice that the color distribution shape for each channel is quite similar with some shifts. Of course the green one is the most prominent due to the dominance of the green colour of the leaves. From this point, we can suppose that the model classifying process may rely on this distribution. After the exploration of the data we will go on to the preparation of the data in order to make it easy for the extraction of knowledge on the model.

2.0.1 Data pre-processing :

it is the concept aiming to slightly modify the original database in order to make it clean. A clean database allows a more exhaustive learning and prevents more overfitting[2]. The different techniques used in data augmentation and preparation have to rely on the specific application needs. In the context of computer vision, we speak about image preparation. There are several techniques coming from the potential, pictures environment related, image distortion. These can be due to the diversity of devices leading to different processing of the the same captured picture. Weather conditions or the point of view that the photograph has chosen for his pictures can also influence the picture. The image pre-processing techniques aim to to simulate those diverging behaviours.

This is a crucial part for our study. Indeed, performing Image classification implies dealing with high dimensions data. So in order to facilitate the training of the data in this crucial context, we have to prepare images. We use the following process :

Canny edge detection is an edge detection algorithm as noticed by its name. It was created by John F. Canny in 1986. The algorithm involves in several steps.

- **Noise reduction:** Since edge detection is susceptible to noise in an image, we remove the noise in the image using a 5x5 Gaussian filter[3]. So recall the Gaussian function in one dimension also call function of density:

  $$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2 \big/ 2\sigma^2}$$

  Let's assume that sigma is the standard deviation of the distribution (in our case of image). The distribution is assumed to have a mean of 0. More simply will reduce the range of the dataset around the mean 0; with the standard or (ecart-type) 1. Let's notice that as well as standard deviation is large the range of the distribution will be large. In our case we will use the gaussian function on two dimensions as following :

  $$P(x,y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\left(x^2+y^2\right) \big/ 2\sigma^2}$$

  Now let's see how we are going to apply this distribution on our dataset or more clearly how to bring back our distribution in normal distribution. Firstly we have to compute the standard deviation and mean of each dimension of distribution of the images. After that, for each value we can do the values minus the mean in order to get the mean of the distribution to 0. And we have to divide the result value with the standard of derivation in order to get the standard of derivation to 1. With these metrics we can define the Gaussian functions in the image field this function is a kernel or matrix. After getting a Gaussian kernel which is a matrix, we have to apply convolution operation between the kernel and each image or each channel of image.

- **Finding Intensity Gradient of the Image [4]:** First, the gradient in computer vision is the directional change in image if we loop on coordinate. Now we have to compute edge gradient and Angle of gradient. Let consider the image I represented in a matrix, the gradient in the pixel with the coordinates x=c and y=r can be compute as follows :

  $$dx = I(c+1,r) - I(c-1,r) \; dy = I(c,r+1) - I(c,r-1) \tag{1}$$

  This means calculating the partial derivative of the function represented in our case by the matrix. I(x,y) is a luminosity in one channel The gradient magnitude is how strong is the change of the luminosity if we can position it in image. To find the Intensity Gradient of the Image, we need two metric : The gradient orientation and the Edge Gradient. The gradient orientation is computed as followed :

  $$\theta = \tanh^{-1}\left(dx^2 + dy^2\right) \tag{2}$$

  The magnitude or edge gradient equals :

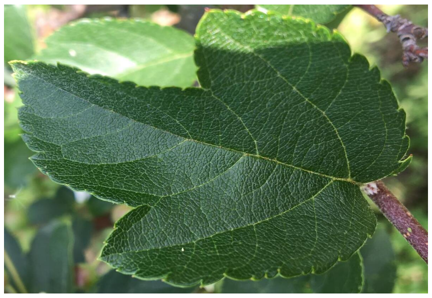
  $$Magnitude = \sqrt{\left(dx^2 + dy^2\right)} \tag{3}$$

- **Rounding :** The gradient is always perpendicular to edges. So, it is rounded to one of the four angles representing vertical, horizontal and two diagonal directions.

- **Non-maximum suppression :** After getting the gradient magnitude and direction, a full scan of the image is done to remove any unwanted pixels which may not constitute the edge. For this, we check every pixel for being a local maximum in its neighborhood in the direction of the gradient.

- **Hysteresis Thresholding:** This stage decides which parts are edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient greater than maxVal are considered edges and those lesser than minVal are considered non-edges, and discarded. Those who lie between these two thresholds are classified edges or non-edges based on their neighborhood. If they are near "sure-edge" pixels, they are considered edges, and otherwise, they are discarded.

(a) Original Image



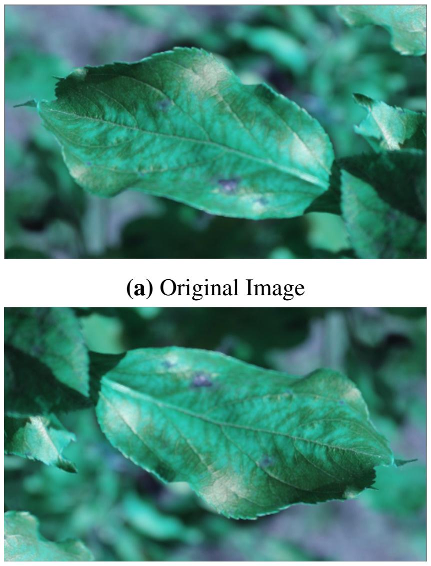
(b) Image Cropped with canny edge detection

The result of these five steps is a two-dimensional binary map (0 or 255) indicating the location of edges on the image. With this method we reduce the size of the image to focus only on the leaf. We cropped the Image as we can see on the following screen.

Flipping is a simple transformation that involves index switching on image channels. In vertical flipping, the order of the lines is exchanged. In vertical flipping, the order of the lines is exchanged. Let's assume that "long" and the "lag" is the size of the In order to perform the horizontal flipping

the pixel (x, y) will be situated at coordinate (long - x + 1, y) in the new image.
The figure 6 illustrates as well as possible the flipping operation.

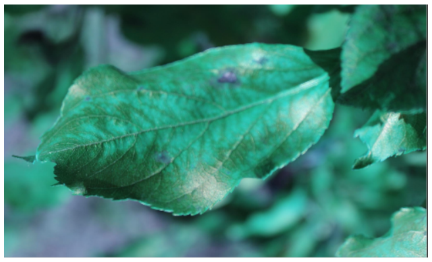
- Horizontal Flip :
$$HorizontalFlip : A_{ij} = A_{i(long+1-j)} \tag{4}$$
- Vertical Flipping :
$$HorizontalFlip : A_{ij} = A_{(lag+1-i)j} \tag{5}$$


(a) Original Image

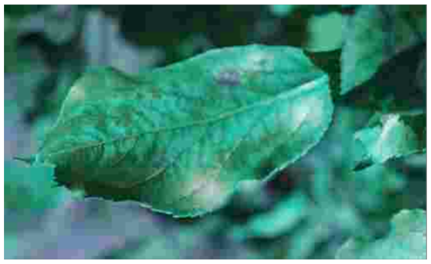
(b) Image flipped

Fig. 6. Illustration of Flipping

Gaussian blur and Compression : As we know that sometimes the image quality taken for prediction can be a bad one, we have to consider the quality of this image by making our model more robust. Blurring is simply the addition of noise to the image, resulting in a less-clear image. The noise can be sampled from any distribution of choice, as long as the main content in the image does not become invisible. Only the minor details get obfuscated due to blurring. We can also use the image compression algorithms in order to decrease the image quality, in view to make the mode The following image is set to 90% jpeg compression. To illustrate this operation in simply way, let's checkout the figure. 7



(a) Original Image



(b) Compressed Image

Fig. 7. Illustration of compression 10%

3 Method:

In order to choose the most efficient model for our task of classification, we have decided to collect different benchmarks over recent papers which perform image classification in general and more precisely which perform plant pathology's detection tasks. In view of the constraints of the domain due to the low power of the device which will host this model, we have to find the most efficient model which requires less memory space and computation power. To reach our goal, we read the following articles : [5], `https://paperswithcode.com/task/image-classification` [6]. The resources present in a synthetic way the benchmarks on many state of art. Through these papers and benchmarks, we keep on our notebook the following models: ResNet, and EfficientNet. Before diving deep into these models let's notice that in order to gain efficiency, we will train these models in transfer learning context. Transfer Learning is the ability to reuse an trained model to solve other problems. The main purpose of Transfer learning is efficiency during training. It has been proven that training a model from scratch is more expensive than training a model from a pre-training model. In our case, our model was pre-trained on `http://www.image-net.org/` dataset. Let talk about the selected models :

3.1 ResNet :

ResNet stands for Residual network. This model appears in response to the gradient vanishing problem[10] in Convolution Neural Network with a large depth. Let explain the context of apparition of this model. In the past to increase accuracy of image classification tasks, the basic approach was to scaling up the model. By adding more layers of convolution to the model or make greater the width of the network. The following schema 8 will explain different kinds of scaling up of network.

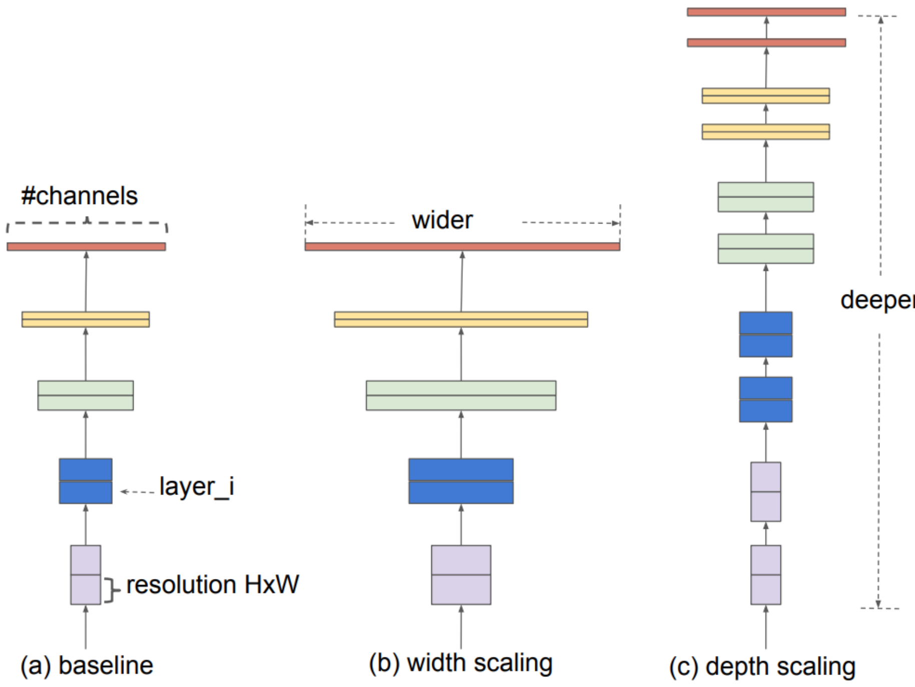


Fig. 8. Illustration of scaling of Deep Neural Network : from [7]

But in this way, we need more and more power to compute the weights of this kind of network. After a long time spent scaling up a single dimension at the same time(either channel, either width either depth) of the networks in order to improve the accuracy of classification, a main problem has been observed. This problem is the gradient vanishing problem [8]. As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train. Some activation functions, such as the sigmoid function, compress a large input space into a small input space between 0 and 1, so a large change in the sigmoid function input will result in a small change in the output. Consequently, the derivative becomes small. For a shallow network with only a few layers using these activation's function, this is not a big problem. However, when multiple layers are used, the gradient can be too shallow for the training to perform well. The gradients of the neural networks are determined using back-propagation. In very simple terms, back-propagation is used to find network derivatives by moving layer by layer from the top layer to the bottom layer. According to the chain rule, the derivatives of each layer are multiplied in the network (from the final layer to the initial layer) to calculate the derivatives of the initial layers. However, when n hidden layers use an activation like the sigmoid function, n small derivatives are multiplied together. Thus, the gradient decreases exponentially as we propagate down to the initial layers. A small gradient means that the weights and biases of the initial layers will not be updated effectively with each training. As these initial layers are often crucial in recognising the essential elements of the input data, this can lead to an overall inaccuracy of the entire network.

- The simplest solution is to use other activation functions, such as ReLU, which does not cause a small derivative.
- Also, batch normalization layers can fix the issue. The problem is occurred when a large range of input is mapped to a tiny one. Batch normalization reduces this problem by normalizing the input.
- Another solution is the Residual network. It is characterized by the residual connections straight to earlier layers. The residual connection directly adds the value at the beginning of the block, to the end of the block (F(x)+x). This residual link does not go through activation functions that "overwrite" the derivatives, resulting in a higher overall derivative of the block.

In view of these problems, we saw the creation of a new network called Residual neural network (ResNet) . The following figure 9 sums up the new approach :

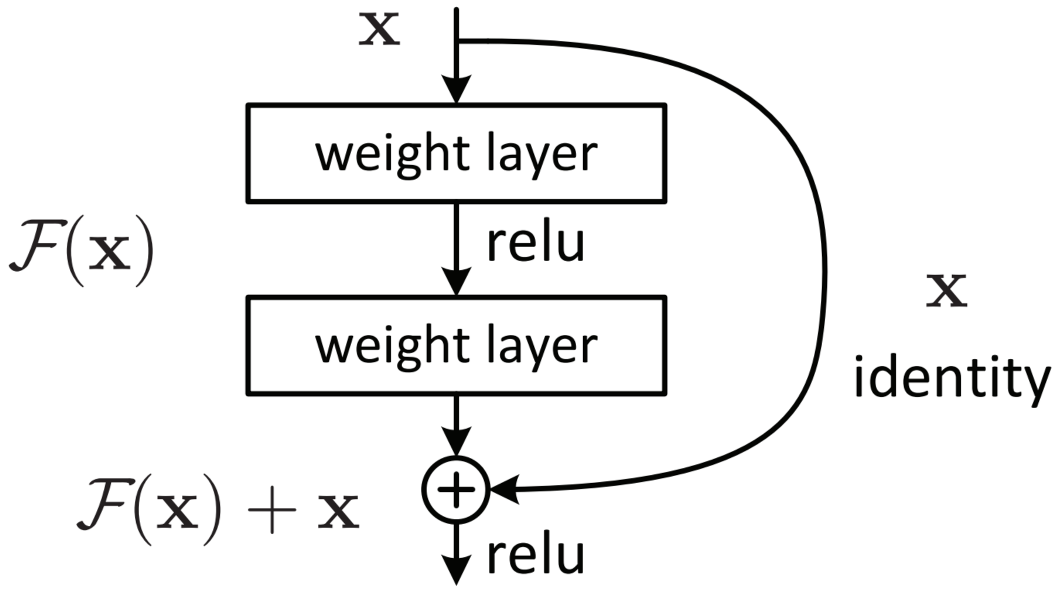

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Fig. 9. Resnet Minimum architecture:from [9]

The main idea of the ResNet is from Cortex visual of the Human. In fact they added the highway to the architecture. This highway (having the identity function on the input) will “amplify” the output change according to the input change. In the ResNet context of scaling up CNNs, many variants have been created. To scale up the ResNet architecture we have to add more layers. For instance From ResNet-18 to ResNet-200, we add more layers. The obtained network will be more accurate if we can train it. Now let’s explain our own resnet model. We use **ResNet-50**. As mentioned in his name it uses 50 layers. Now that our ResNet model has **23,858,500** parameters, **23,805,380** Trainable parameters and non-trainable parameters: 53,120. On the top of this ResNet-50 we add one layer of GlobalAveragePooling2D, another of 128 neurons with relu function for activation, another one of 64 neurons with relu activation and the last one with 4 neurons with softmax activation to get the output like a probability. The optimizer which we have used is the Adam optimizer. The results that we got will be presented in the result section of the document.

3.2 EfficientNet :

Once the gradient vanishing problem was solved with the ResNet architecture, we noticed that there is no formal way to scale up the network. In fact there is no formal way to choose which dimension (width, depth or Resolution) scales in which distribution in order to increase the accuracy and minimize the cost (computational cost, memory cost). Although we succeeded in training these models, we can see that the accuracy will be saturated after a certain level of scaling. As shown in the following figure 10.

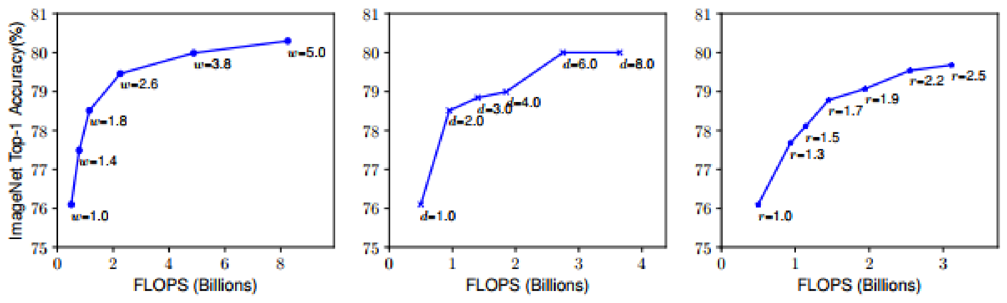


Figure 3. **Scaling Up a Baseline Model with Different Network Width (w), Depth (d), and Resolution (r) Coefficients.** Bigger networks with larger width, depth, or resolution tend to achieve higher accuracy, but the accuracy gain quickly saturate after reaching 80%, demonstrating the limitation of single dimension scaling. Baseline network is described in Table 1.

Fig. 10. Accuracy over scaling on one dimension:from [7]

Let’s notice that with w = 5.0 (scaling on one dimension width) they got the saturation of accuracy. With d=8.0 (scaling on one dimension width) we also get the saturation of accuracy and with r=2.5 we get the saturation of accuracy. But the researchers noticed that in the following schema with depth= 2.0 and r=1.3 the saturation event didn’t occurred.

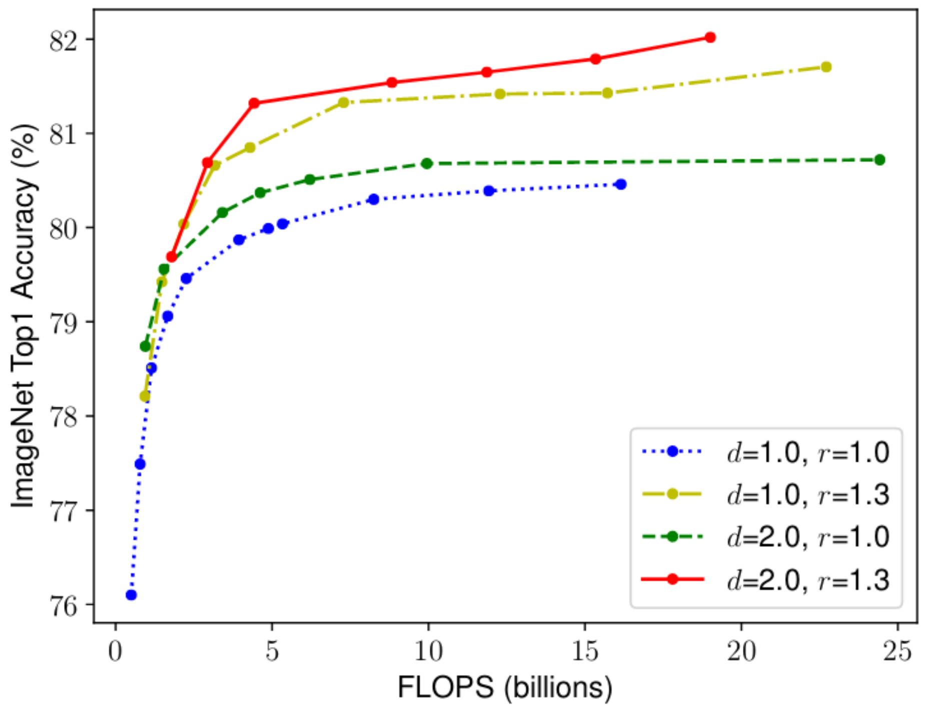


Fig. 11. Impact on accuracy over scaling on multiple dimensions:from [7]

Each dot is a model with different width. Where here, they conclude that the right solution can be scaling up the network on multiple dimensions (depth and resolution). The previous problems or situations caused the creation of the new approach or technique of scaling up the network on multiple dimensions. This technique (model) rethinks the way that we scale CNNs up. In addition, this new technique has posed certain constraints to be satisfied.

- maximize the model accuracy for any given resource constraints.
- To systematically study model scaling and balance the network depth, width and resolution.

In more formal way : with the model N, the objective function will be

$$\max_{d,w,r} \quad Accuracy(\mathcal{N}(d,w,r))$$

Fig. 12. Objective function :from [7]

With the following constraints:

$$\text{Memory}(\mathcal{N}) \leq \text{target\_memory}$$
$$\text{FLOPS}(\mathcal{N}) \leq \text{target\_flops}$$

Fig. 13. Constraints of equations: from [7]

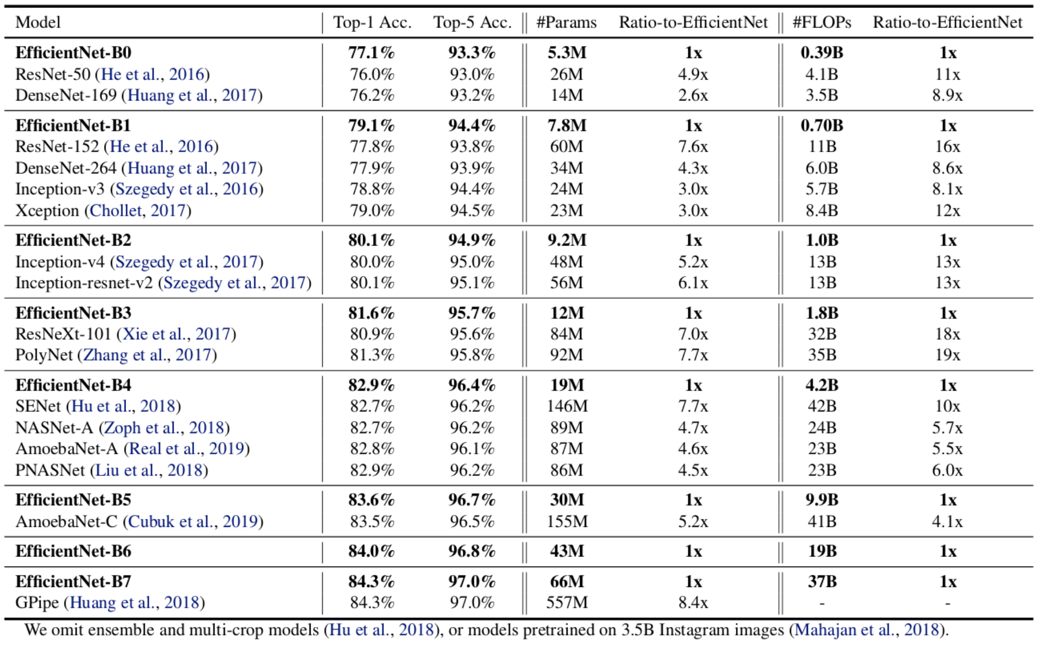
To solve this problem EfficientNet proposes the **Compound scaling method**. This scaling method tries to balance dimensions of width/depth/resolution by scaling with a constant ratio. Let's notice that this new approach uses an existing architecture and propose a way to efficiently scale up this one with a Compound scaling method. This new approach states that :

2^n more computational resources implies $a^n + b^n + c^2$ additional dimensions on the model.

The coefficient a, b and c are constants determined by a small GRID SEARCH on the original small model. Let's notice that a^n is the additional value to the depth dimension of the model , b^n is the additional value to the width dimension of the model and c^n is the additional value to the resolution dimension of the model. The following equation allows to get the different coefficients :

$$a + b^2 + c^2 \approx 2 \tag{6}$$

from the original paper that is formulated as following : Thanks to this new way, the researchers realized the best accuracy with less parameters (less resources). As we can show on the following figure 14 from original paper.



Model	Top-1 Acc.	Top-5 Acc.	#Params	Ratio-to-EfficientNet	#FLOPs	Ratio-to-EfficientNet
EfficientNet-B0	**77.1%**	**93.3%**	**5.3M**	**1x**	**0.39B**	**1x**
ResNet-50 (He et al., 2016)	76.0%	93.0%	26M	4.9x	4.1B	11x
DenseNet-169 (Huang et al., 2017)	76.2%	93.2%	14M	2.6x	3.5B	8.9x
EfficientNet-B1	**79.1%**	**94.4%**	**7.8M**	**1x**	**0.70B**	**1x**
ResNet-152 (He et al., 2016)	77.8%	93.8%	60M	7.6x	11B	16x
DenseNet-264 (Huang et al., 2017)	77.9%	93.9%	34M	4.3x	6.0B	8.6x
Inception-v3 (Szegedy et al., 2016)	78.8%	94.4%	24M	3.0x	5.7B	8.1x
Xception (Chollet, 2017)	79.0%	94.5%	23M	3.0x	8.4B	12x
EfficientNet-B2	**80.1%**	**94.9%**	**9.2M**	**1x**	**1.0B**	**1x**
Inception-v4 (Szegedy et al., 2017)	80.0%	95.0%	48M	5.2x	13B	13x
Inception-resnet-v2 (Szegedy et al., 2017)	80.1%	95.1%	56M	6.1x	13B	13x
EfficientNet-B3	**81.6%**	**95.7%**	**12M**	**1x**	**1.8B**	**1x**
ResNeXt-101 (Xie et al., 2017)	80.9%	95.6%	84M	7.0x	32B	18x
PolyNet (Zhang et al., 2017)	81.3%	95.8%	92M	7.7x	35B	19x
EfficientNet-B4	**82.9%**	**96.4%**	**19M**	**1x**	**4.2B**	**1x**
SENet (Hu et al., 2018)	82.7%	96.2%	146M	7.7x	42B	10x
NASNet-A (Zoph et al., 2018)	82.7%	96.2%	89M	4.7x	24B	5.7x
AmoebaNet-A (Real et al., 2019)	82.8%	96.1%	87M	4.6x	23B	5.5x
PNASNet (Liu et al., 2018)	82.9%	96.2%	86M	4.5x	23B	6.0x
EfficientNet-B5	**83.6%**	**96.7%**	**30M**	**1x**	**9.9B**	**1x**
AmoebaNet-C (Cubuk et al., 2019)	83.5%	96.5%	155M	5.2x	41B	4.1x
EfficientNet-B6	**84.0%**	**96.8%**	**43M**	**1x**	**19B**	**1x**
EfficientNet-B7	**84.3%**	**97.0%**	**66M**	**1x**	**37B**	**1x**
GPipe (Huang et al., 2018)	84.3%	97.0%	557M	8.4x	-	-

We omit ensemble and multi-crop models (Hu et al., 2018), or models pretrained on 3.5B Instagram images (Mahajan et al., 2018).

Fig. 14. Gain of performance on different models: from [7]

We can show on the first row, with EfficientNet-B0 there are 5.3M parameters and they got as accuracy 77.1% and with ResNet-50 there are 26M and they got as accuracy 76.0 %. The gain of the performance here is awesome.

We have used EfficientNetB7 adding GlobalAveragePooling layer, 3 Dense layer as following :

- dense3 (Dense): with the following shape (None, 128) and parameters 327808
- dense4 (Dense): with the following shape (None, 64) and parameters 8256
- dense5 (Dense): with the following shape (None, 4) and parameters 260

The explaining of the adding layer is about under-fitting avoiding strategic. In fact, according to the high dimension of the data, we add these layer to allow to the model to get more knowledge from data. And the last layer is about the the numbers of output class for the classification problem.

We coupled this model with Self-Training Noisy student [10]. This technique allows us to improve the accuracy of the model.

4 Simulation and discussion

4.1 Result

In this section we will present the result of our study. We start with accuracy and Epochs plot in order to show how the accuracy will increase over the time.

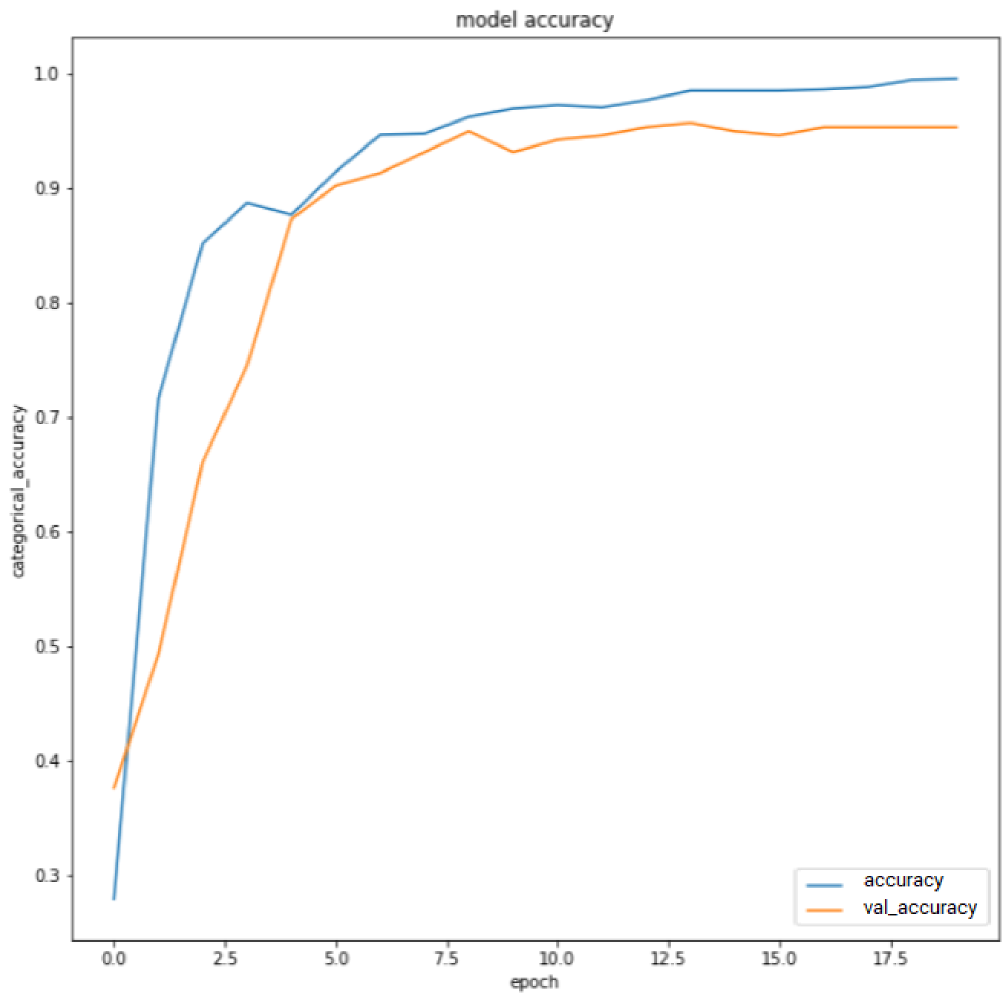


Fig. 15. Dense Accuracy Plot over epochs

This plot 15 shows the convergence of the average. We got 0.97 as accuracy on the training set and the 0.93 on the test set. With this last one we can see easily that the model works well on the training than the training set. Obviously this is normal. Our DenseNet is a standard CNNs.

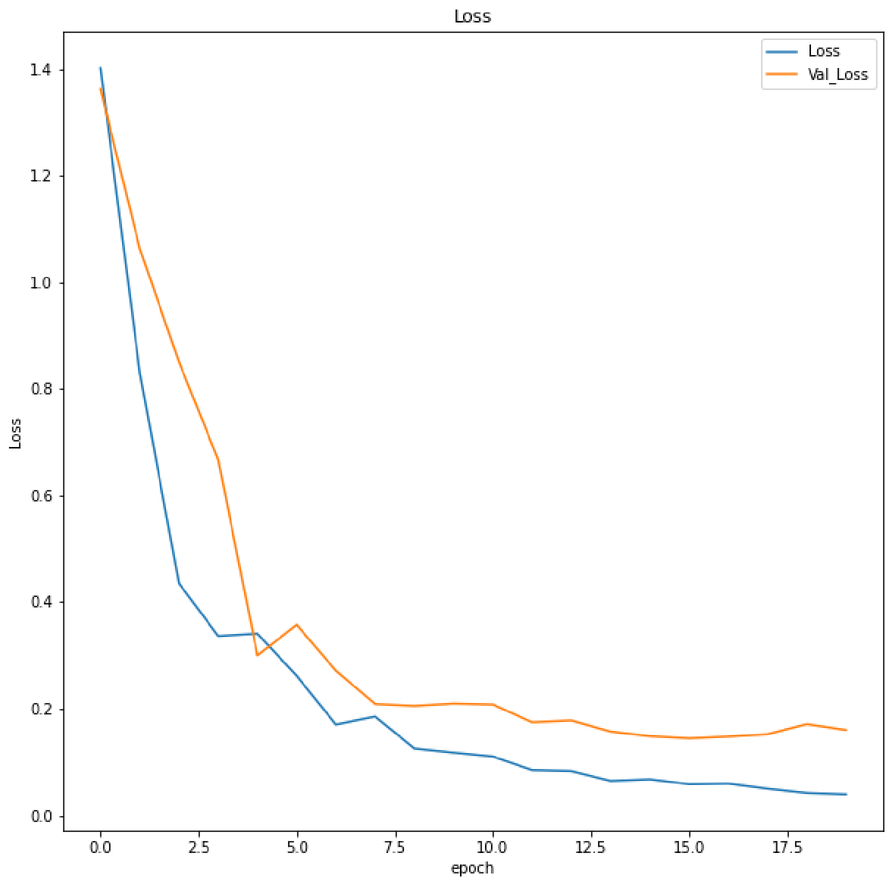


Fig. 16. Dense Loss Plot over epochs

This plot 16 shows the evolution of the loss value over the time. Obviously the loss converges to 0 over contrary the accuracy.

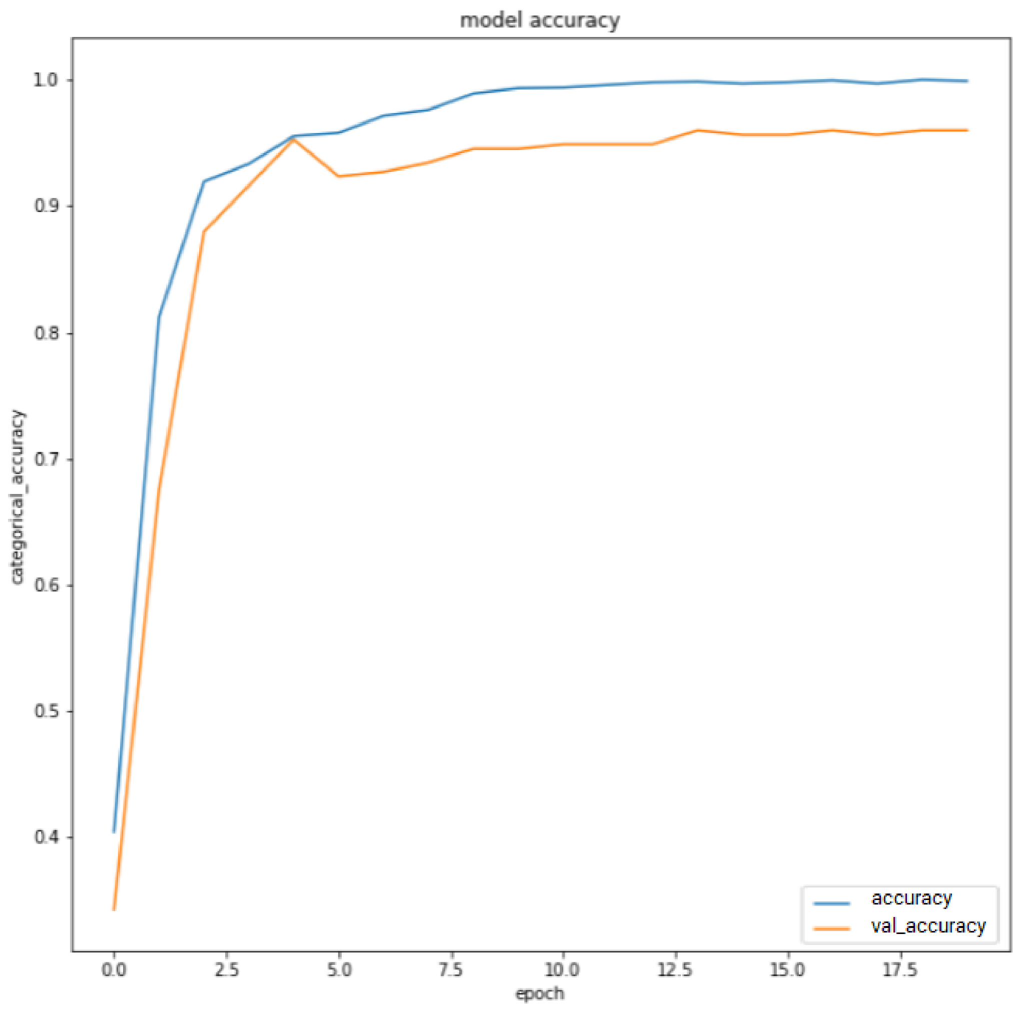


Fig. 17. Dense Accurate with filter Plot over epochs

In this Network we add more filter in image in order to decrease the quality of this ones and we train our dense with them. This manipulation is done in order to make the model more robust. The figure 17 shows the accuracy gotten.

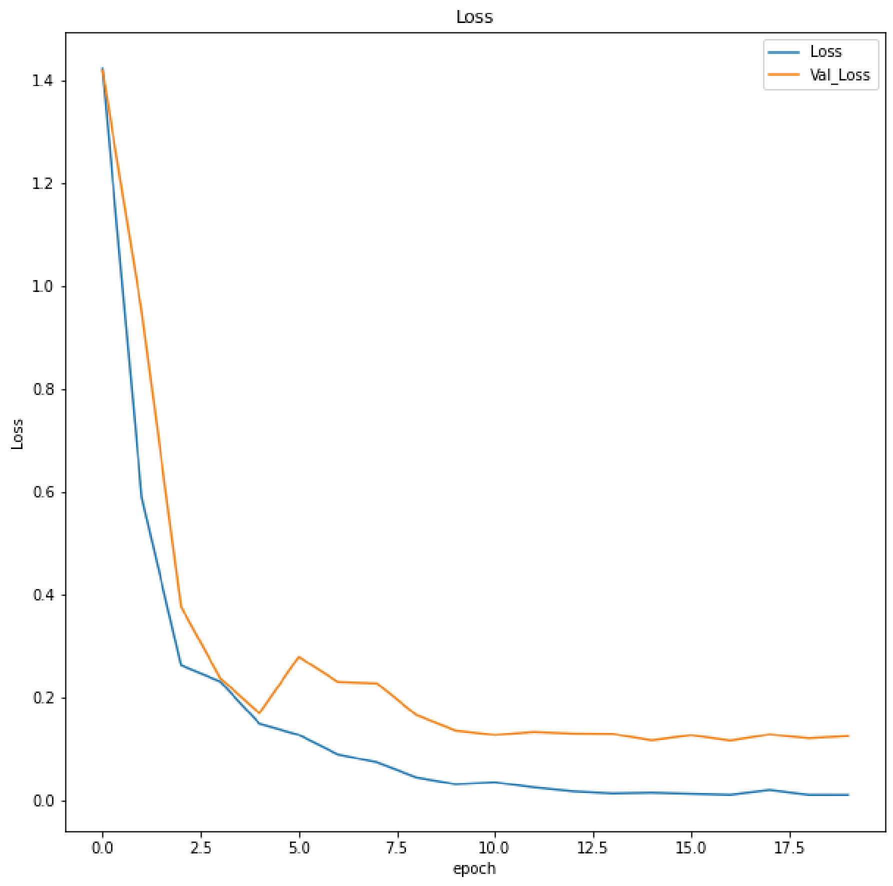


Fig. 18. Dense Loss with filter Plot over epochs

This is the plot 18 distribution over the epochs with the filter applied in the DenseNEt

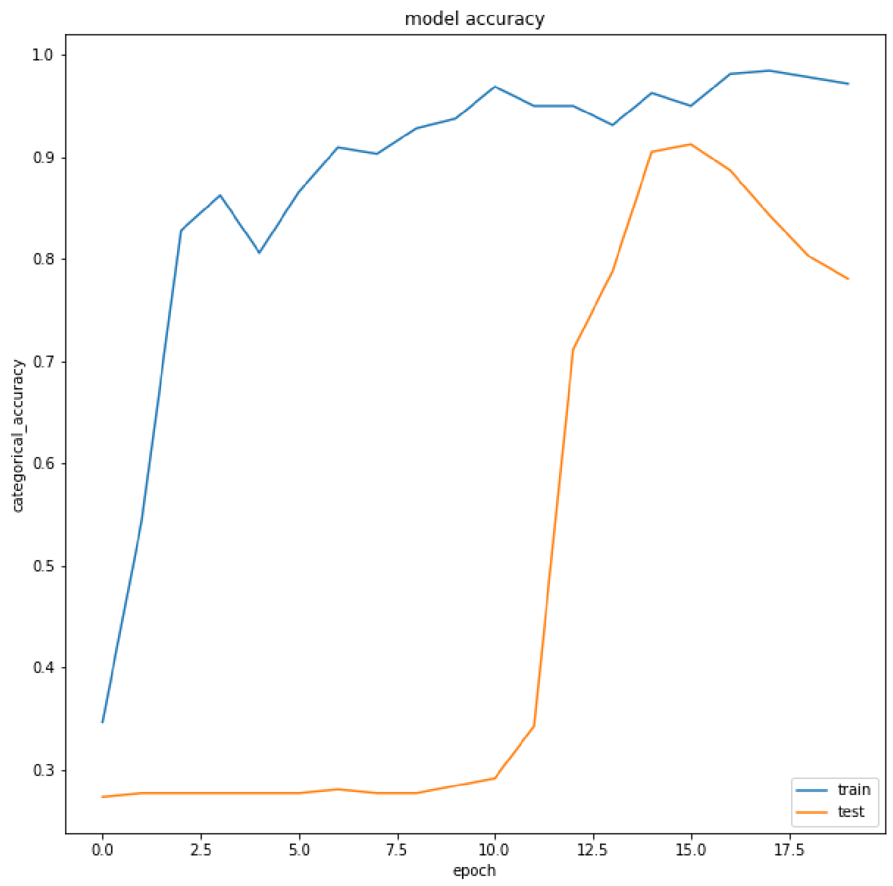


Fig. 19. Resnet Accuracy with filter Plot over epochs

This plot 19 shows the distribution of the accuracy of the epoch with res-net. Obviously we can notice this network make a lot of time before to converge in a test set. This is due to the effect of the filter on the training set and also we didn't apply these filters on the test set.

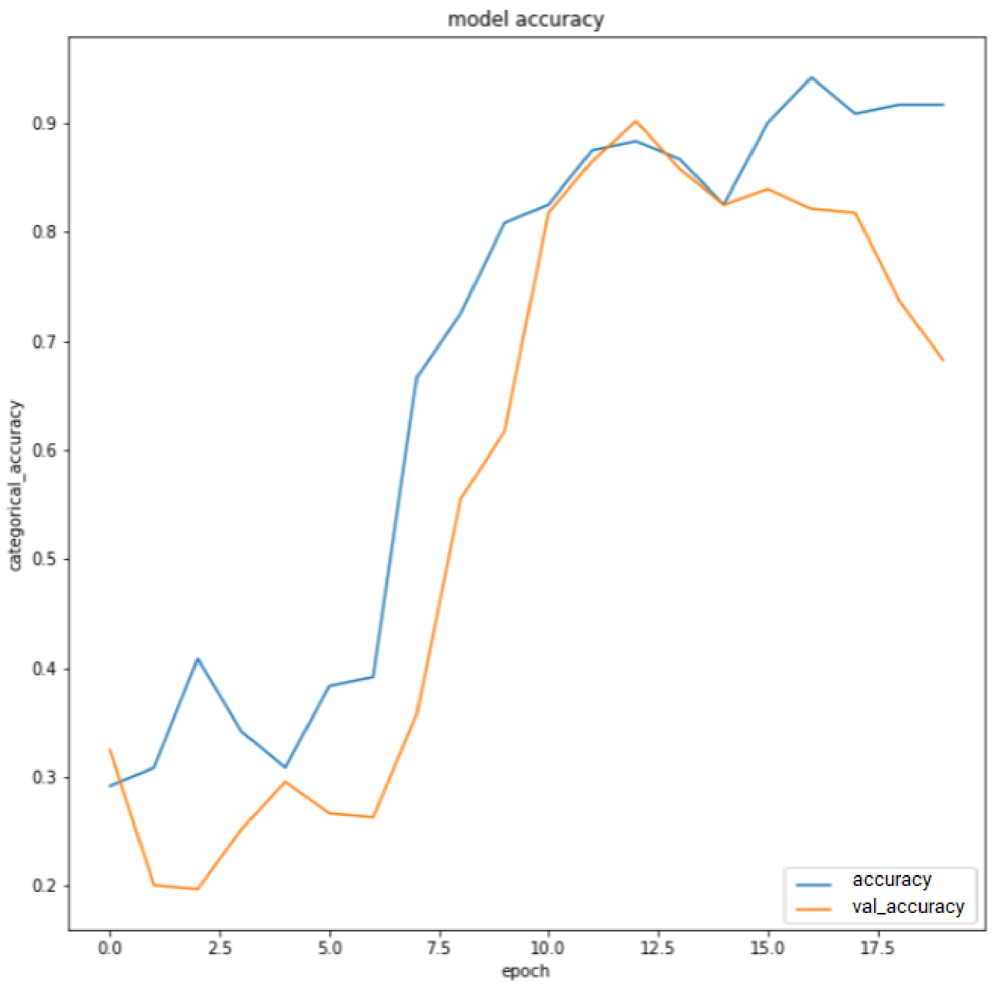


Fig. 20. EfficientNet Accuracy with filter Plot over epochs

This plot 20 shows the performance of the EfficientNet on this Dataset. As specified previously, we have used EfficienNet-B7. We got a couple of the issues due the size of this network. We fixed it, and we got as the accuracy 0.91 on the training set and 0.7 on the test set. We stop training because we got the stack-overflow on Server. But fro, kaggle and others resources; EfficientNet is the suitable network for this task.

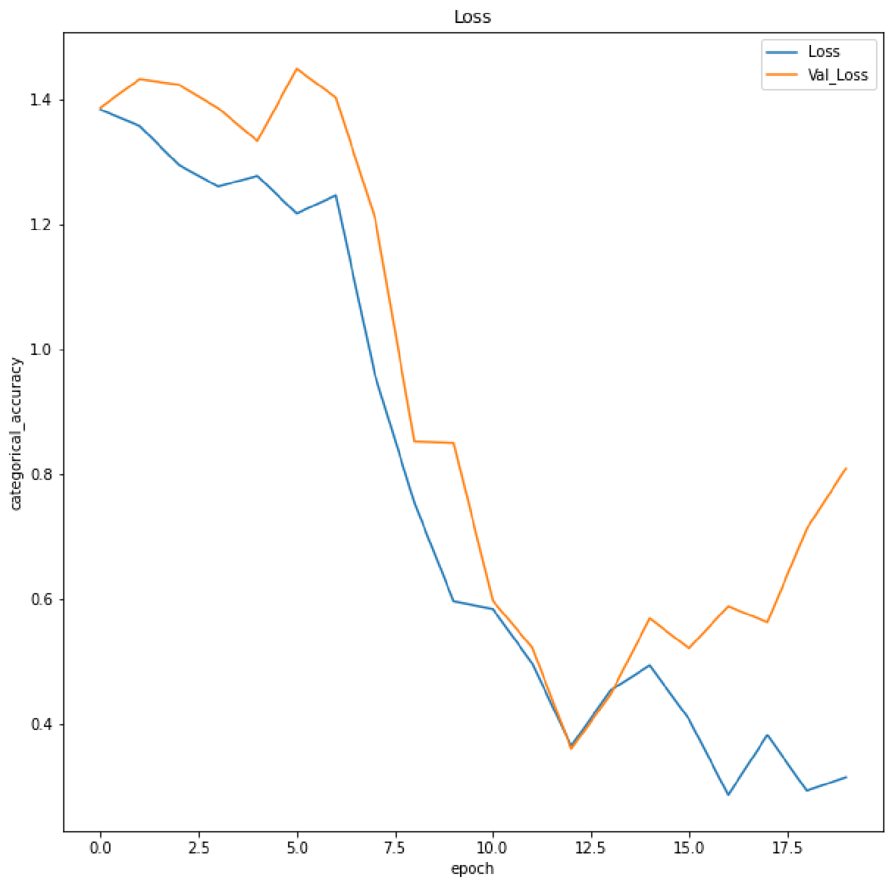


Fig. 21. EfficientNet Loss with filter Plot over epochs

This plot represents the loss distribution of the EfficientNet-B7 on our dataset

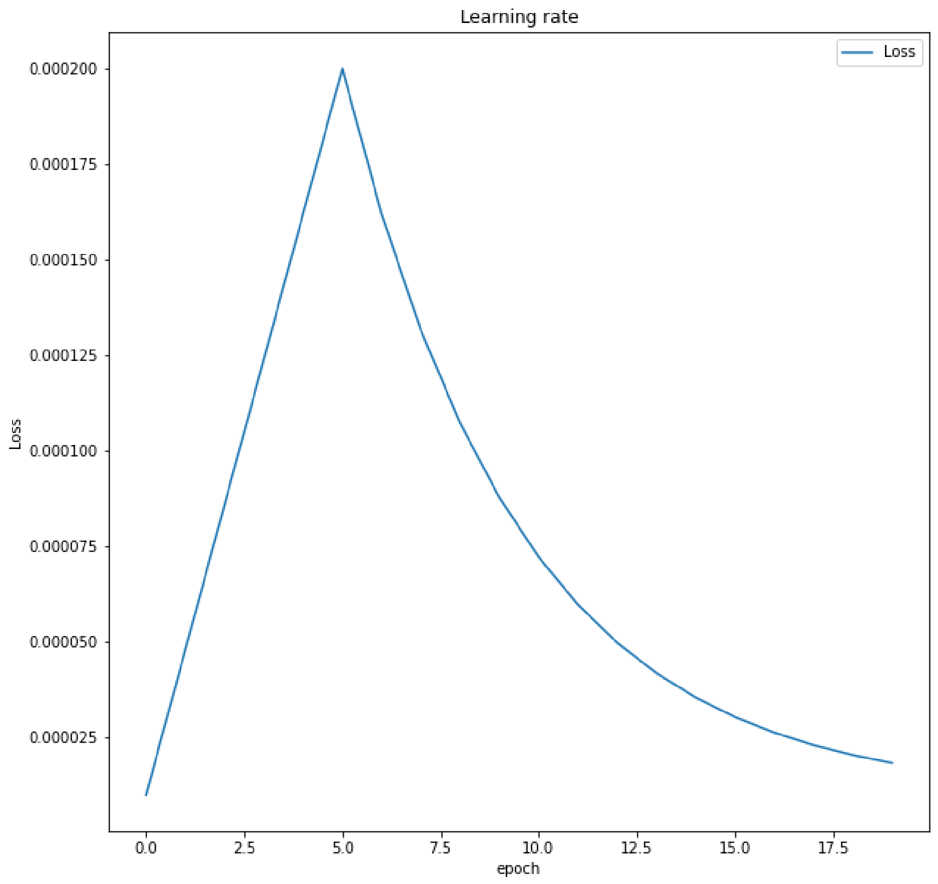


Fig. 22. Learning rate distribution over epochs

4.2 Conclusion and Perspectives

Form different papers on the Plant pathology detection with Deep Learning that we browsed, the most efficient model was the EfficientNet. But after long time on the subject with this dataset, we can say that DenseNet (simple CNN) is more suitable for this specify dataset than the others models. From This result, we can interpret that our dataset is not enough large, and our test server is not powerful enough to train the ResNet and EfficientNet to beat the DenseNet. Given the performance of our model on the training and test data, you can also be sure that we are not in an overfitting or underfitting situation.

In order to be able to run this project, you can see my code at this link : `https://github.com/AbouOpenSource/plantPathology/`

The perspectives of this article are numerous. Once the model have gotten a stable architecture, we have to think about some mistakes linked to the production of this model. In fact, we have to solve the problem that is characterized by the absence of one or more classes during the training phase, but they are present during the test or production phase. More precisely, this problem is characterized by the fact that once put in production our model can be used to predict on diseases that it has never seen. To this end, we must implement strategies to better manage these situations. This problem is known as zero-shot learning [11]. The next step once zero-shot learning handled will be the thinking of the architecture of software. The next step would be to think about a software architecture in order to inter-operate with client applications of any kind

References

[1] Guillian Z. Cornell Initiative for Digital Agriculture (CIDA). none. 2020. Available from: https://www.kaggle.com/c/plant-pathology-2020-fgvc7.

[2] Ghojogh B, Crowley M. The Theory Behind Overfitting, Cross Validation, Regularization, Bagging, and Boosting: Tutorial. arXiv. 2019. Available from: https://arxiv.org/pdf/1905.12787.pdf.

[3] Gaussian Filtering. none. 2012. Available from: https://www.cs.auckland.ac.nz/courses/compsci373s1c/PatricesLectures.

[4] Edge Detection with Gradients. none. 2020. Available from: https://www.youtube.com/watch?v=j7r3C-otk-U.

[5] Foundation O. OpenCV Canny. OpenCV Foundation. 2021. Available from: https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html.

[6] Das P. THE 5 MOST AMAZING COMPUTER VISION TECHNIQUES TO LEARN. analyticsinsight. 2020. Available from: https://www.analyticsinsight.net/the-5-most-amazing-computer-vision-techniques-to-learn/.

[7] Tan M, Le QV. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. arXiv. 2019. Available from: http://arxiv.org/abs/1905.11946.

[8] Wang CF. The Vanishing Gradient Problem. medium. 2019. Available from: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.

[9] He K, Zhang X, Ren S, Sun J. Deep Residual Learning for Image Recognition. arXiv. 2015. Available from: https://arxiv.org/pdf/1512.03385.pdf.

[10] Xie Q, Luong MT, Hovy E, V Q. Self-training with Noisy Student improves ImageNet classification. arXiv. 2020. Available from: https://arxiv.org/pdf/1911.04252.pdf.

[11] Cacheux YL, Borgne HL, Crucianu M. Zero-shot Learning with Deep Neural Networks for Object Recognition. arXiv. 2021. Available from: https://arxiv.org/pdf/2102.03137.pdf.