Jérémy ROS
Kavinkumar THIRUNAVUKKARASU

POVa : Facial recognition from images

Introduction :

Facial recognition from images is the task of using computer algorithms to identify and verify individuals from digital images or videos. This can be used for a variety of applications including security, social media and entertainment.

One of the most common ways to do it is to use a neural network, such as a CNN (Convolutional Neural Network). CNNs are well suited for this task as they are able to learn hierarchical representations of input data, which makes them efficient in extracting relevant features from images.

To perform facial recognition, a neural network is trained on a large dataset of images of faces, along with labels indicating the identity of the person in each image. During the training, the network learns to recognize the unique facial patterns and characteristics of each person. Once the network is trained, it can be used to identify and verify individuals in new images.

Face recognition from images is a complex task and many factors can affect the performance of the model, such as the quality of the input images, the size and diversity of the training dataset, and the complexity of the model.

Experiment :

We did 2 experiments for that. The first one is to have 2 classes but multiple models by having more or less layers and changing the filter size. The second one is to have 100 classes doing the same thing.

I - Construction of dataset :

For the first experiment, we designed our own dataset. To do this, we downloaded images from the internet belonging to 2 people, Dwayne Johnson and Chris Hemsworth. We only took the images where we could clearly see the different features of their faces (nose, eyes, mouth).

Then we cropped the faces using a pre-trained model using the haar cascade method.

Also we have changed the color images to black and white to reduce the number of input data.

Original image cropped image

We built a dataset of 300 images per class. Also to increase the size of our dataset we used the ImageDataGenerator function from the tensorflow.keras library. To increase the size of the images we have rotated, translated and zoomed the original images. In the end we built a dataset of 1000 images per class divided into 2 classes.

rotation zoom translation

For the second experiment, we used a dataset from the internet, named CelabA. We are just going to use 100 classes from this dataset where each class contains 30 images.

CelebA dataset images

II - The goal of the experiments :

The goal of the experiments is to test the different parameters of the CNN in order to see their influences and also to find the best model for the project.

The parameters to test are:

- The number of convolutional layers
- The number of filters for convolutional layer
- The kernel size for the convolutional layers
- The activation function
- The number of Dense Layers
- The number of neurons for Dense Layers
- The value of the Dropout layer
- The learning rate

III - Generic Model :

We import all images in black and white.
The input size of the model is (200, 200, 1).
We rescale the input by dividing by 1/255 to have the inputs between 0 and 1.

Each convolutional layer is composed of 3 parts :

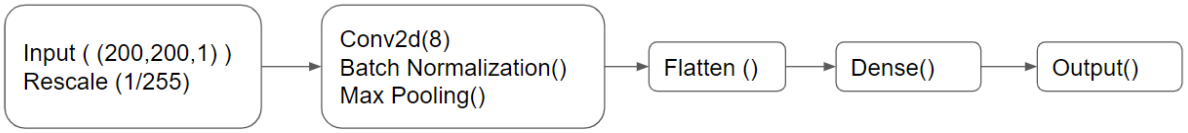
- A Conv2d layer which is the traditional convolutional layer
- A Batch Normalization Layer applies a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. This speeds up the training of the model.
- A MaxPooling2d which allows to keep only important information for training.

Then we have a flatten layer which allows us to go to 1 dimension.

Next we have dense layers and/or dropout layers.

And finally we have the output which is a Dense Layer whose number of neurons is equal to the number of classes and the activation function is softmax.

The epoch is fixed to 50 for every experiment.

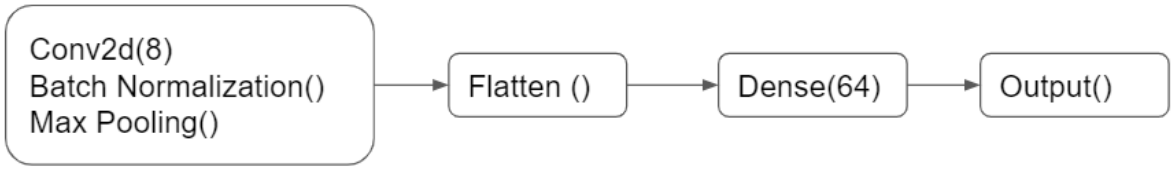


Schema of the generic model

IV - First Experiment :

We started the experiment with a dataset containing 2 classes and each class has 1000 images.

First we tested on a simple model:



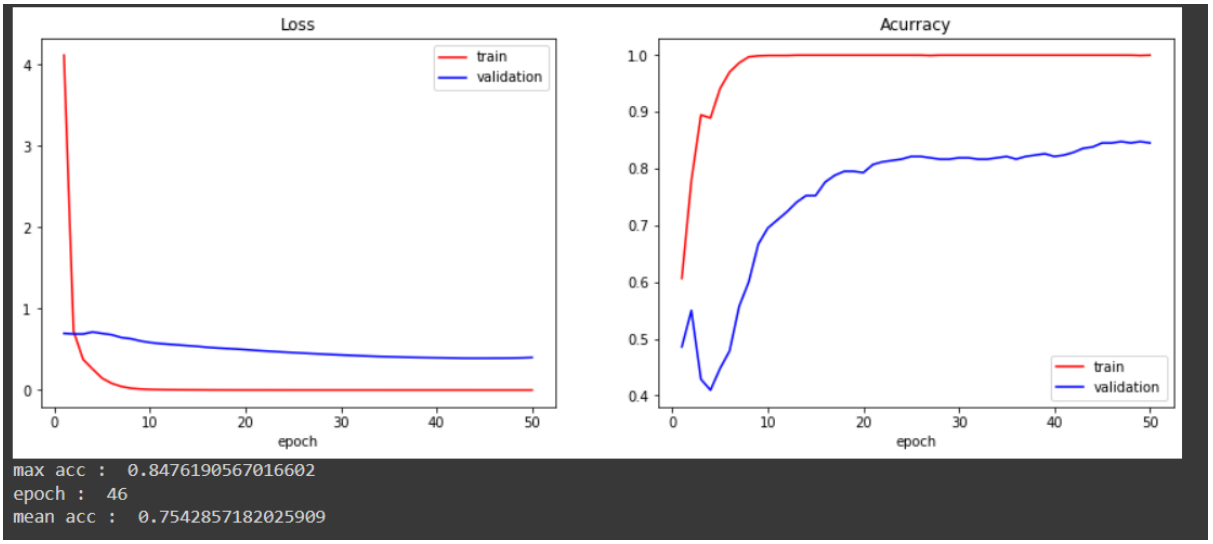
And we had an accuracy of 84%. Then we tested with filters 16, 32, 64 and the precision was around 84%, only the speed of convergence slowed down when we increased the number of filters.

When we added convolutional layers, the accuracy was unchanged.
Then when we add dense layers, the accuracy improves slightly but it is still around 80%.

After, we increased the number of classes to 5 with 1000 images per class and the result was the same.

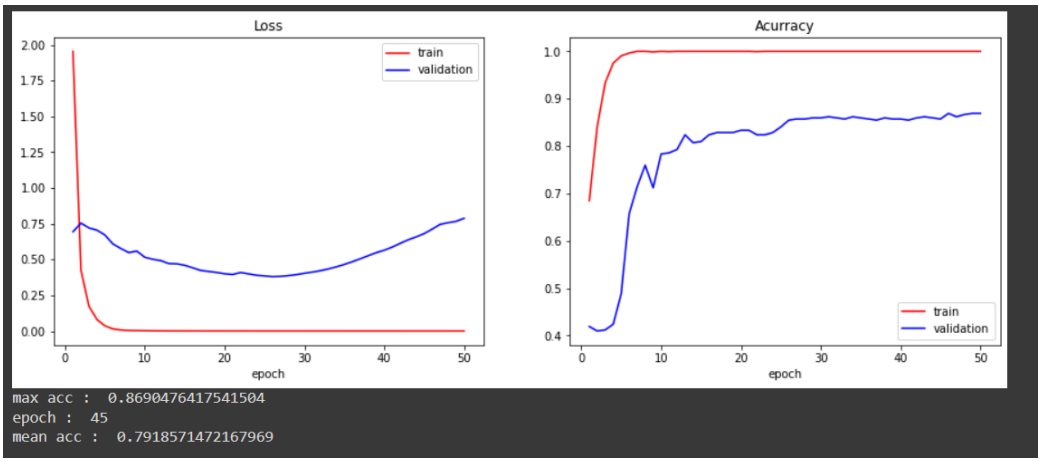
Next, we reduced the number of images per class to 100 per class and the precision dropped around 0.2 for different numbers of convolution layers, dense layers and number of filters.

So we concluded that when we have few classes, the most important parameter is the size of the dataset. The more images there are per class, the better is the precision.
And in our case, the number of filters and the number of convolutional layers had no influence.

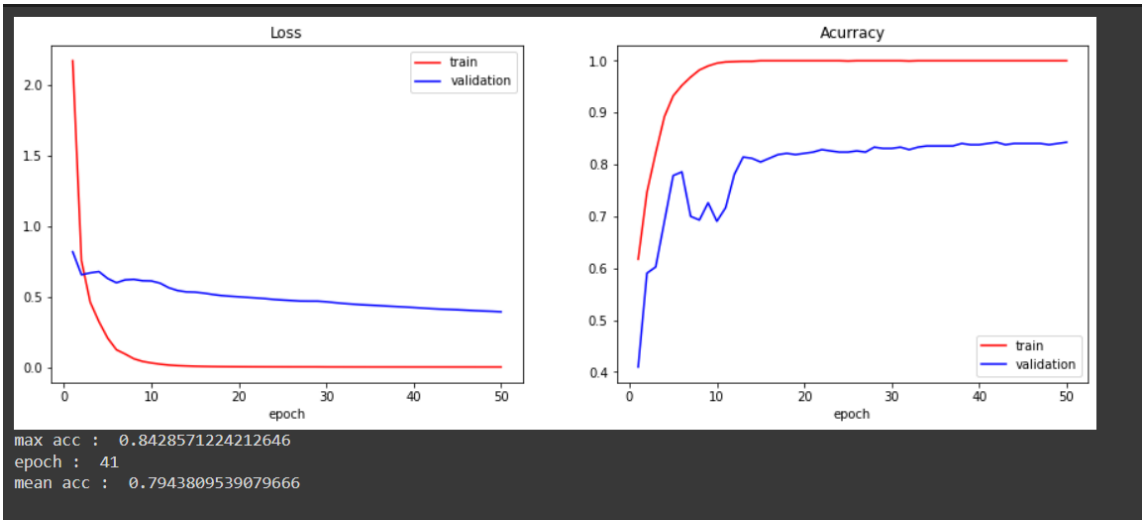


The accuracy and loss for the simplest model : Conv2d(8) - Flatten() - Dense(64)

There was no difference for the accuracy when we tested the kernel size for 5 and 3.

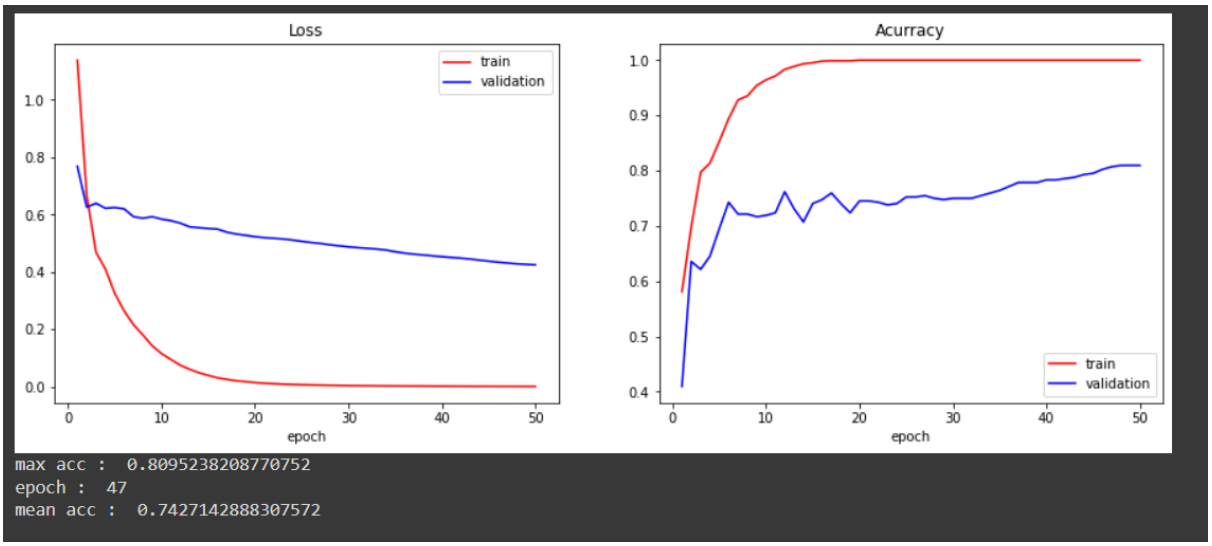


Kernel size = 5, Model : Conv2d(8) - Flatten() - Dense(64)

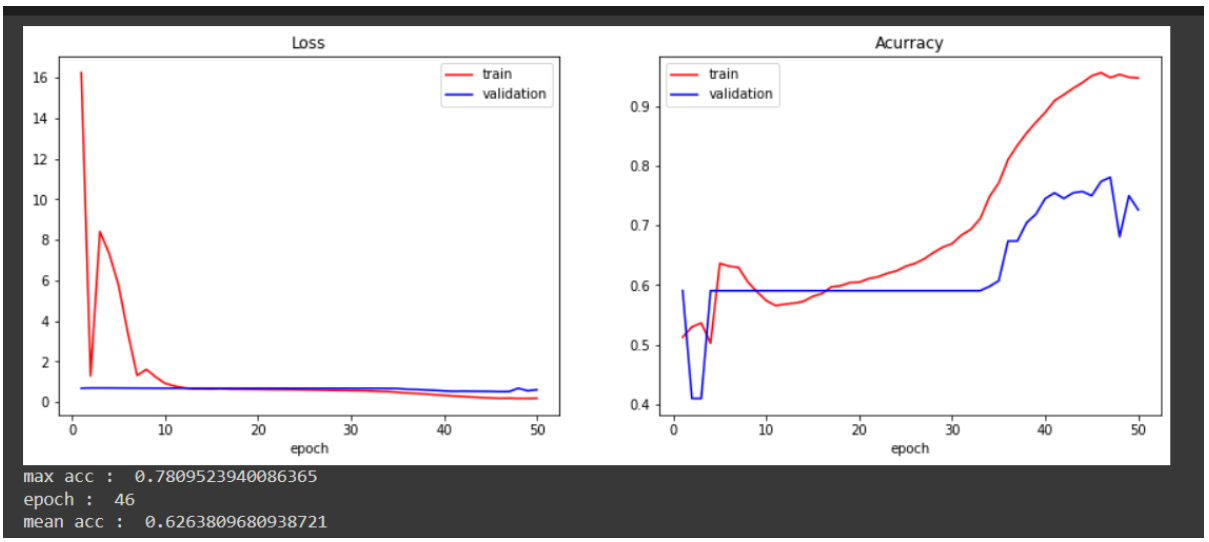


Kernel size = 3, Model : Conv2d(8) - Flatten() - Dense(64)

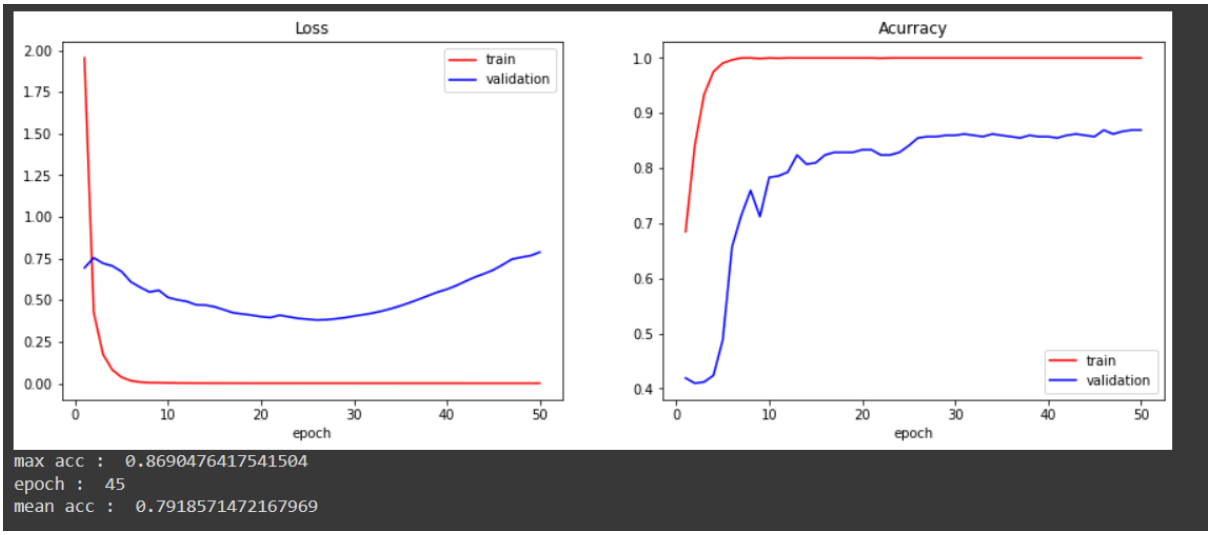
We have also seen that when we have a low learning rate (1e-4), the learning is slow to reach the maximum accuracy. When the learning rate is high 1e-2,the learning is fast but inaccurate. The correct value is 1e-3.



Learning rate = 1e-4, Model : Conv2d(8) - Flatten() - Dense(64)



Learning rate = 1e-2, Model : Conv2d(8) - Flatten() - Dense(64)



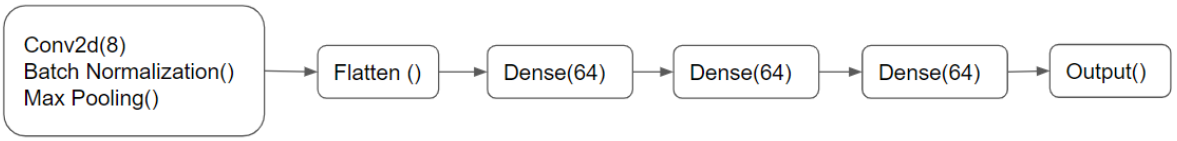
Learning rate = 1e-3, Model : Conv2d(8) - Flatten() - Dense(64)

We also tested the activation functions, the best was "relu".

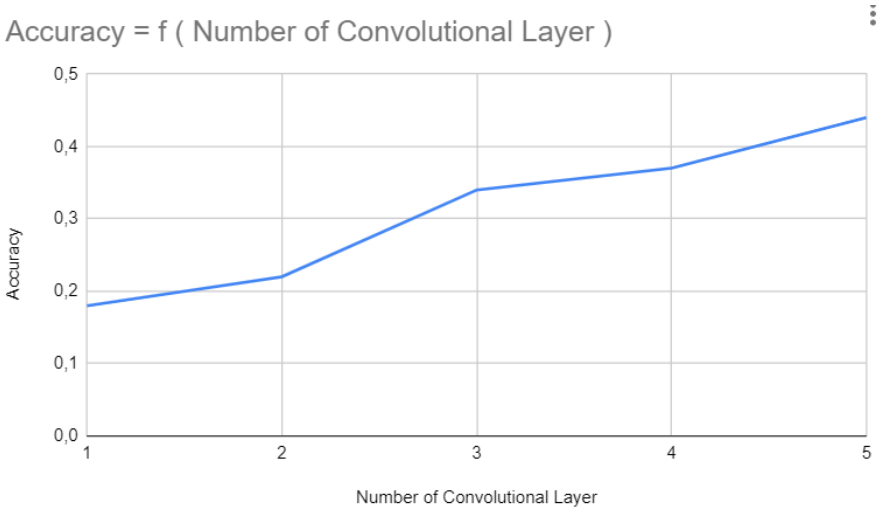
For the next experiments we will keep kernel size = 5, learning rate = 1e-3 and activation function = “relu”.

V - Second Experiment :

For the second experiment, we used 100 classes with 30 images per class.
Then we took a random model :



Then we calculated the accuracy by adding additional convolutional layers.

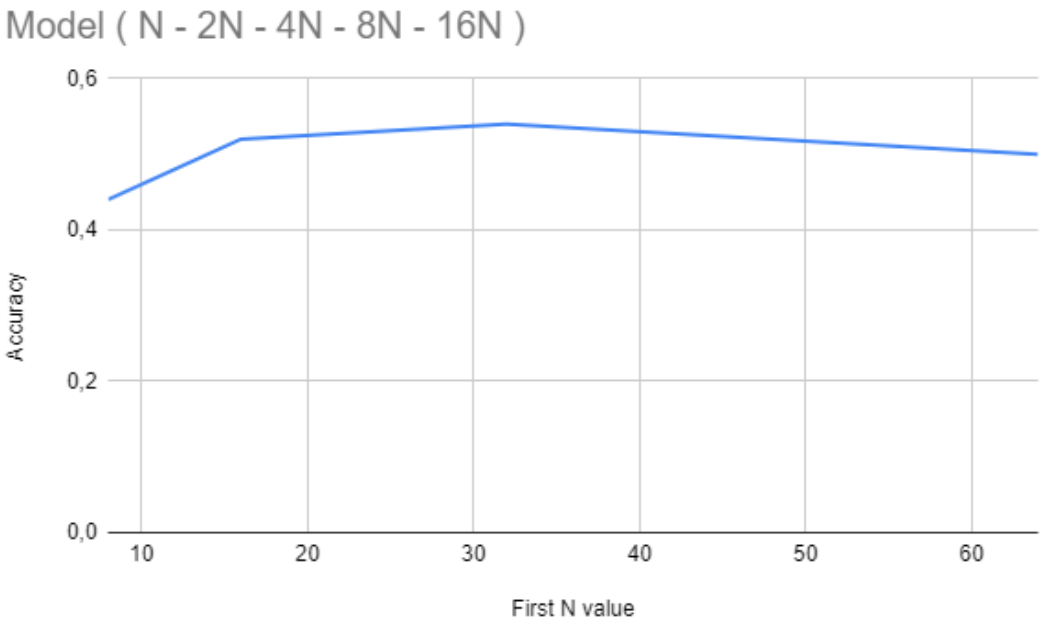


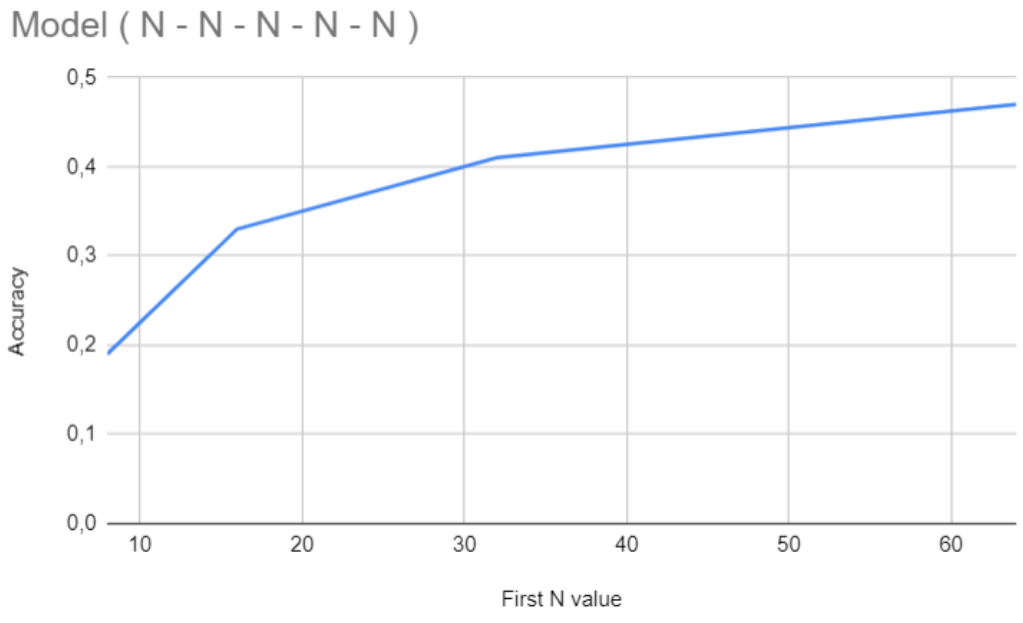
We can see that in our case, the more convolutional layers we have,the better the precision. For the following we will take 5 convolutional layers

We have tested different values of neurons for the 5-layer convolutional model.
We tested the following types of models:

- N - N - N - N - N - N
- N - 2N - 4N - 8N -16N
- 16N - 8N - 4N - 2N - N
- N - 2N - 4N - 2N - N
- 4N - 2N - N - 2N - 4N

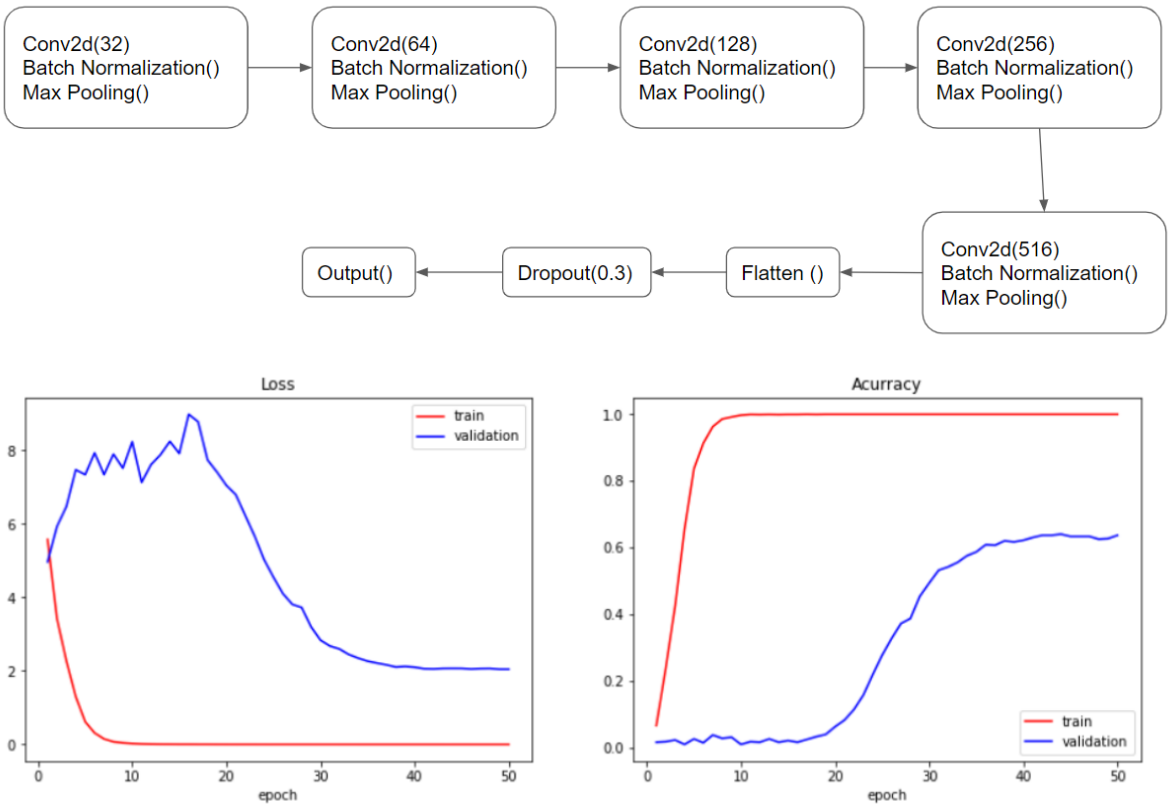
Where each N is a convolutional layer and N represents the number of filter.
Finally we saw that models with type N - 2N - 4N - 8N - 16N had better accuracy. And we took the values: 32 - 64 -128 - 256 - 516 with 0.54 accuracy.





Then we tested the influence of Dense layers, and regardless of the number of neurons and the number of layers, there was no significant improvement in accuracy .Similarly, the combination of dropout layers and dense layers does not improve accuracy.

We tested a last model, without a dense layer and with a dropout layer of value 0.3 between the flatten and output layer. And we had a precision of 0.64.



Accuracy of the last model

VI - Conclusion :

According to our experiences when we have few classes and many datasets, only the size of the dataset has a major influence on the precision.

And when we have a lot of classes and few datasets, the convolutional layers play an important role in the influence of accuracy. One of the best models for the convolutional layer is to have layers with increasing filters (like N - 2N - 4N - 8N 16N).

When we have a saturated model, the dropout layers can be useful to debug the situation.

Unfortunately, we could not see the influence of dense layers.

VII - Advantage and disadvantage of CNN :

- Disadvantage :
    - CNNs require a large amount of labeled data for training.
    - CNNs can be computationally expensive, particularly when training very large models.
    - CNNs are prone to overfitting, particularly when the model is too large or the training data is too small.
    - CNNs are sensitive to the initial values of their weights and biases, and can get stuck in local optima during training if the initialization is not careful. This can make it difficult to train CNNs effectively, and can require significant fine-tuning and experimentation.
    - CNNs can take a long time to train and find the optimal parameters.

- Advantage :
    - CNNs are effective at automatically learning features from images.
    - CNNs can handle large amounts of data and are able to learn from large datasets.
    - CNNs are able to learn and recognize patterns and features in images that are translation invariant.
    - CNNs trained models are relatively fast and efficient, making them suitable for real-time applications.

Comparaison :

ResNet (Residual Network) vs CNN (Convolutional Neural Network) :

Architecture: ResNets are a type of CNNs that are designed to be very deep, with hundreds or even thousands of layers. They introduce residual connections, which allow the model to learn residual functions that are added to the input of the layer. This alleviates the problem of vanishing gradients, which occurs when the model becomes too deep and the parameter gradients become very small, making it difficult to train the model efficiently. CNNs, on the other hand, are composed of several layers of convolution and clustering operations, designed to learn the spatial hierarchies of features from the input data.

Applications: ResNets and CNNs are widely used in a variety of applications. ResNets have achieved state-of-the-art performance on a number of image recognition tasks and are widely used in industry and research. CNNs are commonly used in image classification and object detection tasks, and have also achieved peak performance on a number of benchmarks.

Overall, ResNets and CNNs are powerful tools in the field of deep learning, and the choice between the two will depend on the specific task and the requirements of the model.

VGGNet :

VGGNet is a convolutional neural network (CNN) architecture developed by the Visual Geometry Group (VGG) at the University of Oxford. It was first introduced in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition" by K. Simonyan and A. Zisserman in 2014. It is known for its simplicity and efficiency, and has achieved peak performance on a number of image classification marks. It is composed of several convolutional and fully connected layers, and uses 3x3 convolution filters to learn deep representations of the input data.

It uses up to 19 weight layers which allows the network to learn complex functionality from input data that will lead to good performance on a variety of tasks.

It has been used a lot in the field of computer vision and in a number of research papers and practical applications. VGGNet also inspired the development of other deep CNN architectures, such as ResNet and DenseNet. It is trained using a variant of stochastic gradient descent (SGD) called "batch normalization", which helps to stabilize the training process and improve generalization of the model. It is also trained using a variant of SGD called "momentum", which helps to speed up model convergence by incorporating a moving average of past gradients. For the training, a large dataset containing labeled images is needed. After the training it can learn a set of features from the input data. It can be used for image classification, object detection and segmentation for example.

DenseNet :

DenseNet is a convolutional neural network (CNN) developed by researchers at the Chinese University of Hong Kong in 2016. It is composed of a series of dense blocks connected by dense connections. It allows the network to propagate information efficiently through the network. In a dense block, each layer receives input from all the previous layers, instead of just the immediately previous layer. This will help the network to learn features at multiple scales and alleviate the problem of vanishing gradients, which can occur when the model becomes very deep. It has achieved state-of-the-art performance on image classification benchmarks and is widely used in the field of computer vision. It has also been applied to other tasks, such as object detection and semantic segmentation.

Just like VGGNet, it is trained using batch normalization to stabilize the training process and improve the generalization of the model. It is also trained using momentum, a variant of SGD, in order to accelerate the convergence of the model by adding a moving average of the past gradients. The training is the same as VGGNet.