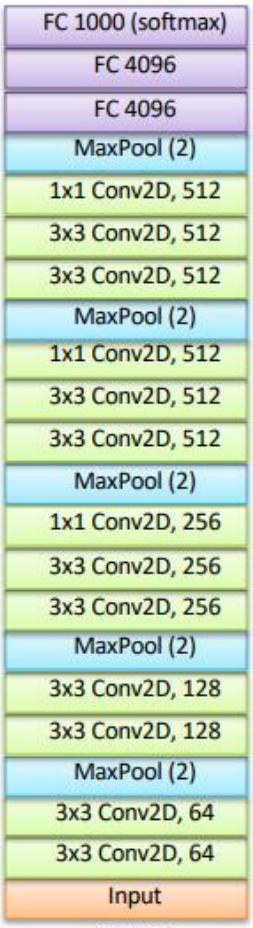
Surname:	Sata	Name:	Kamara

Question 1: Indicate the loss and accuracy values obtained during training and validation of VGG-16 CNN with CIFAR-10 for all datasets used. Indicate the learning rate used during training.

The structure of the VGG16 CNN is the following:

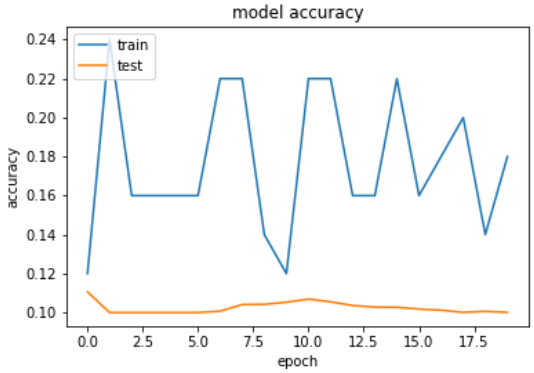


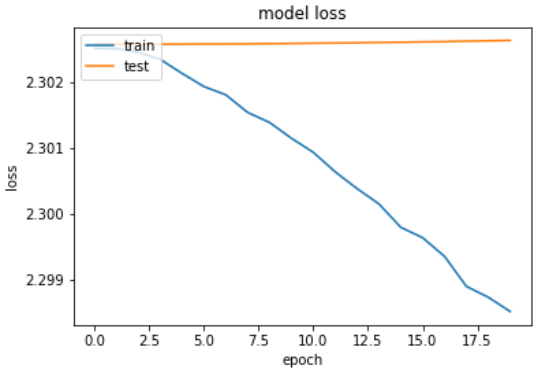
VGG-16

I worked with the full dataset of images. I implemented this neural network for different sizes of training set of images. This architecture has 28.919.626 parameters, which are all trainable. The hyperparameters of the training are the number of images that are trained, the number of epochs which is the times that the training set is passed forward and backward and the learning rate.

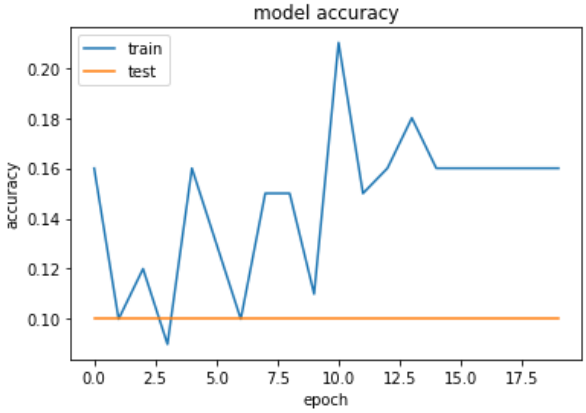
First, I set the number of epochs to 20, the learning rate to 0.001 and the optimizer is the Stochastic Gradient Descent (SGD). I modified the number of images of the training set. For each training set, I computed the accuracy and the loss of the network and I plot the two metrics as functions of the epochs.

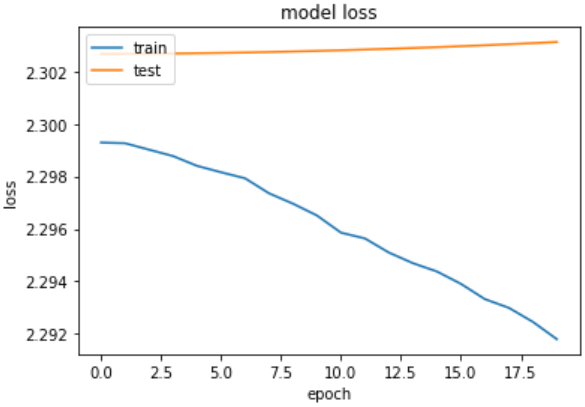
For 50 images, the accuracy is 10.00% and the loss is 2.3027 which is very bad. As the plots show, both metrics are quasi constant with the epochs, which means that the model overfits:



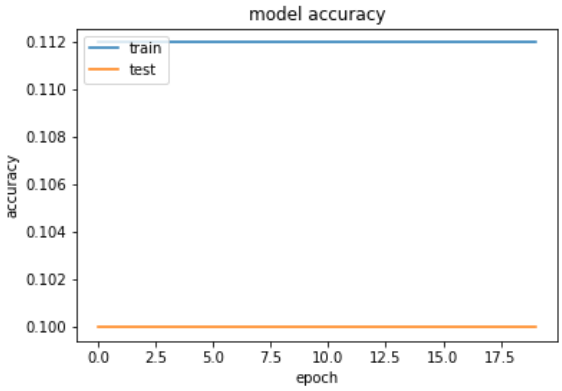


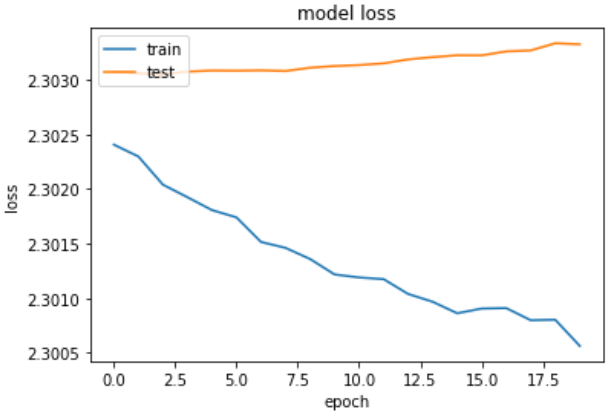
For 100 images, the accuracy is also very low and the loss very high: 10.00% for the accuracy and 2.3031 for the loss, the model still overfits:



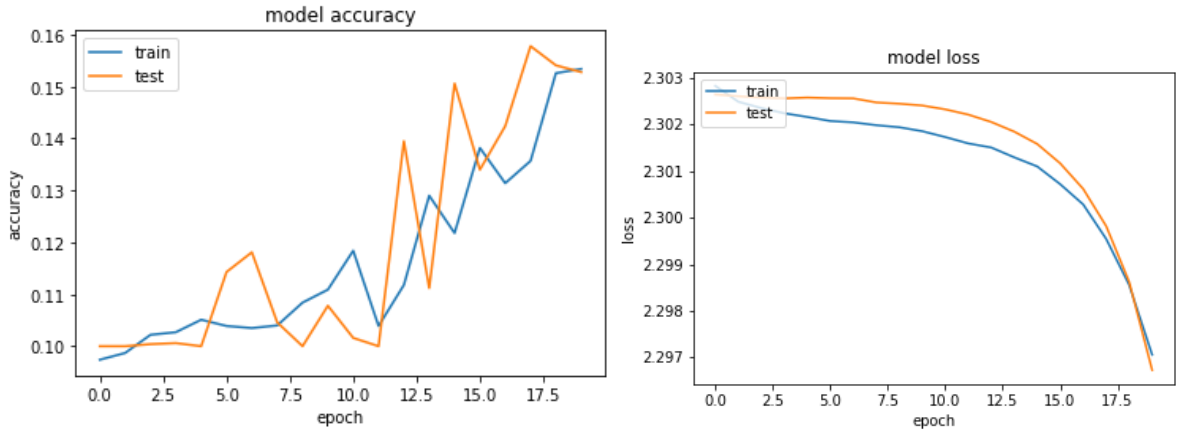


For 1000 images, the results are not better than the two previous ones: the accuracy is 10.00% and the loss is 2.3033. The accuracy for the test set seems to be stable, so it does not depend on the epoch. Moreover, as it is still below the accuracy of the training set, the model overfits as the previous ones.

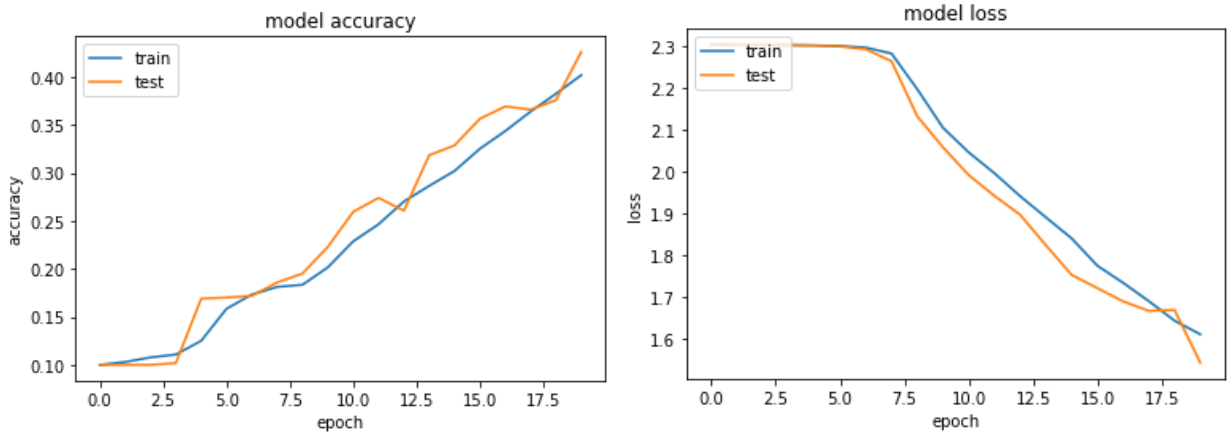




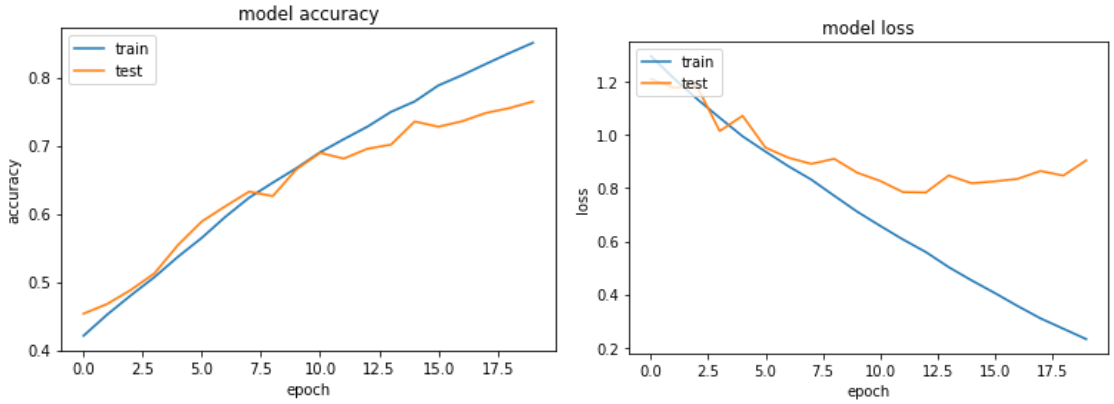
For 10000 images, the accuracy is 15.28% and the loss is 2.2984: we can notice better results than previously but still not good enough to validate the model. Nevertheless, the accuracy is increasing and the model is not overfitting anymore:



For 25000 images, the accuracy is 52.04% and the loss is 1.3032: the classification is becoming better. As the graphs show it, the accuracy of the test set is increasing above the one of the training set and the loss of the test set is constantly decreasing below the one of training set:



For 50000 images, the accuracy is 76.45%, which is almost twice the accuracy than for a half number of images and the loss is 0.9050, which is much better than for the previous ones:



We can see that taking more than 40% of the images for the training set brings about an accuracy that is more than 50%: less than half of the images are misclassified. Under this number of images in the training set, the model is overfitting. Concerning the prediction error, which is represented by the loss, it is always decreasing with the epoch: increasing the number of epochs would improve the loss.

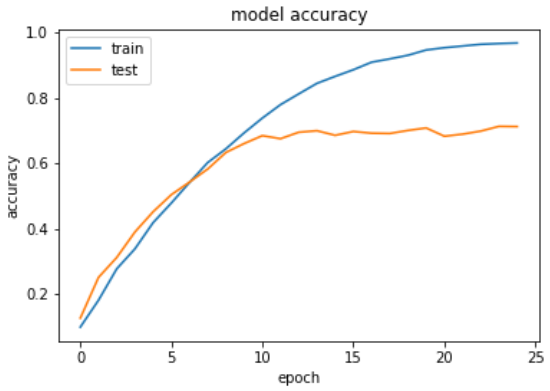
To improve the results, there are different solutions:

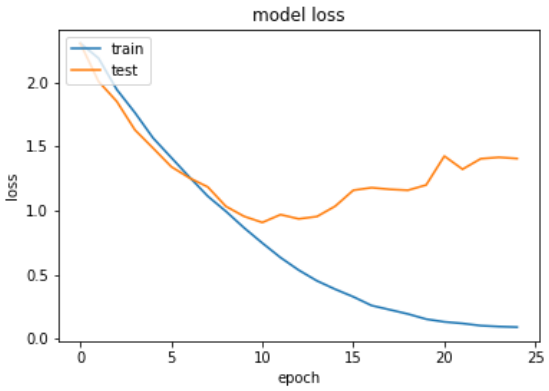
- Method 1: we can increase the number of images of the training set: the more the network has inputs, the better is the computation of the weights and consequently the output. We have already seen that the accuracy of the prediction is better for a certain percentage of image taken as training set.
- Method 2: we can increase the number of epochs of the executions. In fact, we can see that the accuracy increases quasi linearly with the number of epochs and the loss decreases.
- Method 3: we can improve the learning rate of the training set.
- Method 4: we can try other optimizers to obtain the smallest loss as possible
- Method 5: we can explore the different dropout rate to know which one is the best

Application of method 2

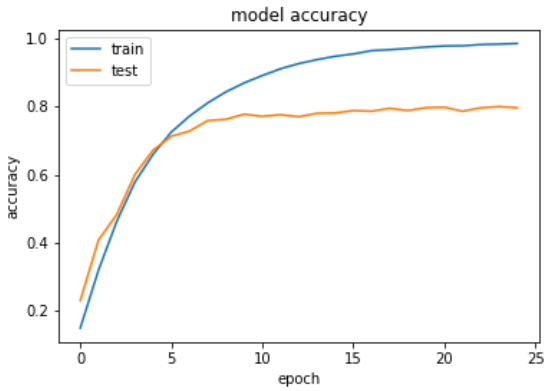
For the training sets of 25000 and 50000 images (which are the more accurate), I increase the number of epochs. We obtain the following accuracies and losses:

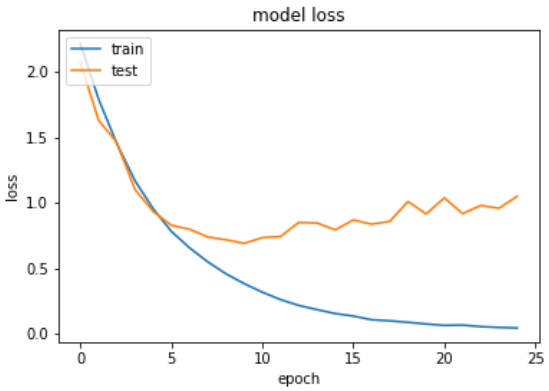
25000 images: accuracy = 71.28%, loss = 1.4052





50000 images: accuracy = 79.74%, loss = 1.0496





We can notice that increasing the number of epochs allows to obtain a better accuracy but does not reduce the loss, as expected. Moreover, we can notice that from 5 epochs, the model is overfitting because the training set has better values for the accuracy and the loss. As a result, modifying the number of epochs can allow to improve the accuracy and the loss but as the model overfits, it cannot be considered as performant because the variability is too high.

Application of method 3

Regarding the previous result, I decided to modify another hyperparameter: the learning rate. The learning rate is the size of the steps of the optimizer. For the same number of epochs, I tested a lower value and a higher value than the initial learning rate and I computed for each rate the accuracy and the loss. The decay is equal to 0 so the learning rate is kept as constant all the long. I obtained the following table:

	Accuracy (%)			Loss		
	0.0001	0.001	0.01	0.0001	0.001	0.01
50 images	10.00	10.00	81.93	2.3023	2.3027	1.3616
100 images	10.00	10.00	81.93	2.3023	2.3031	1.3615
1000 images	10.00	10.00	81.93	2.3022	2.3033	1.3612

10000 images	10.00	15.28	81.93	2.3022	2.2984	1.3526
25000 images	10.00	52.04	80.93	2.3023	1.3032	1.1465
50000 images	21.94	76.45	81.18	2.3010	0.9050	0.8443

We can notice that for a higher learning rate, the result is better than for a low one: the accuracy is very good, and the tends to 0. We can keep 0.01 as a value of the learning rate, which gives neither too small nor too large steps. In fact, if we want to choose a smaller learning rate, we have to increase the number of epochs, but we have seen previously that increasing the number of epochs brings about overfitting.

I tried to improve the performance of the model with another way to modify the learning rate. For this, I changed the value of the decay: now the learning rate is no longer constant. The value set for the learning rate is the initial value of the learning rate and for each iteration, it is updated as following:

$$learning\_rate = initial\_learning\_rate \times \frac{1}{1 + decay \times iteration}$$

I tested different values of decay to determine which one gives the better results. I performed the tests on the training set of 50000 images and I kept the previous parameters. I obtained the following accuracies and losses:

Decay	Accuracy (%)	Loss
0	76.45	0.9050
0.00001	79.50	0.9589
0.0001	75.10	1.3522
0.001	66.34	0.9407
0.01	17.80	2.2624
1	10.00	2.3025

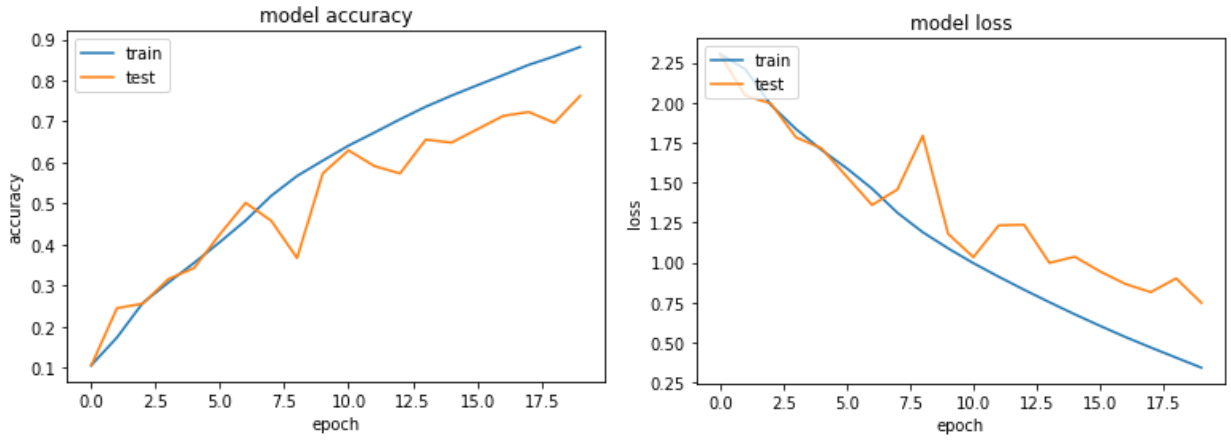
We can see that increasing the learning rate does not improve the results: the accuracy becomes lower and the loss higher. The more the decay is close to zero the better is the classification.

As a result, the learning rate has to be kept constant all the long and the optimal value, that minimizes the loss and maximizes the accuracy is 0.01.

Application of method 4

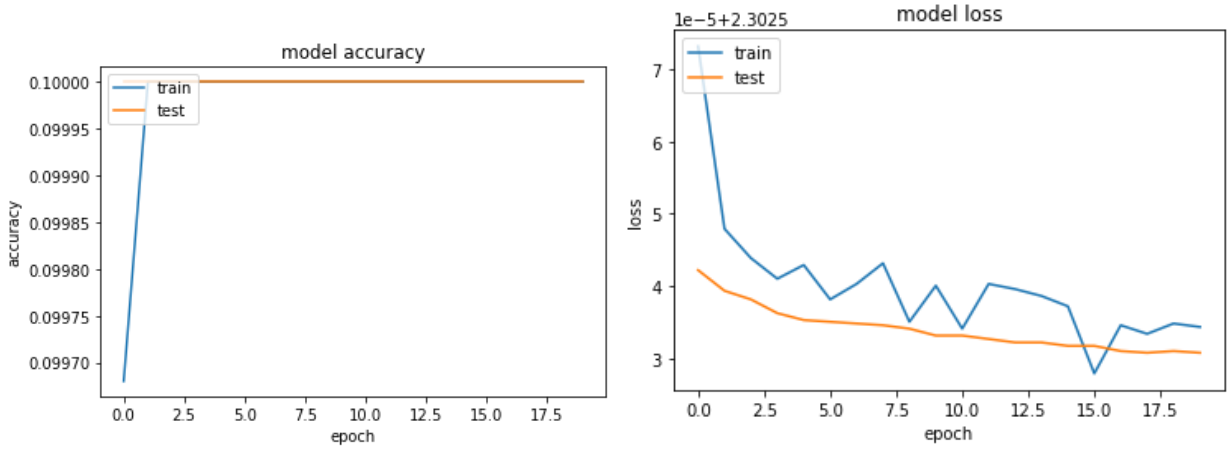
There are different optimizers that can be used to construct the CNN. The Keras documentation of the API Optimizers presents the available optimizers: SGD, that has been used until then and others. I evaluate the models constructed with some of these optimizers: AdaGrad, RMSProp and Adam. The other parameters (learning rate, number of images in the training set…) are the same than previously and the new parameters are set as default.

AdaGrad: it means “adaptive gradient” because the learning rate is adjusted at each iteration. This latter is inversely proportional to the square root of the sum of the past square values. The accuracy and the loss for this optimizer are the following:



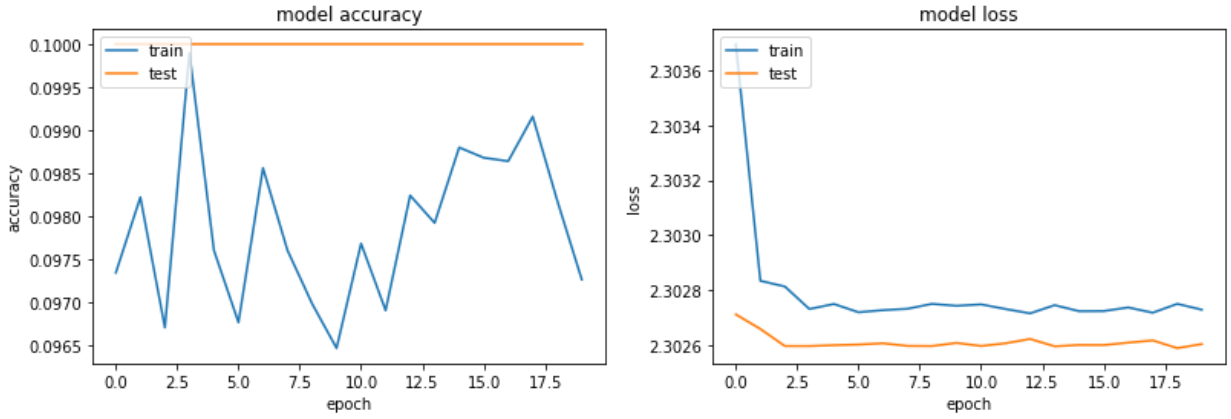
The accuracy is 76.23% which is slightly below the accuracy of the SGD and the loss is 0.7472, which is the closest value to 0 of all the trainings.

RMSProp: it has the same basis than AdaGrad, the only difference is that the gradient accumulation is changed by an exponentially weighted moving average, which only consider the proximal past. The accuracy and the loss for this optimizer are the following:



The model does not overfit. However, the accuracy is very low and the loss very high, even though the training set is composed of 50000 images.

Adam: this optimizer derives from "adaptive moments", it has the same principle than RMSProp because the update is similar, except that a smooth version of the gradient is used instead of the raw stochastic gradient. The accuracy and the loss for this optimizer are the following:



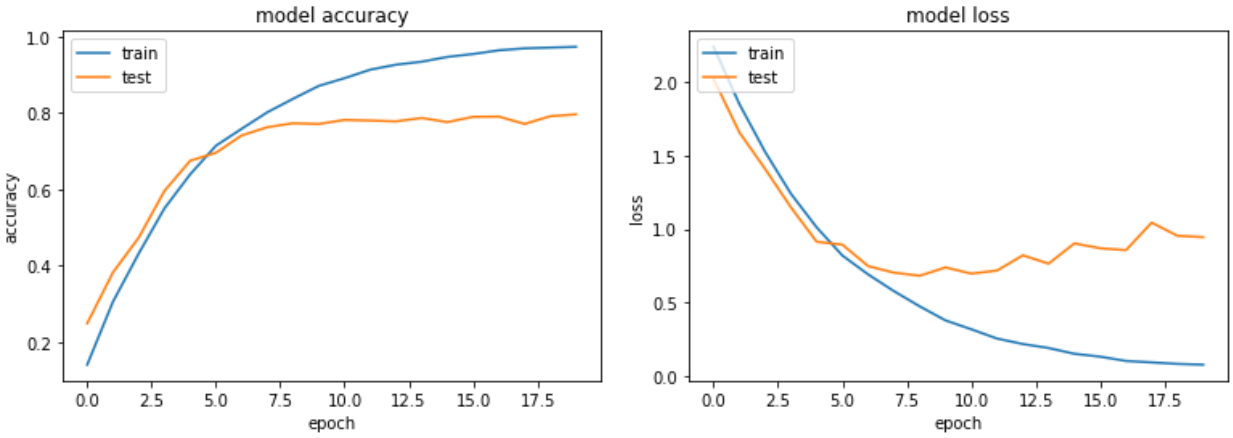
The accuracy and the loss are worst than the other optimizers, it may be improved by changing the parameters of the optimizer.

Consequently, the SGD and the AdaGrad optimizers have the best performance for this CNN.

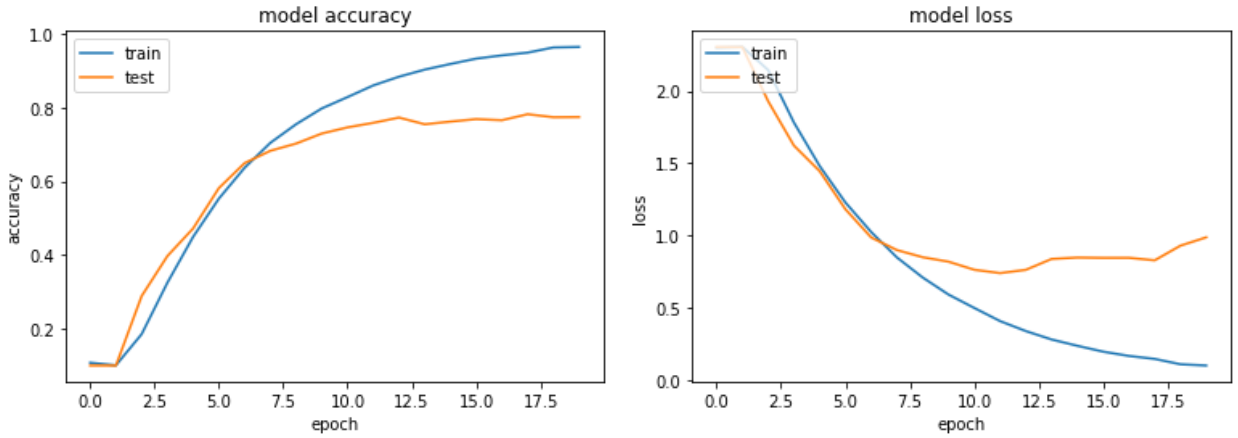
Application of method 5

Dropout allows to regularize the model and thus to prevent overfitting. The dropout rate has been initially put at 0.5 so I tested a higher and a lower value than this rate to determine which one is the best. I chose the dropout rates 0.2 and 0.7 and I obtained:

Dropout=0.2: accuracy=79.68%, loss=0.9446



Dropout=0.7: accuracy=77.47%, loss=0.9896



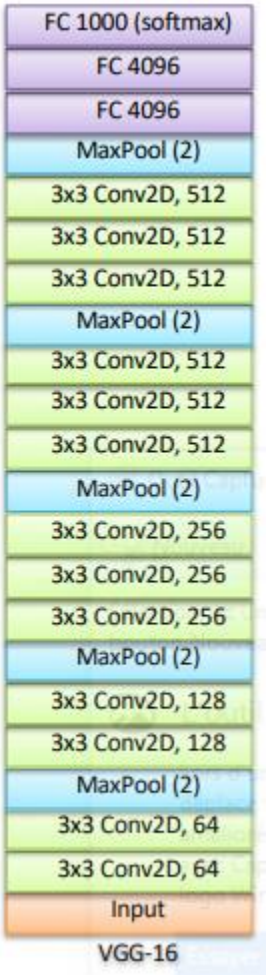
For the two dropout rates, the model is overfitting the first one for more than 5 epochs and the second one for more than 14 epochs. Increasing the dropout rate allows to obtain a better result. However, the accuracy is lower, and the loss is higher for a higher dropout rate: there is a trade-off between the dropout rate and the results obtained. To decide, we can just keep the initial dropout rate which was 0.5.

After testing different hyperparameters on the same VGG16 architecture, we can say that to have the best results when training the CNN, the number of images must be higher than 40% of the full dataset, the optimizer can be either SGD or AdaGrad, the learning rate has to be constant, the value 0.01 gives appropriate results and the number of epochs does not have to be higher than 20 and the dropout rate has to be 0.5, if we want to avoid overfitting.

Question 2: Depict the architecture of the proposed CNN. Indicate loss and accuracy values obtained during training and validation with CIFAR-10

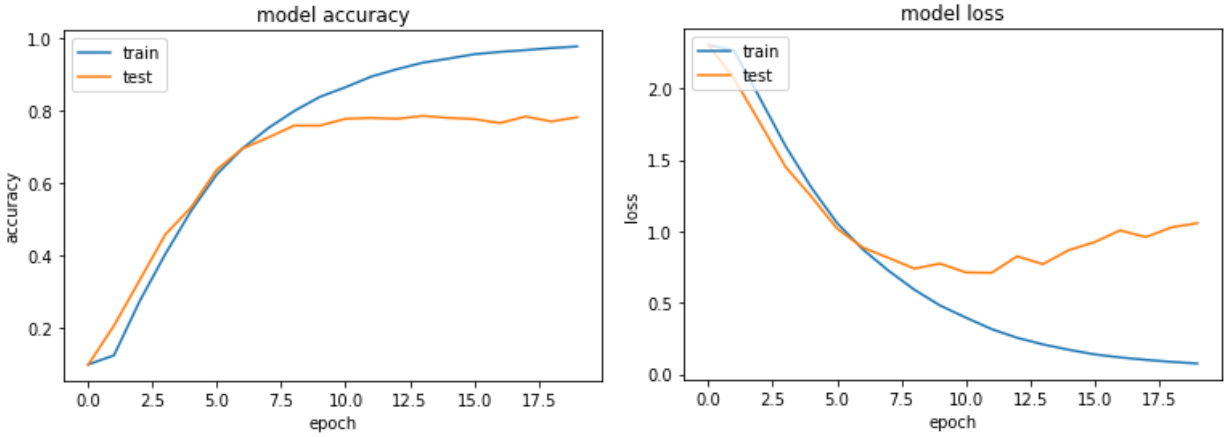
The different hyperparameters has been changed in a way to obtain the best accuracy and loss rates for classification. I keep these parameters to modify this time the architecture of the CNN. We can compare different architecture then elect the one that gives better results. To construct the CNN, we can modify two parameters: the size of the filters and the number of layers.

Firstly, I change the size of the filters of the initial CNN. I replace the 1x1 Conv2D by 3x3 ones, as following:



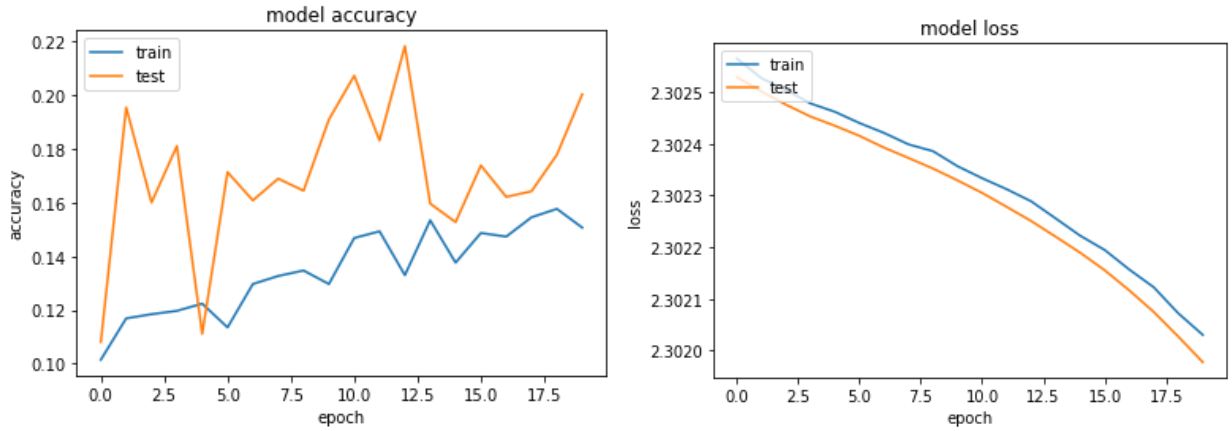
This architecture has 33.638.218 parameters and increase linearity. I computed the accuracy and the loss for the full training set, of 50000 images.

The accuracy is 78.05% and the loss is 1.0566. However, the model overfits for epochs above 6:



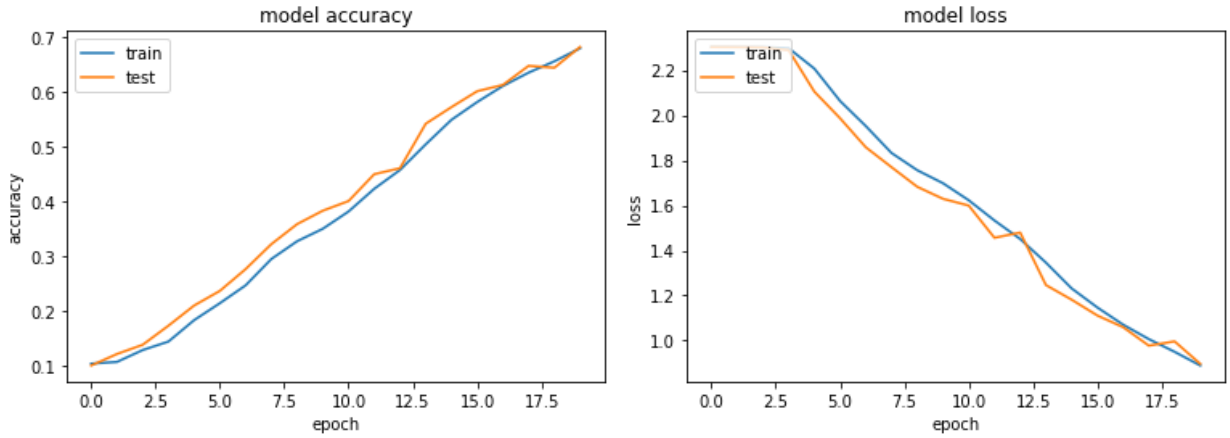
I modified the hyperparameters to obtain better results. I started with the learning rate, I changed several times until obtaining the higher accuracy and lower loss as possible:

LR=0.0001: accuracy=20.03%, loss=2.3020

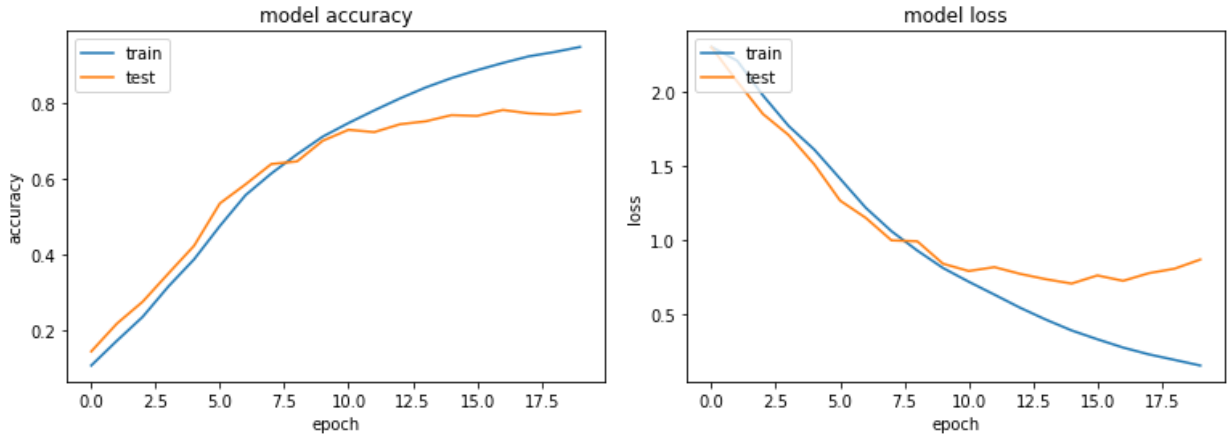


This model has bad results, the learning rate is too low, so the steps of the algorithm are too small. I increased gradually the learning rate:

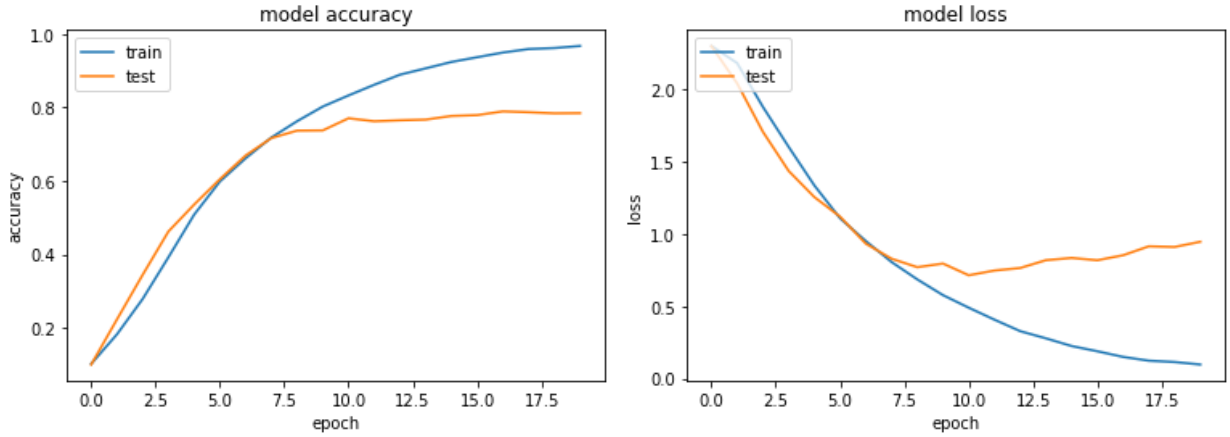
LR=0.001: accuracy=68.09%, loss=0.8968



LR=0.003: accuracy=77.85%, loss=0.8675

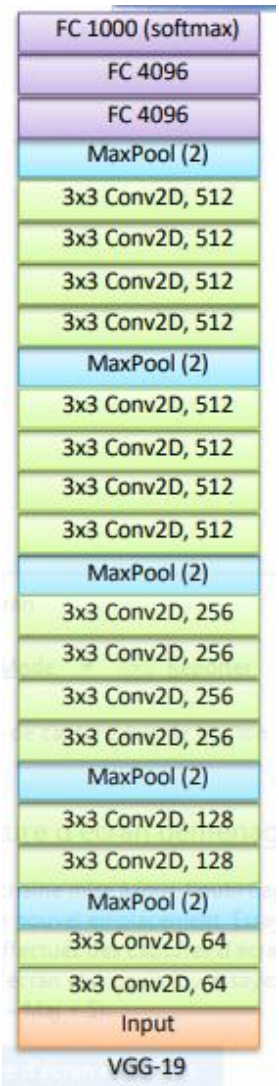


LR=0.005: accuracy=78.41%, loss=0.9464



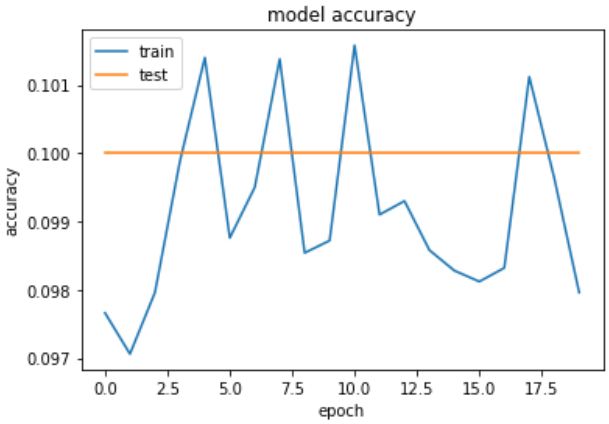
The learning rate 0.001 gives us the best results until now, the accuracy is the highest without overfitting problem. We can see that increasing the learning rate allow to have better results, but for a learning rate above 0.003, the model overfits. When the learning rate is too high, the SGD algorithm does not converge, and the results are bad.

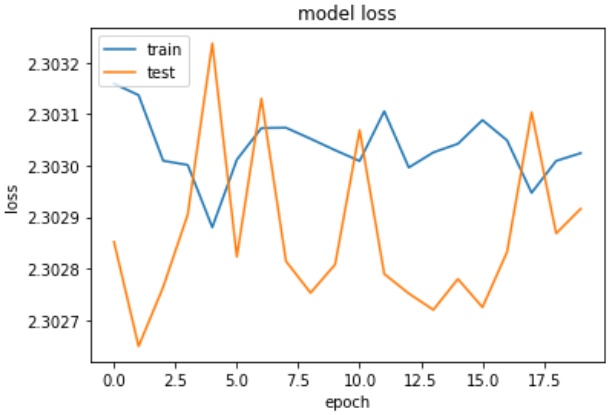
Then I add layers in the architecture, I use the VGG19 model which is the following:



This architecture has 38.947.914 parameters. I used the full training set so 50000 images.

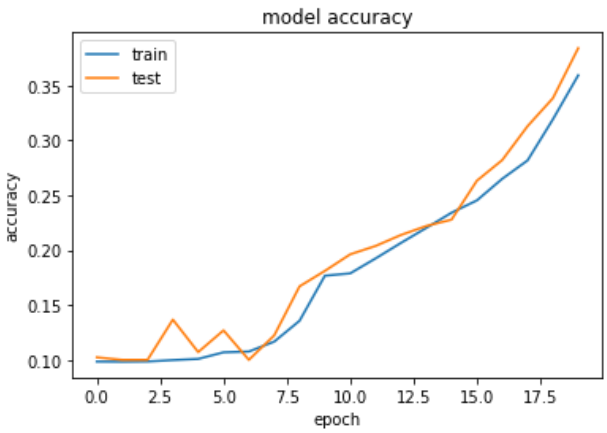
For the learning rate 0.01, the accuracy is 10.00% and the loss is 2.3029, which means that it also has to be modified:

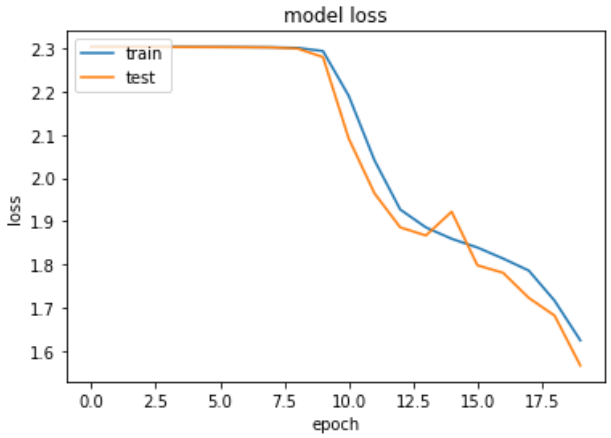




For this architecture, I also tested different learning rates.

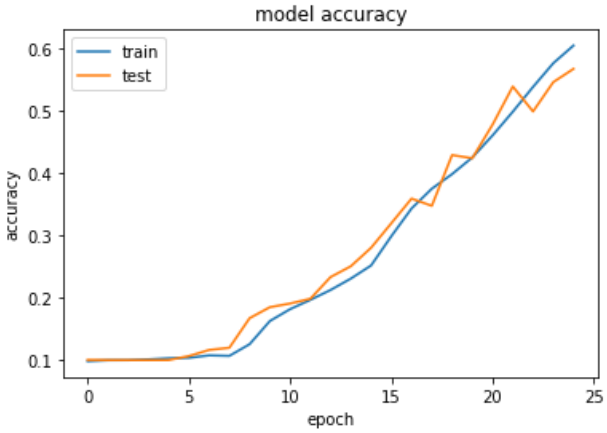
LR=0.001: accuracy=38.41%, loss=1.5431

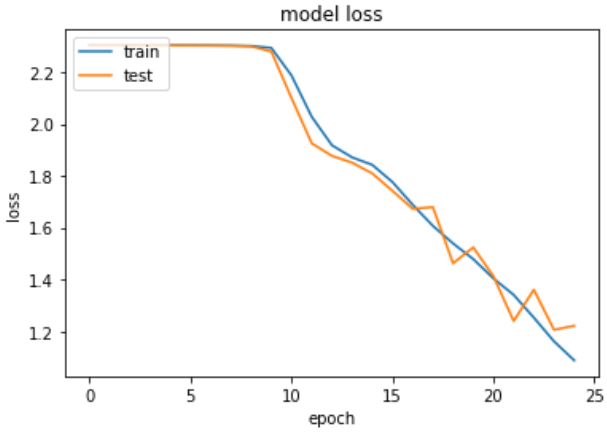




This model is well-fitting so to increase the accuracy, I raised the number of epochs to 25:

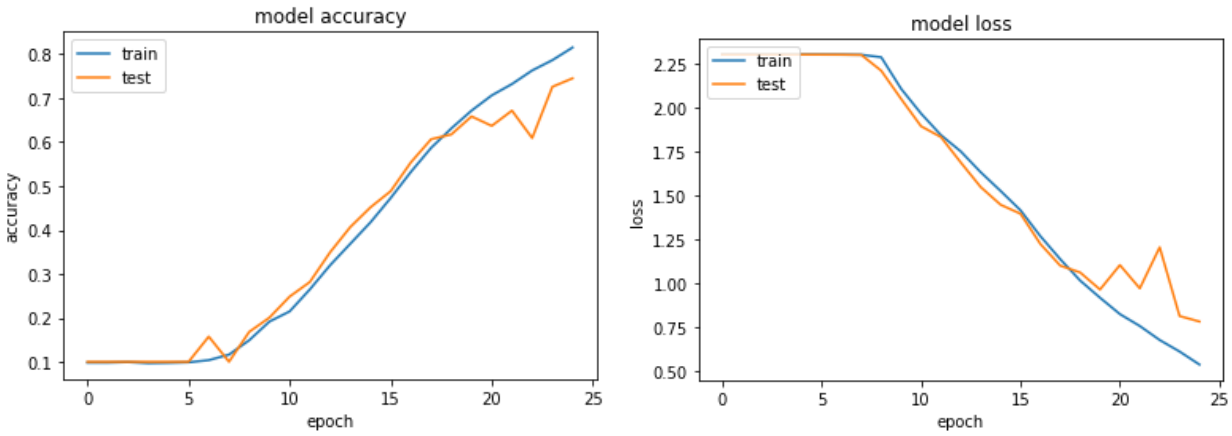
LR=0.001 & epochs=25: accuracy=56.66%, loss=1.2214





Increasing the number of epochs actually improve the results. I keep this number of epochs and I increased the learning rate. I obtained:

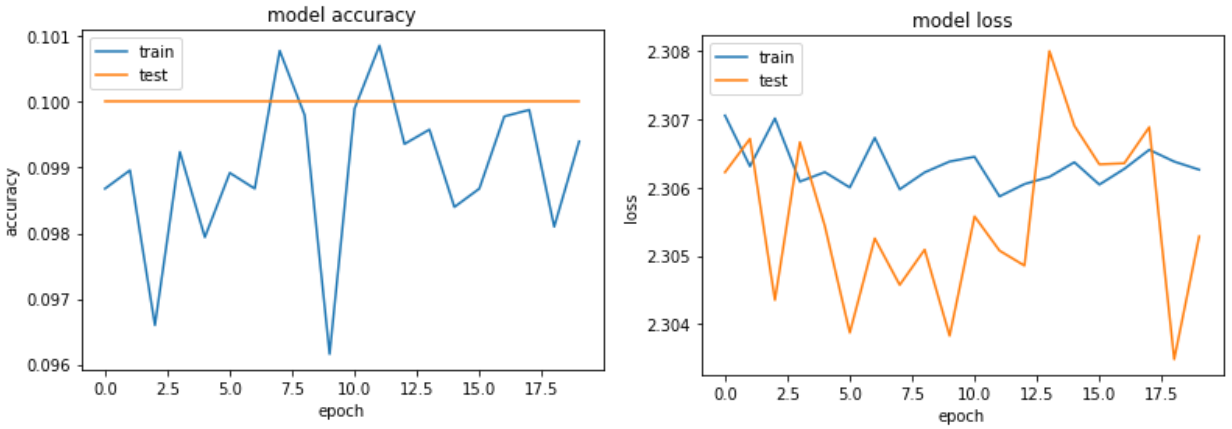
LR=0.002 & epochs=25: accuracy=74.52%, loss=0.7815



The loss is the lowest until now and the model overfits from 20 epochs, which means that it is a better model than the VGG16 with the same parameters.

I changed the structure of the VGG19 by modifying the size of the filter, in the same way than I did for the VGG16. The architecture is the same, I just replaced all the 3×3 sizes by 5×5 sizes and I obtained a 74.537.034 parameters model. The accuracy and the loss for this architecture are:

LR=0.1: accuracy=10.00%, loss=2.3053



I tested other values of learning rate (0.01, 0.001 & 0.002) and the result was still the same. We can assume that this architecture does not match with the set of images.

We have seen that the main issue encountered for the model is overfitting. To avoid it, a solution can be the used is the regularization with the dropout rate, that has already been treated. Another solution can be cross-validation, which is a validation technique for assessing the model based on the repetition of

random data splitting in the training set. An alternative can be weight regularization, which consists in updating the loss function to penalize the model in proportion to the size of the model weights. Finally, another possibility to increase the accuracy and reduce the loss can be data augmentation, which is the generation of additional data by resampling the training dataset with small random modifications.

The proposed CNN has the architecture of the VGG19.