MACHINE LEARNING AND DEEP LEARNING

HOMEWORK 2 **S265348** **SOFIA BORGATO**

1-DATA PREPARATION:

As first step I compile the Caltech Class as follow:
-the init method read and save in a sorted list all the classes in the “101_ObjectCategories”. Then I create a dictionary with the label and an assigned index. By filtering I delete the “BACKGROUND_Google”.
-By reading the text file referred to the split parameter of the Caltech Class each line has been use to access and save the related images from the whole dataset. The image has been saved in the samples list with its index and converted as ‘RGB’
-I also define the len method that return the number of element saved in he samples list, and the method get items that thanks to the index return the image and corresponding label.

Thanks to this class I create to object of Caltech: one from the train.txt file and the other from test.txt .

Then I split the previous train set in a 2 equal portions(odd indexes for the train, even indexes for the validation) in a new train set and a validation set. The two sets also contain the same number of element for each label.

Those are the result of the phase just described:

```
102 ['BACKGROUND_Google', 'Faces', 'Faces_easy', 'Leopards', 'Motorbikes', 'accordion',
'airplanes', 'anchor', 'ant', 'barrel', 'bass', 'beaver', 'binocular', 'bonsai', 'brain', 'brontosaurus', 'buddha',
'butterfly', 'camera', 'cannon', 'car_side', 'ceiling_fan', 'cellphone', 'chair', 'chandelier', 'cougar_body',
'cougar_face', 'crab', 'crayfish', 'crocodile', 'crocodile_head', 'cup', 'dalmatian', 'dollar_bill', 'dolphin',
'dragonfly', 'electric_guitar', 'elephant', 'emu', 'euphonium', 'ewer', 'ferry', 'flamingo',
'flamingo_head', 'garfield', 'gerenuk', 'gramophone', 'grand_piano', 'hawksbill', 'headphone',
'hedgehog', 'helicopter', 'ibis', 'inline_skate', 'joshua_tree', 'kangaroo', 'ketch', 'lamp', 'laptop', 'llama',
'lobster', 'lotus', 'mandolin', 'mayfly', 'menorah', 'metronome', 'minaret', 'nautilus', 'octopus', 'okapi',
'pagoda', 'panda', 'pigeon', 'pizza', 'platypus', 'pyramid', 'revolver', 'rhino', 'rooster', 'saxophone',
'schooner', 'scissors', 'scorpion', 'sea_horse', 'snoopy', 'soccer_ball', 'stapler', 'starfish', 'stegosaurus',
'stop_sign', 'strawberry', 'sunflower', 'tick', 'trilobite', 'umbrella', 'watch', 'water_lilly', 'wheelchair',
'wild_cat', 'windsor_chair', 'wrench', 'yin_yang']
101 {'Faces': 1, 'Faces_easy': 2, 'Leopards': 3, 'Motorbikes': 4, 'accordion': 5, 'airplanes': 6, 'anchor':
7, 'ant': 8, 'barrel': 9, 'bass': 10, 'beaver': 11, 'binocular': 12, 'bonsai': 13, 'brain': 14, 'brontosaurus':
15, 'buddha': 16, 'butterfly': 17, 'camera': 18, 'cannon': 19, 'car_side': 20, 'ceiling_fan': 21,
'cellphone': 22, 'chair': 23, 'chandelier': 24, 'cougar_body': 25, 'cougar_face': 26, 'crab': 27, 'crayfish':
28, 'crocodile': 29, 'crocodile_head': 30, 'cup': 31, 'dalmatian': 32, 'dollar_bill': 33, 'dolphin': 34,
'dragonfly': 35, 'electric_guitar': 36, 'elephant': 37, 'emu': 38, 'euphonium': 39, 'ewer': 40, 'ferry': 41,
'flamingo': 42, 'flamingo_head': 43, 'garfield': 44, 'gerenuk': 45, 'gramophone': 46, 'grand_piano': 47,
'hawksbill': 48, 'headphone': 49, 'hedgehog': 50, 'helicopter': 51, 'ibis': 52, 'inline_skate': 53,
'joshua_tree': 54, 'kangaroo': 55, 'ketch': 56, 'lamp': 57, 'laptop': 58, 'llama': 59, 'lobster': 60, 'lotus':
61, 'mandolin': 62, 'mayfly': 63, 'menorah': 64, 'metronome': 65, 'minaret': 66, 'nautilus': 67,
'octopus': 68, 'okapi': 69, 'pagoda': 70, 'panda': 71, 'pigeon': 72, 'pizza': 73, 'platypus': 74, 'pyramid':
75, 'revolver': 76, 'rhino': 77, 'rooster': 78, 'saxophone': 79, 'schooner': 80, 'scissors': 81, 'scorpion':
82, 'sea_horse': 83, 'snoopy': 84, 'soccer_ball': 85, 'stapler': 86, 'starfish': 87, 'stegosaurus': 88,
'stop_sign': 89, 'strawberry': 90, 'sunflower': 91, 'tick': 92, 'trilobite': 93, 'umbrella': 94, 'watch': 95,
'water_lilly': 96, 'wheelchair': 97, 'wild_cat': 98, 'windsor_chair': 99, 'wrench': 100, 'yin_yang': 101}
iterations without BACKGROUND 5784
iterations with BACKGROUND 6096
Number of images of train = 5784
Number of images of train without BACKGROUND_Google Class = 5784
len 101

102 ['BACKGROUND_Google', 'Faces', 'Faces_easy', 'Leopards', 'Motorbikes', 'accordion',
'airplanes', 'anchor', 'ant', 'barrel', 'bass', 'beaver', 'binocular', 'bonsai', 'brain', 'brontosaurus', 'buddha',
'butterfly', 'camera', 'cannon', 'car_side', 'ceiling_fan', 'cellphone', 'chair', 'chandelier', 'cougar_body',
'cougar_face', 'crab', 'crayfish', 'crocodile', 'crocodile_head', 'cup', 'dalmatian', 'dollar_bill', 'dolphin',
'dragonfly', 'electric_guitar', 'elephant', 'emu', 'euphonium', 'ewer', 'ferry', 'flamingo',
'flamingo_head', 'garfield', 'gerenuk', 'gramophone', 'grand_piano', 'hawksbill', 'headphone',
'hedgehog', 'helicopter', 'ibis', 'inline_skate', 'joshua_tree', 'kangaroo', 'ketch', 'lamp', 'laptop', 'llama',
'lobster', 'lotus', 'mandolin', 'mayfly', 'menorah', 'metronome', 'minaret', 'nautilus', 'octopus', 'okapi',
'pagoda', 'panda', 'pigeon', 'pizza', 'platypus', 'pyramid', 'revolver', 'rhino', 'rooster', 'saxophone',
'schooner', 'scissors', 'scorpion', 'sea_horse', 'snoopy', 'soccer_ball', 'stapler', 'starfish', 'stegosaurus',
'stop_sign', 'strawberry', 'sunflower', 'tick', 'trilobite', 'umbrella', 'watch', 'water_lilly', 'wheelchair',
'wild_cat', 'windsor_chair', 'wrench', 'yin_yang']
101 {'Faces': 1, 'Faces_easy': 2, 'Leopards': 3, 'Motorbikes': 4, 'accordion': 5, 'airplanes': 6, 'anchor':
7, 'ant': 8, 'barrel': 9, 'bass': 10, 'beaver': 11, 'binocular': 12, 'bonsai': 13, 'brain': 14, 'brontosaurus':
15, 'buddha': 16, 'butterfly': 17, 'camera': 18, 'cannon': 19, 'car_side': 20, 'ceiling_fan': 21,
'cellphone': 22, 'chair': 23, 'chandelier': 24, 'cougar_body': 25, 'cougar_face': 26, 'crab': 27, 'crayfish':
28, 'crocodile': 29, 'crocodile_head': 30, 'cup': 31, 'dalmatian': 32, 'dollar_bill': 33, 'dolphin': 34,
'dragonfly': 35, 'electric_guitar': 36, 'elephant': 37, 'emu': 38, 'euphonium': 39, 'ewer': 40, 'ferry': 41,
'flamingo': 42, 'flamingo_head': 43, 'garfield': 44, 'gerenuk': 45, 'gramophone': 46, 'grand_piano': 47,
'hawksbill': 48, 'headphone': 49, 'hedgehog': 50, 'helicopter': 51, 'ibis': 52, 'inline_skate': 53,
'joshua_tree': 54, 'kangaroo': 55, 'ketch': 56, 'lamp': 57, 'laptop': 58, 'llama': 59, 'lobster': 60, 'lotus':
61, 'mandolin': 62, 'mayfly': 63, 'menorah': 64, 'metronome': 65, 'minaret': 66, 'nautilus': 67,
'octopus': 68, 'okapi': 69, 'pagoda': 70, 'panda': 71, 'pigeon': 72, 'pizza': 73, 'platypus': 74, 'pyramid':
75, 'revolver': 76, 'rhino': 77, 'rooster': 78, 'saxophone': 79, 'schooner': 80, 'scissors': 81, 'scorpion':
82, 'sea_horse': 83, 'snoopy': 84, 'soccer_ball': 85, 'stapler': 86, 'starfish': 87, 'stegosaurus': 88,
'stop_sign': 89, 'strawberry': 90, 'sunflower': 91, 'tick': 92, 'trilobite': 93, 'umbrella': 94, 'watch': 95,
'water_lilly': 96, 'wheelchair': 97, 'wild_cat': 98, 'windsor_chair': 99, 'wrench': 100, 'yin_yang': 101}
iterations without BACKGROUND 2893
iterations with BACKGROUND 3049
Number of images of test = 2893
Number of images of test without BACKGROUND_Google Class = 2893
len 101
Train Dataset: 2892
Valid Dataset: 2892
Test Dataset: 2893
```

I created two data-loader objects (train_data-loader ,after shuffling the train set and test_data-loader), I initialized alexnet and set the last level of alexnet to the number of correct classes, defined the loss function, choose the parameters to optimize and finally trained the network.

Then I create the third data-loader for the validation set, without shuffle the data.

2a,b,c-TRAINING FROM SCRATCH

For this phase at the end of each epoch I tested the accuracy on the validation set and calculate the loss function . If an increasing in term of accuracy was registered I saved the current status of the network as best_net.
At the end I test the accuracy on the test set by using the best_net saved.
With the default configuration(BATCH_SIZE=256, LR=1e^{-3}, MOMENTUM=0.9 , WEIGHT_DECAY=5e^{-5}, NUM_EPOCH=30, STEP_SIZE=20, LOG_FREQUENCIES=10) the loss function decrease really slow and the accuracy is bad too.

I use as device “cuda” and so just for not stressed to much the GPU I don’t use the grid-search and I decide to change manually the hyper-parameters evaluating each time the result collected.

First of all I decide to decrease BATCH_SIZE in order to avoid problem with the RAM, and also because a minor size permit to update several times the model and so get better accuracy.
So I choose as batch_size 64.

LEARNING_RATE: 10^{-2} (10^{-3} by default), I opt for a greater learning rate because I observed that the loss function decrease slowly with the first configuration and thanks to this setting I can penalize more the model errors.

NUMBER_OF_EPOCH: 50

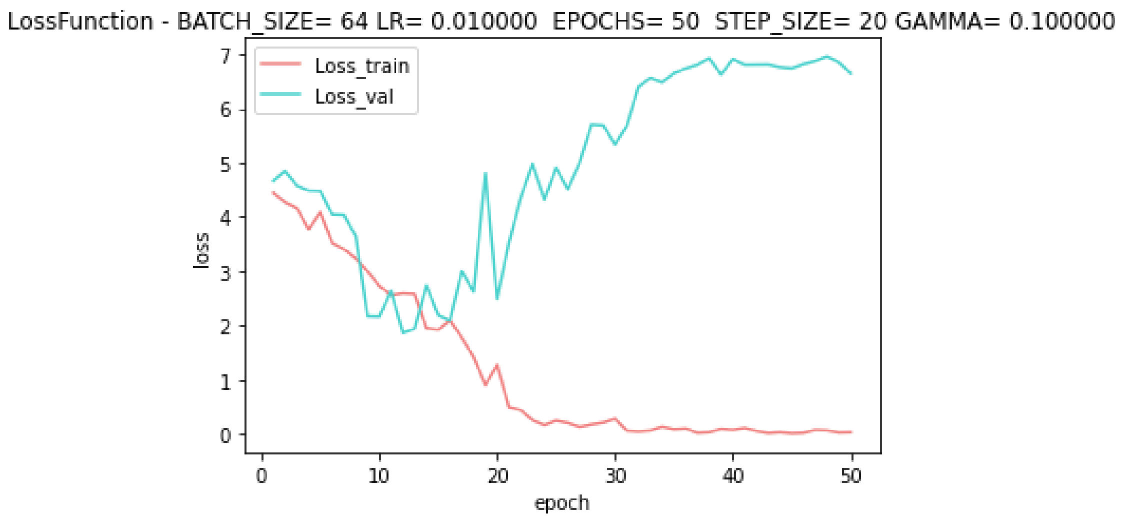
STEP_SIZE: 20.

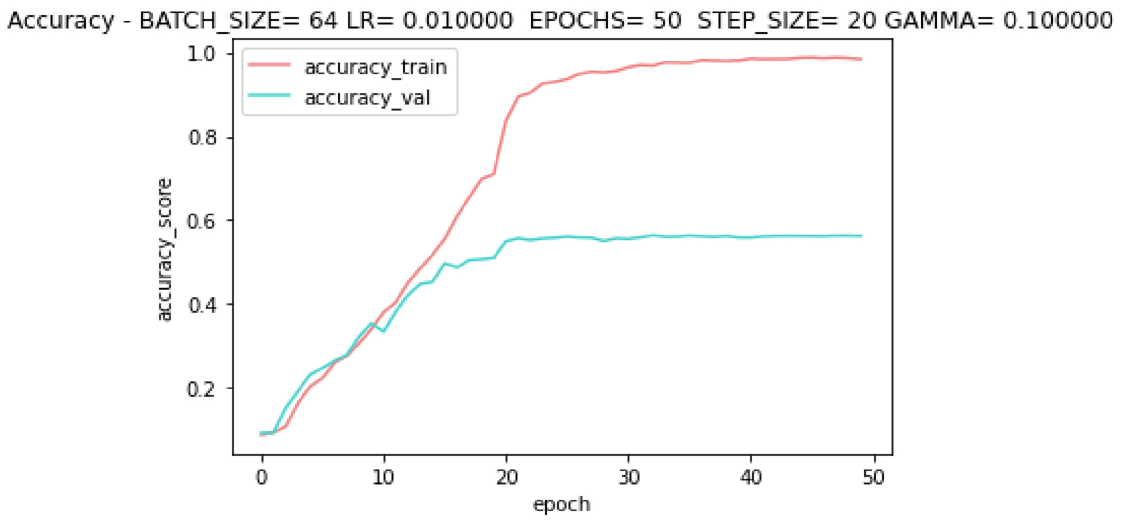
GAMMA: 0.1 , I understand that one of the key points of this analysis is find a good balance between the step size, learning rate and the number of epoch. In this “game” play a key rule the gamma parameter. I observe before that fixed the number of epoch to 30, has good result by increasing the learning rate. For this reason is useless choose a minor gamma value. On the other hand putting a value of gamma equal to 1 stop the updating of the learning rate after the step size, and this is useless to.

WEIGHT_DECAY: 5e^{-5}

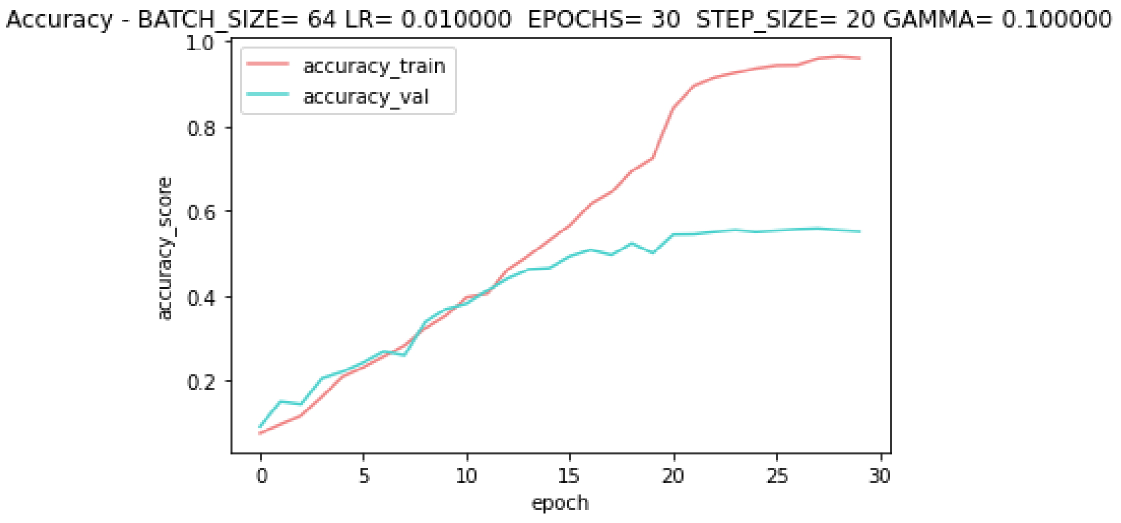
LOG_FREQUENCY: 10.

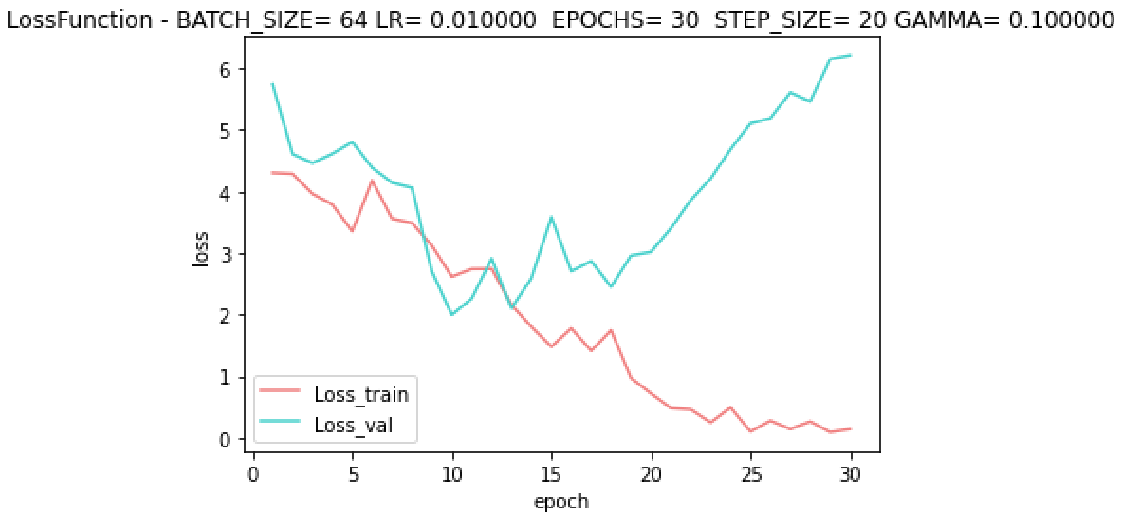
The result of the described configuration are the following :





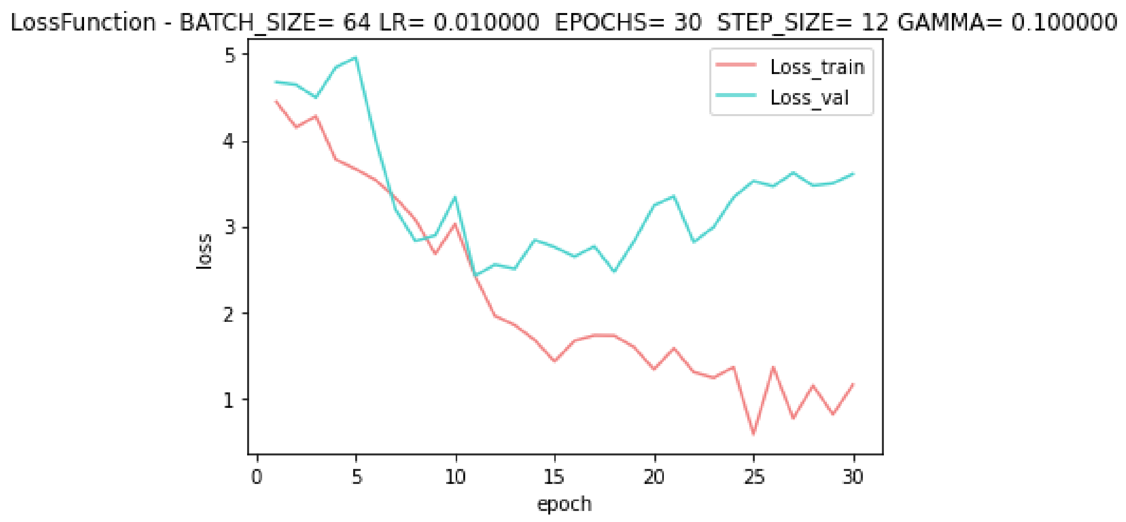
As we can see in the graph representing the loss function, the loss function calculated with the validation set , at a certain point (near by the 20th epoch) restart increase and as a consequences we have a stabilization of the accuracy near by its maximum values. This behaviour is caused by the over-fitting of the training data.
The best accuracy calculated on the validation set is 0.561.
So after this attempt I decide to decrease the number of epoch until 30, with all the previous parameters.
And I get the following result:

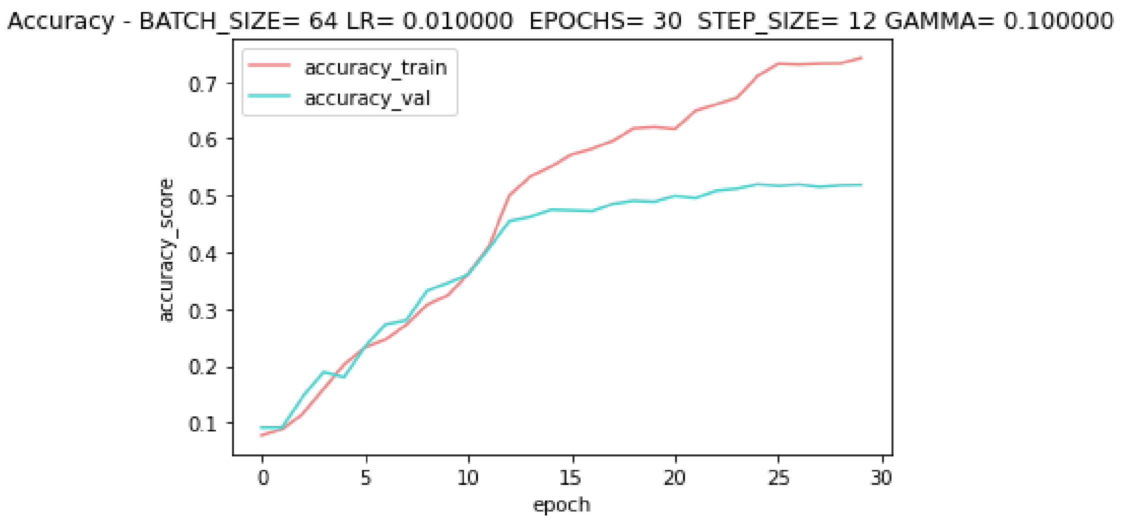




As we can see the over-fitting starts near by the 20th epoch.
In term of accuracy on validation set it was scored as best 0.556.

Finally I decided to try with STEP_SIZE =12 , with all the parameters previous discussed.





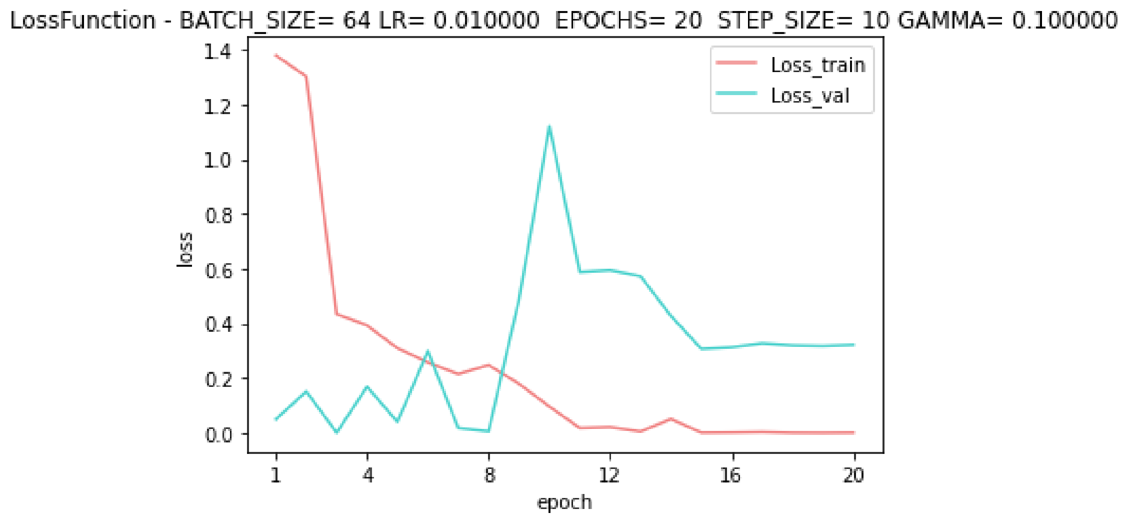
With this configuration I get an accuracy score on the validation set of 0.533. The previous model is the best.

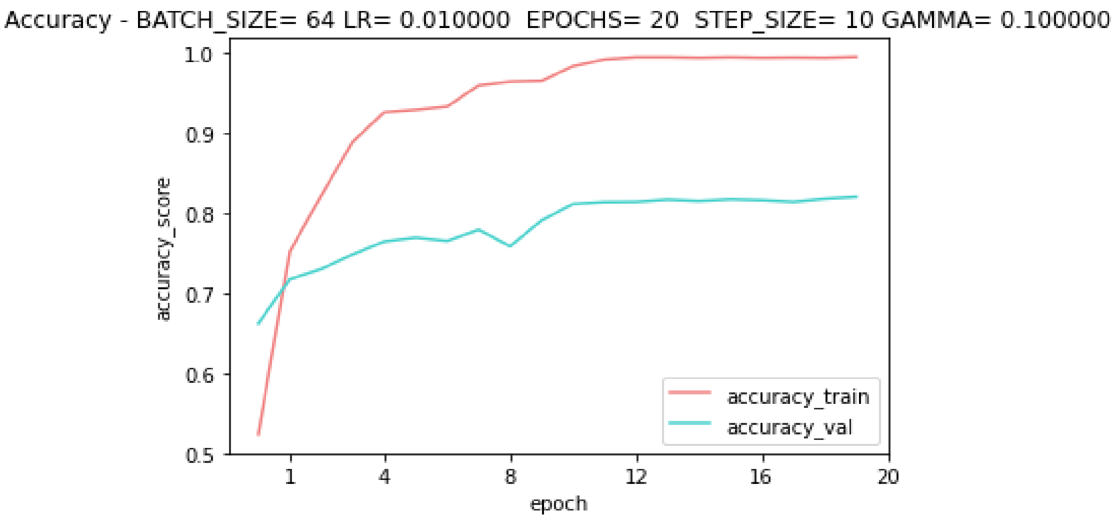
3-TRANSFER LEARNING A,B,C:

I set the parameter pre-trained=TRUE, for use the pre-trained model on image-net dataset and also update the normalize function with these values (0.485, 0.456, 0.406), (0.229, 0.224, 0.225). corresponding to mean and standard deviation of the image-net dataset.

I train the model on the train set and at each epoch evaluate the accuracy and loss function on both the 2 set.
As first hyper-parameters configuration I choose instead of the default ones:
BATCH_SIZE =64
LR=0.01
STEP_SIZE =10

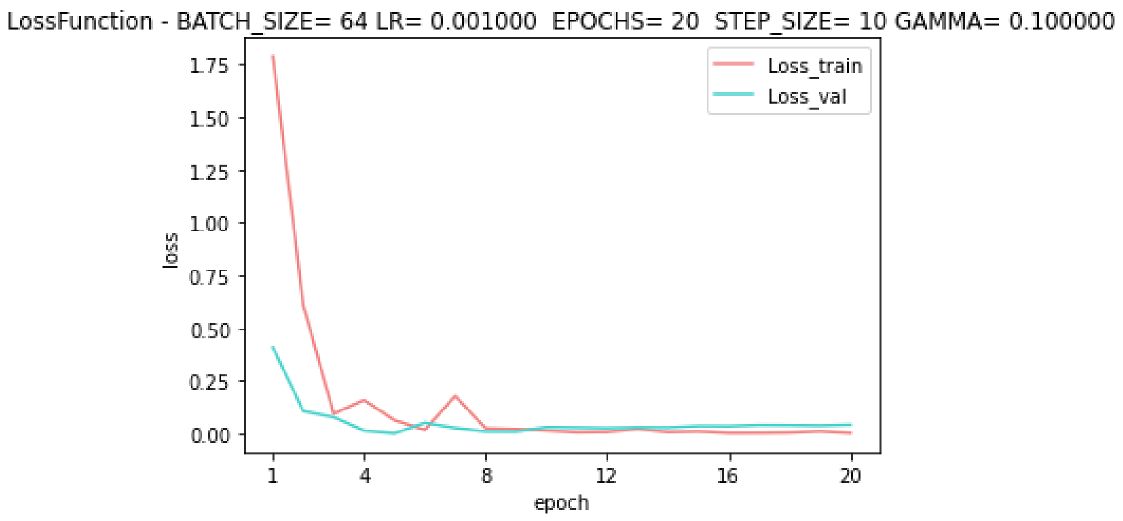
GAMMA=0.1
I get the following result:

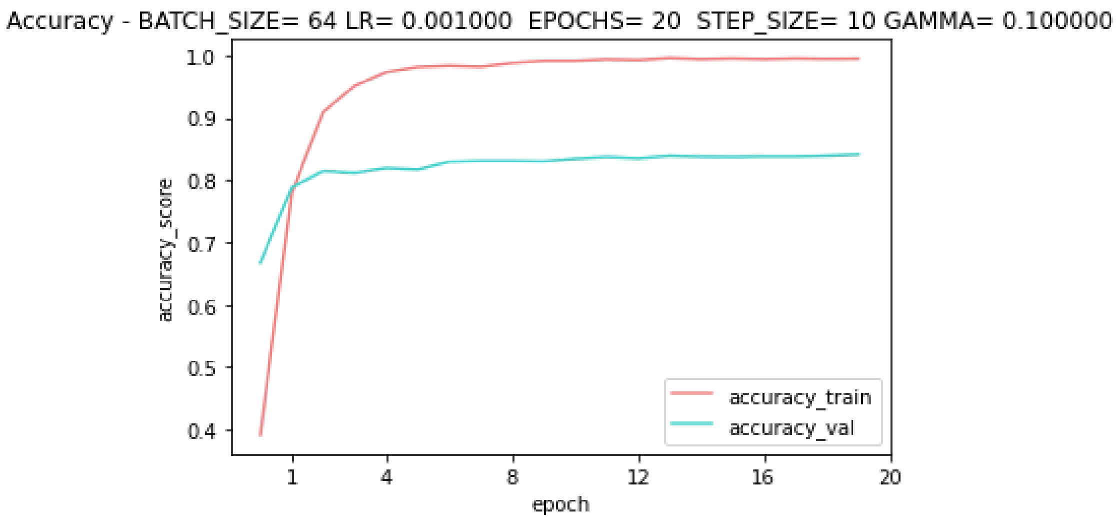




The best value for the accuracy is 0.81, a good value compared with the others get without pre-training.

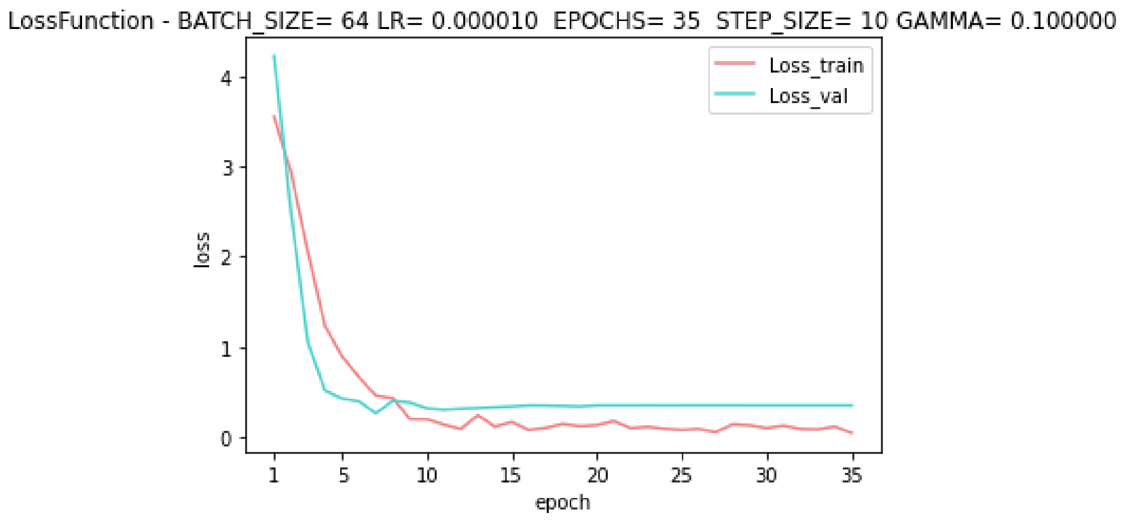
The loss function on the train test decrease really fast but there is a problem on the loss function of the validation set.
On the test set is scored 0.81 , also from there we can see the benefits of training on a bigger-set: the difference between the validation and the test set is really small.
As I expect a smaller number of epochs (20) is enough to make the loss function converge at zero, so I decided to take this as fixed.
For fix the problem of the excessive to oscillate of the loss function of the validation set I opt for decrease the learning rate because I think that it could be inputted to the too strong updating of the weight.
So in the new configuration with learning rate= 1e^{-3} I get the following result:

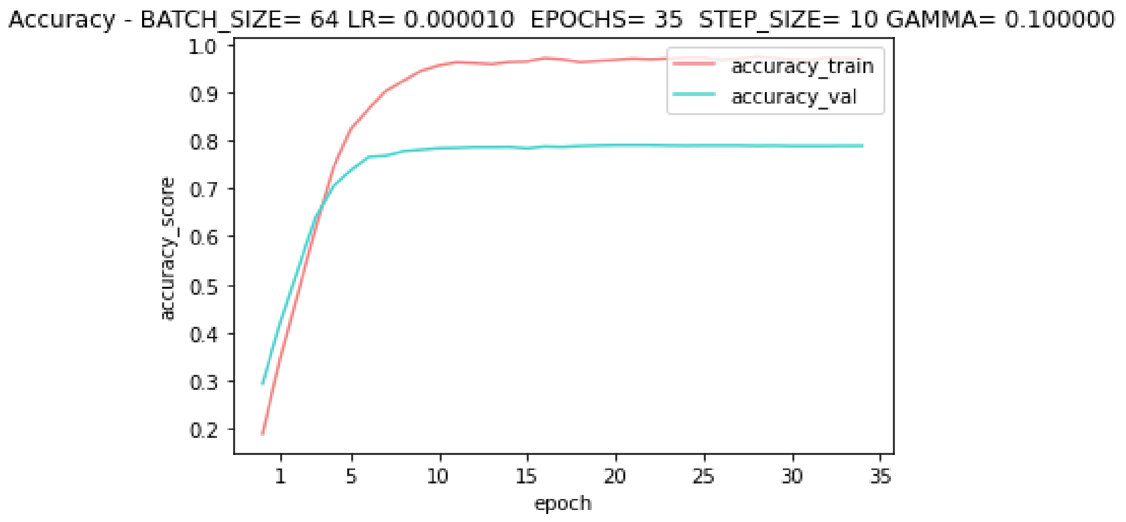




In this case the loss function converge to zero both for the train set and the validation set.
The best accuracy on the validation set is 0.8385. Really close to the one on the test set that is 0.834.

As last configuration I decide to change the optimizer, I choose for the Adam module provided by py-torch.
After some attempts to fit a good model by using this optimizer, I get the following configuration as best:
BATCH-SIZE=64
LR=1e^{-5}
WEIGHT-DECAY=5e^{-5}
NUM_EPOCHS=35
STEP_SIZE=10
GAMMA=0.1
LOG_FREQUENCY=10
In this set I get the following results:

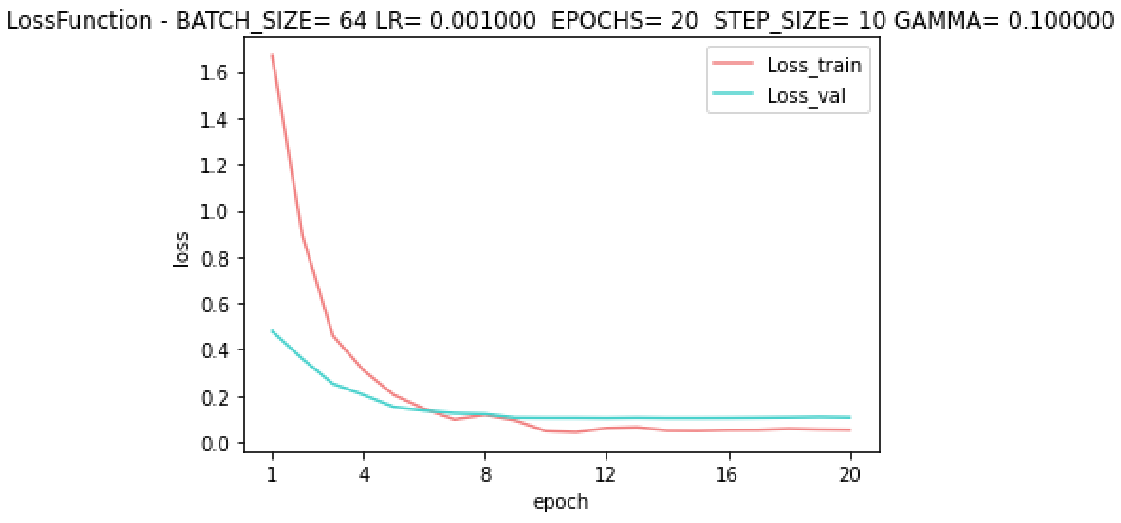


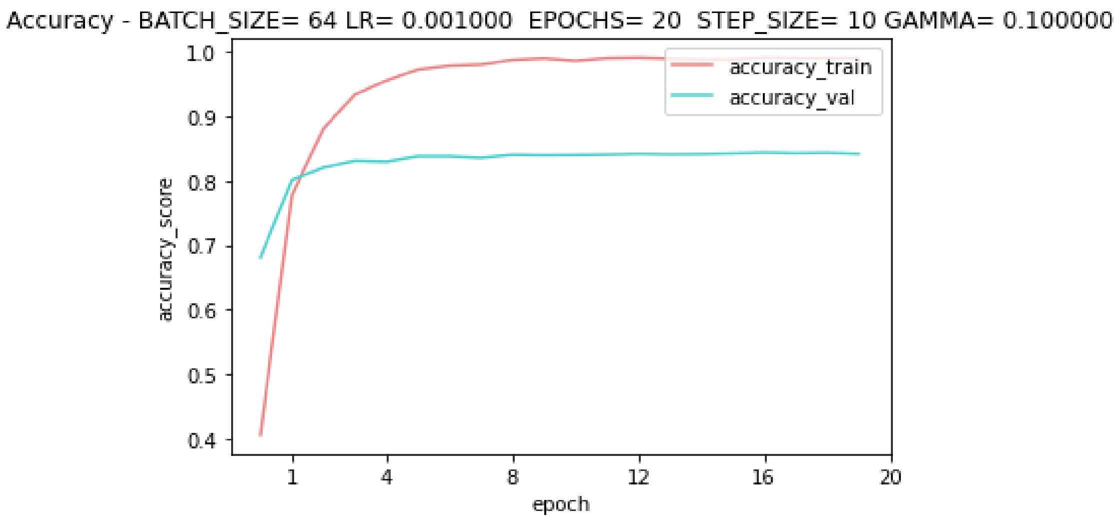


The best accuracy reached by testing on the validation set is 0.790.
And testing on the set test we get 0.789

D-TRAIN ONLY THE FULLY-CONNECTED LAYER.

Thanks to the method param.required_grad=false, it’s possible to freeze some layer and train only on what we desire. As first try I freeze the CNN layer and train only on the fully connected(the last 4) . In this way I train the net setting the best parameters I found with the previous case(BATCH-SIZE=64, LR=1e$^{-5,}$,WEIGHT-DECAY= 5e^{-5} , NUM_EPOCHS=20,STEP_SIZE=10 GAMMA=0.1,LOG_FREQUENCIES=10) . I registered the following result:

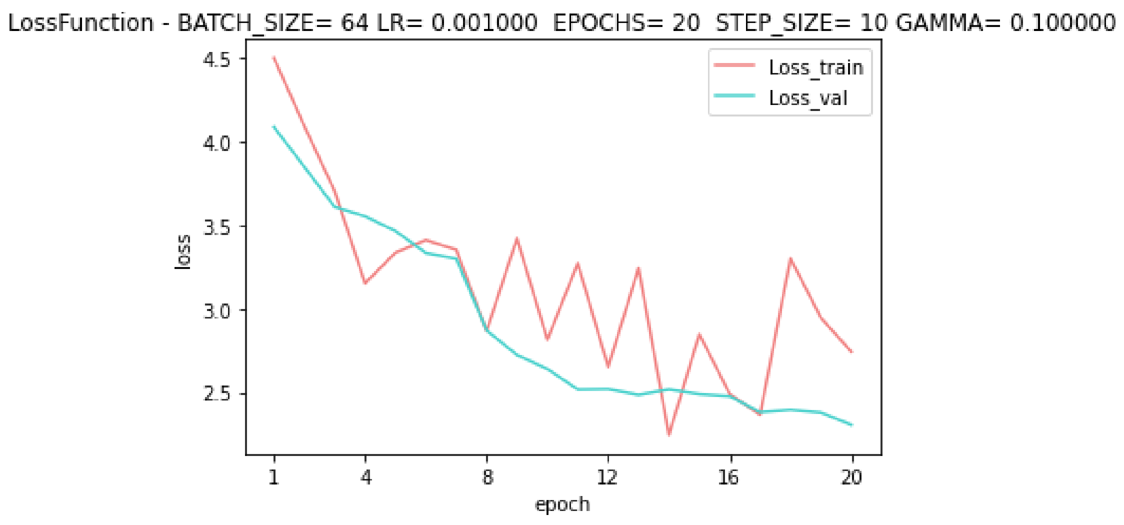


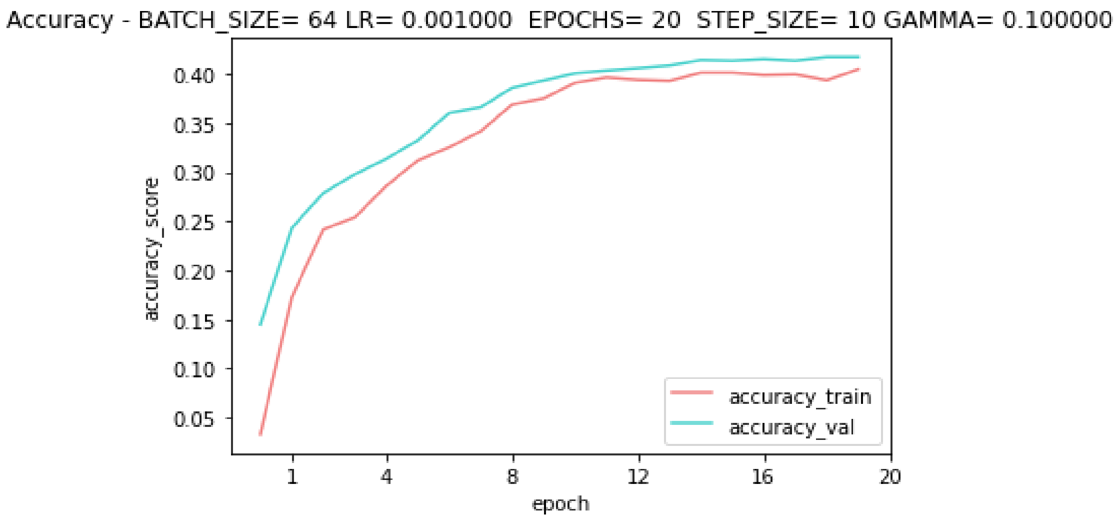


As we can see the loss function of both the train and the validation decrease and converges to a value near by zero, and so the accuracy is stable.
The best accuracy and the lost function minimum is reached near by the 12th epoch, the best value of the first one is 0.8473.
By testing the model on the test set is scored 0.850 in accuracy.
So thanks to this method I have a good feedback in terms of accuracy score.

E-FREEZE THE FULLY CONNECTED LAYER:

For do this step we freeze the layer that has index>9 ,by setting for the respective index require_grad=FALSE.
By using the hyper-parameters associated to the best configuration of the previus point(BATCH-SIZE=64, LR=1e^{-5},WEIGHT-DECAY= 5e^{-5} , NUM_EPOCHS=20,STEP_SIZE=10 GAMMA=0.1,LOG_FREQUENCIES=10) I get the same result:



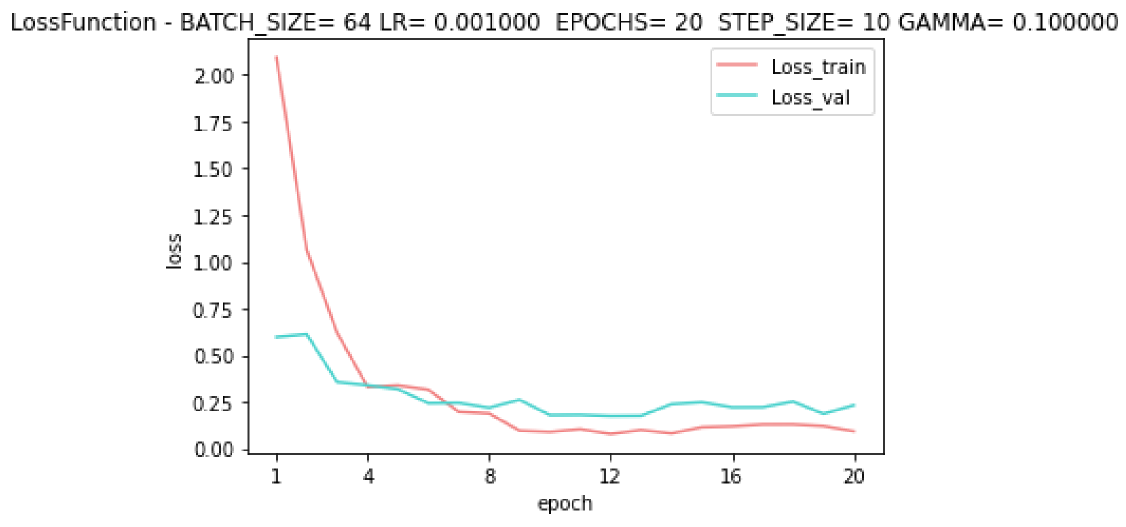


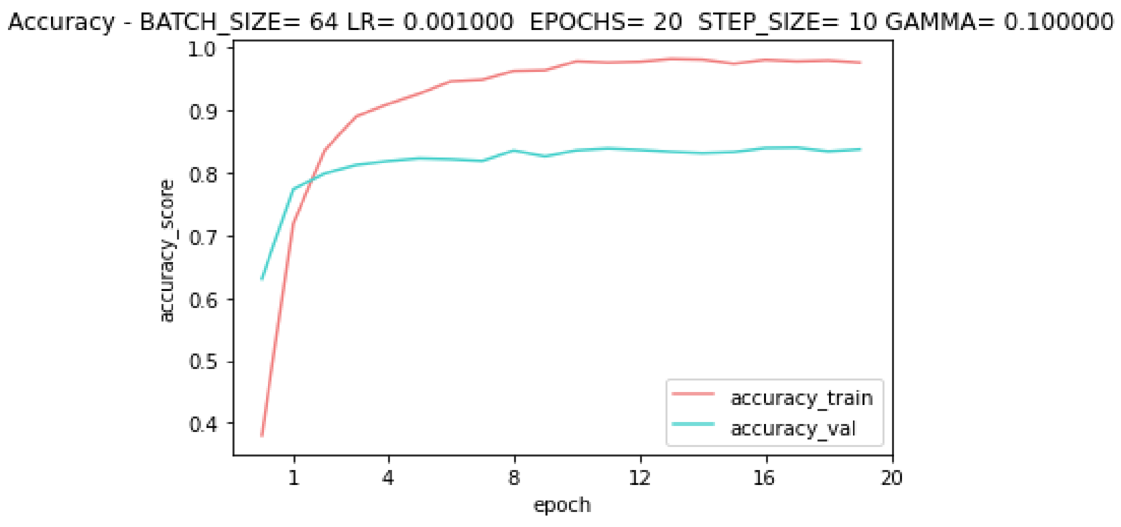
As we can see from the graph the loss function is not smooth as before, and I registered I strong decrease in accuracy on the evaluation set: 0.394. This bad result is not a surprise because the fc-layers are the ones used for classification, and freezing them the net is not able to classify well our dataset, it just based the classification on the weight learned with the image-net dataset.

4-DATA AUGMENTATION:

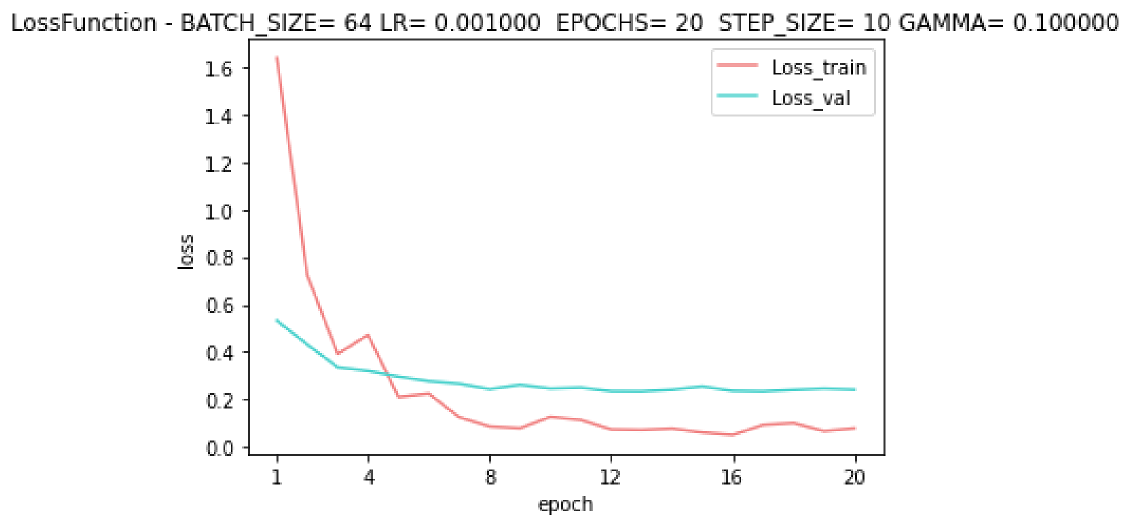
I try different transforms and different combinations of transforms, all the model are trained with the best configuration reached till now, so I'm training only on the fully connected layers and the net parameters are :BATCH-SIZE=64, LR=1e$^{-5,}$,WEIGHT-DECAY= 5e^{-5} , NUM_EPOCHS=20,STEP_SIZE=10 GAMMA=0.1,LOG_FREQUENCIES=10, with SGD-Optimizer.
All the this configuration are added to the other and used before.

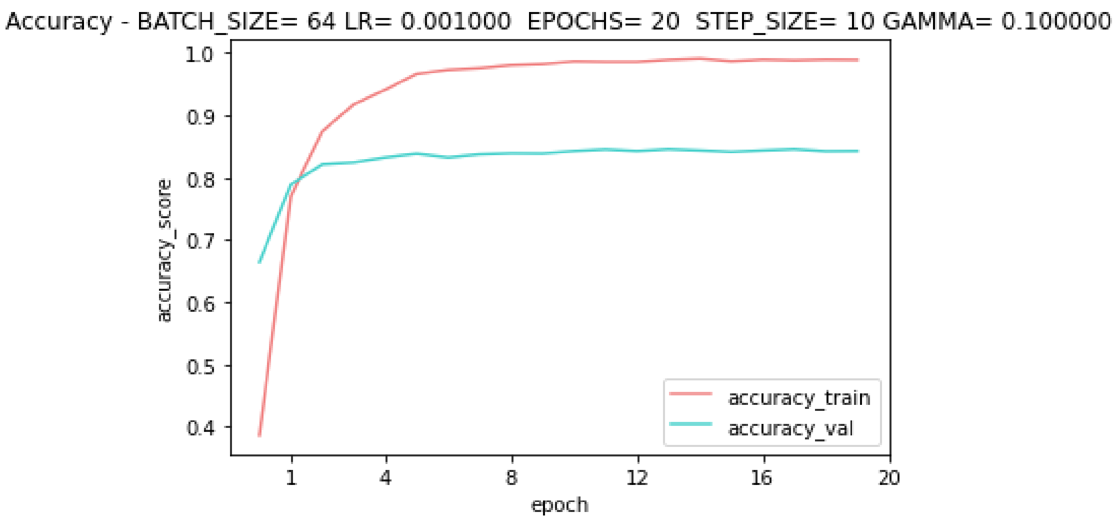
As first data transform I try RandomGrayScalewith probability 0.5 setting.
And I collect the following result:



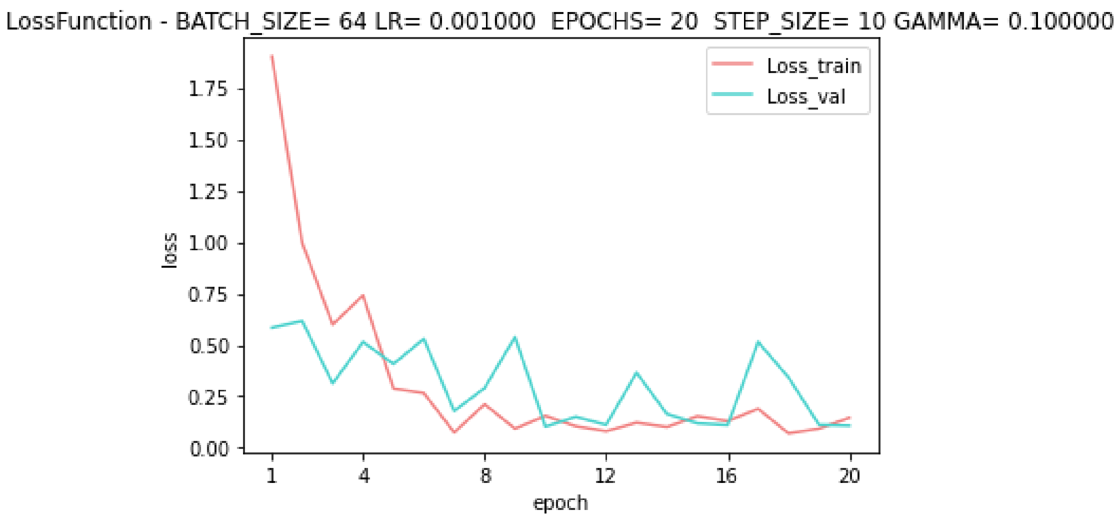


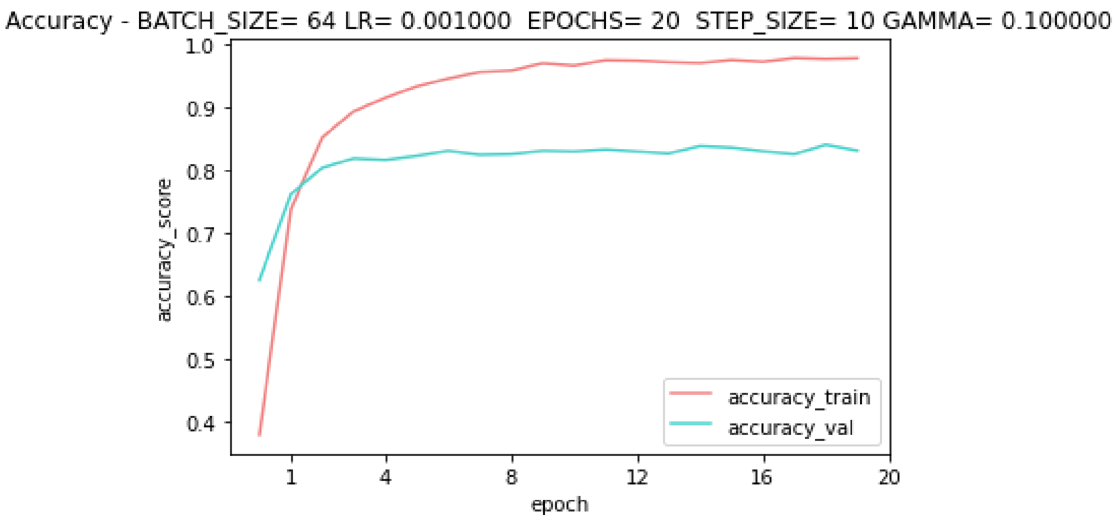
As best accuracy on the validation is scored 0.8459, and 0.8520 on the test set.
Then I try with RandomHorizontalFlip(p=0.5), as result I have:





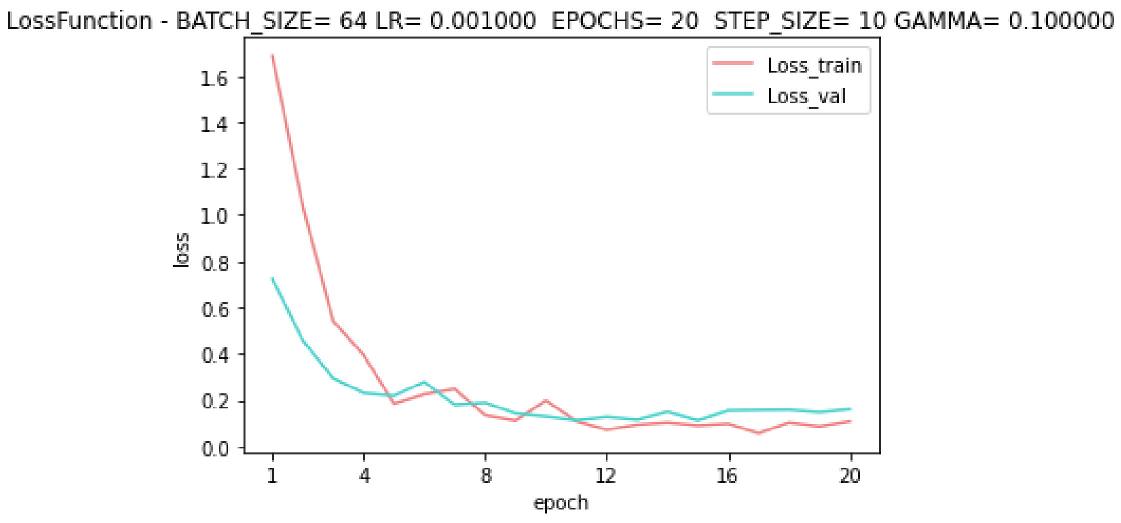
And get a value of 0.8457 of accuracy for the validation set, and 0.8492 for the test set.
Then I substitute the CenterCrop(224), with the RandomCrop(224).
And get :

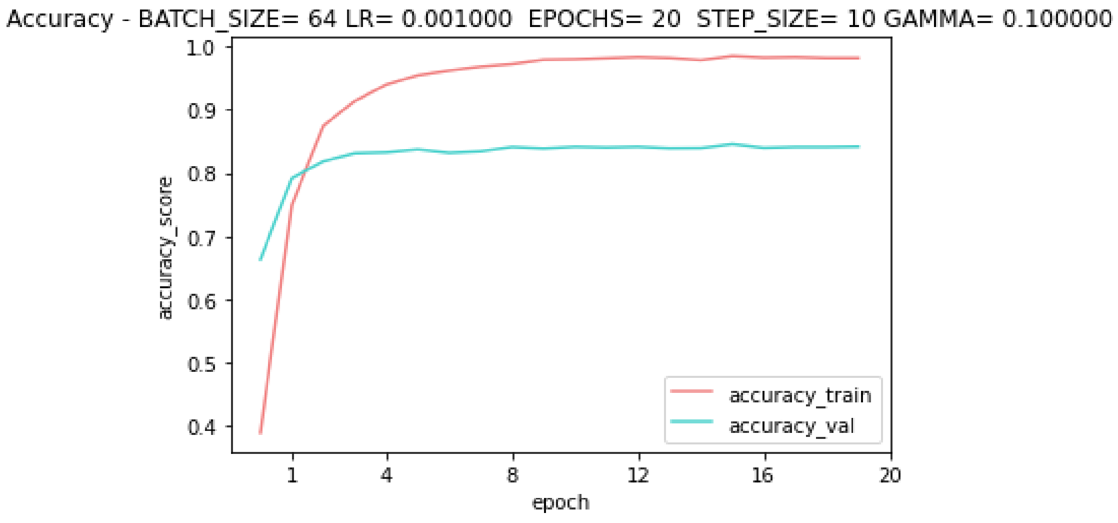




As we can see in this case the loss function decrease till zero but not smooth as before.

As a consequences is registered a small loss in accuracy both on the validation both on validation and test sets. The best accuracy on the validation set is 0.839 , and the test set 0.840
for this reason I decide to prefer the CenterCrop transformation.
Finally I compose the 3 transformations and apply together CenterCrop, RandomGreyScale(p=0.1),RandomHorizontalFlip(p=0.1). What I registered is



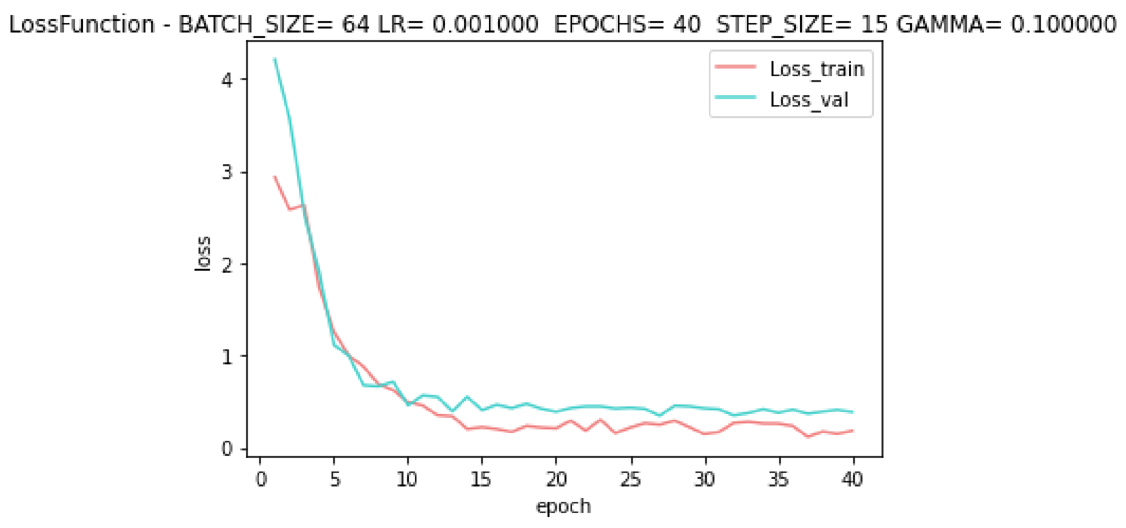


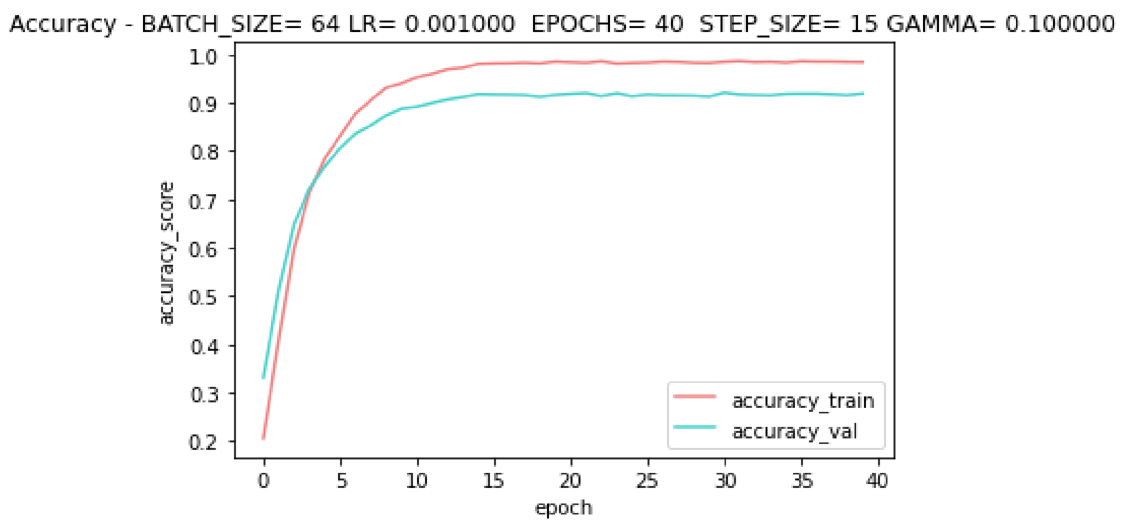
The result are comparable with the previous get without data augmentation, but in this case the loss function oscillate near by a value a little bit higher than zero, in the previous case was quiet smooth
I get 0.8487 of accuracy on the validation set, 0.8501 on the test set.

5-BEYOND ALEXNET

Finally I try with the Resnet18. I do some tuning in order to set best hyper-parameter for this model balancing LR,NUM_EPOCHS AND GAMMA . I also try training only the convolutional layers or the fully connected layer but it cause a great loss in accuracy and the loss function doesn’t converge at all.
I maintain the last described preprocessing.

So I train the model on the train set and after each epoch evaluate the accuracy on both train and validation and the accuracy too. Finally I test the model and on the test set where it's scored 0.9219. A good increase in comparison with AlexNet.
The behaviour of the loss function and the accuracy is described in the following graph:





As expect this is the best result scored. Resnet18 has 18 layer and this permits a better fit of the data.