MACHINE LEARNING AND DEEP LEARNING

HOMEWORK 2

Sofia Perosin S269748

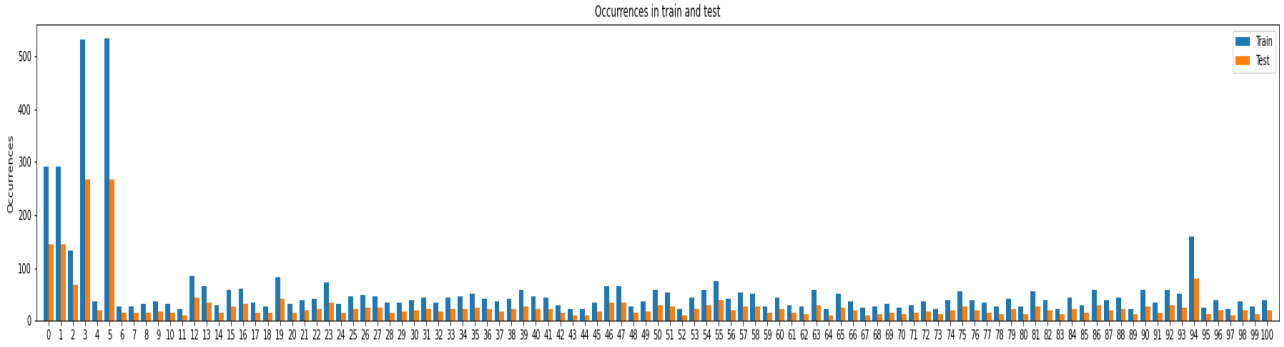
OVERVIEW

The aim of this homework is to train a Convolutional Neural Network for image classification using AlexNet as CNN and Caltech-101 as dataset.

The Caltech 101 dataset contains pictures of objects belonging to 101 categories, there are about 40 to 800 images per category and most categories have about 50 images. The size of each image is roughly 300 * 200 pixels[1].

DATA PREPARATION

The first step is the data preparation, so the creation of the dataset for Caltech, removing the *"Background"* folder and splitting the images in the remaining 101 classes into the train and test dataset, according to the specified splits.



Since there are a lot of classes the corresponding labels are not very easily readable, but it's possible to see how there are much more samples from the categories number 0 (*"Faces"*), 1 (*"Faces_easy"*), 3 *("Motorbikes"*), 5 (*"airplaines"*). So because of the model will be trained with a large amount of observations of these categories, their classification will be more efficient.

TO BE UNDERLINE

The AlexNet classifier is slightly modified because we have to specify that the output size is equal to the number of classes specified during the *"Set Arguments"* step.

HOW TO CHOOSE THE BEST PERFORMING MODEL

In order to not be redundant, it's previously shown the logic according to which a model is chosen as the best performing: both accuracy and loss behaviour are taken into account.

The loss is a good measure which allows to understand if there is underfitting or overfitting: are preferred those models which reach the smallest lost and where the gap between the loss measured on the train and the loss measured on the validation is as small as possible (if it's too big there is overfitting).

Then if the loss in two or more models is similar is preferred the one which gives the biggest accuracy on the validation set.

[1] Information from http://www.vision.caltech.edu/Image_Datasets/Caltech101/

TRAINING FROM SCRATCH

The first approach is to train AlexNet from scratch, so in order to tune the hyperparameters the train dataset has to be subdivided into the train and validation datasets. This is done by taking care to maintain the proportion of each class, in order to not have some categories underrepresented or overrepresented.

The loss function used is the *"Cross Entropy Loss"* which is a criterion that combines *"Log Soft max"* and *"NLL Loss"* and it's very useful for the training part of a classification problem.

Now it's needed to optimize over all the parameters of AlexNet since the model is trained from scratch; it will be not a good idea to optimize only some of the layers inside the network since the initial weights are random.

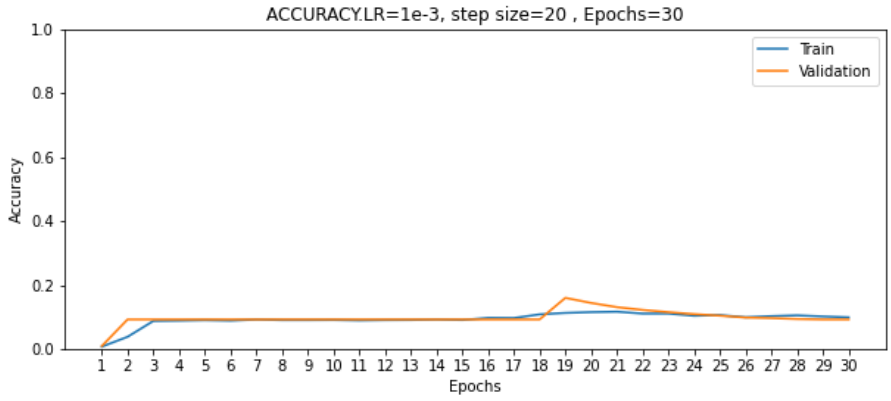
The algorithm used for updating network weights is the SGD (Stochastic Gradient Descent), faster than the Gradient Descent. In the optimizer it has to be specified the parameters to optimize and the learning rate, then it's possible, but not mandatory, to specified the momentum, the weight decay, the dampening for momentum and if it's allowed to used the Nesterov accelerated gradient.

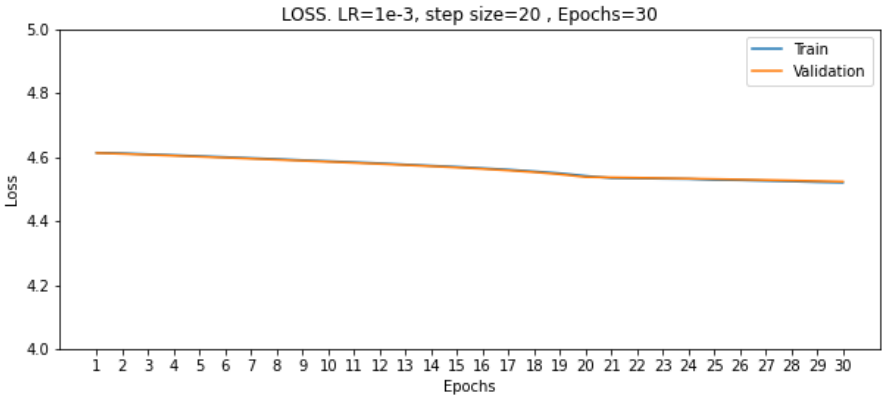
Finally the dynamically changing of the learning rate is done by the scheduler.

Using a *"Step LR"* scheduler the learning rate of each parameter group is decayed by a parameter gamma after a fixed number of epochs.

The learning rate is one of the most important hyperparameters, since it decides how much the model has to be changed according to the estimated error when the weights are updated. For this reason the main attention, at first, is focused on the *"learning rate"* parameter, in order to find an acceptable range for it and then tune also the other parameters.

- LR = 1e-3, STEP_SIZE = 20, NUM_EPOCHS = 30





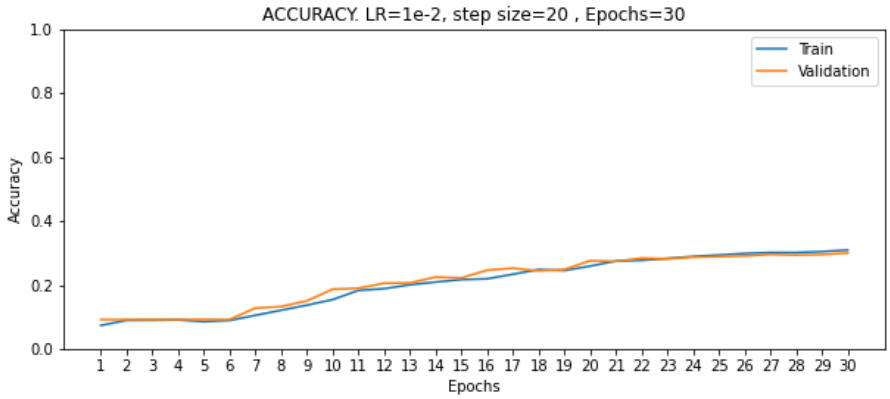
[2]

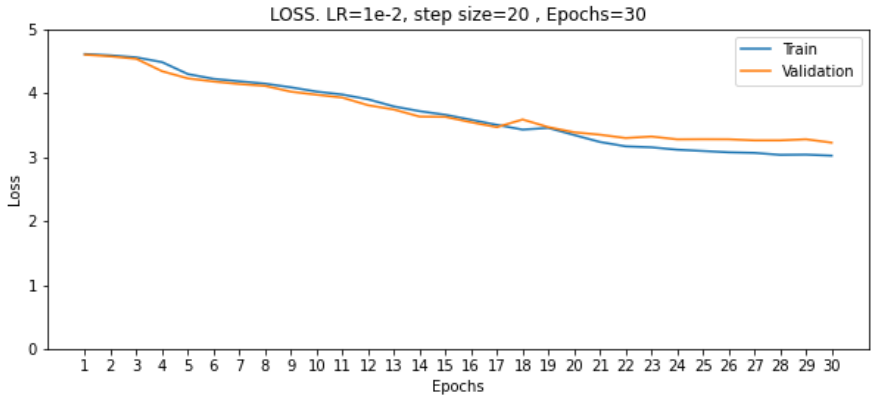
[2] Please note that the range in y-axis is different from the following.

How it's possible to see this rate is too low, since the loss, both in validation and training set decreases too slowly and the accuracy calculated in the validation set is stuck at 0.092.

So the parameter has to be increased.

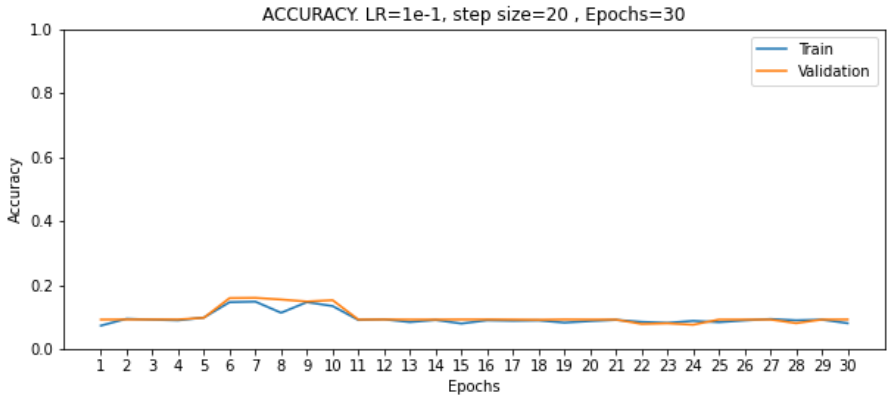
- LR = 1e-2, STEP_SIZE = 20, NUM_EPOCHS = 30

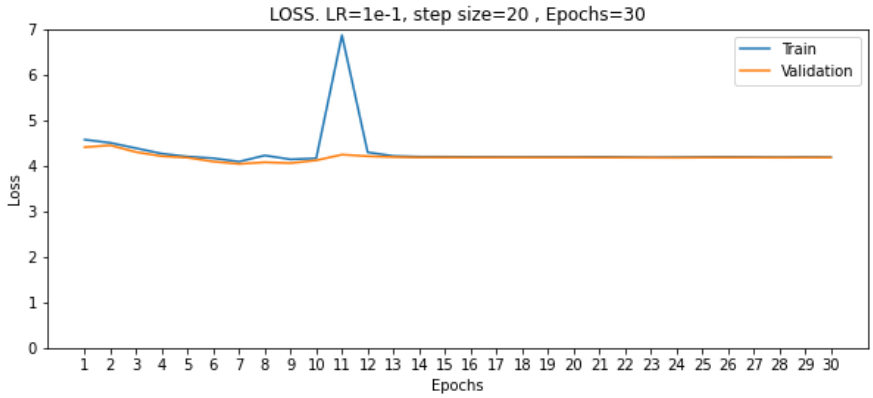




Now, with a bigger learning rate the efficiency of the model improves a lot. The accuracy reached in the validation set is 0.314 and the losses decrease faster.

- LR = 1e-1, STEP_SIZE = 20, NUM_EPOCHS = 30



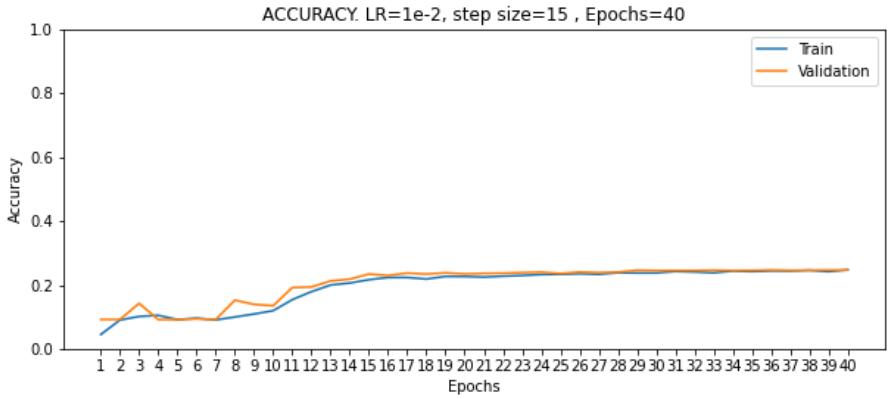


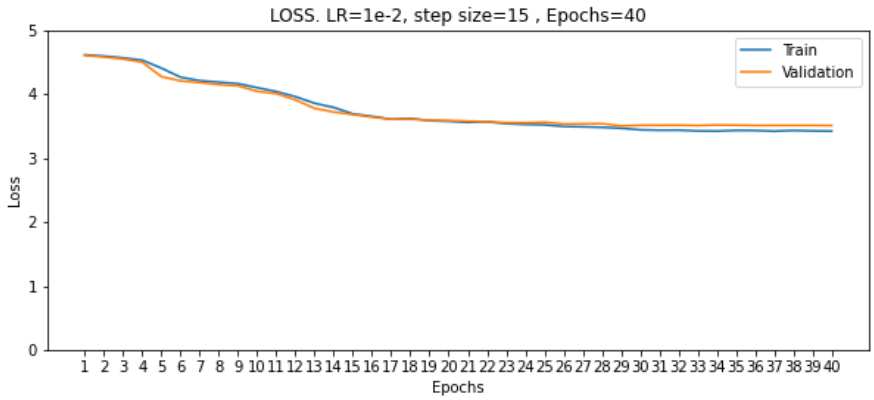
[3]

These performances are very poor: this learning rate is too high.

It's clear that a good value for the learning rate is around 1e-2; and now it could be useful also to change the number of epochs and the step size.

- LR = 1e-2, STEP_SIZE = 15, NUM_EPOCHS = 40
  It's decreased the step size, and increase the total number of training epochs.
  This because of the learning rate will be reduced more often and so the algorithm will be slower, and it will need more epochs to reach a good accuracy, always paying attention to overfitting.

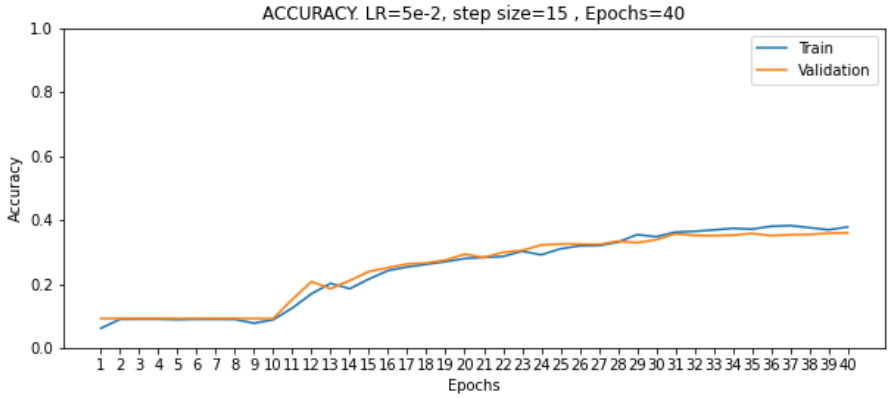


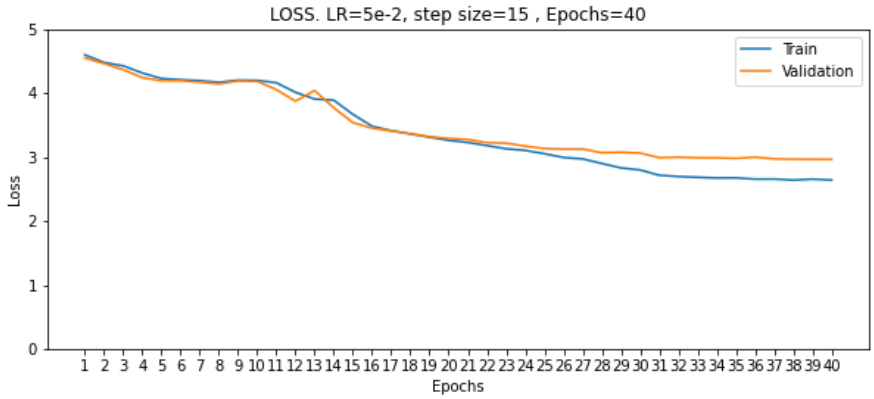


The accuracy reached in the validation set is 0.247, so lower than the one reached with the model *"LR=1e-2, step size = 20, epochs = 30"*; but it has to be noticed that the gap between the two losses is smaller.

[3] Please note that the range in y-axis is different from the others.

- LR = 5e-2, STEP_SIZE = 15, NUM_EPOCHS = 40
  Finally to try to reach faster a good accuracy can be useful to try the previous model with a bigger learning rate.





With this configuration the accuracy on the validation set is the highest: 0.360.

Accuracy on validation set summary:

Configuration	Accuracy
LR = 1e-3, STEP_SIZE = 20, NUM_EPOCHS = 30	0.092
LR = 1e-2, STEP_SIZE = 20, NUM_EPOCHS = 30	0.314
LR = 1e-1, STEP_SIZE = 20, NUM_EPOCHS = 30	0.08
LR = 1e-2, STEP_SIZE = 15, NUM_EPOCHS = 40	0.247
LR = 5e-2, STEP_SIZE = 15, NUM_EPOCHS = 40	0.36

In the testing phase is used the model *"LR = 5e-2, STEP_SIZE = 15, NUM_EPOCHS = 40"*, since it gives the smallest loss and the biggest accuracy on the validation set.

Using the network trained on the training set the accuracy on the test set is 0.351; instead if the network is trained on the entire training set (training set and validation set) the accuracy on the test set is 0.531.

TRANSFER LEARNING

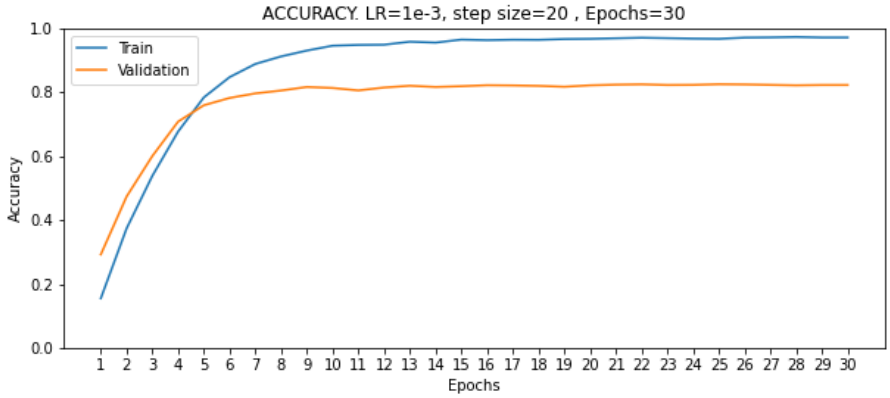
The results obtained by the training from scratch are not very satisfactory, and this was predictable since in order to build an efficient Neural Network, namely find the right weights, a very large dataset is needed: the images available now to train are not sufficient. Furthermore neither a k-fold cross validation can be used (an approach usually adopted in machine learning to tune the hyperparameters in the case in which there are not enough data to train the model) since it's required a lot of time to complete trainings.

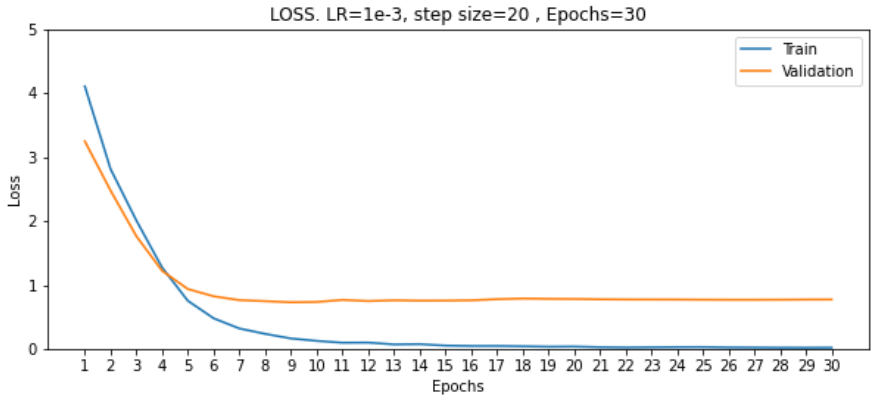
The best solution so it's to exploit transfer learning, and then apply finetuning.

The *"Torchvision"* package provides pre-trained models which can be easily used by passing the parameter *"pretrained=True"* when the network is constructed; in this way are loaded the weights learned by training the network on *"ImageNet"*.[4]

Also in this case, first the attention is on finding a good learning rate.

- LR = 1e-3, STEP_SIZE = 20, NUM_EPOCHS = 30

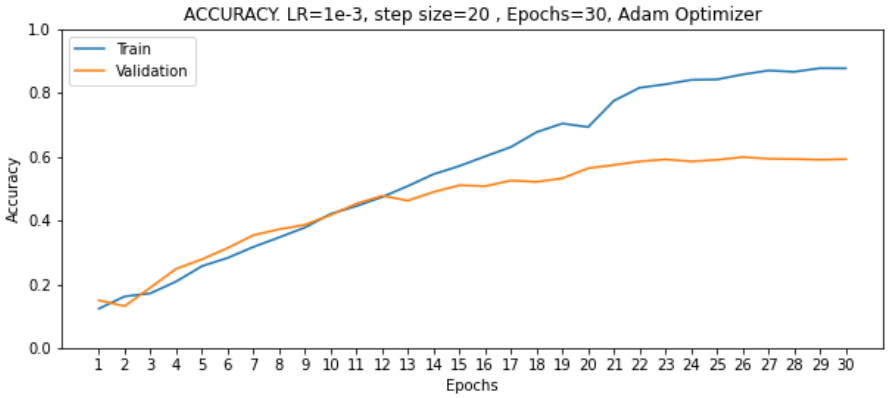


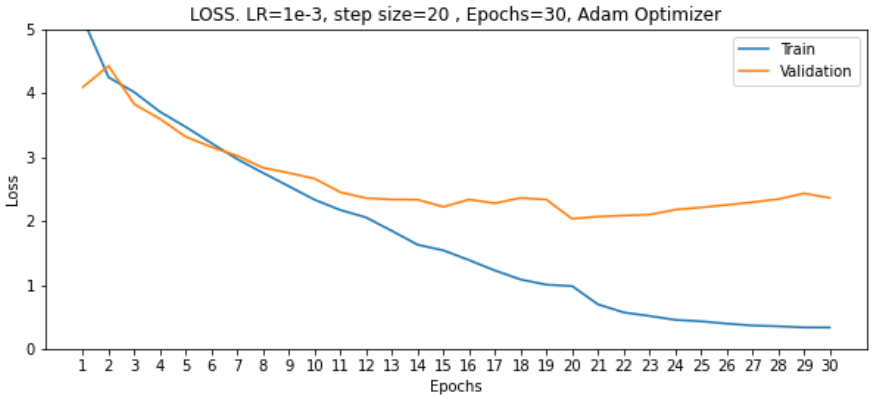


It's possible to see already now, how the performances significantly improve respect to the models trained from scratch: the accuracy reach 80% after few iterations and the loss decreases faster reaching smaller values. Particularly, in this case the accuracy on validation set 0.822.

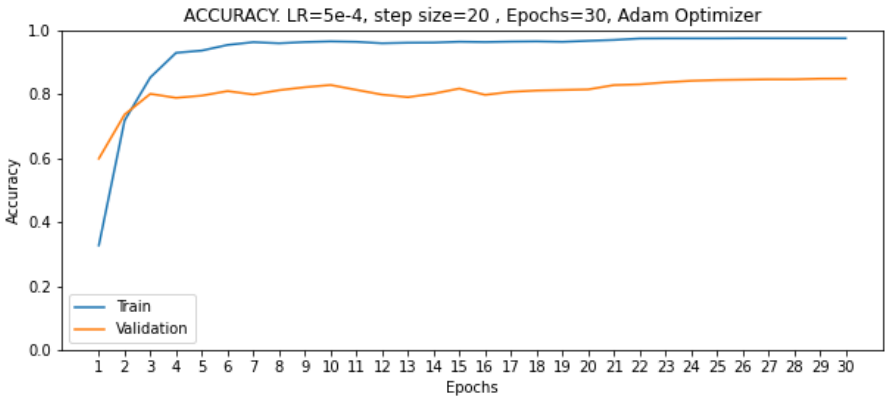
It could be interesting to observe what happens if it's changed the optimizer: instead of the Stochastic Gradient Descent, in the following model is used Adam, another method for stochastic optimization.

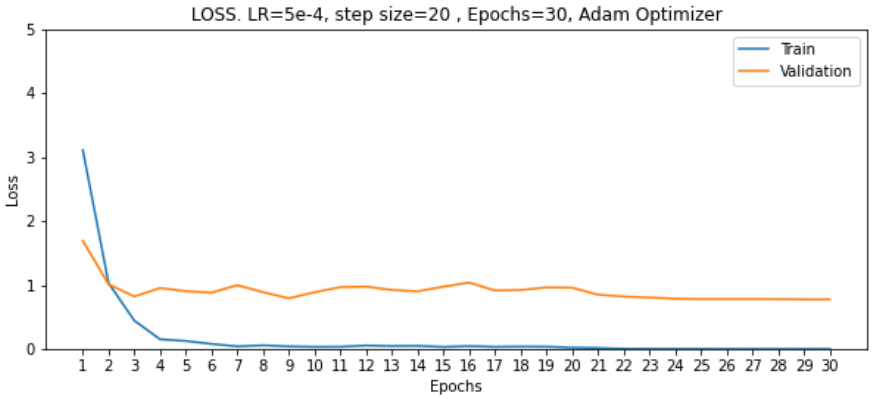
[4] Consequently the normalization has to be done using ImageNet's mean and standard deviation





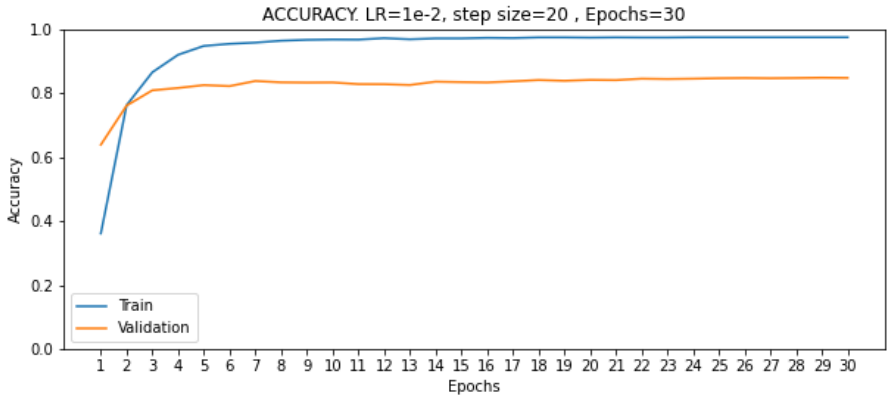
The performances are poor, the gaps are very large: these are clear signs of overfitting.
If the learning rate is decreased the results improve:

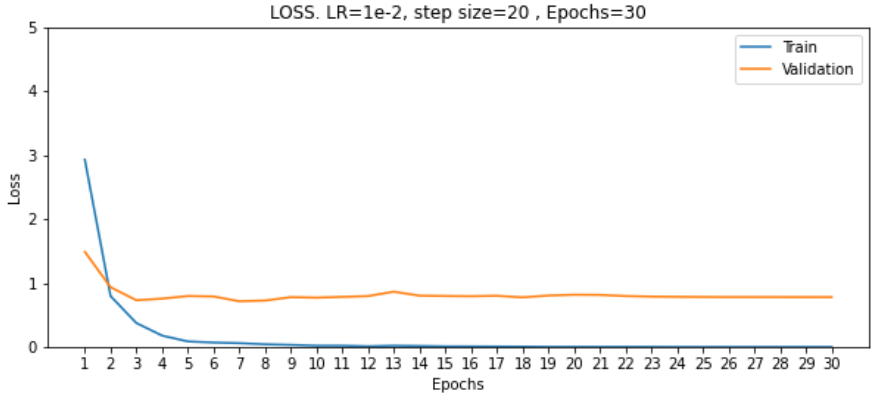




It’s possible to see how the accuracy is very good: 0.845, also the behaviour of the losses is improved.

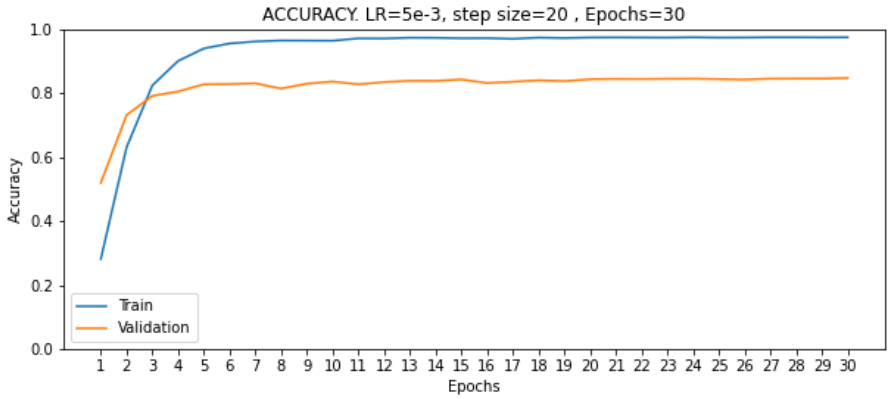
- LR = 1e-2, STEP_SIZE = 20, NUM_EPOCHS = 30

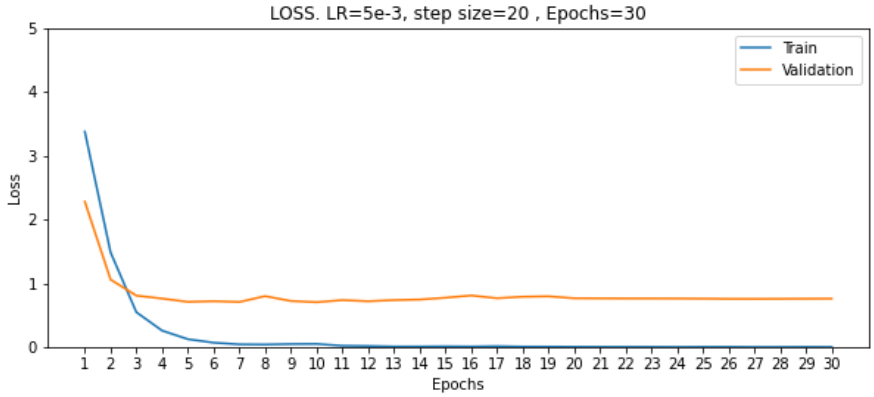




With this configuration the accuracy reached on the validation set is 0.847.

- LR = 5e-3, STEP_SIZE = 20, NUM_EPOCHS = 30

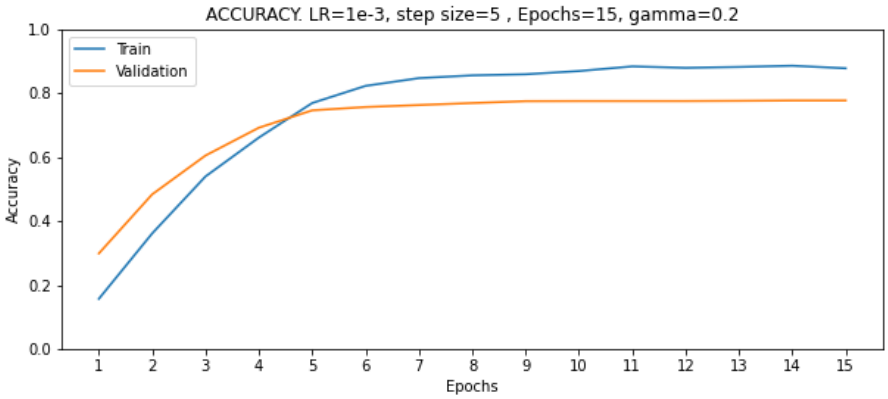


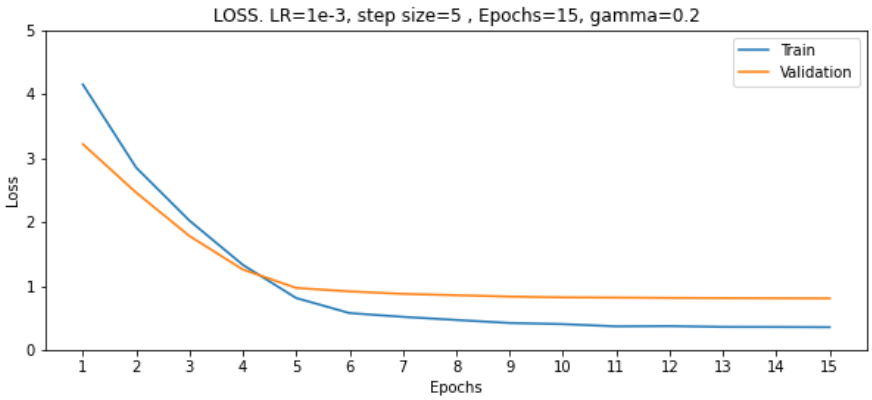


Also with this configuration the accuracy on the validation set is 0.847.

In all the cases it's possible to observed that after few iteration good values of accuracy and loss are reached, and this is due to the fact that the weights are initialized according to the ones of ImageNet. For this reason it could be useful to try to improve the performances using small learning rate and decrease it in a little number of epochs using a bigger *"gamma"* (the multiplicative factor for learning rate step-down) in order to avoid to do too strong changes that could affect the stability of the model.

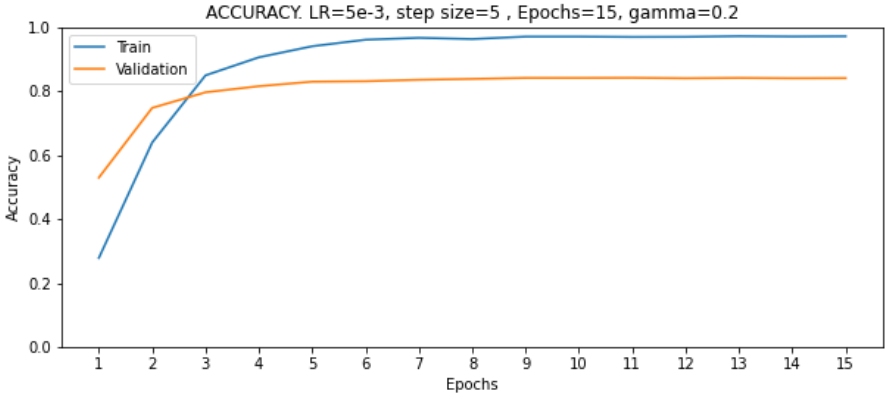
- LR = 1e-3, STEP_SIZE = 5, NUM_EPOCHS = 15, gamma = 0.2

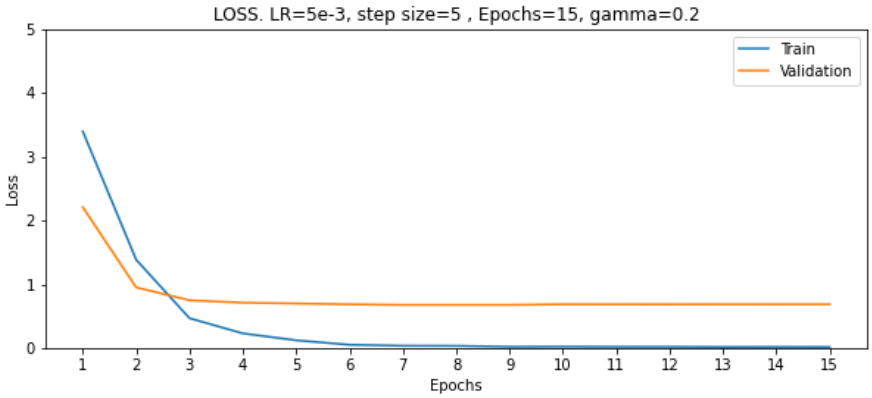




The accuracy on the validation is 0.776.: the model is too slow, and so could be useful to start using a bigger learning rate

- LR = 5e-3, STEP_SIZE = 5, NUM_EPOCHS = 15, gamma = 0.2





These results are satisfactory: it's reached a good accuracy with few iterations: 0.84, and also the loss plot does not highlight sign of overfitting, furthermore the loss in this case is the smallest reached.

Accuracy on validation set summary:

Configuration	Accuracy
LR = 1e-3, STEP_SIZE = 20, NUM_EPOCHS = 30	0.822
LR = 5e-3, STEP_SIZE = 20, NUM_EPOCHS = 30	0.847
LR = 1e-2, STEP_SIZE = 20, NUM_EPOCHS = 30	0.847
LR = 5e-4, STEP_SIZE = 20, NUM_EPOCHS = 30, Adam optimizer	0.845
LR = 1e-3, STEP_SIZE = 5, NUM_EPOCHS = 15, gamma = 0.2	0.776
LR = 5e-3, STEP_SIZE = 5, NUM_EPOCHS = 15, gamma = 0.2	0.84

So, according to what observed, the final model used for the testing phase is "*LR = 5e-3, STEP_SIZE = 5, NUM_EPOCHS = 15, gamma=0.2"*.

Using the network trained on the training set the accuracy on the test set is 0.852; instead if the network is trained on the entire training set (training set and validation set) the accuracy on the test set is 0.879. Contrarily to what was observed in the testing of the model training from scratch, here there is not a bigger improvement in the accuracy if the model is trained on the entire training set. This could be due to the fact that the initial weights have already good values and so it's not so relevant to add more observation in the training sample.

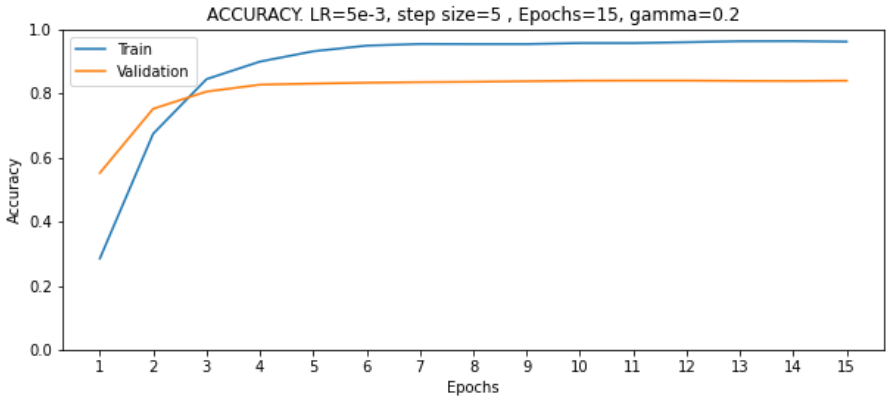
Furthermore to prevent overfitting and speed up the training phase, it could be useful to "freeze" some layers of the network and train only a part of it.

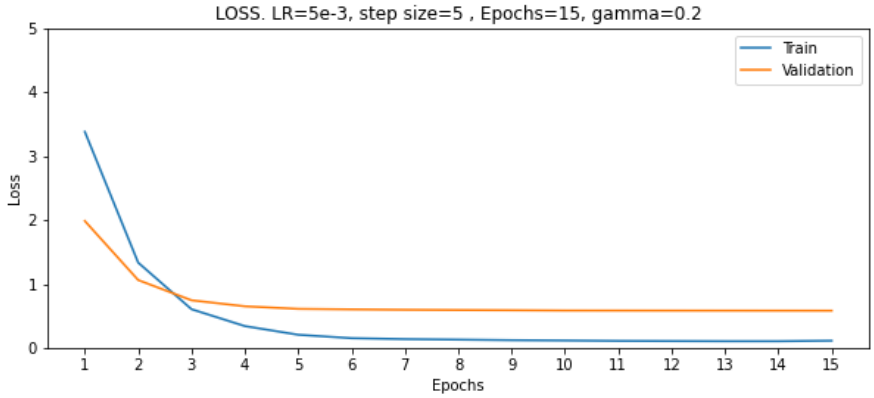
To do that it's enough to pass to the optimizer only the parameters of the fully connected layers or only the parameters of the Convolutional layers.

Remembering that the first Convolutional layers capture more general features, instead the higher layers extract specific features, characteristic of the particular observation.

So, said that, it's possible to consider that could be a good idea to "freeze" the Convolutional layers, since the general features are more or less the same among different, but similar, datasets.

- “FREEZE” CONVOLUTIONAL LAYERS

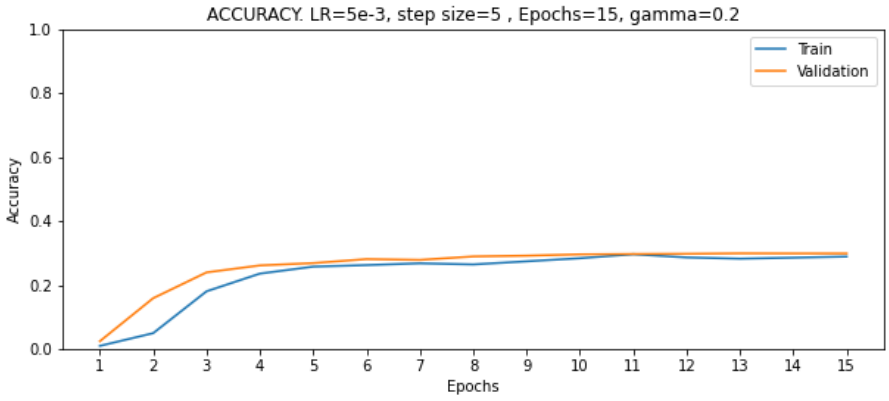


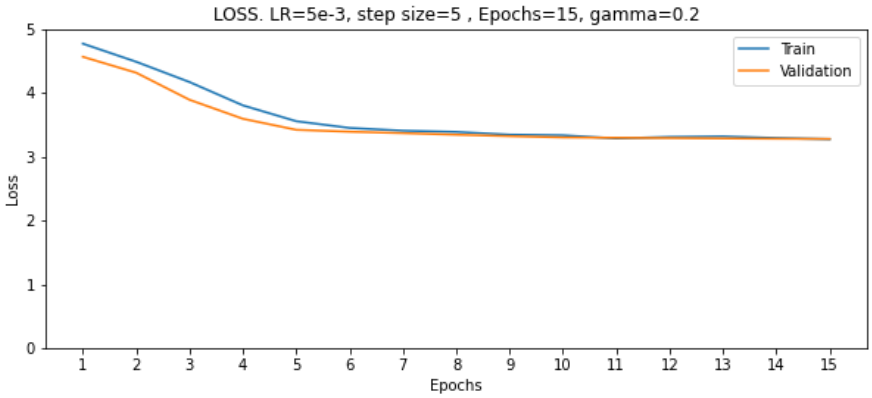


The accuracy on the validation is 0.839, so slightly lower than the one reached without freezing any layers; but the loss on the validation set is better; so as said before there is an improvement respect to the overfitting.

Using the network trained on the training set the accuracy on the test set is 0.832; instead if the network is trained on the entire training set (training set and validation set) the accuracy on the test set is 0.872.

- “FREEZE” FULLY CONNECTED LAYERS





So, as hypothesized, in this case the performances are very poor; and the reason is simple: here it's trying to changed the weights related to the general features, that have been already tune in a better way using a bigger dataset; and the weights related to specific features which depends on the specific dataset, are not changed.

Accuracy on validation set summary:

Configuration	**Accuracy**
without ferezing	0.84
freeze convolutional layers	0.839
freeze fully connected layers	0.23

DATA AUGMENTATION

Finally, since Deep Learning needs an huge amount of data to properly train the model, data augmentation is a good practice to increase the dataset size avoiding overfitting.

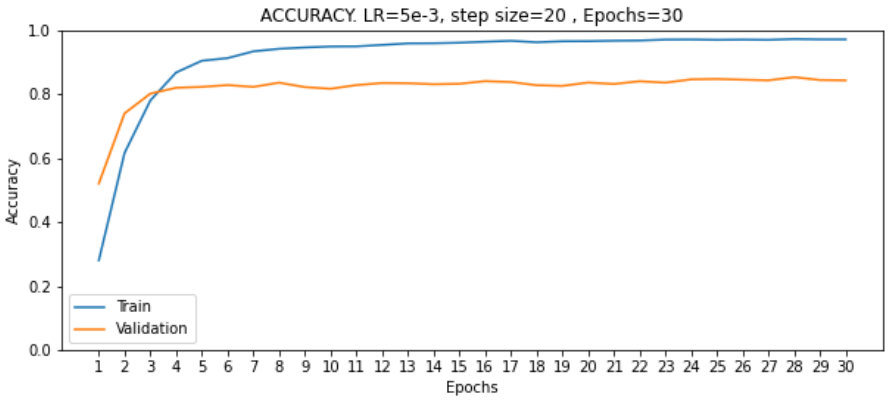
At first, if it's possible, it's a good practice to look at the images inside the training dataset and the images inside the test dataset, in order to identify which transformation can be useful in data augmentation: if training and test images are very similar, strong transformations are useless, furthermore they may worsen the accuracy.

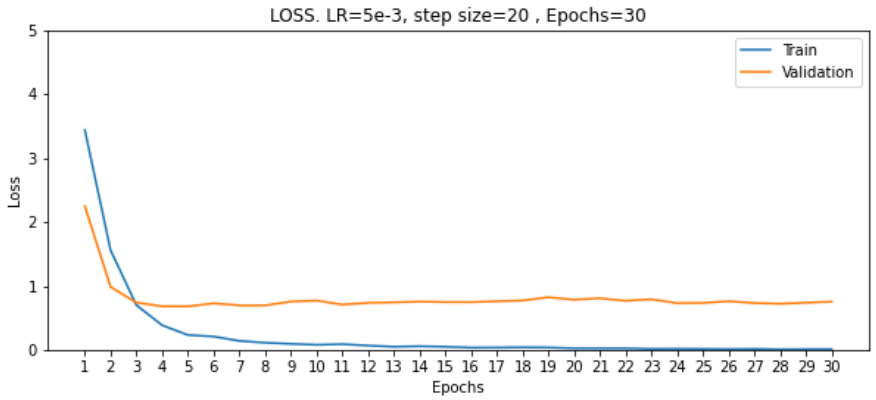
It's important to note that, by construction, data augmentation is automatically applied also to the validation dataset; so the validation dataset is subject to the randomness due to data augmentation, for this reason the accuracy on the validation set is calculated multiple time in order to obtain the mean and the standard deviation.

Since using data augmentation it's possible to have a bigger (in terms of cardinality) training set, the number of epochs used is 30 (instead of 15 which was the parameter of the best configuration found in the previous step).

- COLOR JITTERING & RANDOM CROP

  In "Caltech – 101" almost all the images are not flipped, but there some images black and white, other brighter or darker. So for this reason the first transformation applied is *"color jittering"*, and also *"random crop"* is used.

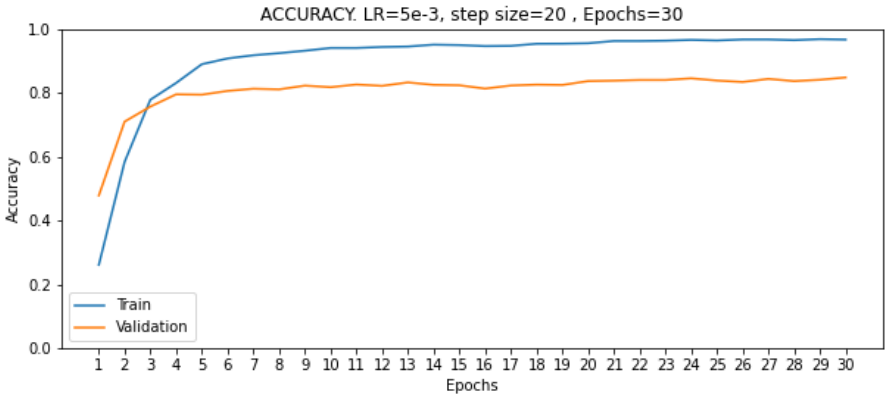


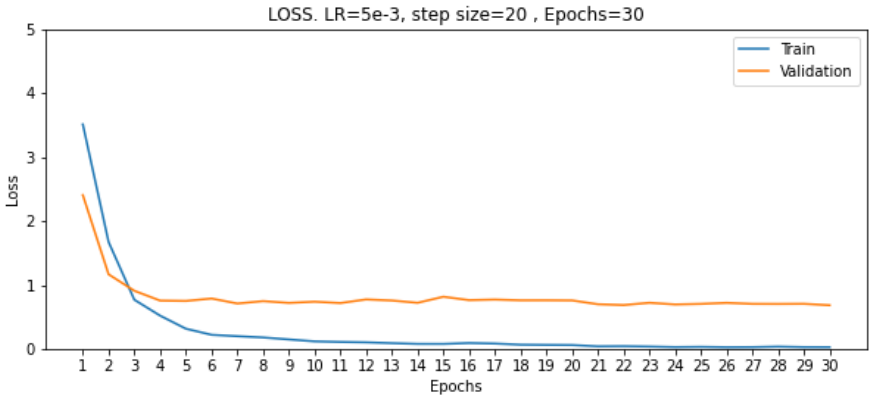


The accuracy is 0.843 ± 0.004.

- COLOR JITTERING (ONLY SATURATION) & RANDOM CROP

  It's possible to set that only the saturation can be changed.

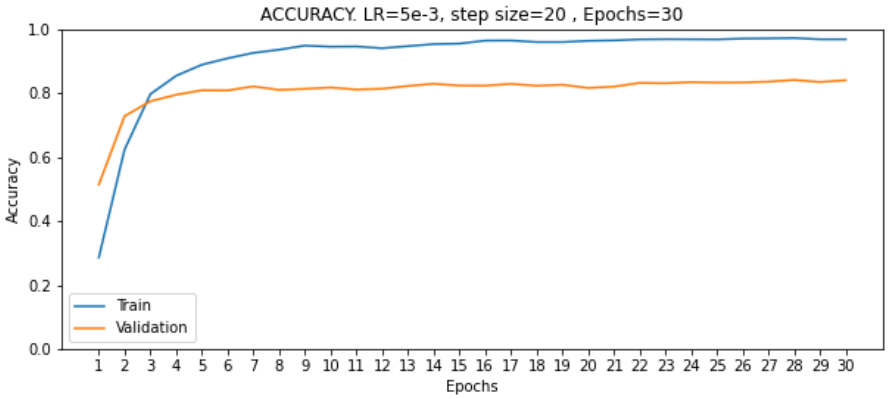


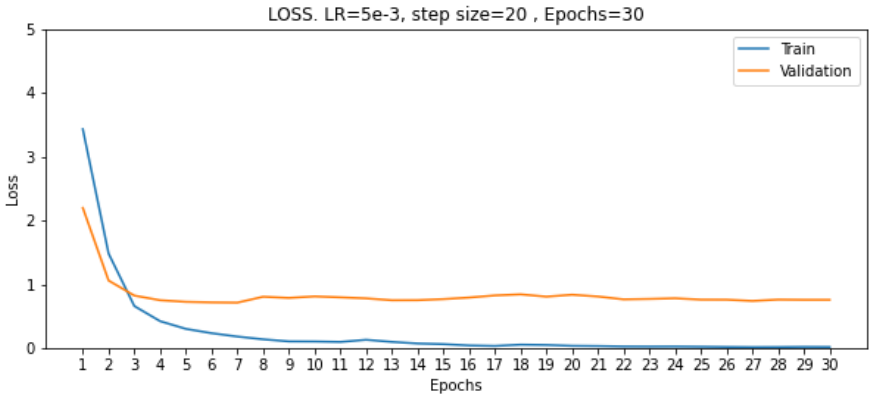


In this case the accuracy is 0.844 ± 0.004, and also the loss behaviour is better.

- COLOR JITTERING & RANDOM CROP & RANDOM FLIP

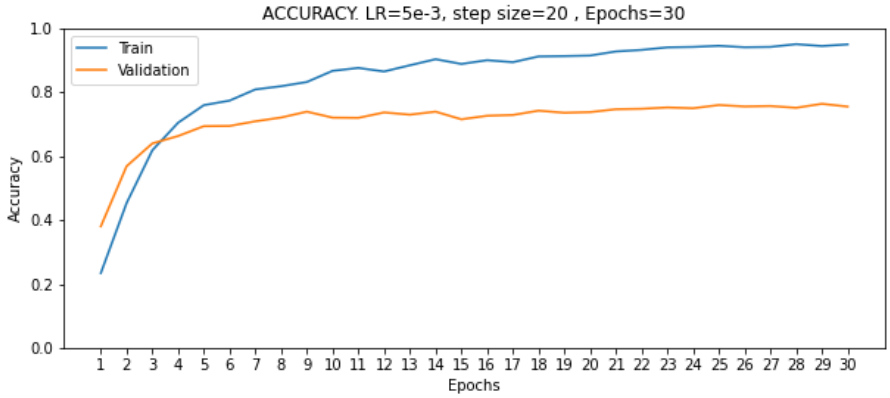
  Even if "Caltech – 101" does not contain many flipped images, it could be interesting to apply *"random flip"* transformation anyway and see what happens.

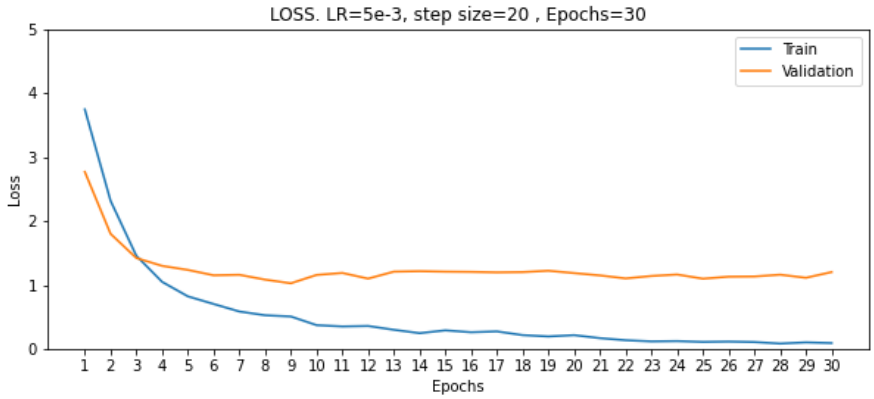




In this case the accuracy is 0.836 ± 0.004, so it's smaller, and the loss is slightly higher than the previous cases.

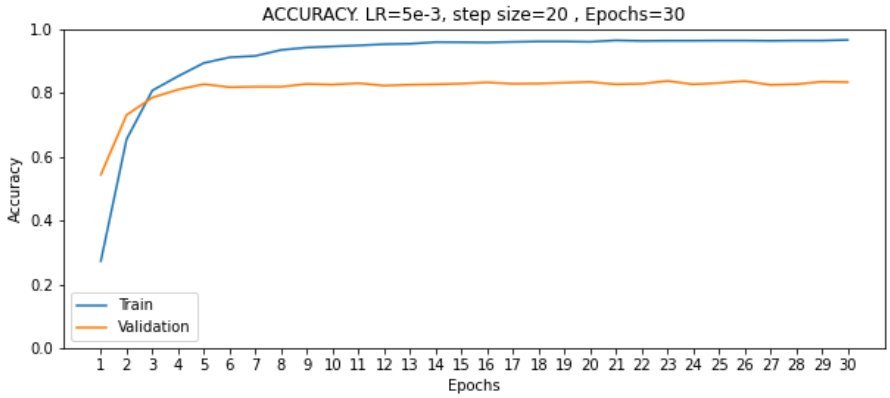
- COLOR JITTERING (ONLY SATURATION) & RANDOM CROP & ROTATION
  Since in *"Caltech – 101"* some images are tilted over by 45° approximately, it could be interesting to see if *"rotation"* transformation works.

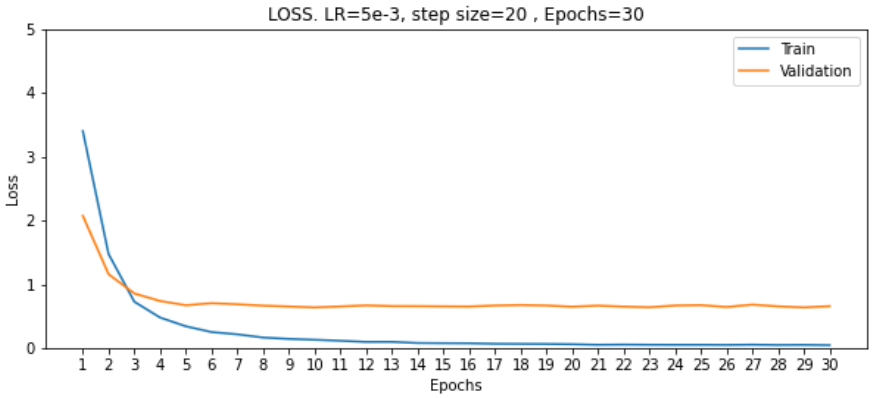




In this case the performances are not very good: the accuracy is 0.757 ± 0.004 and the gap between the losses is enormous. This could be due to the fact that the images are tilted or not only by 45°; so do some rotations of different degrees affects the performances.

- COLOR JITTERING & RANDOM CROP with frozen Convolutional layers
  Finally it's applied data augmentation in the case in which Convolutional layers are frozen.





The accuracy is 0.831 ± 0.003.

Accuracy on validation set summary:

Configuration	**Accuracy**
COLOR JITTERING & RANDOM CROP	0.843 ± 0.004
COLOR JITTERING (ONLY SATURATION) & RANDOM CROP	0.844 ± 0.004
COLOR JITTERING & RANDOM CROP & RANDOM FLIP	0.836 ± 0.004
COLOR JITTERING (ONLY SATURATION) & RANDOM CROP & ROTATION	0.757 ± 0.004
COLOR JITTERING & RANDOM CROP with frozen Convolutional layers	0.831 ± 0.003

The second model *"COLOR JITTERING (ONLY SATURATION) & RANDOM CROP"*, looking at the accuracy and the loss trends, seems to be the best one.

Using the network trained on the training set the accuracy on the test set is 0.863; instead if the network is trained on the entire training set (training set and validation set) the accuracy on the test set is 0.888.

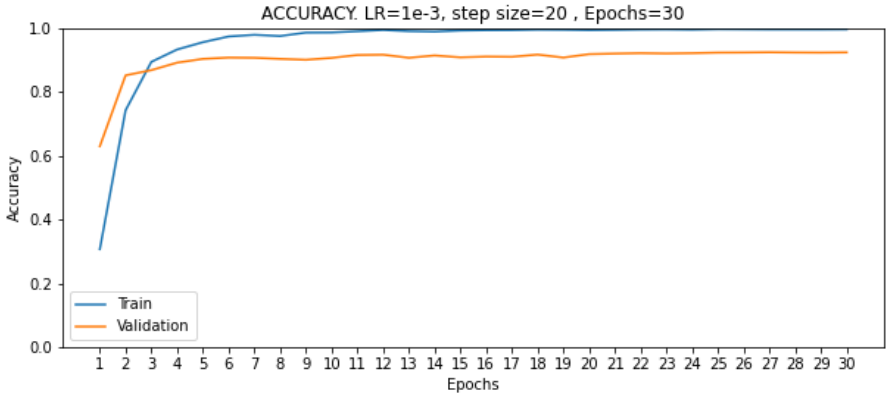
BEYOND ALEXNET

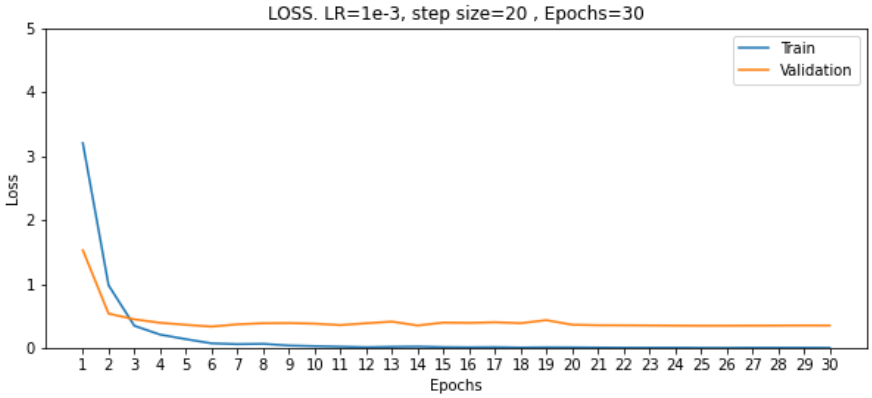
All the processed done before can be applied also to different Convolutional Neural Network.

VGG – 19

The last network used is VGG – 19, it has the same architecture of AlexNet, but thanks to the bigger number of parameters and layers (19), it allows to achieve better result.

The important thing that has to be taken into account while this network is used it's that since VGG has an high number of parameters there could be the risk of overfitting, so it's not advisable to use it when the dataset is small.



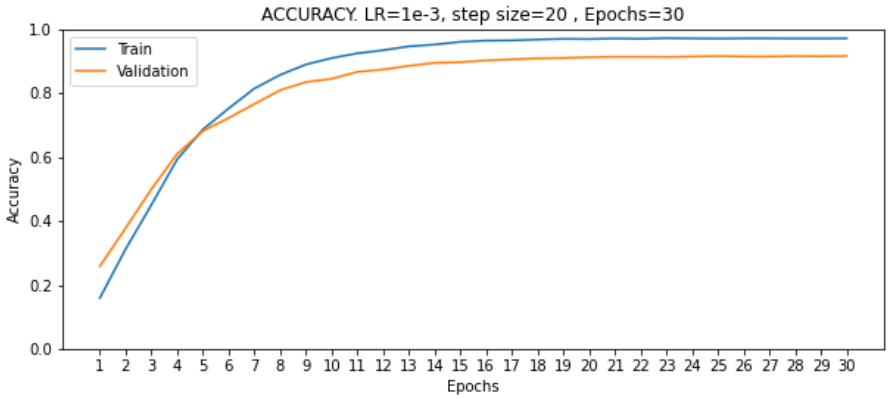


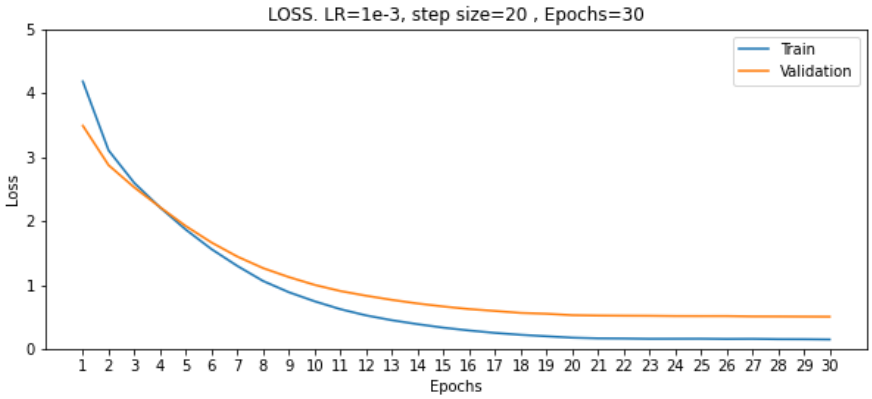
Using the network trained on the training set the accuracy on the test set is 0.922; instead if the network is trained on the entire training set (training set and validation set) the accuracy on the test set is 0.936.

RESNET - 34

In VGG when more layers are added, in general this happens in all the plain networks, the accuracy improves, but this is true until a point: in fact from 34-layer plain network exhibit strong vanishing gradient problem.

RESNET solves this problems thanks to skip connections, that allow the loss to go faster to 0, and also the accuracy is very high.





These are clearly the best graphs obtained until now: the accuracy increases with regular course, and also the loss on validation decrease uniformly, and the gap between train and validation loss is very thin.

Using the network trained on the training set the accuracy on the test set is 0.913; instead if the network is trained on the entire training set (training set and validation set) the accuracy on the test set is 0.948.