POLITECNICO DI TORINO

MSc in Data Science and Engineering

Machine learning and Deep learning

Homework 2

Professor:
Barbara Caputo

Student:
Paola Privitera

Academic Year 2019-2020

Index

Introduction

The aim of the project is to train a convolutional neural network for image classification by using the AlexNet architecture on the Caltech-101 dataset.

Due to the incompatibility of the original dataset with the network, we set up a data preparation phase, through which we defined a set of transformation functions to adjust and adapt the images, to ultimately create a suitable dataset for the neural network.

Once the database has been set, we trained the network "*from scratch*" and we tuned it on different sets of hyperparameters. As neural networks usually require large datasets to be both efficient and effective, we deployed a *transfer learning* technique, which consists in using a pre-trained network (on a larger dataset of similar pictures, in this case the ImageNet dataset) for the classification. This practice allows to take advantage of the classification abilities entailed in the pre-trained network and *transfer* them on a new dataset, avoiding poor model training due to the lack of data, and in turn unsatisfactorily classification performance.

Nevertheless, basing on the idea that a larger amount of data might lead to better model performances, it is possible to increase the size of the dataset by using different approaches, especially when new fresh data are not available. Therefore, we created additional pictures by applying several transformations to the input images (still preserving the likelihood of belonging to the original distribution) and thus, enriching the original dataset with several variations. In this way, the network can rely on a greater data pool without necessarily add brand new images.

To conclude, we executed once again the classification algorithms, but this time using a deeper convolutional neural network (VGGNet), to check whether a more stratified network (higher number of layers) leads to better performances with respect to the AlexNet.

1 Network

The AlexNet architecture consists of eight layers: the first five are convolutional layers and the other three are fully connected layers, where the last layer provides a 1000-dimensional vector as output. Finally, the AlexNet uses the ReLU (Rectified Linear Unit) as activation function and Max Pooling to down sample.

2 Dataset

The Caltech-101 dataset consists of 9,146 pictures of objects belonging to 101 categories. Each category can have from 40 to 800 images, but most categories have about 50 images. The size of each picture is roughly 300 x 200 pixels. The subjects of the images vary from vehicles, animals to common domestic objects.

3 Data preparation

In order to load the dataset, we defined an ad-hoc dataset class Caltech (which is a subclass of VisionDataset) by enriching the skeleton code on GitHub repository and taking the ImageFolder class of PyTorch as reference.

We customized Caltech functions to partition the dataset into train and test sets, according to the two text files, respectively “train.txt” and “test.txt”, contained in the GitHub repository. Moreover, we ensured to discard the additional category “BACKGROUND_Google” which is incorrectly provided in the splits.
By doing so, we created two Caltech objects containing 5,782 images for the training set and 2,893 for the testing one.

4 Training from scratch

The common practice suggests that the training set should be partitioned into train and validation sets. Despite the K-Fold algorithm is the preferred one for such purpose, it is not highly recommended in deep learning, due to the inefficient execution time during the training phase. To solve the problem, we split the training set into two equal subsets, one representing the actual train set and the other the validation one, keeping the same number of images (2,892) per category in each set.

Before starting the training phase, we adapted the AlexNet architecture to our problem, by replacing the last layer, which typically has 1000 output neurons, with a new fully connected layer with 101 outputs, corresponding to the Caltech-101 categories.

To estimate the difference between the prediction of the labels from the ground truth, we used the Cross Entropy Loss which is defined as the negative logarithm of the softmax function; furthermore, we used the Stochastic Gradient Descent (SGD) with momentum as optimization technique to update the weights based on loss.
Finally, we defined the scheduler which is responsible for the decay of the learning rate after a given number of epochs.

At each epoch, the model is trained and parallelly evaluated on the validation set (loss and accuracy are computed) and the best performing model is chosen whenever the loss on the validation set is minimized.

The procedure is repeated several times, by changing the initial hyperparameters. The first set is the default one, the others are customized hyperparameter sets.

The learning rate (LR) is the most important hyperparameter for neural network tuning because it controls how quickly the model adapts to the underlying database, and thus, improving the classification ability. In particular, the learning rate scales the weight updates magnitude in order to minimize the loss function. If the learning rates are small, the training phase would be very slow due to the tiny updates of the weights (requiring more training epochs) whereas larger learning rates result in rapid changes with less epochs.
The selection of this value is tricky:

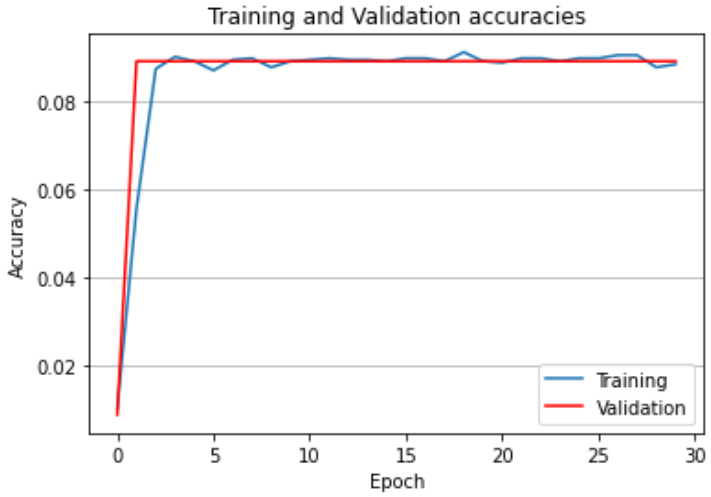
- if the learning rate is too low, it could cause the process to get stuck,
- if the learning rate is too high, it could cause drastic updates that would lead to divergent behaviors.

For each set of hyperparameters, two graphs are proposed. The first one showing the loss variation at each epoch, and the second one the corresponding accuracy. Alongside, the resulting values of Loss, Validation accuracy and Test accuracy at the best epoch are summarized at the bottom.

As mentioned, the first set of default hyperparameters is:

LR	= 0.001
EPOCHS	= 30
γ*	= 0.1

* *this represents the multiplicative factor for learning rate step-down*

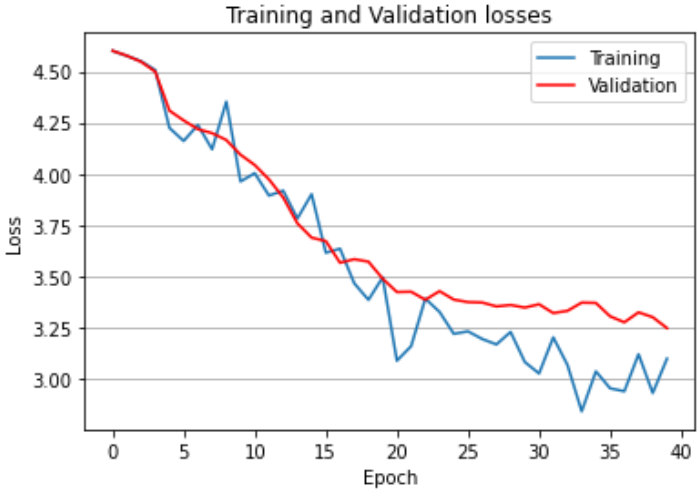


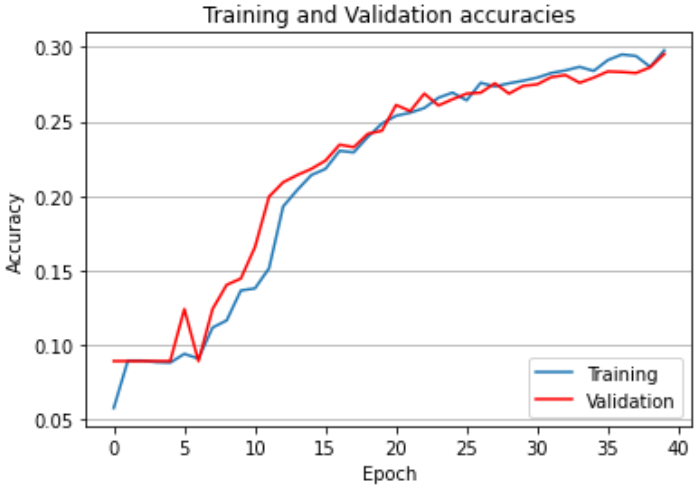

```
The best epoch is: 30
Loss = 4.549631118774414
Val Accuracy = 0.08921161825726141
Test Accuracy = 0.09194607673695127
```

The results of the first model run with default hyperparameters are disappointing, as proven by an extremely low accuracy on the test set equal to approximately 9%. The overall high loss value (though it decreases, but at a very shallow rate), across all epochs, suggests that the model is not able to "learn" from the training set (underfitting). This means that, either the learning rate is too small, or the number of epochs is insufficient. Since the LR is the leading driver, in order to improve the result, we decided to increase the number of epochs, but more importantly slightly increase the learning rate, keeping γ constant.

The second set of hyperparameters is the following:

LR	= 0.01
EPOCHS	= 40
γ	= 0.1



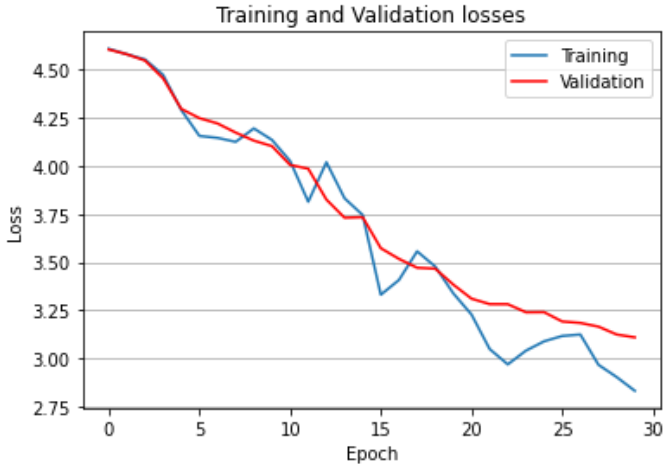


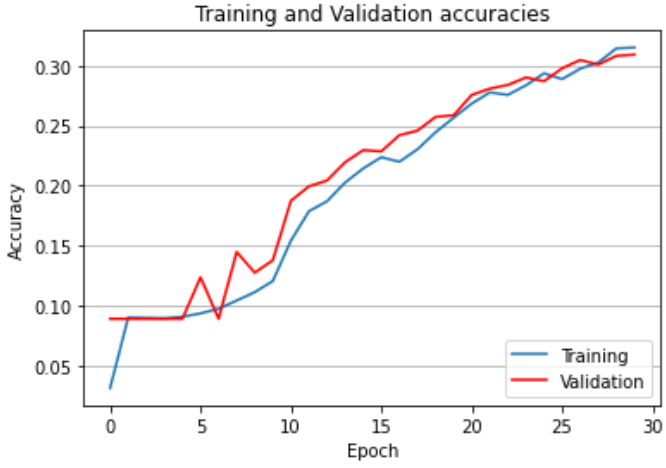
```
The best epoch is: 40
Loss = 3.2478489875793457
Val Accuracy = 0.29529737206085754
Test Accuracy = 0.2945039751123401
```

As one might expect the loss decreased at a faster rate with respect to the previous model. Coherently, the accuracy (both on validation and test) improves to roughly 29%. Despite this, by looking at the first graph, the model is subjected to a small data overfitting (phenomenon which would be considerably remarkable considering a higher number of epochs); therefore, in order to avoid it, we decided to decrease once again the number of epochs and select a more stringent decaying factor (γ).

The new hyperparameters set configuration is:

LR	= 0.01
EPOCHS	= 30
γ	= 0.2



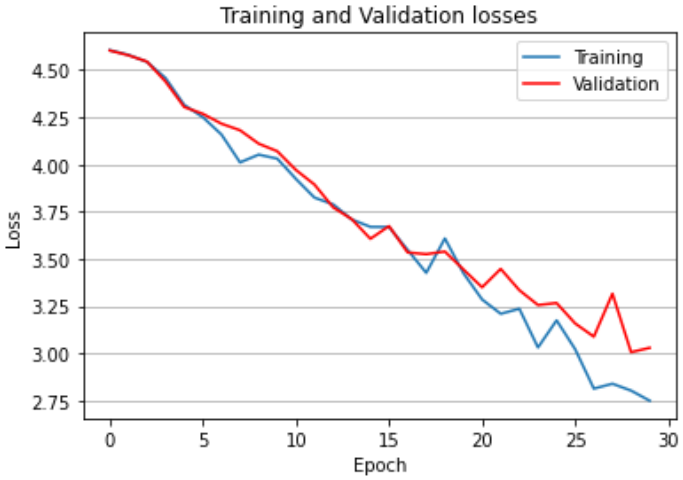


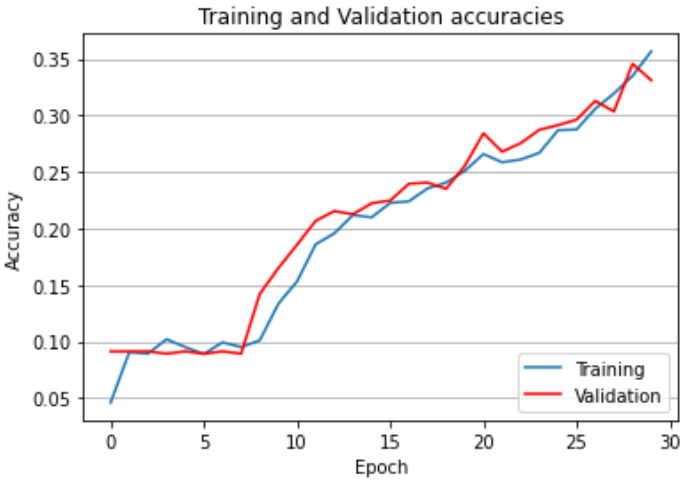
```
The best epoch is: 30
Loss = 3.110236167907715
Val Accuracy = 0.3091286307053942
Test Accuracy = 0.32423090217767025
```

In this case, the model delivered a small improvement in loss terms, as well as in accuracy (~31% on validation and ~32% on the test set).
For the last run, we decided to furtherly increase the decaying factor to understand the sensitivity of the model to a slower learning rate, after the 20th epoch (we set the step size equal to 20, for all sets of hyperparameters).

The last set of hyperparameters is as follows:

LR	= 0.01
EPOCHS	= 30
γ	= 0.8





```
The best epoch is: 29
Loss = 3.0058562755584717
Val Accuracy = 0.3457814661134163
Test Accuracy = 0.3546491531282406
```

In conclusion, the last model run provided the best overall results, both in terms of loss and accuracy (~35% both on validation and test).

5 Transfer Learning

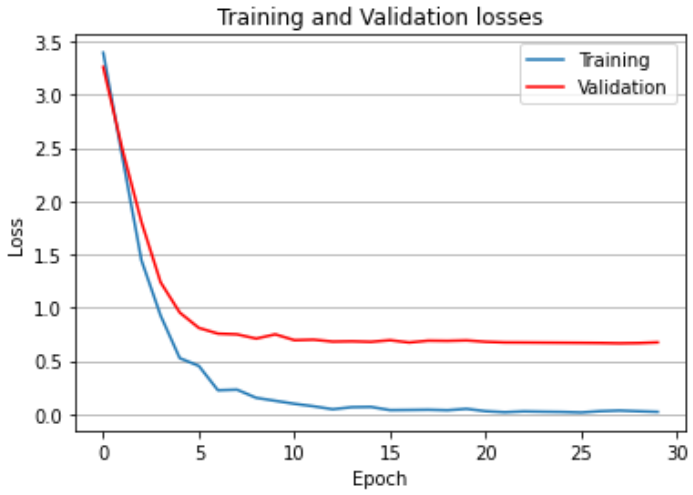
Neural networks typically require enormous amount of data in order to be operationally effective, for example in classification problems. In our case, the training dataset accounts for only 2,892 images, which is a relatively small set compared to the needs of the network. Hence, we decided to deploy a pre-trained AlexNet which exploits the ImageNet dataset, containing more than 20,000 pictures. The use of a similar but larger dataset for model training is the key idea beneath the *transfer learning* concept, for which the model is already endorsed with the properties and abilities (e.g. feature extraction) to correctly classify upcoming data.

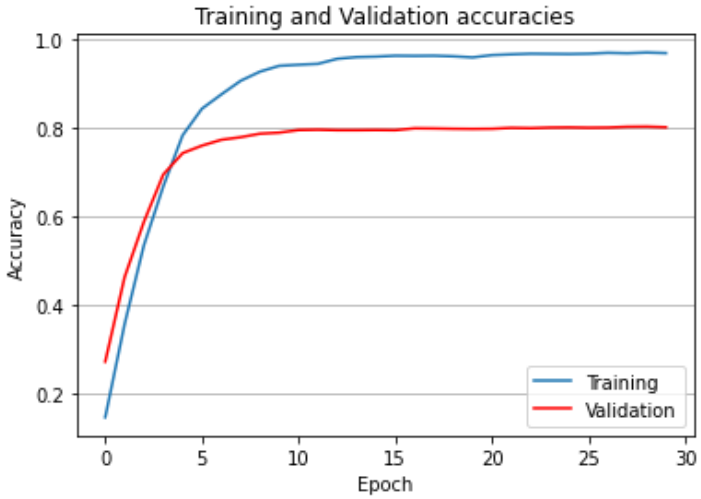
Since the training set comes from a different source with respect to the initial Caltech class we created, a further normalization is needed as if they were originally taken from Caltech dataset. Thus, we normalized the input images with the mean and standard deviation from ImageNet, namely the vectors (0.485, 0.456, 0.406) and (0.229, 0.224, 0.225), where each value refers to red, green and blue attributes in the RGB system.

Analogously to the first analysis, we run the model on three different sets of hyperparameters, and the results are shown as in the previous format.

The first set of values is the default one:

LR	= 0.001
EPOCHS	= 30
γ	= 0.1





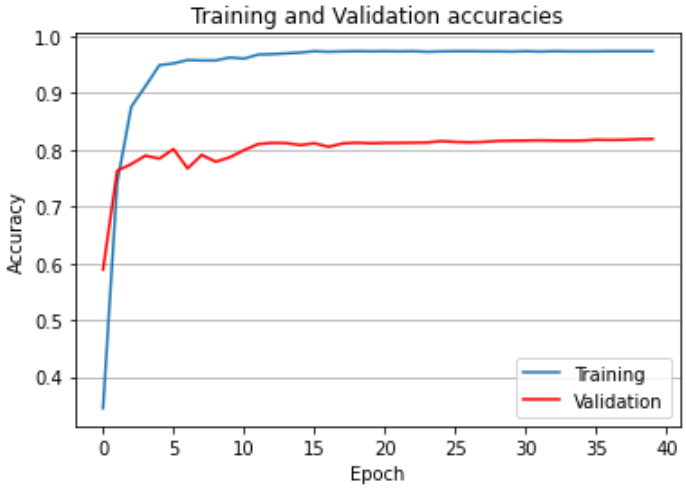

```
The best epoch is: 28
Loss = 0.6647233963012695
Val Accuracy = 0.8032503457814661
Test Accuracy = 0.8240580712063602
```

First off, the pre-trained model delivered a significant improvement both in loss and accuracy, the latter equal to approximately 80% on the validation set and 82% on the test set, compared to roughly 9% (on both sets) in the non-pre-trained neural network. Moreover, the loss is 0.66 versus 4.54 for the same set of hyperparameters.
From the first graph, we can easily notice that the model converges after about 6-7 epochs and, since the gap in between the two losses is not increasing over time, the model is not showing overfitting.

The second set of hyperparameters is as follows:

LR	= 0.01
EPOCHS	= 40
γ	= 0.1

As shown in the next graph, the training loss is virtually tending to zero, while the "best" loss on validation is about 0.65.

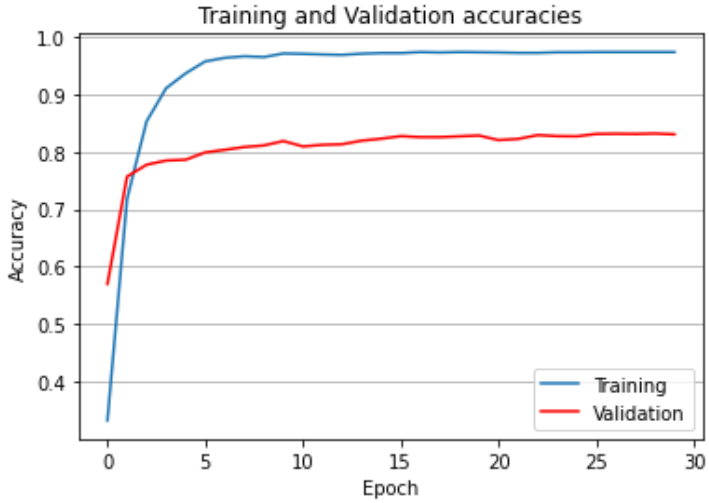



```
The best epoch is: 14
Loss = 0.6571099758148193
Val accuracy = 0.8118948824343015
Test Accuracy = 0.8240580712063602
```

The last hyperparameters set is:

LR	= 0.01
EPOCHS	= 30
γ	= 0.8
STEP SIZE	= 10

In this last round, we changed an additional parameter, STEP_SIZE, which represents the epoch frequency at which the learning rate decay factor is activated.

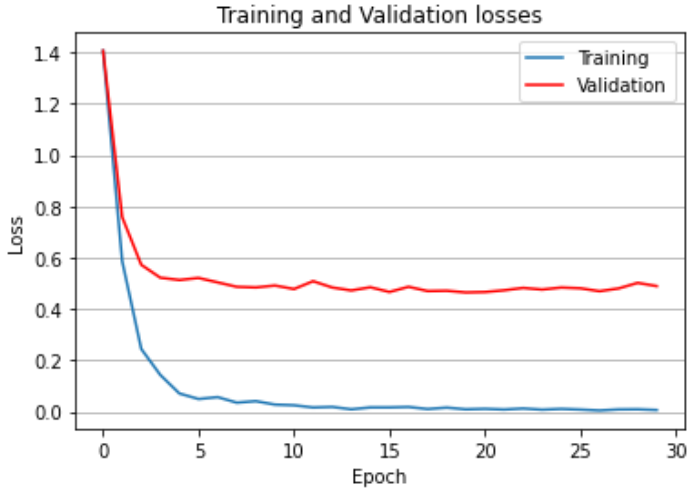


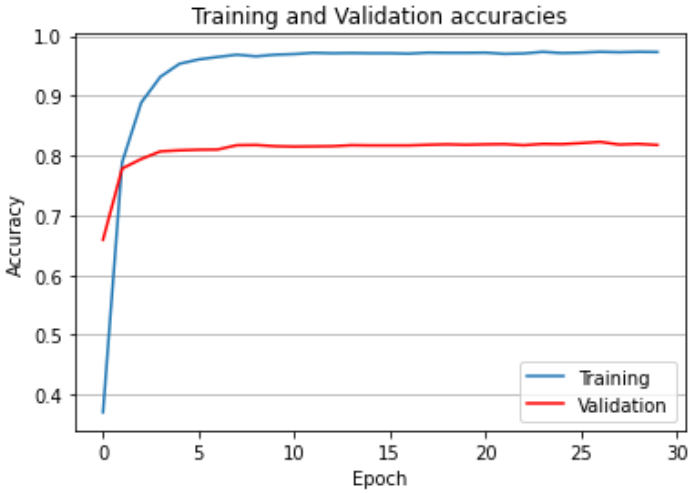

```
The best epoch is: 20
Loss = 0.5237555503845215
Val Accuracy = 0.8284923928077456
Test Accuracy = 0.8444521258209471
```

This configuration provided the best results so far; indeed, the loss decreased to 0.52, whilst the accuracy increased to 83% and 84%, on the validation and test set, respectively.

To address the overfitting issue, it is possible to "freeze" some of the network layers and train those left. This technique is often times not the optimal solution, but in some cases might prevent the model from overfitting. Following this lead, we set up a two-step procedure where in the first run we trained only on the fully connected and froze all the others; in the second step, we froze the fully connected and trained on the convolutional layers.

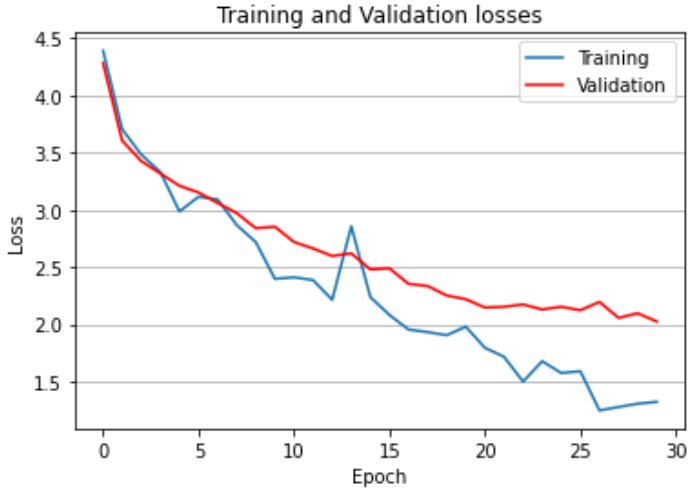
The results are presented sequentially in the following charts.

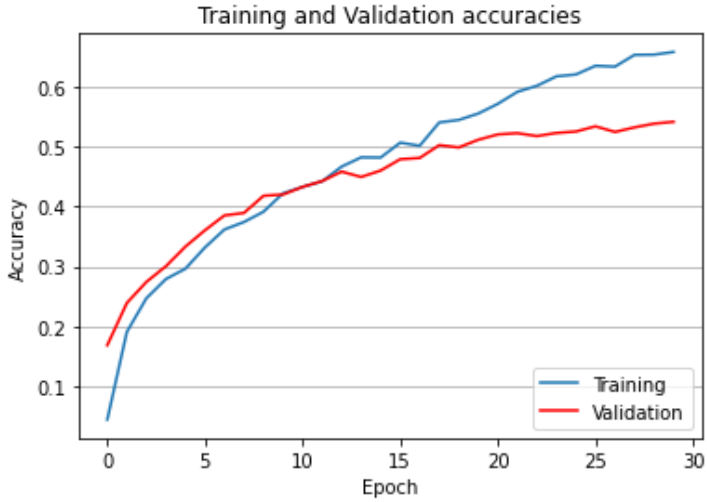





```
The best epoch is: 20

Loss = 0.46448853611946106

Val Accuracy = 0.818118948824343

Test Accuracy = 0.8486000691323885
```

By training only on the fully connected and freezing the convolutional layers, despite an improvement in the loss, a small overfitting is concerning (indeed, the validation loss seems to increase from the 25 epoch on). Obviously, the training phase is faster than in the previous scenarios due to the lower number of layers in which the training is executed.






```
The best epoch is: 30
Loss = 2.0246822834014893
Val Accuracy = 0.5421853388658368
Test Accuracy = 0.5537504320774282
```

While in the first case, we did not identify any worth-noting difference, except for the improvement in computational and execution speed of the training phase, by freezing the fully connected layers (and therefore training on the convolutional layers) the model performance considerably worsens. The loss is equal to 2.02 (vs 0.46) and the accuracies are 54% (vs 82%) and 55% (vs 85%) on validation and test set, respectively.

For the next experiment of data augmentation, we selected the solution that trains exclusively on the fully connected layers as the best configuration.

6 Data Augmentation

Data augmentation consists in augmenting the input data with some "fake" variations. In this way, we can artificially increase the size of the dataset (to obtain better performances on the test). Data augmentation can be carried out by adopting different techniques; for example, we can change the color, the brightness, the contrast of one image or even rotate the initial image by a certain angle degree.

It should be noticed that the dataset has already been subjected to transformations by centrally cropping the image to reshape it to a specific resolution of 224x224, in order to adapt the pictures to the needs of the AlexNet architecture.

To serve purpose of data augmentation, we applied different transformations to the training set:

- *random crop*, which consists in cropping the image in a random position. In this way, the network can recognize the category by looking only at pieces of the object;
- *horizontal flip*, which implies exchanging the left side of the picture with the right one;
- *horizontal flip* plus *color jitter*, where the latter consists in changing the values of brightness, contrast and saturation of the image.

In order to compare the results, both across the different transformation methods, and the models with transformed dataset with the latest one pre-data augmentation, we provide the following summarizing table:

	Val Loss	Val Accuracy (%)	Test Accuracy (%)
No augmentation	0.464	81.81	84.86
Random crop	0.437	81.85	83.93
Horizontal flip	0.453	82.01	84.65
Flip + color jitter	0.422	82.39	84.86

The results showed no tangible improvements in terms of loss and accuracy and this might be for several reasons. The most likely is that the results are strictly dependent of the transformation applied. The chosen transformations might not be suitable for this specific dataset, which implies there could be other transformation methods available that can deliver better (or worse) results. Nonetheless, by doing so, it would require a trial-and-error procedure, which often times is counter-productive.

7 Beyond AlexNet

In conclusion, we trained the convolutional neural network using the VGGNet architecture, which is significantly deeper than the AlexNet. Indeed, the former is characterized by sixteen layers with respect to the eight of the latter. In order to compare the results of the two networks, we tested the VGGNet by using the default set of hyperparameters with transfer learning. In addition, we reduced the batch size (i.e. the number of images that pass through the network at the same step) to 16, to avoid reaching the maximum memory capacity due to the high number of layers of the model. Moreover, since the reduction of the batch size would bring to longer epochs, we reduced the number of epochs to 10, without losing anything in the number of update iterations.

The following table shows the results obtained with the two networks:

	Val Loss	Val Accuracy (%)	Test Accuracy (%)
AlexNet	0.664	80.32	82.41
VGGNet	0.038	90.14	91.01

As expected, the VGGNet performed significantly better with respect to the AlexNet, thanks to the larger network depth.