Machine Learning and Deep Learning

Report - Homework 2

Roberto Franceschi (s276243)

1 Introduction

In this report we are going to perform an image classification task by means of a Convolutional Neural Networks. The goal of this homework is to better inspect all the stages needed to train and fine-tune a CNN that is able to correctly assign a class to previously unseen images.

For this analysis we will use the *Caltech-101 dataset* [1], which consists of 9146 images, split unevenly between 101 distinct object categories and a background category. According to the official website, there are about 40 to 800 images per category, additionally most of the categories have around 50 images in the dataset. Such varying images per category can be a real problem when trying to achieve high accuracy in deep neural network training. Figure 1 shows as an example some images taken from this dataset.

The network architecture we will use is given by *AlexNet* [2], which significantly outperformed the second runner-up at ImageNet ILSVRC challenge in 2012. This network had a very similar architecture to *LeNet*, but was deeper, bigger, and featured Convolutional Layers stacked on top of each other.

Since the dataset is really unbalanced and for some classes we have a very small number of samples we will explore some methods to achieve better performance (in terms of accuracy) on the test set. In section 5 we will implement *Transfer Learning* which consists in the transfer of knowledge from a related task that has already been learned. Furthermore, we will analyze the *Data Augmentation* approach which is a strategy that enables to significantly increase the diversity of data available for training models, without actually collecting new data.

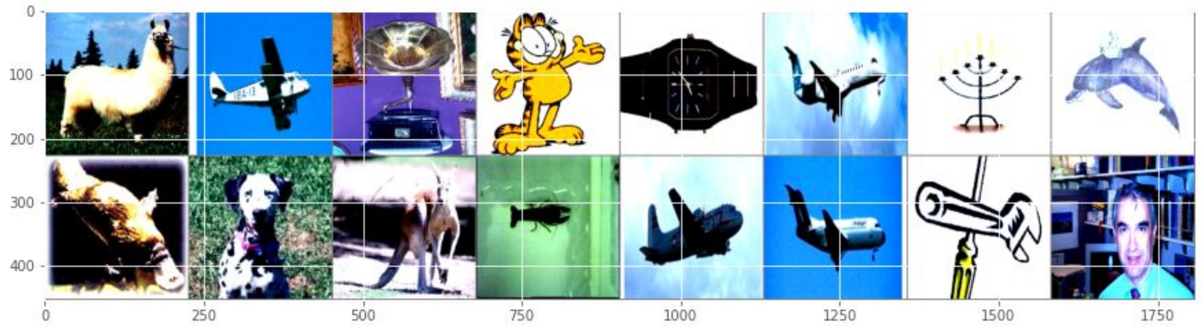


Figure 1: *Sample images from the Caltech-101 dataset*

In the last section, we will explore two of the successors of AlexNet in the ImageNet competition: in particular, we use the *VGGNet* [3], that got the 2nd place in 2014 behind the GoogleNet, and the *ResNet* [4], which was the winner of the 2015 edition.

2 Data preprocessing

As already mentioned above, the dataset is a relatively small for a deep learning task, it contains only 9146 samples, divided in 101 classes, these are not enough images to get very high accuracy. Furthermore, ignoring the `BACKGROUND_Google` label and its images, we obtain a total of 8677 images. Apart from the less number of images, another problem is the distribution of the images per class. To get a better idea about the distribution of the images in each category, the following figure might be useful (see Fig. 2).

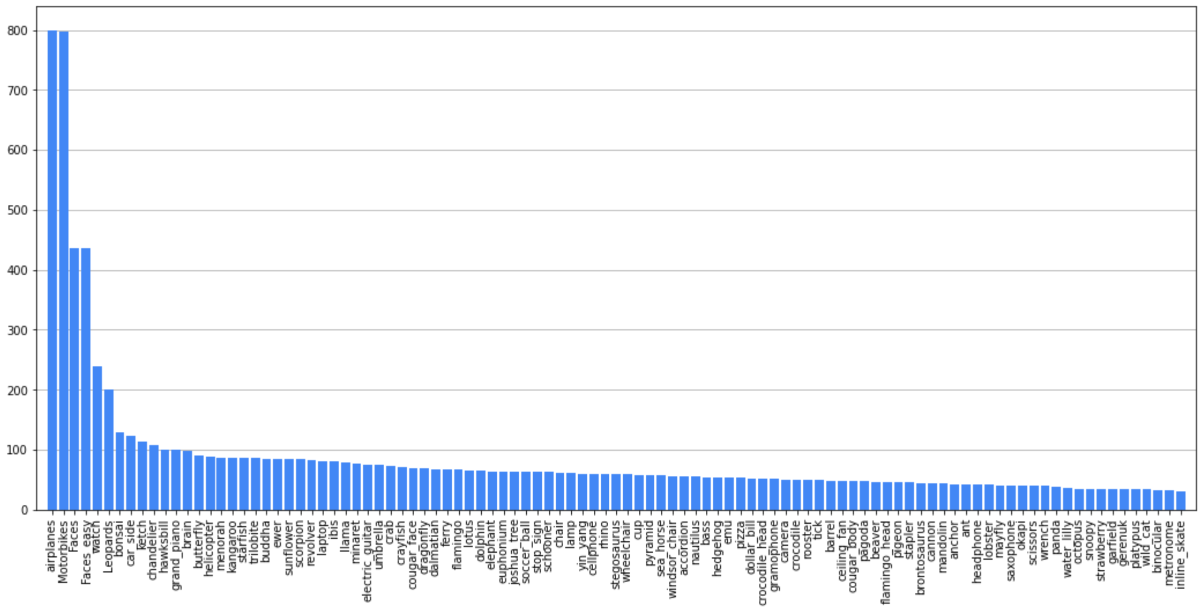


Figure 2: *Distribution of number of images in each category*

We can clearly see that `airplanes` and `Motorbikes` categories have the highest number of images, around 800, and the lowest number of images may be as low as 40 for most of the labels. Such an imbalanced dataset, as we will see soon, is one of the major reasons for the bad performance of the network trained from scratch.

Train, Validation and Test sets

The division between training and test data is provided through two text files in the Github repository [5], called `train.txt` and `test.txt`, that indicate which image files belong to each set. In order to load the dataset, we define a subclass `Caltech` of `VisionDataset` and we customize the initialization function (i.e., `make_dataset`), which provides the logic to retrieve the images as dictated in the text files and to store them in the Caltech object. Furthermore, the initialization function perform the desired transformation on the input images, normalizing them and adapting their size to the input required by AlexNet, which is

224x224 pixels. As a next step, the training set is splitted in half into the actual training samples, that will be used to train the neural network, and the validation set, which serves to evaluate the performance of the network after each epoch. In order to maintain class balancing between the sets the split has been implemented by using `StratifiedShuffleSplit` (`from sklearn.model_selection`). At the end of this phase, we will have the dataset has been split in training, validation and test set, with the same proportion. The following table shows the division between sets.

	Number of samples
Training set	2892
Validation set	2892
Test set	2893

Table 1: *Size for each set*

The code and the implementation details can be found at:
`https://github.com/robertofranceschi/Image-classification-on-Caltech101-using-CNNs`.

Note that the results contained in the following report are not fully deterministic since both the dataset splits and the mini-batches are random. For this reason the result reported in this report are the means computed over three runs.

3 Training from scratch

As already mentioned in the introduction, for this task the AlexNet has been used to perform training from scratch (i.e., without pretrained weights).
AlexNet consists of 8 main layers, divided in two segments: the first five are convolutional layers, some of which are followed by max-pooling layers, while the last ones are fully-connected. The last layer outputs a 1000-dimensional vector, which contains the score of each class. Inside the network a non-saturating *ReLU* activation function and *Dropout* are employed, in order to prevent respectively problem of 'killing' the gradients and co-adaptation in the hidden layers.
Before starting the training, we need to adapt the AlexNet structure to our problem, which considers only 101 categories instead of the default value of 1000. Therefore, we replace the last fully-connected layer in the network with another layer of the same type, but with a 101-dimensional output.

There are multiple useful quantities we should monitor during training of a neural network. The first quantity that has been tracked during training is the *loss*, as it is evaluated on the individual batches during the forward pass, while the second one is the *validation/training accuracy*, that gives a valuable insights into the amount of overfitting in the model. Loss and accuracy plots were used to gain insights into the different hyperparameters settings and to know how to change them for more efficient learning.

Baseline model. At the beginning, we train the network with the default values provided in the skeleton of the code. In particular we use the following parameters 30 epochs basing on the *Cross Entropy* loss function. This function use the softmax to normalize the scores output in the forward pass, and use the negative logarithm in order to make it easy to perform the minimization. The gradients are updated at each step according to the *Stochastic Gradient Descent* (SGD) with momentum (`MOMENTUM=0.9` and `WEIGHT_DECAY=5e-4`), with an initial learning rate of 10^{-3}. The learning rate is subject to a decay by a factor 0.1 (`GAMMA`) every 20 epochs (`STEP_SIZE`). Each step is executed on a batch of 256 images (`BATCH_SIZE = 256`). The following graph shows how the model initially performed.

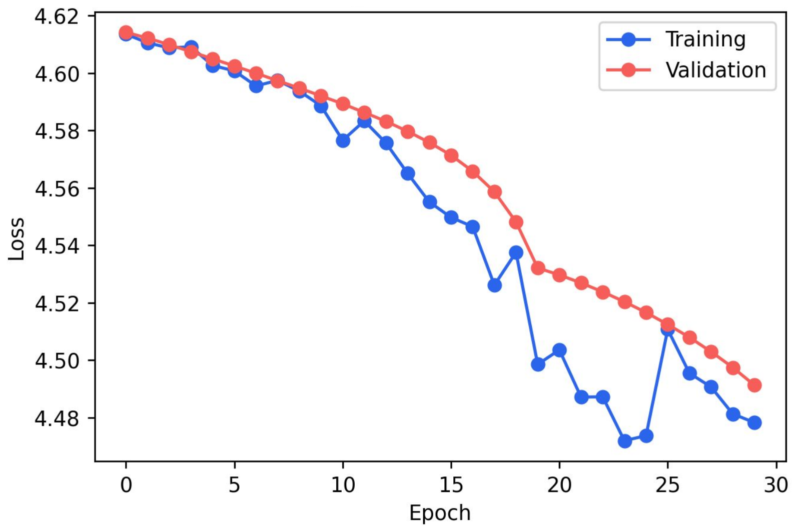
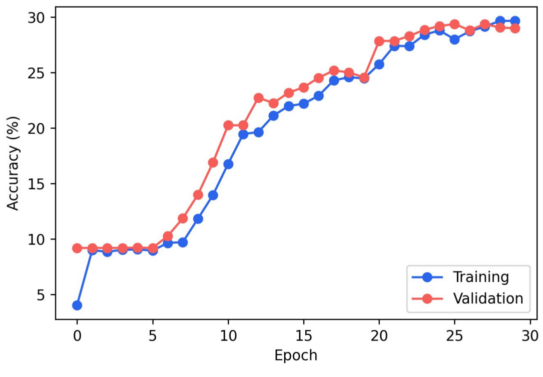


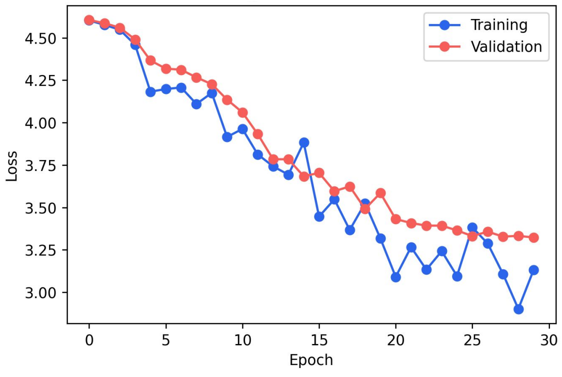
Figure 3: *Training and Validation losses with default hyperparameters (LR=0.001)*

As can be seen the loss is decreasing very slowly (at the end the cost is around 4.49) and so the network is not learning enough. Indeed, also the training and validation accuracies after few epochs remains fixed at the same constant value (around 0.092). Furthermore, also the decay policy of the learning rate act forcing the model to learn even slowly. All this considerations suggest to choose higher values for the LR or to increase the number of epochs.

Tuning LR. Therefore, a new training has been done with a higher learning rate of 0.01 (10 times higher), keeping the other parameters unchanged. As can be seen in the charts (see Figure 4) with a higher learning rate the model is able to reduce the cost function. Both graphs indicate how performances improve when increasing the learning rate, without showing overfitting as can be seen in Fig. 4 (a) the validation accuracy “follow” the training one without diverging.

For this reason other value for the learning rates has been tried out (in particular 0.005, 0.001, 0.05, 0.01, 0.1). In Figure 5 are reported the results of this grid search.





(a) *(b)*

Figure 4: *Accuracies (a) and losses (b) on training and validation with LR=0.01*

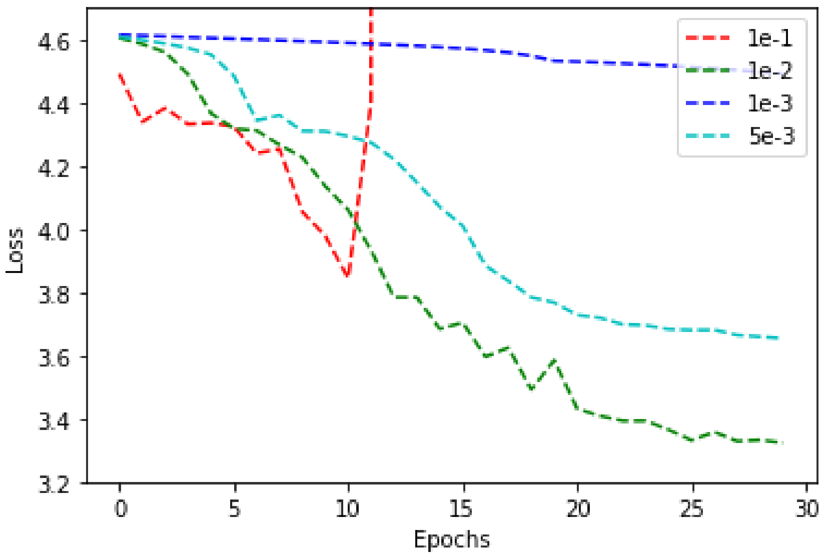


Figure 5: *Validation losses with different learning rates*

As can be seen in the previous graph there are some interesting consideration to point out.

- The figures shows that the loss diverge after few epochs for the red line (with a learning rate of 0.1). The network showed to be very sensitive to changes in the learning rate, especially with very high values as in this case. Therefore, the value related with this curve can be considered as an upper bound regarding the learning rate.
- The default model (i.e., LR=1e-3) as already stated is an example of *underfitting*, because the network is not learning enough to achieve good results (the loss is decaying too slowly).

- Finally, the green line can be considered the best configuration found in terms of goodness of the learning curve since it seems that it will continue to decrease if the epochs will increase.

Tuning Epochs. To better understand the learning curves the same kind of tuning has been executed with a larger number of epochs. For this reason the number of epochs has been doubled (and consequently the step size has been increased). In Figure 6 are reported the results with this setting and some values of the learning rates.

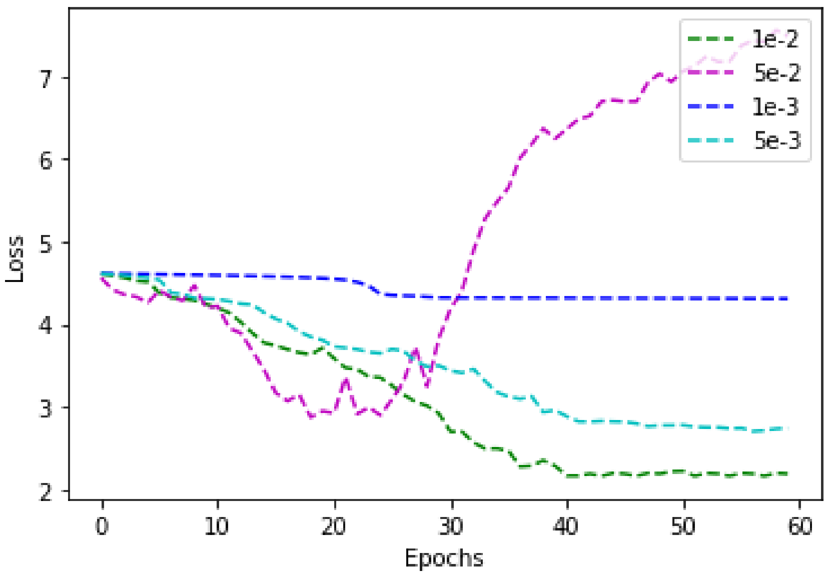


Figure 6: *Training and Validation losses with different learning rates and 60 epochs (note that the step size is set to 40)*

As can be clearly seen the best learning rate found with 30 epochs confirmed to be a "good learning curve". Instead, the higher LR=5e-2 (purple line) ended up displaying a catastrophic *overfitting* scenario, this means that the model has learned the training dataset too well and is less good to generalize to previously unseen data, resulting in an increase in the error.

Tuning Step-Size. Then, also the step-down policy has been tuned for the current best configuration (i.e. `LR=1e-2` and `NUM_EPOCHS=60`). Note that the decay factor (`GAMMA`) remained unchanged wrt the default model. Specifically, the `STEP_SIZE` hyperparameter has been set to different values (i.e. 25, 30, 35, 40, 45) as shown in Figure 7.
As can be seen the best model is the same as before, because it has the lowest loss at the last epoch. Furthermore, when step size becomes higher than 30, the model start to show overfitting (this can be clearly seen with step size equals 45).

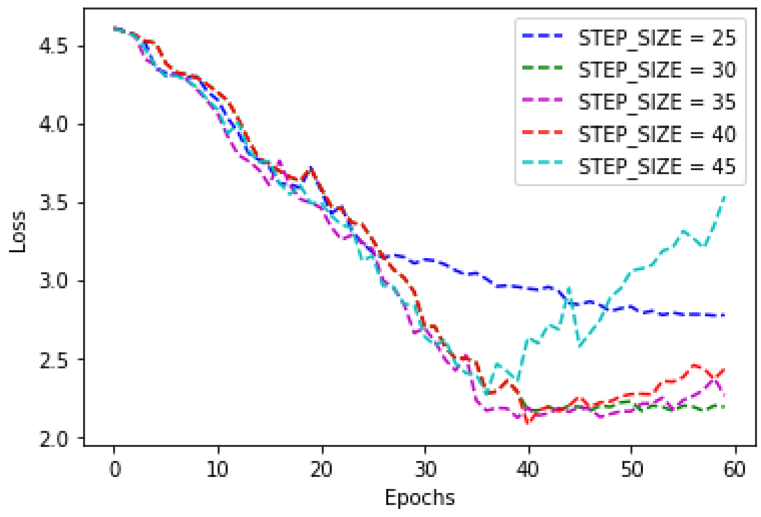


***Figure 7:** Training and Validation losses with different step-sizes (LR=1e-2 and 60 epochs)*

Then, the best hyperparameter found so far for the training from scratch are:

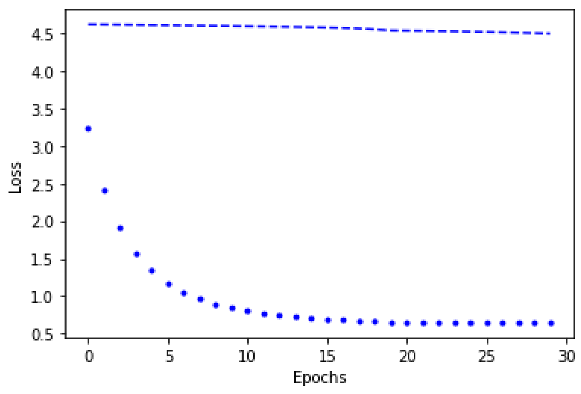
- `LR = 1e-2`
- `NUM_EPOCHS = 60`
- `STEP_SIZE = 30`

Finally, the model is retrained on *training plus validation* sets with the best configuration found. The final accuracy on the test set obtained with this setting is 62.5%.

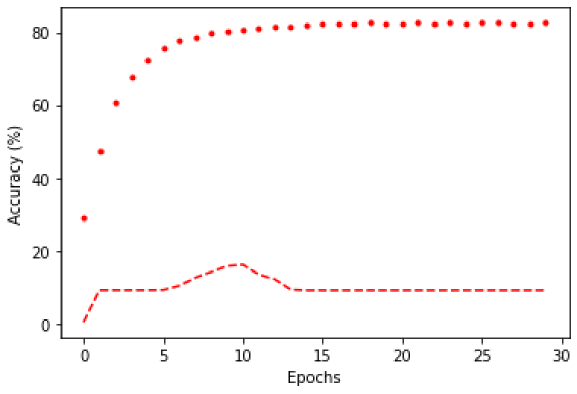
The charts saw in this first part of the report how, by training a network from scratch with such few samples, it is not possible to obtain a good model which learns well from the training set and at the same time is also capable to well generalize on unseen data. As a conclusion, we can state that, in the case of small datasets, training a network from scratch is not an easy and effective task.

4 Transfer Learning

Neural networks typically need huge amounts of data in order to be operationally effective, in this case for classification. In order to address the issue of low accuracy in training from scratch, it's possible to use *Transfer Learning* technique which upload AlexNet with its weights (already learned on a large dataset, in our case Imagenet) and it is used as starting point for training on our small dataset (i.e. Caltech-101).
Before feeding the net, the images are preprocessed in the same way described above (see Section 2), to obtain normalized images, but this time we normalize the input images with the mean and standard deviation from ImageNet, which are the vectors (0.485, 0.456, 0.406) and (0.229, 0.224, 0.224). By using the same parameters of the first implementation trained the following results are obtained. Figure 8 shows the impressive performance achieved with transfer learning in comparison with the training done with the default hyperparameters: in particular, the validation accuracy is boosted from 9.20% to as much as 82.54%.



(a)



(b)

Figure 8: *Comparison between the baseline "--"and transfer learning "•" in terms of loss (a) and validation accuracy (b)*

Now that we have achieved satisfactory results, we can use some well-known strategy to accelerating the training phase. In order to do that, it's possible to *'freeze'* some part of the network, i.e. performing the training only on a subset of the layers.
By training the net with the best set of hyperparameters found so far, the results obtained are reported in the following table.

	Validation accuracy (%)	*Test accuracy (%)*
All layers	82.54	83.63
FC only	82.34	83.55
Conv only	34.62	33.98

Table 2: *Results obtained training different parts of the network*

From the table we can see that freezing the convolutional layers does not impact the overall precision of the model. This is likely since that convolutional part typically deals with generic features of the images, that can be learned from ImageNet in pretraining. Instead, freezing the fully-connected layers brings to poor performances, since those are in charge of the final classification according to the specific categories in the Caltech dataset.

Note that for the last part of this work, we will freeze the first part of the network (i.e. convolutional layers), in order to drop significantly the training time without losing too much in terms of accuracy.

5 Data Augmentation

One of the best way to reduce overfitting, and train a more robust deep neural network model is to use image augmentation. Data augmentation can include flipping, rotating, cropping, changing the color palette and many more operations.
In addition to transfer learning, a data augmentation task is now performed to further prevent overfitting on the given dataset. Augmenting the images will make the neural network model see different types of images at each mini-batch. This adds a regularizing effect as well which makes the network robust to augmented images during test time as well.

Three different transformations are applied:
- Random Crop instead of a central one, perform a crop of the desidered size on the image. This
- Horizontal Flip of the images with probability of 0.5 to be applied combined with a crop applied at a random location of the images (output size 224x244, with no padding);
- A random rotation of the images of 45 degrees.

By applying these transformations to the baseline model (with frozen convolutional layer), the new scores are collected in the table below:

	Validation accuracy (%)	*Test accuracy (%)*
No augmentation	82.34	83.55
Random Crop	85.34	**84.42**
Random Crop + Horizontal Flip	85.12	84.28
Random Crop + Horizontal Flip + Random Rotation	84.04	83.92

Table 3: *Results with data augmentation*

Note that has been also tested the Vertical Flipping, i.e. inverting the upper and the lower part, but this transformation reveals to be unsuccessful wrt the results obtained in the table above. This can be easily justified since it's unlikely that we get a test time an animal or car turned upside down. Indeed, providing training data that are different from test data might bring the network to learn wrong features, with a decrease in the accuracy.

6 Beyond AlexNet

In this section the classification is performed by using two successor of AlexNet, in particular a Resnet and a VGG. The peculiarity of these networks, that allowed them to overcome the performance of Alexnet, is that they are significantly deeper: we use a 16-layer VGG, a 18-layer and 50-layer ResNet (with respect to the 8 layer in the AlexNet).
As a consequence, those networks consume more resources (i.e. memory), so that we have to reduce the batch size, by doing so we will have longer epochs, thus also the number epochs is reduced (in this last part we set the number of epochs to 10). In the case of the VGGNet, we set the batch size to as little as 16 samples and we train the network for 10 epochs; when using the ResNet instead we have batches with 64 elements and 30 epochs.
As already done for the AlexNet we adapt the last layer of each net to perform classification on 100 classes and finally, all networks are pretrained on the ImageNet dataset.

	Validation accuracy (%)	Test accuracy (%)
VGGNet-16	96.85	93.60
ResNet-18	93.35	92.24
ResNet-50	99.64	95.89

Table 4: *Performances comparison of the network architectures implemented*

In conclusion, as can be seen in the following table, more recent and deeper models are able to significantly improve performances achieving higher test accuracy.

References

[1] Caltech101 - Image Dataset

[2] AlexNet: ImageNet Classification with Deep Convolutional Neural Networks

[3] VGG: Very Deep Convolutional Networks for Large-Scale Image Recognition

[4] ResNet: Deep Residual Learning for Image Recognition

[5] Dataset and split: https://github.com/MachineLearning2020/Homework2-Caltech101