POLITECNICO DI TORINO

MACHINE LEARNING AND DEEP LEARNING

ALBERTO MARIA FALLETTA - S277971

HOMEWORK 2 – REPORT

DEEP LEARNING

INTRODUCTION

In this homework we will train a Convolutional Neural Network for image classification, more specifically an AlexNet.

For the purpose of this assignment we will be using Caltech 101, a dataset containing pictures of objects belonging to 101 categories with about 40 to 800 images per category (Most categories have about 50 images). Therefore, we can say the dataset is rather unbalanced. Having to work with unbalanced data is not optimal most of the times but this is particularly true when it comes to deep learning since the network may learn to classify an image as the most present class since statistically it will be right most of the times, without focusing on the features it should really be learning instead. A solution for that may be under-sampling the most present class.

Another thing to notice is that the size of each image varies and the average size is roughly 300x200 pixels, however AlexNet requires 224x224 pixels images therefore many images are under the required dimension but fortunately this is not a problem and it is automatically handled by Pytorch's trasforms.CenterCrop( ) which adds padding when needed.

Furthermore most images have little to no clutter (i.e. no random objects in the image that makes it difficult to focus on the main object), the objects tend to be centered in each image and most objects are presented in a stereotypical pose. This means, from this point of view, we are in the most ideal scenario possible.

This image found on Caltech's offical website is a composition of averages of the images of each category.

It is possible to recognize some categories as Faces, Motorbikes, Yin Yangs and more.

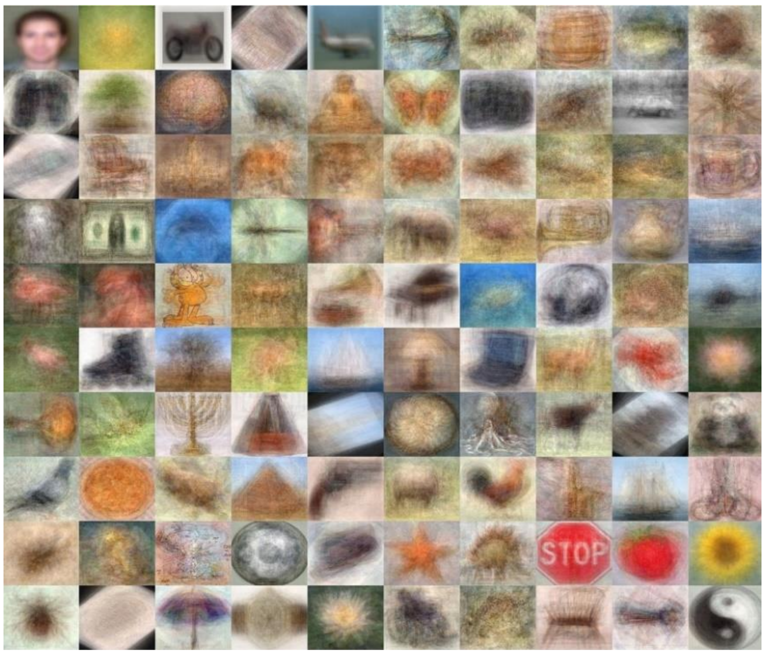


Figure 1 composition of averages of the images of each category

DATA PREPARATION

As a first step we create the dataset for Caltech images taking as reference torchvision's ImageFolder, therefore building dataset classes, one for training + validation and one for testing (being the split files provided by mean of txt files), from data structured in nested folders. (The code for this part of the task is in my_caltech_dataset.py)

By creating a Caltech class a make_dataset function is called, this function reads the appropriate split textual file and it builds from that a list of tuple items made of PIL image and label making sure BACKGROUND_Google images are filtered out.

The following images shows the distribution of the two splits provided, one for training and validation and one for testing, as we can see there are classes way bigger than others in both the two sets as motorbikes, faces and airplanes. Fortunately, the distribution is the same in both the two sets.

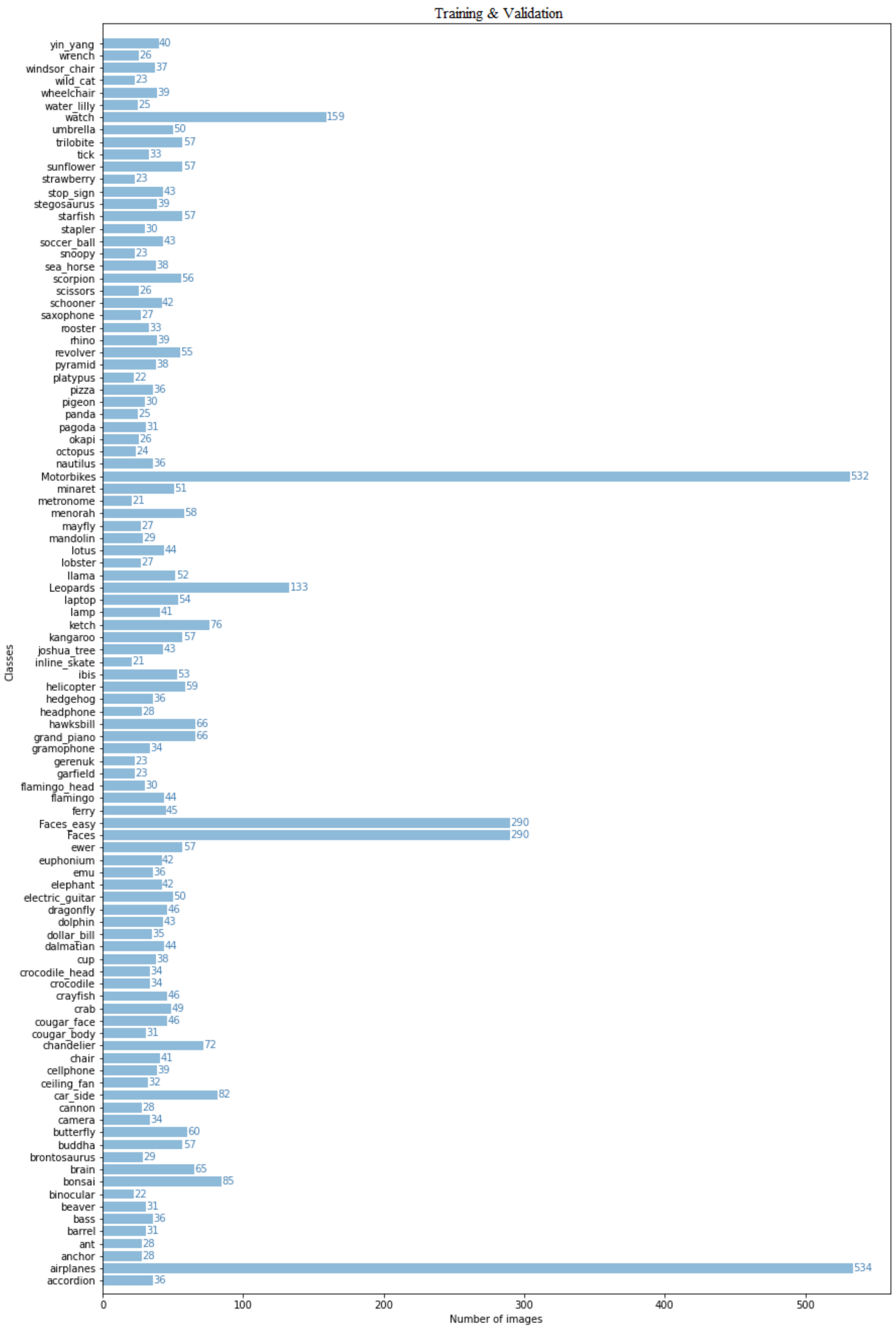


Figure 2 Number of images per class in training & validation split

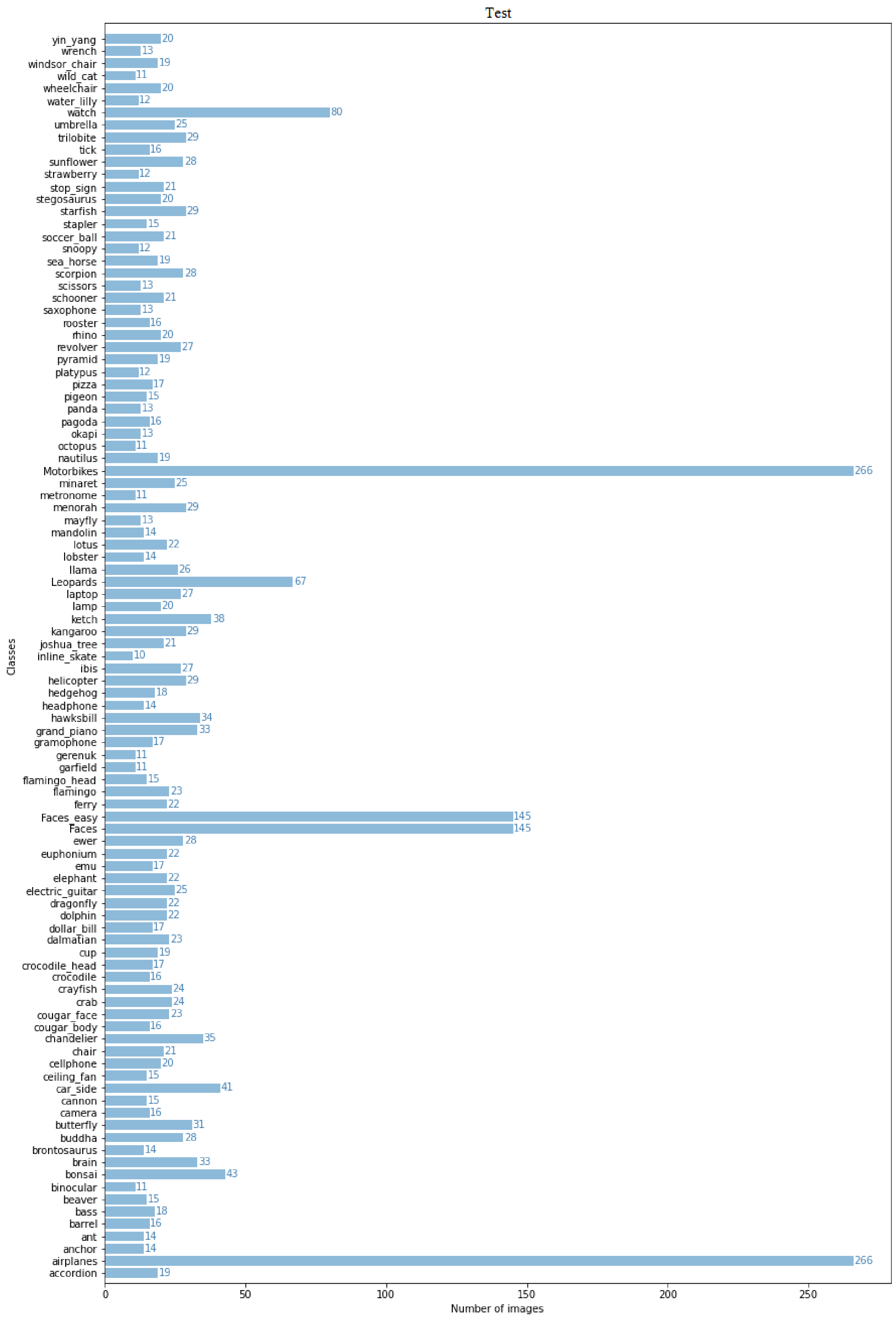


Figure 3 Number of images per class in test split

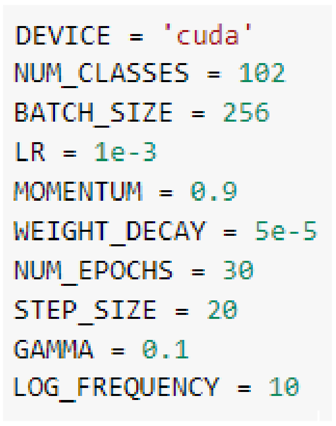
1. TRAINING FROM SCRATCH

We can proceed in splitting the training + validation set in training set and validation set maintaining the original distribution by dividing in even-indexes and odd-indexes and therefore exploiting the order of train.txt.

The starting implementation of the template trains the network using Stochastic Gradient Descent (SGD) with the parameters specified in figure 4.

Running the training with this configuration leads from a starting loss of 4.62 and accuracy on validation set of 0.09 of the first epoch to a loss of 4.55 and a not improved validation accuracy by the 30th epoch as shown by figure 5 and equal accuracy on the test set of 0.09.

Of course, any good performance was not expected since this is the run of the template straight "out of the box" but still visualizing plots helps deciding the direction to take.



```
DEVICE = 'cuda'
NUM_CLASSES = 102
BATCH_SIZE = 256
LR = 1e-3
MOMENTUM = 0.9
WEIGHT_DECAY = 5e-5
NUM_EPOCHS = 30
STEP_SIZE = 20
GAMMA = 0.1
LOG_FREQUENCY = 10
```

Figure 4 template's default parameters

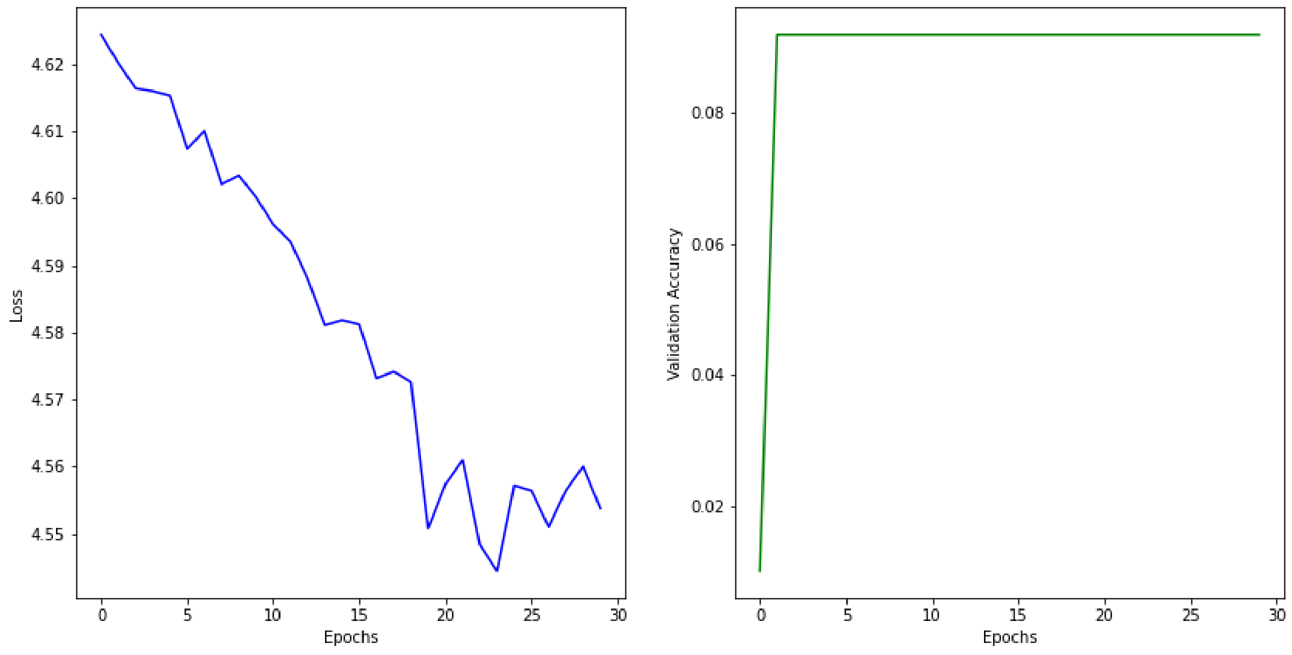


Figure 5 Loss and validation accuracy with parameters: Template's default

1.1. Changing Learning Rate

As suggested by the homework text the first parameter to optimize is the learning rate, in fact with the initial value of 0.001 we can see from the plots that the model converges too slowly and by the 20th epoch is probably stuck in a local minimum. Therefore, we will now train the network with different learning rate values.

The following table summarizes some of the experiments performed with the parameter[1]:

Learning Rate	Validation Accuracy
1e-3	μ = 0.092, σ = 0.000
1e-2	μ = 0.298, σ = 0.002
5e-2	μ = 0.446, σ = 0.016
6e-2	μ = 0.344, σ = 0.058
1e-1	μ = 0.160, σ = 0.085

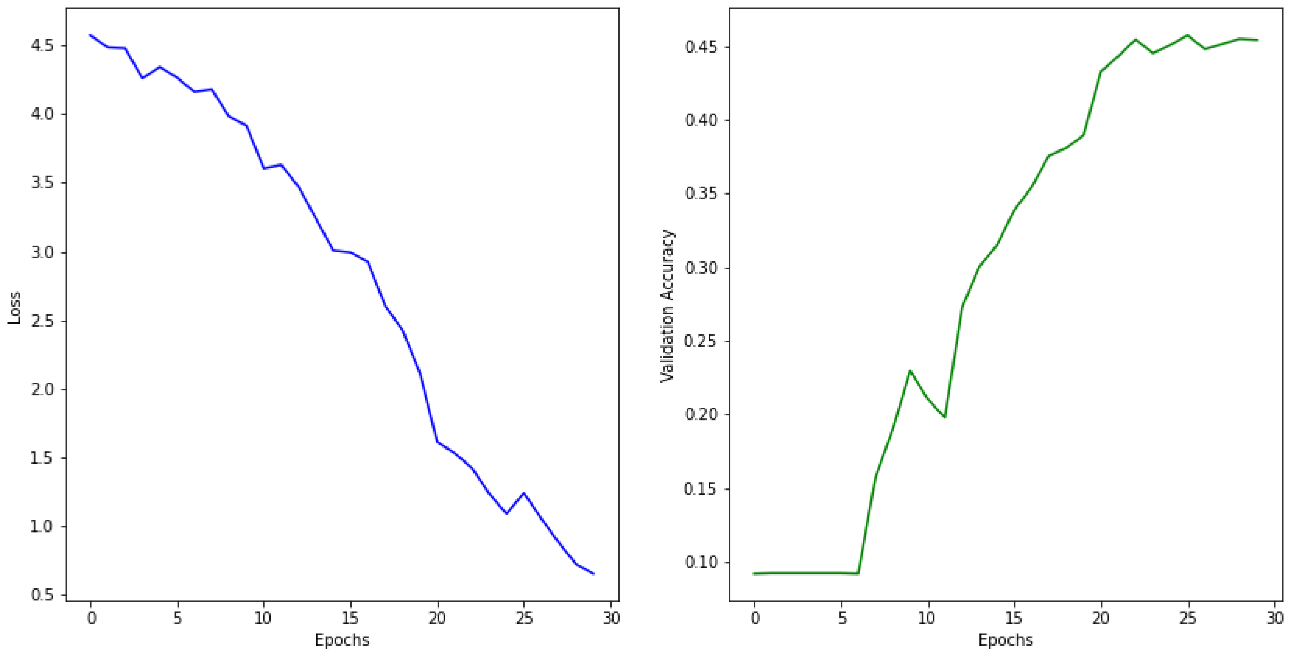


Figure 6 Loss and validation accuracy with parameters: LR = 5e-2, BATCH_SIZE = 256, NUM_EPOCHS = 30, STEP_SIZE = 20

From the experiment associated with the best validation accuracy we can see that changing only the learning rate of a little more than one magnitude order we go from a starting loss of 4.62 and accuracy on validation set of 0.09 of the first epoch to a loss of 0.71 and a 0.446 validation accuracy by the 30th epoch, as shown by figure 6, and almost equal accuracy on the test set. This is not only a huge improvement in accuracy but also a proof of the importance of the learning rate parameter.

[1] Validation Accuracy is the result of three runs. It is therefore expressed by expected value and standard deviation.

1.2. Changing Epochs Number

By looking at the previous plot we can see there is still room for experiments, more specifically it seems the model could use some more epochs to better reach convergence, therefore, I will now train the network with different NUM_EPOCHS values, keeping the learning rate at 0.05.

The following table summarizes some of the experiments performed with the parameter[2]:

Number of Epochs	Validation Accuracy
20	μ = 0.328, σ = 0.057
30	μ = 0.446, σ = 0.016
40	μ = 0.434, σ = 0.018
50	μ = 0.451, σ = 0.011
60	μ = 0.498, σ = 0.012

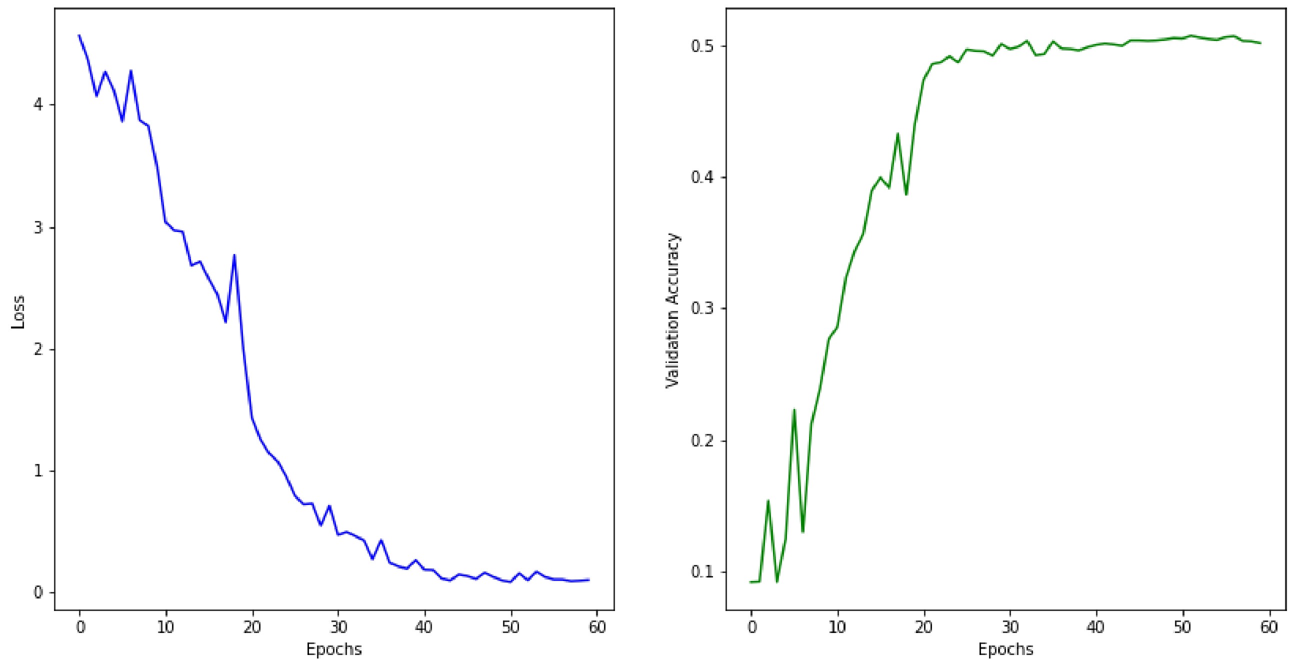


Figure 7 Loss and validation accuracy with parameters: LR = 5e-2, BATCH_SIZE = 256, NUM_EPOCHS = 60, STEP_SIZE = 20

This time we go from a starting loss of 4.62 and accuracy on validation set of 0.09 of the first epoch to a loss of 0.16 and a 0.498 validation accuracy by the 60th epoch, as shown by figure 7, and almost equal accuracy on the test set. This experiment proved the network needed more time for the training phase as speculated by looking figure 6.

[2] Validation Accuracy is the result of three runs. It is therefore expressed by expected value and standard deviation.

1.3. Changing Step Size

From figure 7 we can see how the slope of the line abruptly changes by the 20th epoch. In the next run I will train the network with the best configuration so far and various step size values.

The following table summarizes some of the experiments performed with the parameter[3]:

Step Size	Validation Accuracy
10	μ = 0.321, σ = 0.020
20	μ = 0.498, σ = 0.012
30	μ = 0.476, σ = 0.006
40	μ = 0.476, σ = 0.008
50	μ = 0.504, σ = 0.006

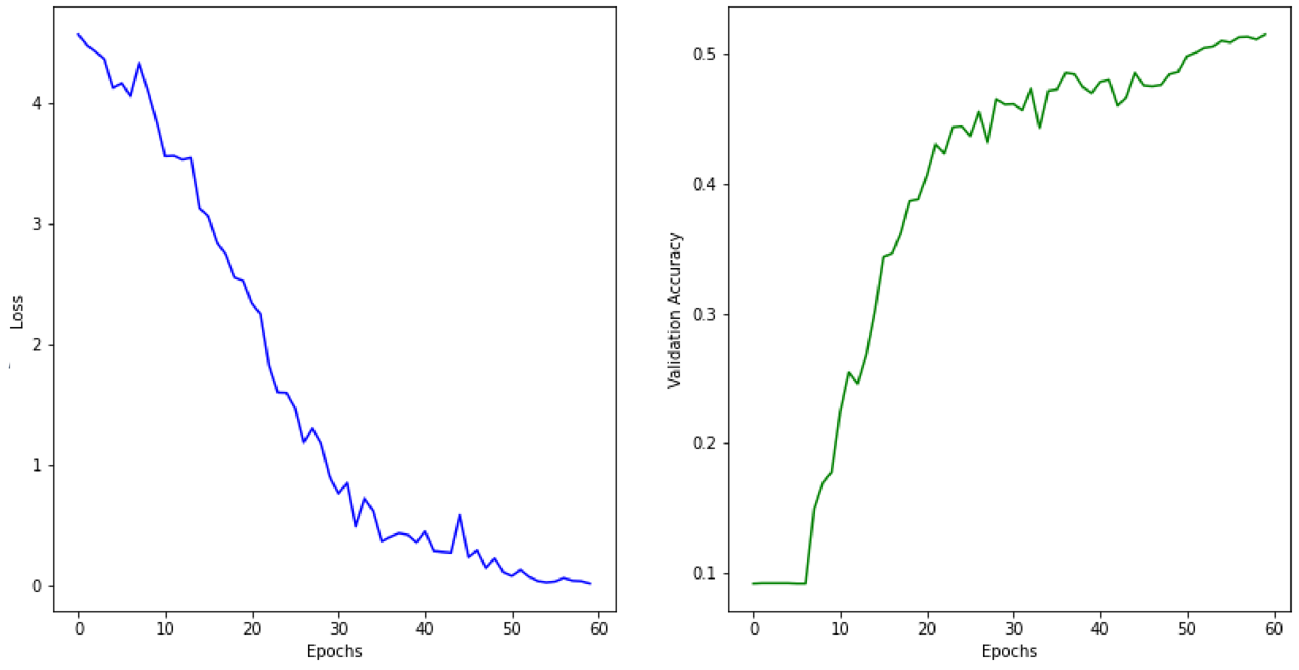


Figure 8 Loss and validation accuracy with parameters: LR = 5e-2, BATCH_SIZE = 256, NUM_EPOCHS = 60, STEP_SIZE = 50

This time we go from a starting loss of 4.62 and accuracy on validation set of 0.09 of the first epoch to a loss of 0.08 and a 0.50 validation accuracy by the 60th epoch, as shown by figure 8, and almost equal accuracy on the test set.

By looking at the plots we can see that maybe convergence is not quite reached yet so we could gain a few more accuracy points, obviously we would have to deal with overfitting going forward in training from scratch.

[3] Validation Accuracy is the result of three runs. It is therefore expressed by expected value and standard deviation.

2. TRANSFER LEARNING

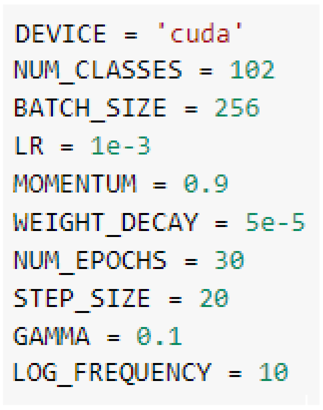
As we have seen, deep learning needs large datasets to train good features but, unfortunately large datasets are not always available like in our case. Caltech101 in fact is a dataset of just 9146 images for 101 categories (that means an average of just 89 images per category).

Therefore, one thing we can do is “finetuning”, that is the exploitation of weights learned by the network trained on a large related dataset as a starting point for training on our small dataset. In fact, by loading AlexNet with “pretrained” set to true we load the weights learned in the training on ImageNet, a dataset of 14 million images and around 20.000 categories.

2.1. Changing Learning Rate

Training the pretrained network using the hyperparameters that gave us the best accuracy score in the previous step makes the loss diverge since the learning rate is too high for a pretrained network, in fact, when using pretrained models we want to prefer lower learning rate values from the moment we just want to adjust a little (fine-tune) the knowledge they acquired in their previous trainings.

One interesting thing to notice is that running the training on the pretrained network, once again with the starting “out of the box” configuration that previously led to such poor accuracy results (0.09), this time makes us achieve a way different result and this happens because we expect the pretrained weights to be quite good already as compared to randomly initialized weights, therefore we do not want to distort them too quickly and too much and that is why a smaller learning rate works.



```
DEVICE = 'cuda'
NUM_CLASSES = 102
BATCH_SIZE = 256
LR = 1e-3
MOMENTUM = 0.9
WEIGHT_DECAY = 5e-5
NUM_EPOCHS = 30
STEP_SIZE = 20
GAMMA = 0.1
LOG_FREQUENCY = 10
```

Figure 9 template's default parameters

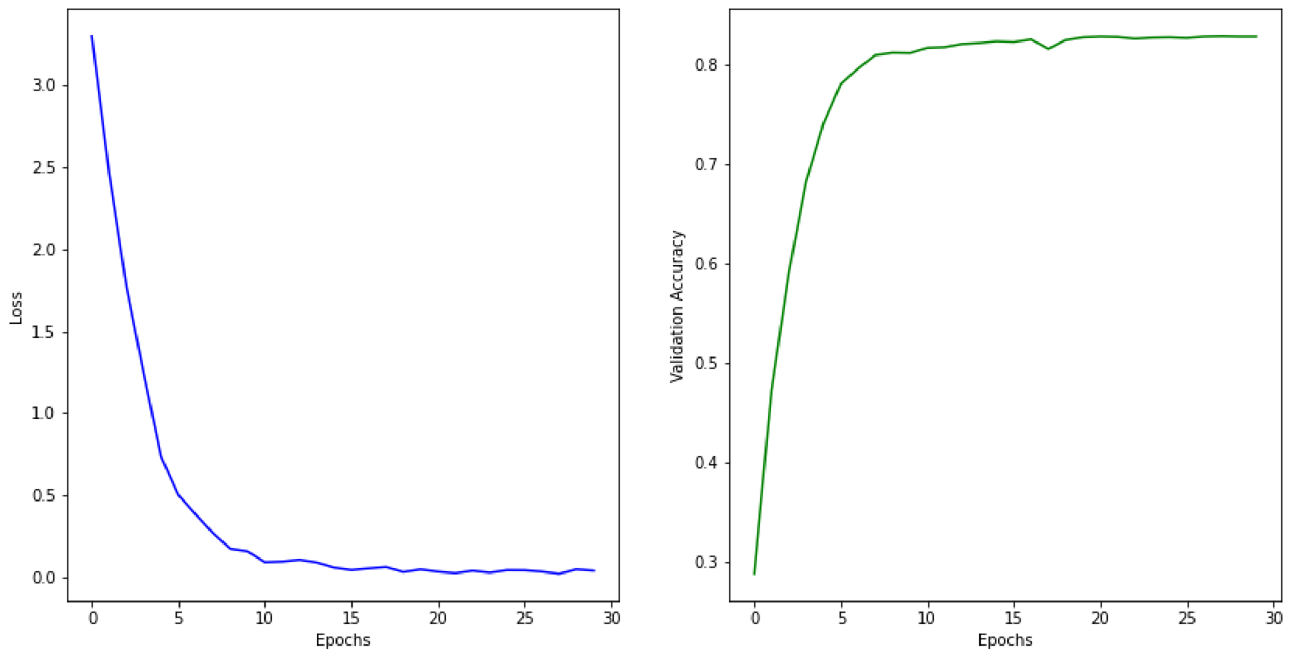


Figure 10 Loss and validation accuracy with parameters: LR = 1e-3, BATCH_SIZE = 256, NUM_EPOCHS = 30, STEP_SIZE = 20

This time, in fact, we go from a starting loss of 4.92 and accuracy on validation set of 0.29 of the first epoch to a loss of 0.02 and a 0.83 validation accuracy by the 30th epoch, as shown by figure 10, and almost equal accuracy on the test set.

Furthermore we can see from figure 10 that loss convergence is reached way sooner than how it was reached when training the network from scratch, in fact, the accuracy score does not change much from the 20th epoch to the 30th.

Other experiments have been performed with different learning rate values on the pretrained model, using the previous best validation accuracy achieving parameters. The following table summarizes some of the experiments performed with the parameter[4]:

Learning Rate	Validation Accuracy
1e-4	μ = 0.801, σ = 0.005
1e-3	μ = 0.842, σ = 0.003
1e-2	μ = 0.858, σ = 0.001

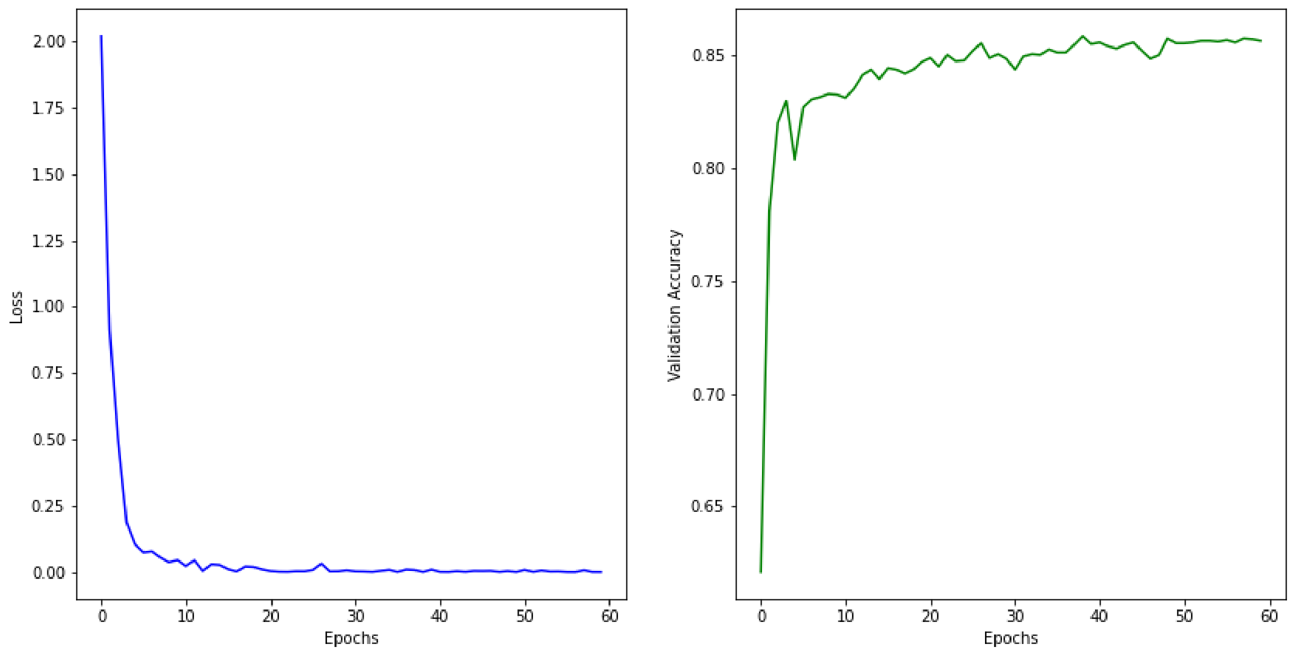


Figure 11 Loss and validation accuracy with parameters: LR = 1e-2, BATCH_SIZE = 256, NUM_EPOCHS = 60, STEP_SIZE = 50

[4] Validation Accuracy is the result of three runs. It is therefore expressed by expected value and standard deviation.

2.2. Changing Step Size

Other trials have been performed with different step size values though no accuracy improvement or notable behavior changes have been experienced. The following table summarizes some of the experiments performed with the parameter[5]:

Step Size	Validation Accuracy
20	μ = 0.857, σ = 0.000
30	μ = 0.859, σ = 0.001
40	μ = 0.855, σ = 0.002
50	μ = 0.858, σ = 0.001

2.3. Changing Batch Size

Another experiment we could do is changing the batch size that currently has been used with a value of 256. The first of the two following figures has batch size of 64 while the second one has batch size of 1024.

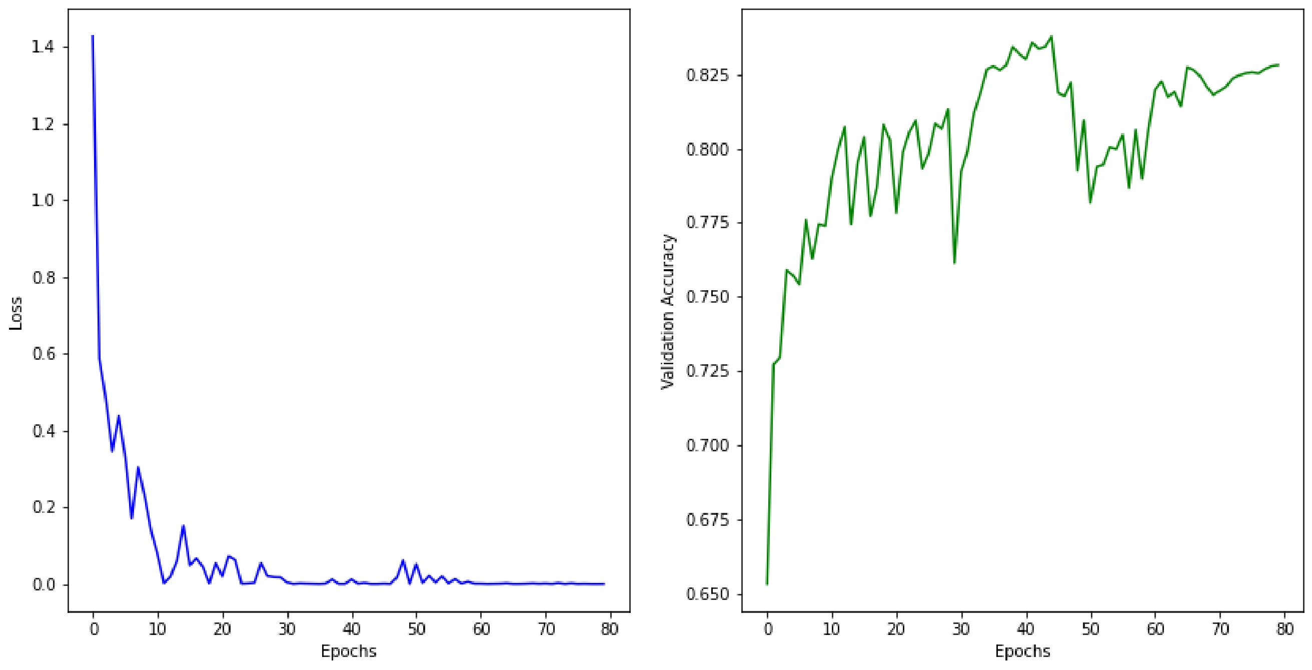


Figure 12 Loss and validation accuracy with parameters: LR = 1e-3, BATCH_SIZE = 64, NUM_EPOCHS = 80, STEP_SIZE = 50

[5] Validation Accuracy is the result of three runs. It is therefore expressed by expected value and standard deviation.

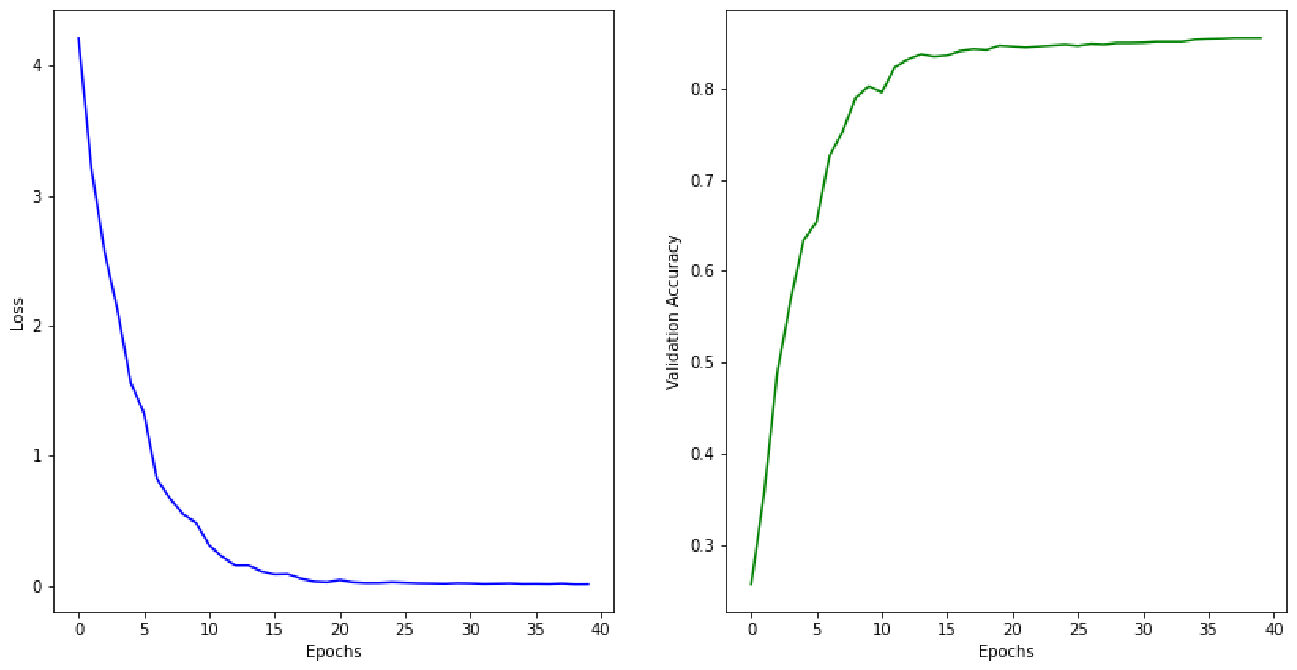


Figure 13 Loss and validation accuracy with parameters: LR = 1e-2, BATCH_SIZE = 1024, NUM_EPOCHS = 40, STEP_SIZE = 30

The first thing we can notice is that in order to achieve the same performance results achieved with the larger batch size we have to adjust learning rate and number of epochs for the model trained with smaller batch size.

The size of batches is essentially the frequency of updates since the gradient is calculated at the end of every epoch and so is the weights update phase, therefore, the smaller the batches the more the updates.

Furthermore, empirical heuristic suggests that when using a larger batch there is a significant degradation in the quality of the model, as measured by its ability to generalize.

The following table summarizes some other experiments performed with batch size[6] and LR=1e-2, STEP_SIZE=30, NUM_EPOCH=60:

Batch Size	Validation Accuracy
64	μ = 0.831, σ = 0.001
128	μ = 0.856, σ = 0.002
256	μ = 0.858, σ = 0.001
512	μ = 0.858, σ = 0.003
1024	μ = 0.858, σ = 0.001

[6] Validation Accuracy is the result of three runs. It is therefore expressed by expected value and standard deviation.

2.4. Freezing the Network

We will now experiment with training just one part of the network and "freezing" the weights of the other part, using the highest accuracy scoring hyperparameters: LR=0.01, STEP_SIZE=30, NUM_EPOCHS=60 and BATCH_SIZE=256.

First, we freeze the convolutional layers and we train only the fully connected ones:

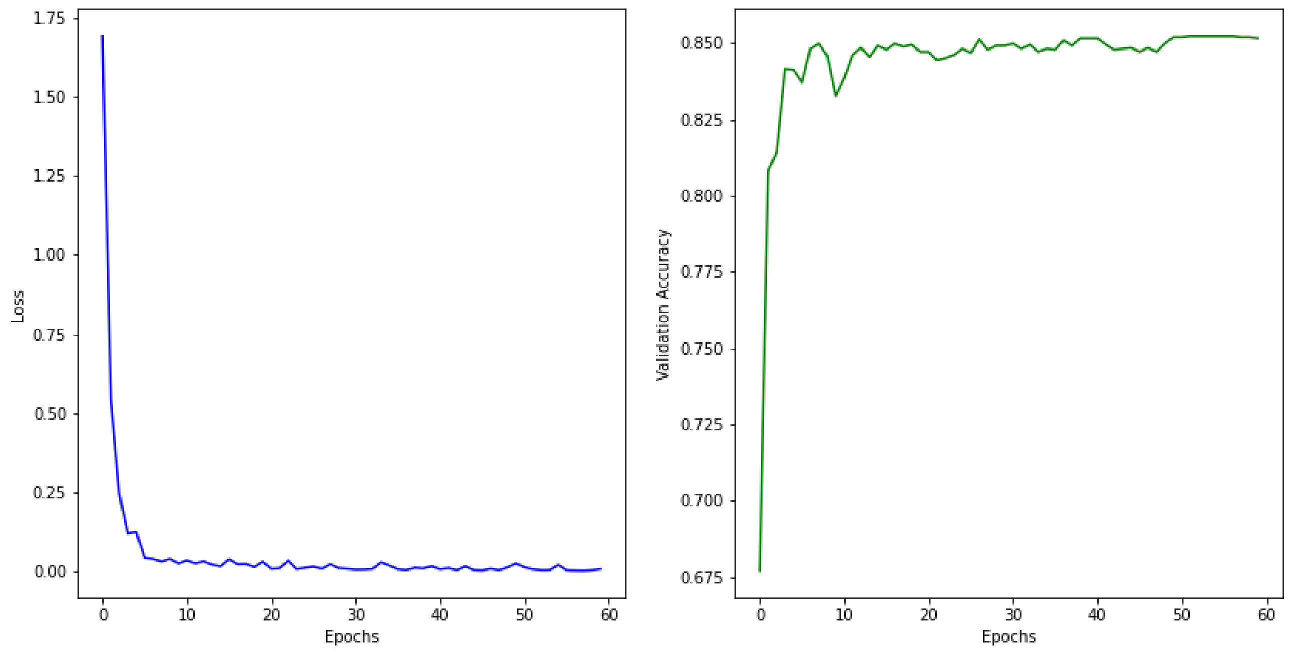


Figure 14 Loss and validation accuracy with parameters: LR = 1e-2, BATCH_SIZE = 256, NUM_EPOCHS = 60, STEP_SIZE = 30

Then we freeze the fully connected layers and we train only the convolutional ones:

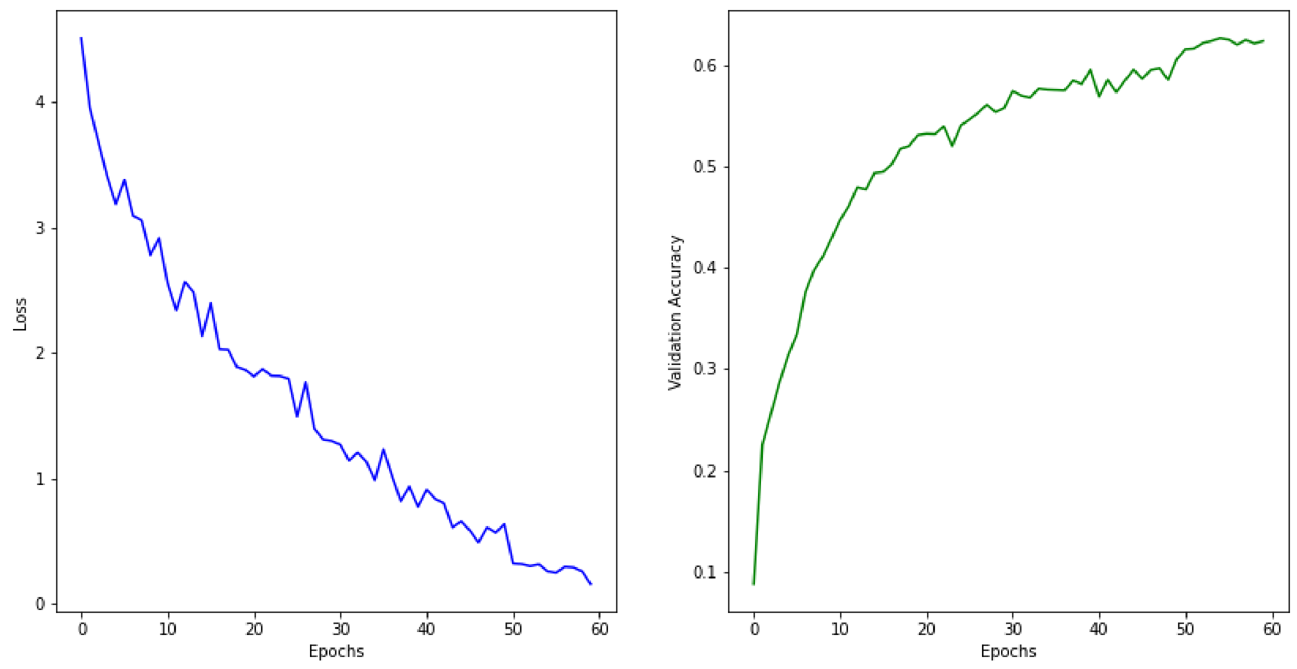


Figure 15 Loss and validation accuracy with parameters: LR = 1e-2, BATCH_SIZE = 256, NUM_EPOCHS = 60, STEP_SIZE = 30

The results are completely different, from the accuracy to the convergence and the cause for this is simple. When doing finetuning it is a common practice to freeze the weights of the first few layers of the pretrained network (in our case the convolutional layers), this is because the first few layers capture the universal features of the images used in the pre-training (ImageNet), like curves and edges, that are also relevant to our new problem. We want to keep those weights intact. The last layers (fully connected layers) instead, capture dataset-specific features that is why these are the layers we want to train and by doing so we achieve the desired results.

3. DATA AUGMENTATION

Having a large dataset is crucial for the performance of the deep learning model. However, we can improve the performance of the model by augmenting the data (i.e. creating variations of the images) we already have.

For this task I will be training only the fully connected layers over 40 epochs, with step size of 20 and initial learning rate of 0.01.

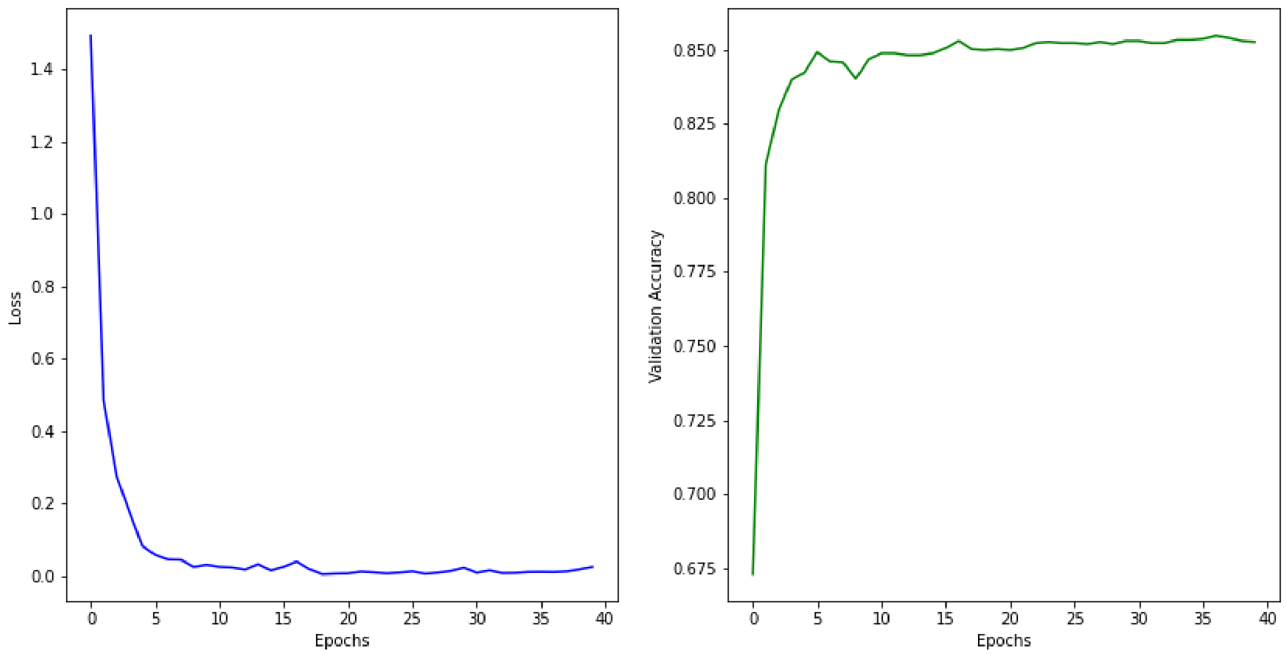


Figure 16 Loss and validation accuracy with parameters: LR = 1e-2, BATCH_SIZE = 256, NUM_EPOCHS = 40, STEP_SIZE = 20

Pytorch's transforms operations are applied on our original images at every batch generation, so the dataset is left unchanged, only the batch images are transformed every iteration.

I’ll first apply RandomCrop(224, pad_if_needed = True, padding_mode = 'edge') instead of transforms.CenterCrop(224) therefore cropping the given PIL image at a random location instead of always in the center, RandomHorizontalFlip(p=0.5) that flips horizontally images with probability 0.5 and RandomGrayscale(p=0.1) that remove the color from the images with probability 0.1

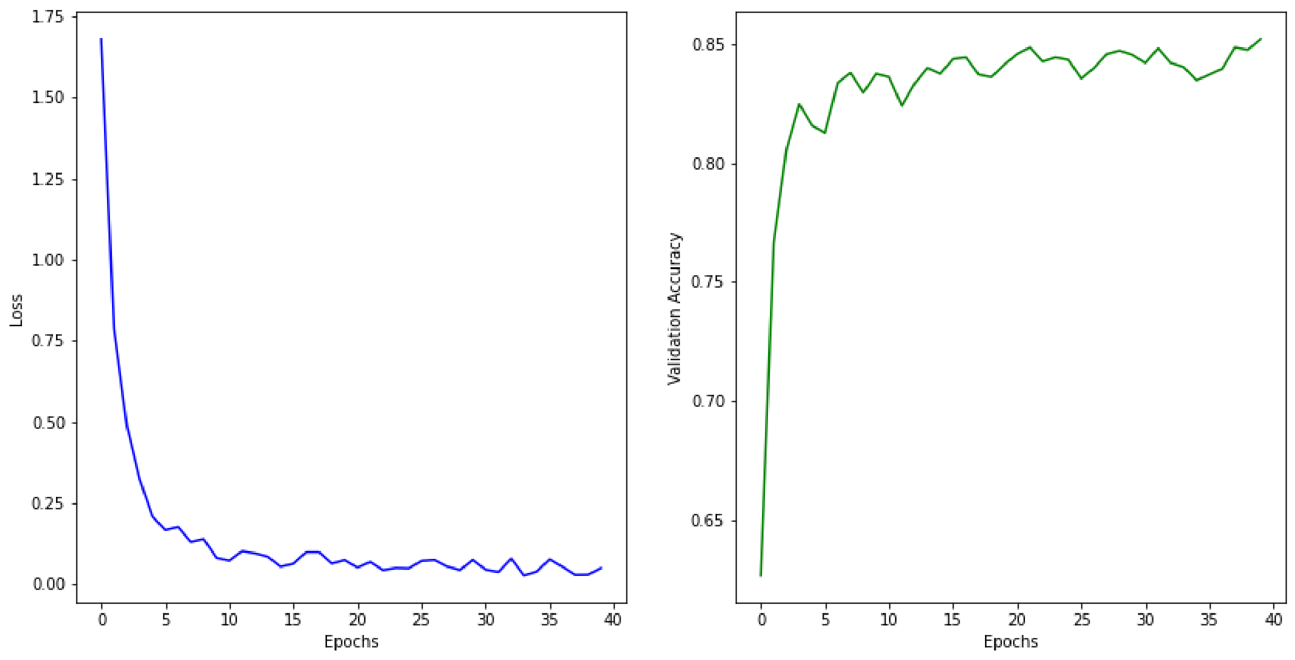


Figure 17 Loss and validation accuracy with RandomCrop

Next I will try RandomPerspective(distortion_scale=0.5, p=0.5, interpolation=3) that distorts images with probability 0.5, RandomRotation(30, resample=False, expand=False, center=None, fill=None) that tilts images of 30 degrees and RandomVerticalFlip(p=0.5) that flips images vertically with probability 0.5.

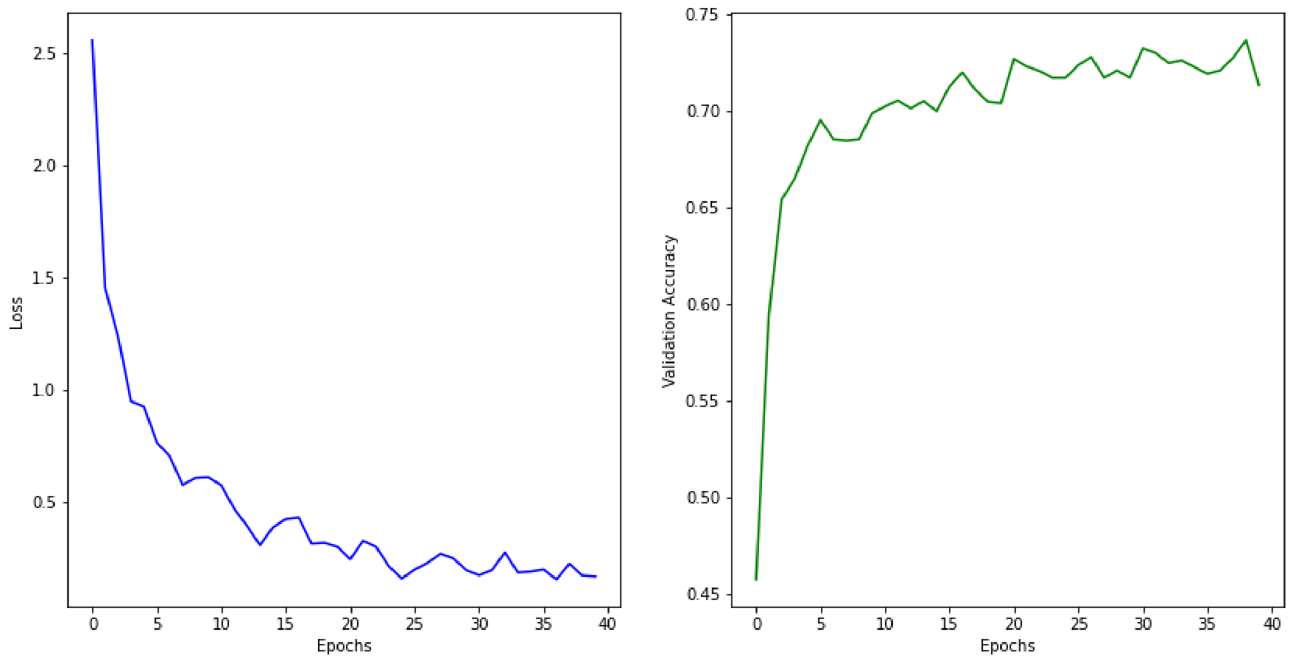


Figure 18 Loss and validation accuracy with RandomPerspective

This last set of preprocessing transforms actually degraded the accuracy and I think this happens because it makes the training set very different from the test set especially with RandomVerticalFlip, in fact, while many categories could be recognized flipped (ex. chairs, scissors..) many other do not appear flipped in real life (ex. bonsai, pyramids).

In the last trial I will use all the ones mentioned above with the exception for RandomVerticalFlip.

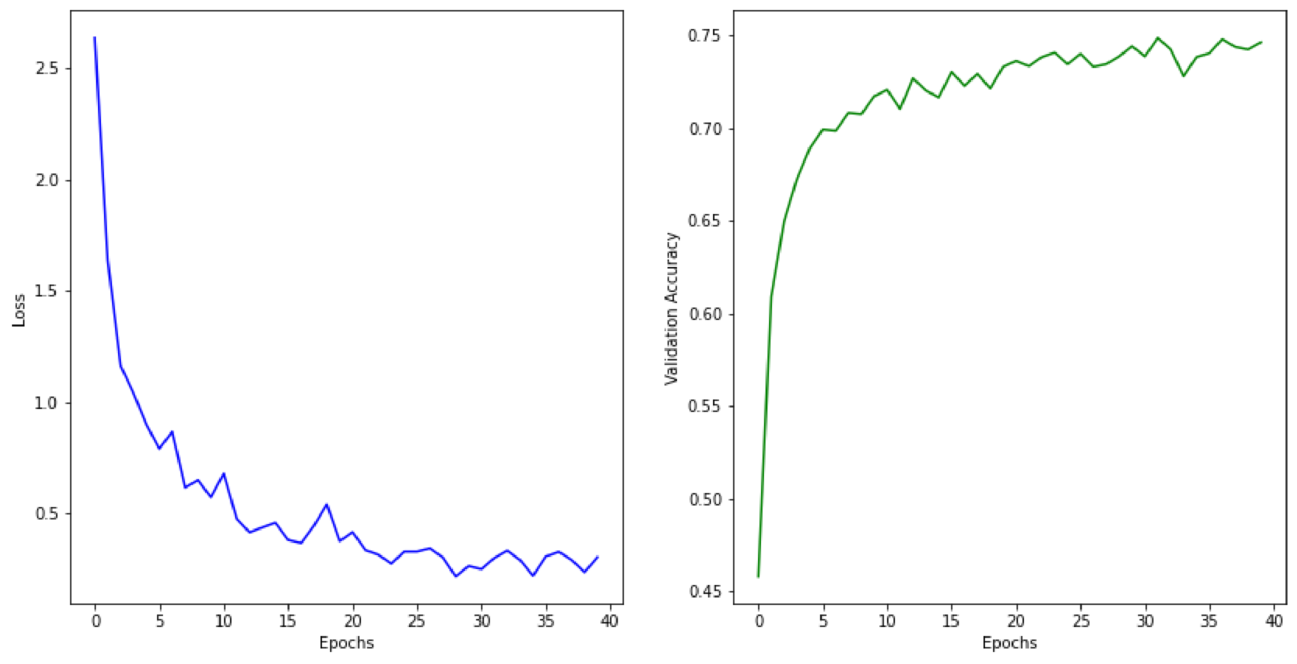
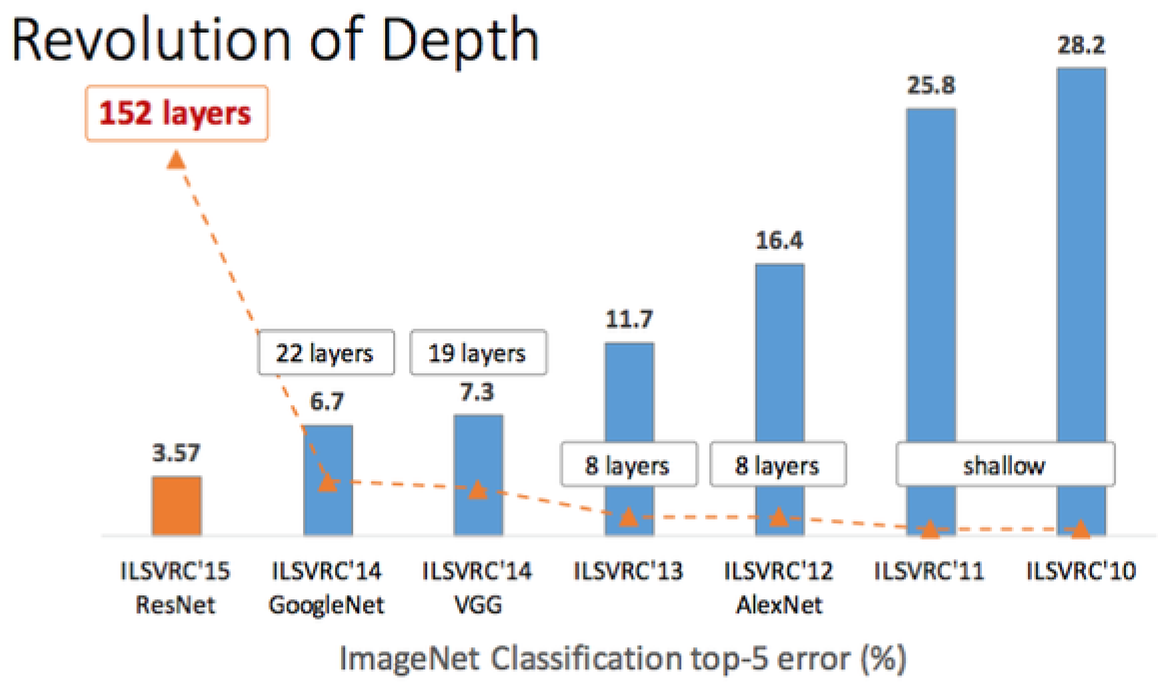


Figure 19 Loss and validation accuracy with RandomVerticalFlip

Even in this case accuracy is slightly worse, that is probably because we are making the training set too different from the validation and test sets since this Caltech dataset is small and almost too ideal (no clutter and subject in perfect image center in stereotypical pose).

4. EXTRA

In this section I will experiment with ResNet18 which is the smallest version of ResNet152 (the most accurate network on image classification).



Larger models are very accurate, but they need a longer time to train and more importantly they consume way more GPU memory, for this reason we have to drop the batch size to 128 otherwise GPU goes out of memory on Google's Colab.

4.1. ResNet from Scratch

Training ResNet from scratch leads to better accuracy results with respect to AlexNet since bigger network have higher learning capabilities since they have way more parameters.

The following table summarizes some of the experiments performed with the learning rate[7] with BATCH_SIZE=128, NUM_EPOCHS=60, STEP_SIZE=50 and pretrained = False:

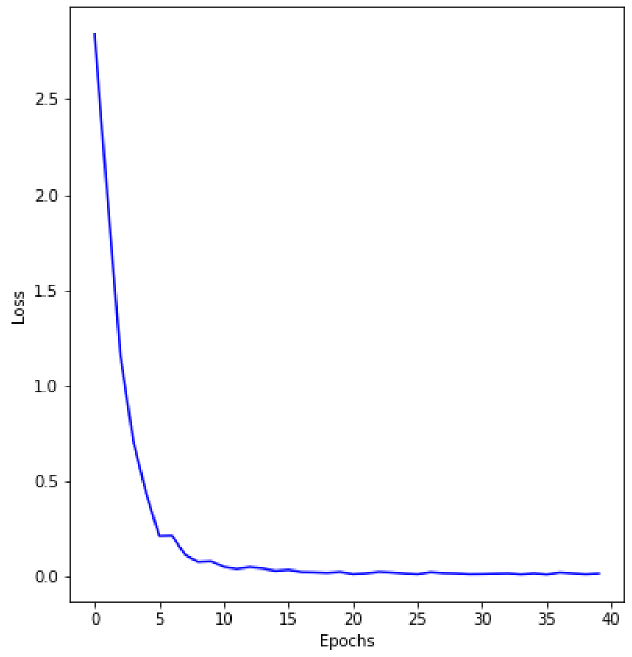
Learning Rate	Validation Accuracy
1e-3	$\mu = 0.442, \sigma = 0.002$
1e-2	$\mu = 0.564, \sigma = 0.001$
5e-2	$\mu = 0.616, \sigma = 0.000$

[7] Validation Accuracy is the result of three runs. It is therefore expressed by expected value and standard deviation.

4.2. Transfer Learning with ResNet

The following table summarizes some of the experiments performed with learning rate[8] with BATCH_SIZE=128, NUM_EPOCHS=60, STEP_SIZE=50 and pretrained = True:

Learning Rate	Validation Accuracy
1e-3	μ = 0.923, σ = 0.003
1e-2	μ = 0.931, σ = 0.001
5e-2	μ = 0.934, σ = 0.000



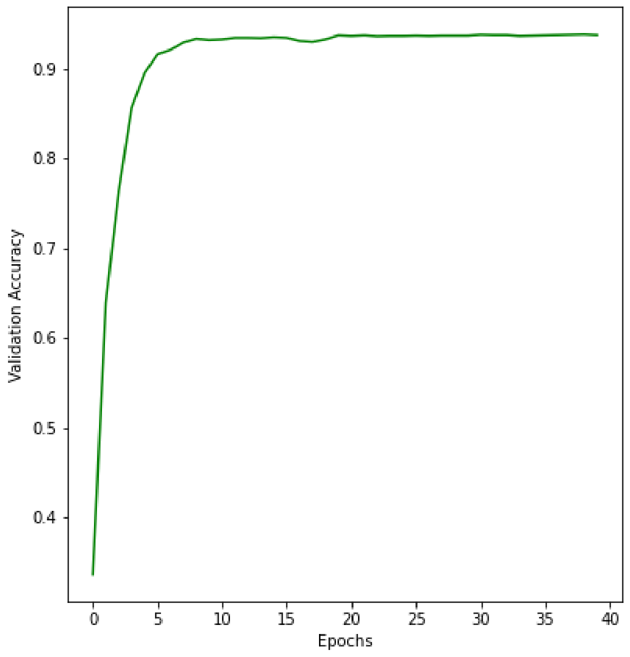


Figure 20 Loss and validation accuracy with parameters: LR = 5e-2, BATCH_SIZE = 256, NUM_EPOCHS = 40, STEP_SIZE = 30

As we see from figure 20, Resnet18 achieves 0.93 in validation and test accuracy making the model more accurate of AlexNet, which has only 8 layers, of almost 10 points in percentage.

5. CONCLUSIONS

In this homework we have seen how neural networks train to learn for an image classification task, how does changing training parameters affect their learning capability, the difference between training from scratch and using a pretrained model to perform fine-tuning, training on pretrained models only the convolutional layers, training only the fully connected one, applying data augmentation techniques and finally comparing AlexNet to a bigger model.

[8] Validation Accuracy is the result of three runs. It is therefore expressed by expected value and standard deviation.