Homework 2
Marco Di Nepi 277959

Data Preparation

Caltech101 is a data set composed by 101 different object categories and a backgroud class.
In order to correctly prepare the data, it has been used the Caltech class.
This class receives the path of the folder that contains the images and the path for the file of the split (test.txt or train.txt). In the init function the class internally defines a list of classes and a list of elements. The background class, that we don't need, is filtered out through a for loop.
Another function that has been implemented is getitem; it receives an index as an integer number, load the image in a variable using the pil_load function and finally returns the image as a tensor (a set of transforms is applied since AlexNet needs a 224x224 image for input) and the corresponding label.
Finally, the method len returns the lenght of the list of elements.

Training from scratch

The next step is to split the original training set in 2 parts: training and validation. An easy way to perform this task is taking an element and alternatively putting it in one of the two sets. Doing this, train and validation have approximately half of the elements of each class and result to have the same length. This is a simple solution, a more complex idea could be using scikit-learn's train_test_split with stratify

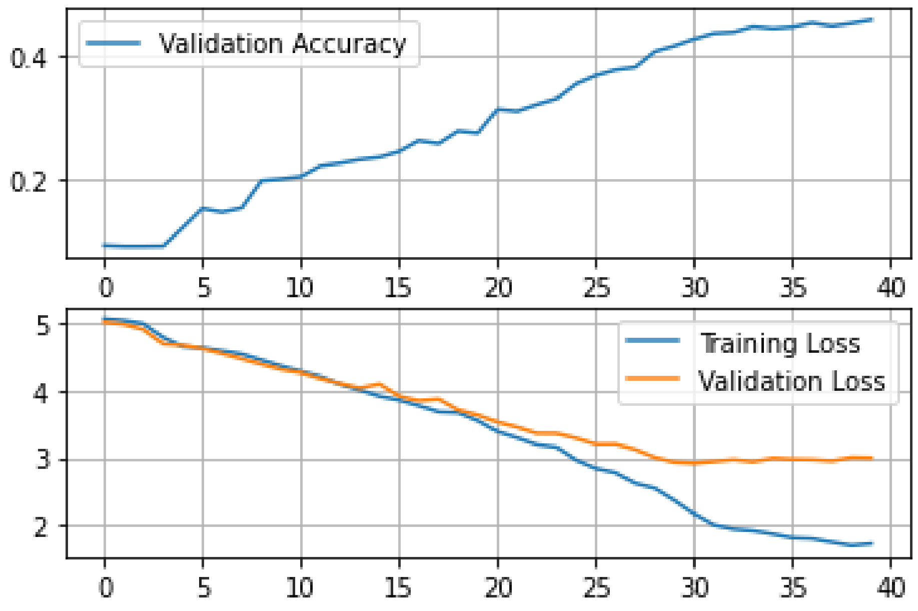
The neural network is now ready to be trained. In this homework has been used the Alexnet model, only the last fully connected layer is changed because there are only 101 classes.
During the training, at each epoch it is computed the accuracy of the model on the validation set. If the accuracy is better than the previous one the current model is copied in another model, called best_model.
After the training phase, the accuracy of the model is computed evaluting the model on the test set.
The first attempt is made by using the parameter provided by the teacher:
30 epochs with learning rate of 0.001 and step_size = 20. The batch size is 256 and the value of gamma is 0.1.
The resulting accuracy of the best model on test set is 0.15 while the final loss during the training phase is 4.5. On the other hand, using the model trained for the whole number of epochs, the results are worse, about 9%.
It is not very high, but it is a good start. To improve the results different values for the hyperparameters can be used.

The first changement is increasing the learning rate to 0.01 and the number of epochs to 40. With an higher LR the accuracy on the test test dramatically increases to 32%.
Next, I decided to increase the step_size to 30, yet to keep the higher learning rate for a larger number of epochs in order to reach the minimum loss. The obtained result is around 42% of accuracy. Again, I tried to increase the Learning rate even more (0.02) and doubling the Batch size from 256 to 512. With these hyperparameters the accuracy decreases to 25%, probably due do the fact that a batch size too big leads to poor generalization.
Afterwards, I made an attempt with a very large learning rate, like 0.5, and as expected the

model diverges.
However, a big learning rate is not always a good choice. A small LR means that is less probable to avoid a local minima but the model will converge slowly, a large LR is better if the number of epochs is low but the model could overshoot the minima or even diverge.
Lastly, I tried with a different optimizer, Adam, but the results were comparable with the ones obtained with Stochastic Gradient Descent.

In any case, looking at these numbers, it is clear that the best score achievable with AlexNet trained from scratch is around 45-48%.

The following graph shows the evolution of the validation accuracy and the training and validation losses. The latter are the average of the losses computed for each batch.



LR = 0.01 Batch_size = 256, Step_size = 30

Transfer Learning

Since caltech101 is a very small dataset, a good idea may be using a pretrained network.

net_trained = alexnet(pretrained=True)

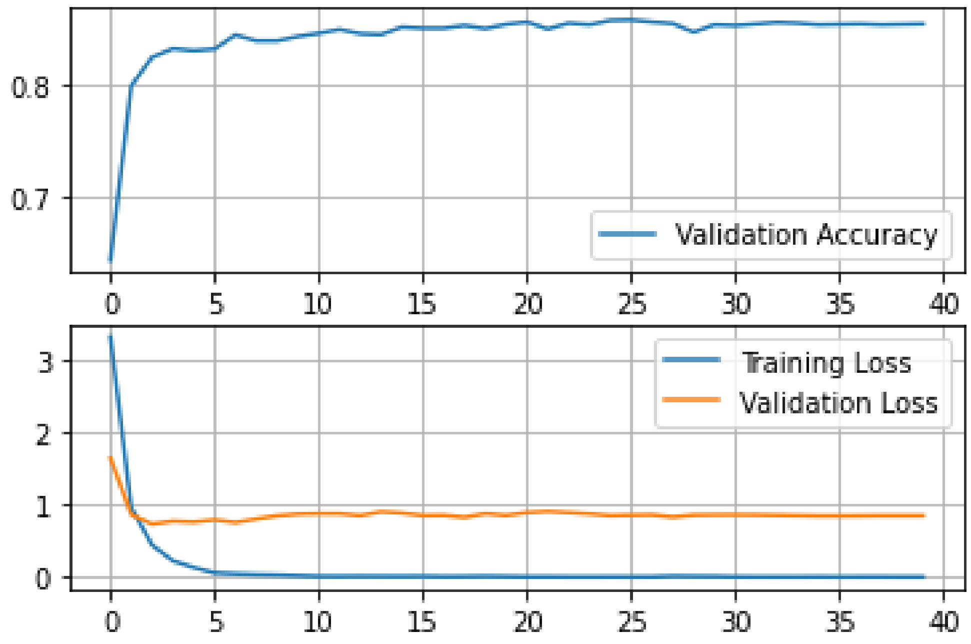
This command load an alexnet aldready trained on the ImageNet data set which is an huge set with 1000 object categories. In order to use this network, the normalization function must be changed with the mean and std of ImageNet. I found that the values are:

(0.485, 0.456, 0.406), (0.229, 0.224, 0.225)

I also defined a new optimizer and a scheduler for this new network.

Using this new pretrained network as a starting point, we can perform a new training phase and, as expected, the results are much better.

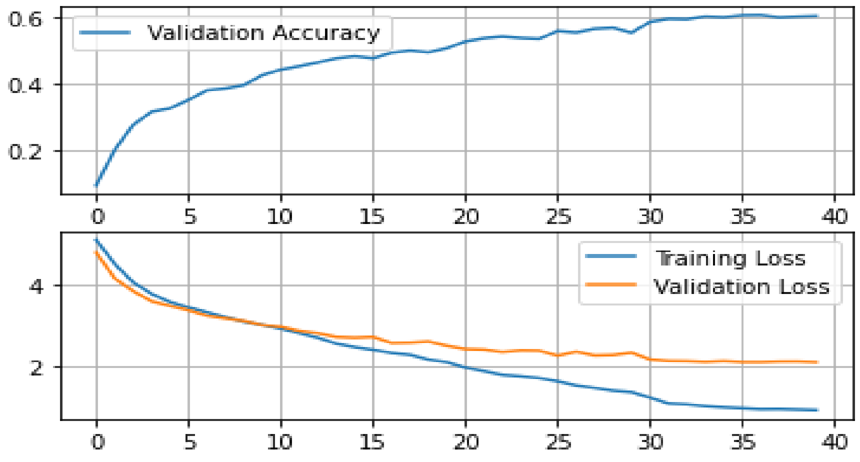
- With the default hyperparameters and 40 epochs (no significant improvements after 20 epochs) the accuracy on the test set is 0.827.
- With LR = 0.01 and 40 epochs the test accuracy is 0.842.
- With LR = 0.02, 40 epochs, step_size = 30 the test accuracy is 0.846.



LR = 0.01 Batch_size = 256, Step_size = 30

We can conclude that with fine-tuning an accuracy around 85% is achievable.

Another advantage of transfer learning is that we can also optimize some parts of the network keeping unaltered the others. To train only the convolutional layers is enough to change the optimizer from net.parameters() to net.features.parameters(). Instead, to train only the fully connected we can use net.classifier.parameters()



Training with fully connected layers frozen

In the first case the accuracy is very low, only 56%, and the training phase is also slower. In the second case, keeping the best hyperparameters found so far, we get again an accuracy about 85% and the training phase is similar to the full newtork's one.

Therefore, the training of the fully connected layers is much more crucial for the classification task while the convolutional layers can be frozen since the extracted features are already good enough. This choice is a good compromise and also makes the training faster.

Data Augmentation

Since the data set is very small, we can increase it using some transformations.
The goal of augmentation is to introduce more variability in the data and avoid overfitting.
Previously, every time the Caltech's method getitem was called, it applied a Resize and a CenterCrop.
Now we can try to use in addition other preprocessing functions that with a certain probability perform a transformation. The hyperparameters are always the same.

I report here the test accuracies with the following transformations:

- RandomGrayscale(): 84%
- RandomHorizontalFlip(0,5): 83%.
- RandomRotation(30): 83%.
- RandomErasing(): 84%.
- RandomHorizontalFlip and ColorJitter and training for 50 epochs: 82%.

For the pretrained AlexNet there are no accuracy improvements but rather a slighty decrease of it. This could be explained by the fact that if test set and training set are very similar this type of transformations may worsen the accuracy or just because the number of epochs is too small and the augmentation is not so effective.

Extra Beyond AlexNet

I made an experiment with a pretrained resnet18 without augmentation. This network is much better than AlexNet for this task. In order to use this network the last layer (the fully connected one) must be changed to another with 100 neurons.

With 30 epochs, Batch size = 256, and LR = 0.01 I obtained a test accuracy of 92%.
Similar results are achievable with 15 or 20 epochs as well since the accuracy in the validation set stop increasing.