Homework 2: Deep Learning

Sanguineti, Francisco Javier (s279265)

Professor: Barbara Caputo

Teaching Assistant: Fabio Cermelli

01TXFSM Machine Learning and Deep Learning - A.A 2019/20 - Data Science and Engineering Master's Degree - Politecnico di Torino

Contents

1 Introduction

The main purpose of this report is to describe the utilized data, the implementation choices, results, and conclusions of the second homework of the Machine Learning and Deep Learning course. The assignment consists of training a convolutional neural network (CNN) for image classification. More specifically, we will use the AlexNet CNN [3] on the Caltech-101 dataset [2]. To perform this homework, the *Python* programming language has been used, and the *PyTorch* framework. As it requires a lot of processing time, it was implemented in the *GoogleColab* platform.

2 An overview of the data

As it has been mention before, the dataset used in this experience is the Caltech-101 dataset [2]. It consists of pictures of objects belonging to 101 classes and a background class. Each image is about 300x200 pixels and is labeled with a single object. Each class contains between 40 and 800 images, giving a total of 9,146 images.
To provide an arbitrary criterion to select the training and testing data, two files describing this were given with the assignment. One file for the training set and other for the test set, consisting of a list of paths to the images.

2.1 A brief analysis of the class distribution

To know more about the Caltech-101 dataset, an histogram counting the number of images that belong to each class was created. This was also done for the files containing the training set and the test set. For a better comparison of them, on the x-axis, we have ordered the classes by their name and then assigned their corresponding index.

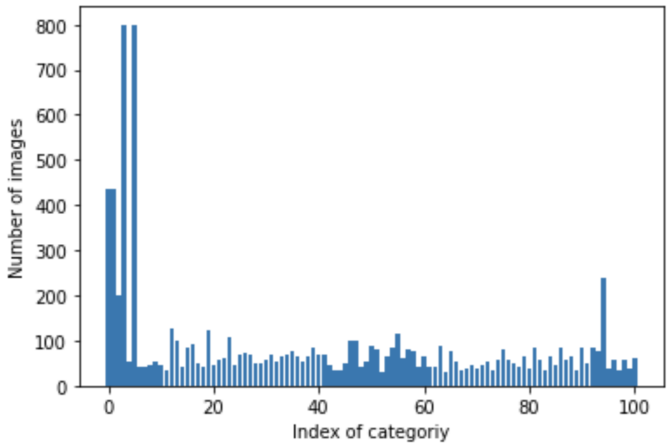

Figure 1: Illustration of the class distribution of the whole dataset.

Looking at Figure 1, it is clear that there is a severe unbalance of the data.

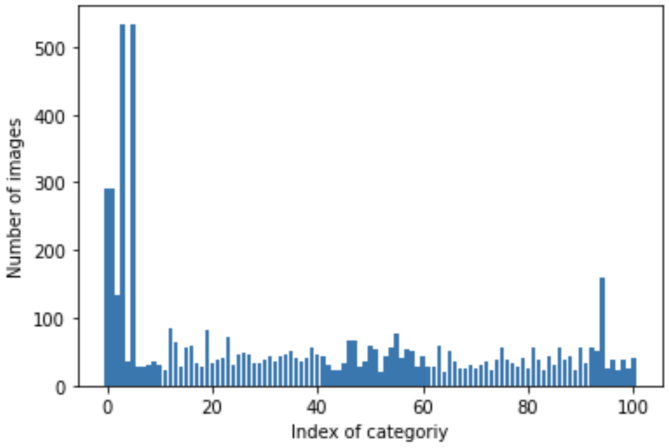

Figure 2: Illustration of the class distribution of the training set.

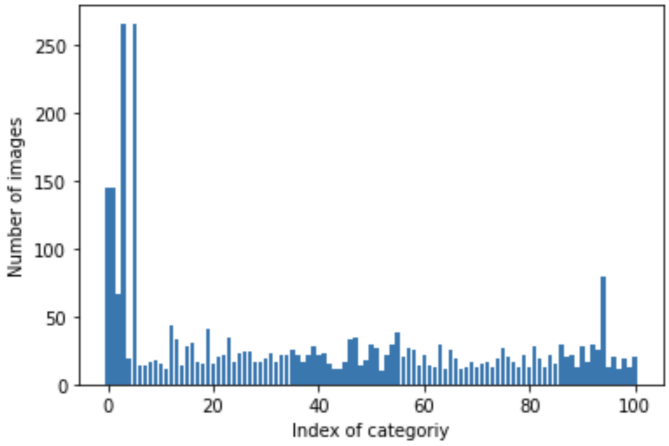

Figure 3: Illustration of the class distribution of the test set.

Looking the Figures 1, 2 and 3 we can notice that these sets were made in a controlled way to preserve the distribution of the class.
The total number of images that are available in the dataset is 8677 (we have to remove some that were in the background class), the total images in the training set is 5784 and 2893 for the test set, giving a relation of two-third to training and one-third to test. It is not a large dataset, it could fit in the main memory of the computer.

3 Methodology

This is a description of how the assignment's tasks are stipulated. The goal will be to train a convolutional neural network (CNN) that can identify objects in images and find the one that generalizes better.

1. **Data preparation:** Consists of how we are going to manage our dataset.

2. **Training from scratch:** This task is about the optimization of hyper-parameters of the network, leaving all the internal weights being initialized randomly, that is why it is called training from scratch (TFS).

3. **Transfer learning:** This a common machine learning technique, which consists of reusing a developed model for a task as the starting point for a model on a second task. In this task we will use this method to train a network.

4. **Data augmentation:** Another important factor of machine learning is the data pre-processing. In this task we have to perform training with different sets of data transformation.

3.1 Training procedure

For those stages in which the training of a model is needed, we will use the data already prepared, with some variation in its pre-processing in some cases, and we will follow the following procedure:

1. The model is initially fitted on a training dataset.

2. Successively, evaluating the fitted model, making it predict the responses for the observations in a second dataset called the validation dataset.

3. Choose others parameters and repeat 1. and 2. trying to find those who work best on the validation set.

4. The final model is evaluated on the test set to provide an unbiased evaluation.

3.2 Evaluating a model

Evaluation metrics explain the performance of a model. An important aspect of evaluation metrics is their capability to discriminate among model results. The metrics were calculated every certain number of batches or steps, and for its visualization the *TensordBorad* toolkit was used.

3.2.1 Accuracy

In the majority of this experience we are going to use the **accuracy** as a metric to evaluate the performance of a model. Accuracy is defined as the ratio of the number of correct predictions to the total number of input samples. The higher the accuracy, the better a model (unless the model has overfitted the training data).

3.2.2 Loss

Another useful metric is the loss. Unlike accuracy, the loss is not a percentage. It is a summation of the errors made for each sample in training or validation sets. In our case, we make use of the **cross entropy loss**.
Loss value implies how well or poorly a certain model behaves after each iteration of optimization. Ideally, one would expect a reduction of loss after each, or several, iterations.

4 Dataset preparation

To be able to use the dataset with the provided files, an ad-hoc dataset class was developed. It was designed to read the list of images paths of the selected file and load all the images to the memory for speeding up the training.

4.1 Splitting the dataset

One point to keep in mind is how we are going to divide our dataset. We will need a part of our data dedicated to train our models, and another part to evaluate them. Moreover, one of our goals is to make our models be able to generalize correctly responding to input samples that has never seen before. Evaluating the model with the same data that was used for training is not useful, because it rewards models that can "remember" the training data, as opposed to generalizing from it. A common strategy to solve this is to split the data in three subsets: **training**, **validation** and **test**. The general definition and main purposes of these subsets are described below:

- **Training set:** The subset of the data used to fit the model. Depending on the comparison result and the specific learning algorithm used, the model parameters are adjusted.
- **Validation set:** The subset of the data used to provide an unbiased evaluation of a model fit on the training dataset while tuning the model's hyperparameters.
- **Test set:** The subset of the data used to provide an unbiased evaluation of a final model fit on the training dataset.

As we saw, there is a file that refers to the training set and another to the test set that each maintain the class distribution. Therefore, since we already have a good proportion in the test set given by the file, we have to split the training set into two subsets: training and validation sets. The proportions given by the assignment are 50% and 50% respectively, but remembering that the training and tests sets are keeping the same distribution as the entire dataset, we must find a way to maintain this.
The method used to accomplish this task is **Stratified Sampling**, which splits a sample or population ensuring that there will be a selection from each sub-group (or categories) and avoiding the possibility of omitting a subgroup leading to sampling bias.

5 Training from scratch

In this task, we have to train a given convolutional neural network (AlexNet) over all its parameters by using the Caltech-101 dataset. The network's parameters are randomly initialized that is why its called training from scratch (TFS).

5.1 Network construction and preparation

This network was built using the AlexNet's model, with random initialization of its parameters, and replacing its last layer of 1000 output for a new fully connected layer with 101 outputs due to the number of categories in the Caltech-101 dataset. We will train on all its parameters, those of the fully connected layers and those of the convolutional layers. The optimizer of these is going to be the stochastic gradient descent (SGD), its parameters are: momentum equal to 0.9, weight decay equal to 5e-5, and the learning rate. Also we define a scheduler that dynamically changes the learning rate by providing two parameters, gamma, and step size, which multiplies the learning rate by gamma every step epochs. A brief explanation of the hyper-parameters:

- Learning rate (LR): This value can drastically influence the learning capacity of a neural network. If it is too high, the training will not converge, resulting in a network that "remembers only the last thing you saw." On the other hand, with a too low learning rate, the network learns very slowly and in some cases, the weights can get stuck at a local minimum of the loss function.

- Epochs: One epoch is when an entire dataset is passed forward and backward through the neural network only once.

- Step size: To prevent the issues due to the value of the learning rate, this is modified during the training step by decreasing it every "step size" epochs. The more knowledge the net has, the more slowly the network converges to the optimal value.

- Gamma: This is the factor in which the learning rate is multiplied every "step size" epochs.

- Batch size: Total number of training examples present in a single batch.

5.2 Data preparation

For this task, we used the data with a brief pre-processing given by the assignment. It consists of resizing the images to 256x256 pixels, then applying a central crop of 224x224, converting it to a tensor, and finally normalizing it.

5.3 Experiments and results

Various experiments were performed with different parameter configurations to find out how these influence the network and also to stochastically find the model that obtains the best score in the validation set. These are reported below.

1. **LR: 0.001, number of epochs: 30, step size: 20, gamma: 0.1, and batch size: 256.**

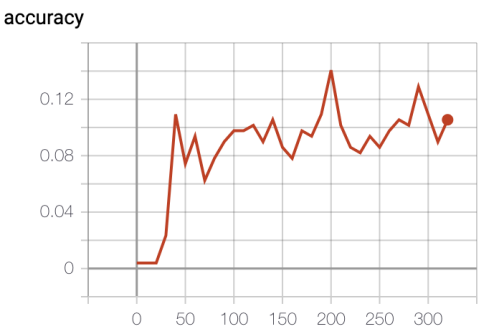

Figure 4: Training accuracy by steps of the TFS experiment 1.

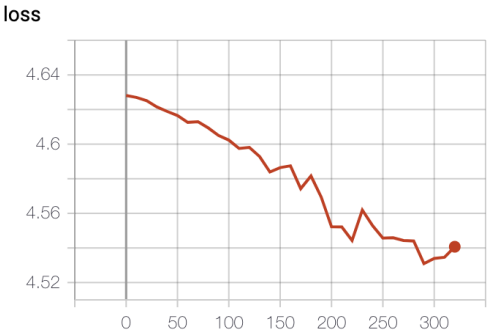

Figure 5: Training loss by steps of the TFS experiment 1.

This one is the one provided by the assignment, consequently it has been taken as a starting point. The mean accuracy reported in the validation set was 0.0958. From Figures 4 and 5 it is seen that it was not enough training.
This is not a good performance for a model. What we can conclude from these results is that it probably can learn more from the training phase. Perhaps it is a low learning rate, because the model converges too slowly.

2. **LR: 0.05, number of epochs: 30, step size: 20, gamma: 0.1, and batch size: 256.**

In this experiment the learning rate was increased compared to the previous one by a factor of 50, expecting it to learn more from the data.

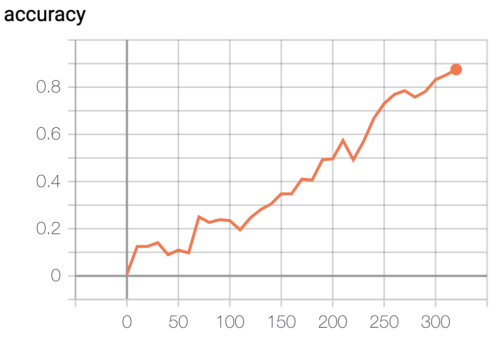

Figure 6: Training accuracy by steps of the TFS experiment 2.

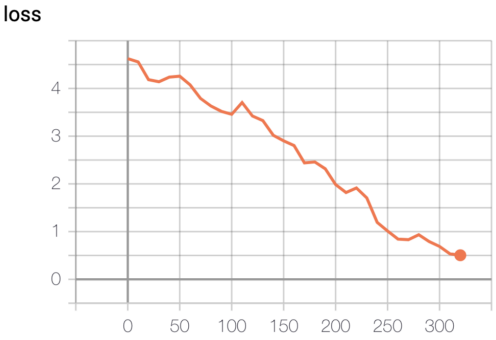

Figure 7: Training loss by steps of the TFS experiment 2.

The training stage illustrated in Figures 6 and 7 gives us a clear indication that the network gained much more knowledge of the data than before and probably could gain even more.
The mean validation accuracy was 0.4910. This could be the case of a high learning rate, because the model converges fast although this accuracy may be sub-optimal. It still is an improvement from the previous one, but now we could try to do two experiments: one with a larger LR and, if it fails, one with the same LR but with more epochs.

3. **LR: 0.1, number of epochs: 30, step size: 20, gamma: 0.1, and batch size: 256.**
   This is the experiment in which the LR is increased even more than in the second experiment.

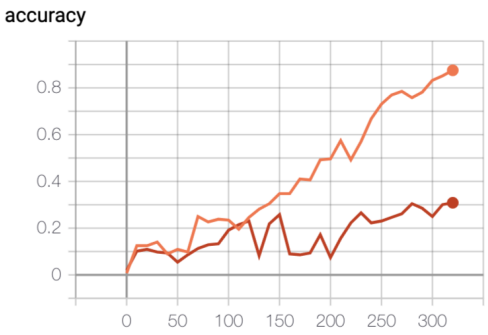

Figure 8: Comparison of training accuracy by steps of the TFS experiments 2 (orange) and 3 (red).

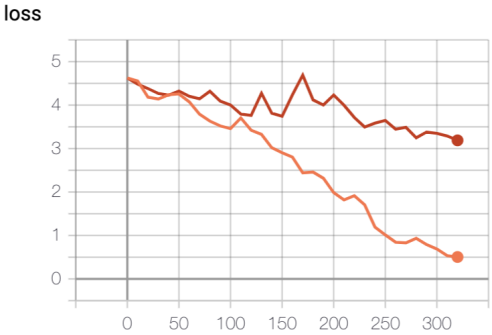

Figure 9: Comparison of training loss by steps of the TFS experiments 2 (orange) and 3 (red).

For a better comparison, the accuracy and loss of this model and the previous one were plotted together on Figures 8 and 9. It can be seen that in the first 200 steps of the model, the loss was oscillating, giving an indication that it is a too high learning rate. Its mean validation accuracy was 0.3241. Probably the top of the LR parameter was reached in the previous one.

4. **LR: 0.05, number of epochs: 120, step size: 80, gamma: 0.1, and batch size: 256.**
   This is the experiment in which the number of epochs is increased more than in the second experiment.

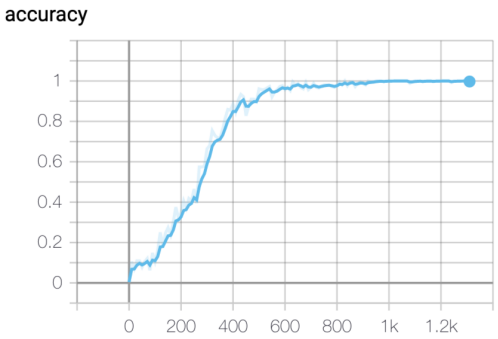

Figure 10: Training accuracy by steps of the TFS experiment 4.

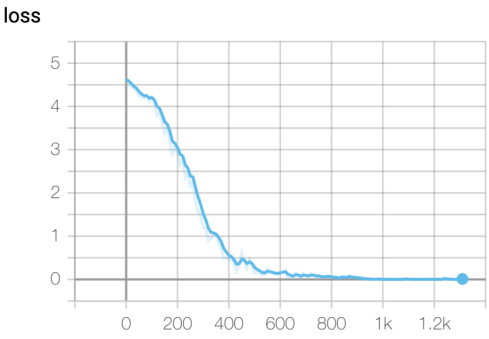

Figure 11: Training loss by steps of the TFS experiment 4.

By Figures 10 and 11 we can conclude is that this model was severely overfitted as it has a perfect performance in the training data, but its

validation score does not reflect better learning, as it only reaches an average of 0.4810.

5. **LR: 0.025, number of epochs: 120, step size: 80, gamma: 0.1, and batch size: 128.**
   To look for better hyper-parameter, a new configuration was created. It was derived from dividing by 2 the LR and the batch size of experiment 2. An empirical heuristic suggests that when changing the batch size the learning rate should change by the same factor to have comparable results.

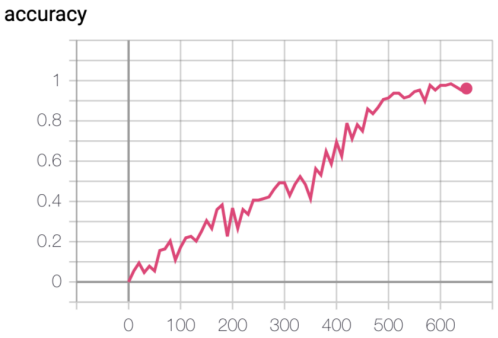

Figure 12: Training accuracy by steps of the TFS experiment 5.

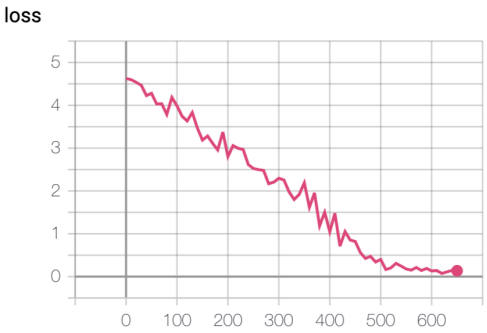

Figure 13: Training loss by steps of the TFS experiment 5.

Figures 12 and 13 shows the training of this model. It seems like it reaches an overfitting, with more accuracy and less loss than the experiment 2. It got an accuracy mean score of 0.5484 in the validation set, giving us the best score in this analysis. This model was selected to be tested in the test dataset and it reaches an accuracy of 0,5541.

6 Transfer Learning

This deep learning technique allows us to use a neural network trained for one task and apply it to another domain. Transfer learning (TL) is useful when you have insufficient data for a new domain and there is a large pre-existing data pool that can be transferred to your problem.
As mentioned, most categories only have 50 images, which is generally not enough for a neural network to learn with high precision. Therefore, instead of building and training a CNN from scratch, in this task, a pre-built and pre-trained model is used applying transfer learning.
This task is divided into three parts or sub-task: one training the whole network, other training only the fully connected layers, and the other part training only the convolutional layers.

6.1 Network construction

This network is similar to the one of the previous task. It is the AlexNet's architecture, with its parameters (weights) initialized with the ones trained for ImageNet [1], and replacing its last layer for a new fully connected layer with 101 outputs with its parameter randomly initialized. The optimizer and scheduler are exactly the same as the ones in the Training from the scratch task, only that in each part, different subsets of the parameters are going to be optimized.

6.2 Data preparation

The data preprocessing just consists of resizing the images to 256x256 pixels, then apply a central crop of 224x224, convert it to a tensor, and finally normalize it with ImageNet's the mean and deviation standard values.

6.3 Training the whole network

The training has been run with 3 different sets of parameter values as it is stipulated in the assignment. Then the best one is used in the next sub-tasks.

6.3.1 Network preparation

We will train on the parameters of fully connected layers and the ones of the convolutional layers. Consequently, we configure the optimizer to adjust them.

6.3.2 Experiments and results

1. **LR: 0.025, number of epochs: 30, step size: 20, gamma: 0.1, and batch size: 128.**
   This is the one that performs better in the previous task, consequently it is has been taken as a start point. The mean accuracy reported in the validation set was 0.751383126. It seems that transfer learning makes a huge difference from training from the scratch.

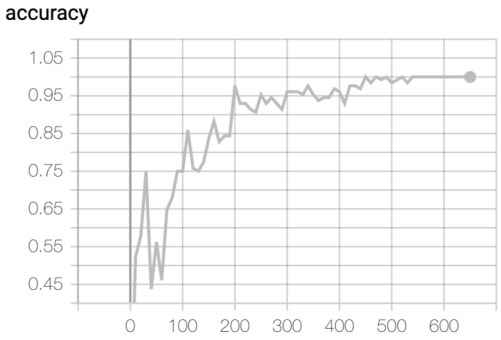

Figure 14: Training accuracy by steps of the TL experiment 1.

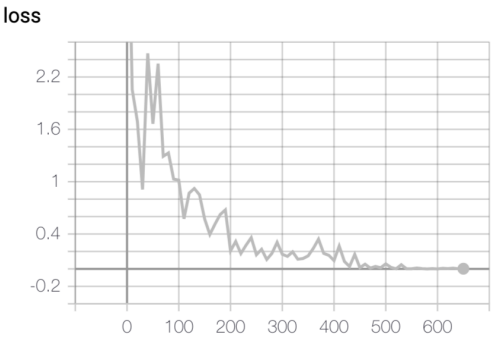

Figure 15: Training loss by steps of the TL experiment 1.

   This is a good performance for a model. Some oscillations in the loss are seen in Figure 15. Perhaps, decreasing the learning rate or increment the batch size could help to solve this. Also it seems overfitted due to many epochs of training.

2. **LR: 0.025, number of epochs: 10, step size: 6, gamma: 0.1, and batch size: 256.**
   It was decided to increment the batch size to 256 in order to control better the loss. The number of epochs was decreased so as not to over-train the network.

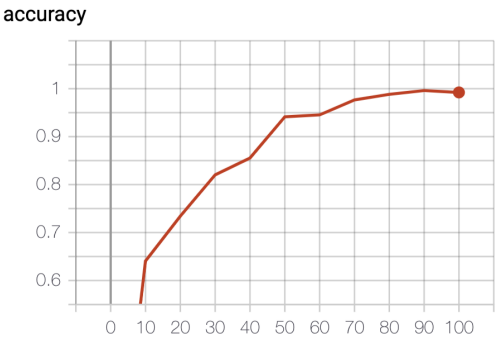

Figure 16: Training accuracy by steps of the TL experiment 2.

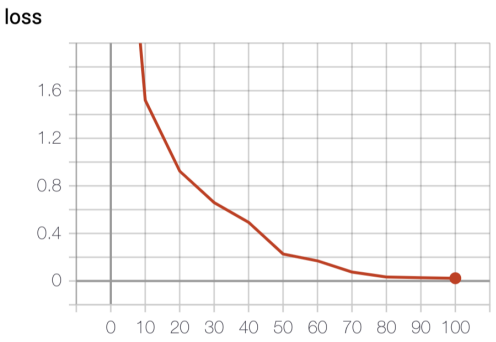

Figure 17: Training loss by steps of the TL experiment 2.

Another good improvement in reference to the previous one. Its mean validation accuracy was 0.8082. It is appreciated in Figures 16 and 17 that the curves are almost in its asymptotic. In only a third of the times, this model was able to achieve incredible results that would not be possible when training from scratch with this dataset.

3. **LR: 0.02, number of epochs: 10, step size: 2, gamma: 0.1, and batch size: 256.**
   The best configuration was found while reducing a little bit the learning rate, and the step size, in order to have a longer training without overfitting.

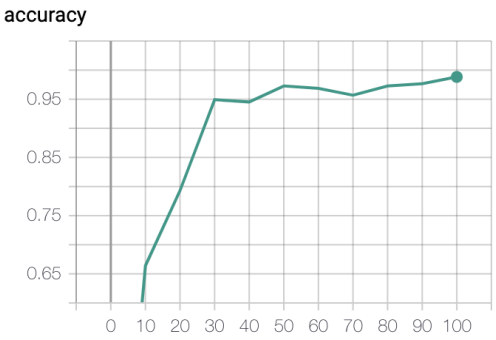

Figure 18: Training accuracy by steps of the TL experiment 3.

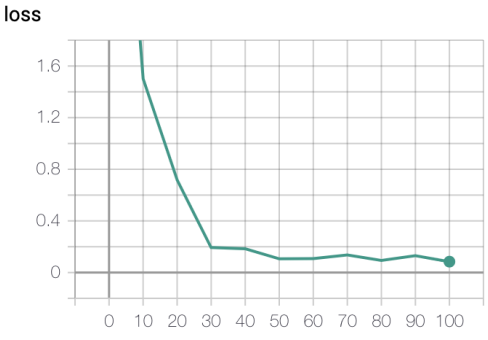

Figure 19: Training loss by steps of the TL experiment 3.

In the training phase, showed by Figures 8 and 9, we can observe that it reach quickly a good learning that the one in experiment 2, and also it is not so overfitted. The mean accuracy on the validation set was

0.8220. It is a bit better than the previous one, so it was the best score in this training analysis.
This model was selected for the next sub-tasks, and it got an accuracy in the test set of 0,8176.

6.4 Training only the convolutional layers

In this sub-task only the convolutional layers weights are trained. The hyper-parameters are those of the best score of the section 6.3: Learning rate 0.02, number of epochs 10, step size 2, gamma 0.1, and batch size 256.

6.4.1 Network preparation

We will train on the parameters of convolutional layers, and the optimizer then is configured to tune only them, freezing all the fully connected layers.

6.4.2 Experiment and results

This type of training is not so common because it is freezing parameters with random initialization in the fully connected layers, so it is like the convolutional layers have to learn how to deal with them. The last convolutional layers are those who are going to be more influenced by this due to the vanishing gradient problem, and maybe in a long training they could overcome it.

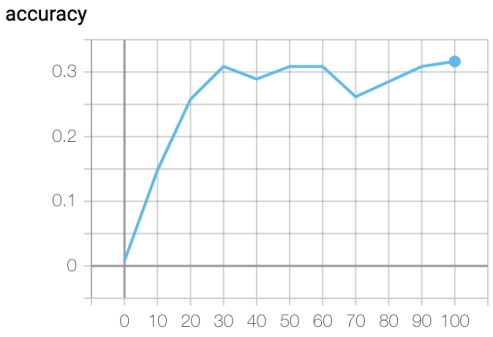

Figure 20: Training accuracy by steps of the transfer learning and tuning the convolutional layers.

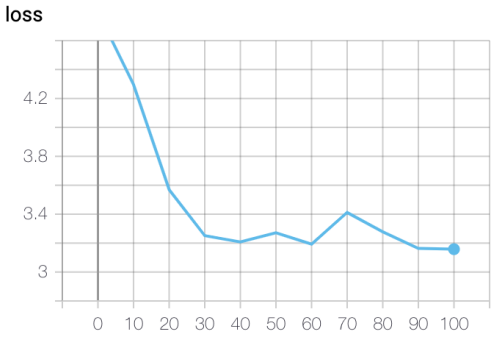

Figure 21: Training loss by steps of the of transfer learning and tuning the convolutional layers.

The metrics of a training phase are shown in Figures 20 and 21, by observing them we can realize that this was a short train for this model, and also that it could learn more from the data with a larger learning rate. The mean

accuracy in the validation set was 0.2988. Certainly, this type of training would cost more computational time than the others in this section.

6.5 Training only the fully connected layers

In this sub-task we will just train over the weight of the fully connected layers, and the hyper-parameters are those of the best score of the section 6.3: Learning rate 0.02, number of epochs 10, step size 2, gamma 0.1, and batch size 256.

6.5.1 Network preparation

We will train on the parameters of fully connected layers, and the optimizer then is configured to tune just them, freezing all the convolutional layers. In fact, when a layer is frozen, its weights cannot be modified further. This technique may cut down on the computational time for training while losing not much on the accuracy side.

6.5.2 Experiment and results

The metrics of a training phase is showed in Figures 24 and 25.

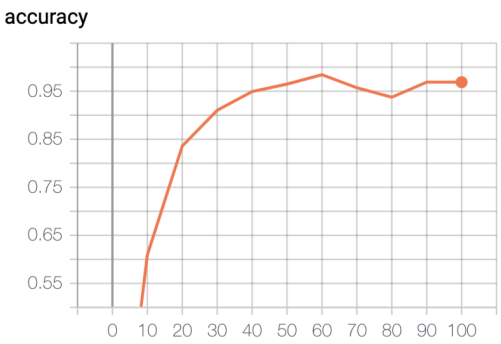

Figure 22: Training accuracy by steps of the transfer learning and tuning the fully connected layers.

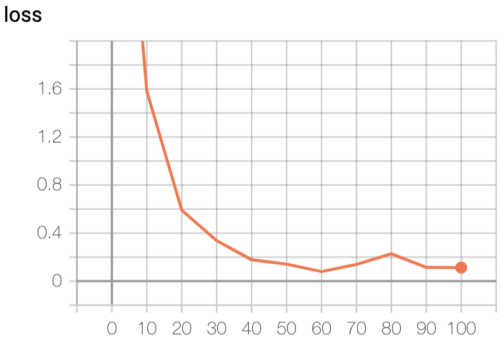

Figure 23: Training loss by steps of the of transfer learning and tuning the fully connected layers.

This configuration got a mean validation accuracy of 0.8184, so we can realize that by training only the last fully connected layers the models can be trained faster and preserving overfitting.

7 Data Augmentation

The idea behind data augmentation is to artificially increase the number and the diversity of data that we can use in the training stage, applying transformations to the images. So the idea is that we want to expand our data by "creating" some similar data to simulate situations in the test stage.
To make our model able to generalize better and not choose a category because it recognizes some patter that it "saw" in the training stage, we can use this method, executing some transformations, also at random, to always train our model with different images.
This task is composed of three training, each one with different sets of preprocessing for training images.

7.1 Augmentation techniques

There are lots of different methods and tricks that can be used to augment the data. In this section, several common and useful transform methods will be introduced below.
There are lots of different methods and tricks that can be used to augment the data, we will discuss those functions that are encapsulated in *Pytorch*, specifically the package: $torchvision.transform$. In this section, several common and useful transform methods will be introduced below.

7.1.1 Crop

This is a common and simple transformation. It crops an image by a given size, or in the center if it is defined. It also can be randomly cropped with parameters that specify the size, it is called randomly resized. If the crop size is larger than the original size, it will then output an image with the larger size and add paddings around the original image.

7.1.2 Normalize

It aims to normalize the images (or tensors) by channels (e.g. red, blue, and green), with a specific mean, std.

7.1.3 Flip

There are two commonly used flip transformations: the one that flips the image vertically and the one that flips the image horizontally. The probability that this happens could be a parameter.

7.1.4 Rotation

This transformation rotates the image by angle. It can help our model become robust to the changes in the orientation of objects. Also, it has a lot of configurations as the center of rotation, also zoom or expansion of the image. It will add padding if it is necessary.

7.1.5 Color Jitter

It can change the brightness, contrast, and saturation of an image. Also it can be set with a random probability that occurs.

7.2 Network construction and preparation

As from here, the best model found was in the previous task, the one in the section 6.3, the AlexNet's architecture, with its parameters initialized with the ones trained for ImageNet and replacing its last layer for a new fully connected layer with 101 outputs with its parameter randomly initialized. The optimizer was configured to tune over all the parameters of the network.

7.3 Data preparation

The sets of preprocessing for training images for this task are detailed in each experiment. All of them have the last part that consists of applying a central cut of 224x224, converting it into a tensor, and finally normalizing it with the mean and standard deviation of the ImageNet dataset.

7.4 Experiments and results

Following, the experiments and the results of different preprocessing sets are detailed.

1. **Preprocessing set 1:** The preprocessing is composed by a random resized crop of size 256 and scale between 80 and 100 percentage, followed by a random rotation with an angle of maximum 20 degrees, and finally a random horizontal flip.
   A first training with hyper-parameters used in section 6.3 was performed. The reason that they were selected is for a better comparison with the previous analysis. Then the hyper-parameters of this part are: Learning rate 0.02, number of epochs 10, step size 2, gamma 0.1, and batch size 256 (a).

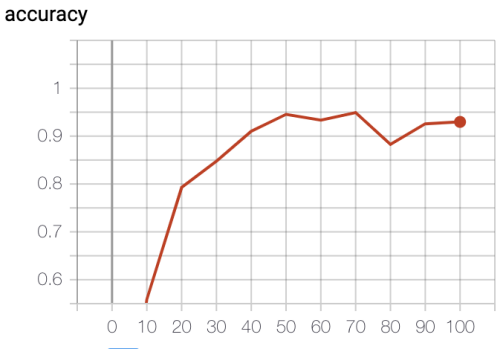

Figure 24: Training accuracy by steps of the DA experiment 1.a.

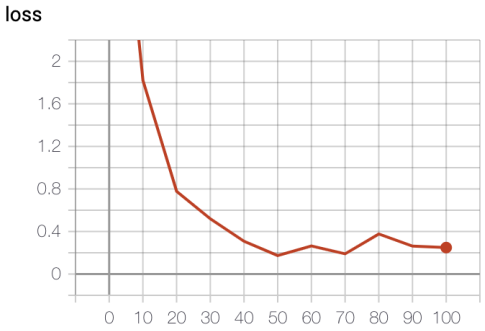

Figure 25: Training loss by steps of the of DA experiment 1.a.

This model had a mean accuracy on the validation set of 0.8067. Analyzing the Figures 24 and 25 we can realize that the model is not overfitting, it could learn more. Therefore more experiments were performed increasing the number of epochs. The next training is performed with the following parameters: Learning rate 0.02, number of epochs 20, step size 4, gamma 0.1, and batch size 256 (b).

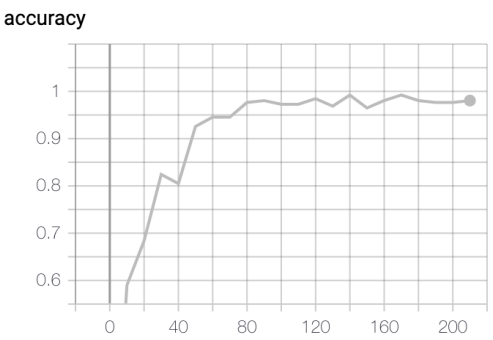

Figure 26: Training accuracy by steps of the DA experiment 1.b.

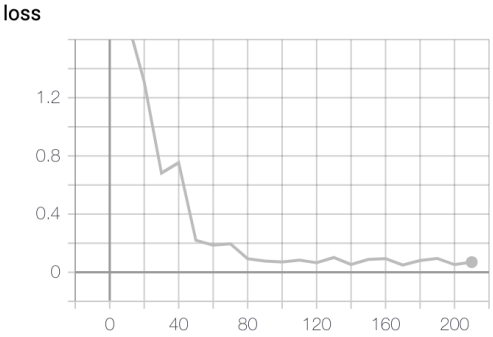

Figure 27: Training loss by steps of the of DA experiment 1.b.

We can see a progress in respect to the previous one as it got a mean accuracy of 0,8191 on validation, but in comparison with the normal pre-processing in other sections, it seems like it is not improving much more than the others. So, a logarithmic scale was made in order to find what the problem is. It is appreciated from Figure 28 that the loss is oscillating (in this logarithm scale), consequently, more experiments have been conducted.

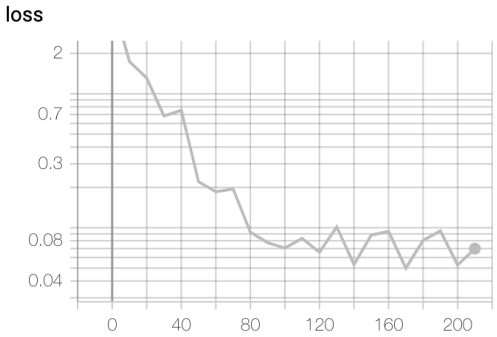

Figure 28: Training loss by steps in a logarithmic scale of the of DA experiment 1.b.

We tried to drop even more the loss with different values of the gamma hyper-parameter and leaving the others constant.

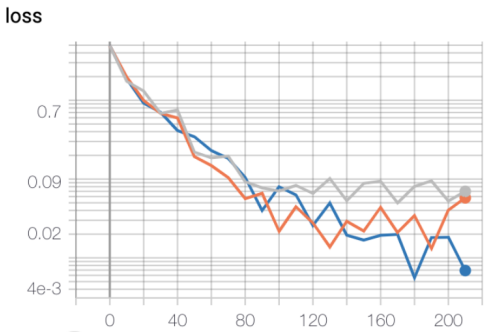

Figure 29: Training loss by steps in a logarithmic scale of the of DA experiment 1.c, in grey gamma 0.1, in orange gamma 0.3, and in blue gamma 0.5.

In Figure 29 we can see the comparison of the previous experiment and two new values of gamma. It is found that with this increase of gamma the model can learn a bit more. The mean validation accuracy of the model with gamma equal to 0.5 is 0.8283, which is the greatest in this experiment.

2. **Preprocessing set 2:** This pre-processing is similar to the previous one adding a color jitter transformation after the random rotation.

The training was performed with the best hyper-parameter found in the previous experiment: Learning rate 0.02, number of epochs 20, step size 4, gamma 0.5, and batch size 256.

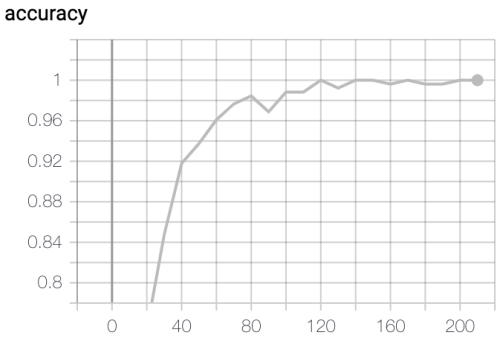

Figure 30: Training accuracy by steps of the DA experiment 2.

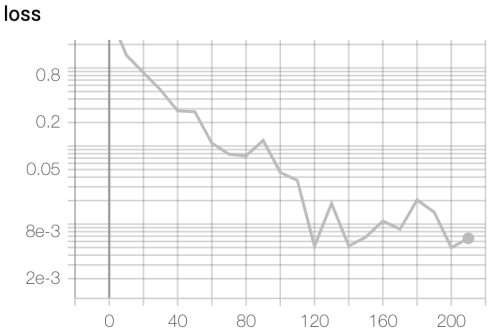

Figure 31: Training loss by steps in a logarithmic scale of the of DA experiment 2.

There was an good improvement, the mean accuracy on the validation is 0.8390. There was even less loss, showed in Figure 31. What we tried to do in this experiment was to reduce the rotation and add some variations in the colors of the image.

3. **Preprocessing set 3:** This pre-processing is similar to the previous one but without the rotation transformation, and the hyper-parameters are the same: Learning rate 0.02, number of epochs 20, step size 4, gamma 0.5, and batch size 256.

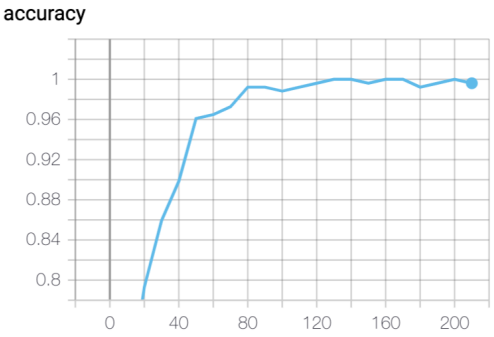

Figure 32: Training accuracy by steps of the DA experiment 3.

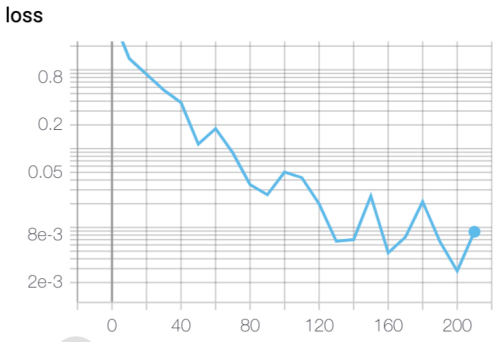

Figure 33: Training loss by steps in a logarithmic scale of the of DA experiment 3.

In Figure 33, we can see that the loss has not dropped more compared to the previous model. This model got a mean validation accuracy of 0.8502, a really good improvement. This model was selected to be tested against the testing set and it got a mean accuracy of 0.8220, making this one the best result.

8 Conclusion

What we can conclude from the experiments and analysis of training from scratch, is that the main problem encountered could be that the model is not able to generalize, caused by a dataset not large enough. For this reason, transfer learning could increase a lot the learning of a model that performs a new task, if the domains are similar.
When applying data augmentation, better results were found, but we have to be cautious in what transformation we perform because the model could be trained using one pre-processed set that changes too much that data and does not correspond to the test or real data.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, *ImageNet: A Large-Scale Hierarchical Image Database*, CVPR09, 2009.

[2] Li Fei-Fei, Rob Fergus, and Pietro Perona, *Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories*, Computer Vision and Pattern Recognition Workshop (2004).

[3] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in Neural Information Processing Systems 25 (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), Curran Associates, Inc., 2012, pp. 1097–1105.