Machine Learning and Artificial Intelligence
Homework 3

Matteo Borghesi *(s268199)*

January 9, 2020

1 Introduction

In this report we will explore a classification problem using different domains. This is a particularly hot topic in deep learning research and has several application in real life situations where we have limited or no access at training time to the domain (or distribution) where the test images are taken from. This requires a domain adaptation step, which involves a different architecture of the network that allows not only to classify input images in the source domain, but also to transfer this capability to the target domain.
In order to tackle the issue we will use the PACS dataset, that contains overall 9991 images belonging to 7 classes and 4 domains: Art painting, Cartoon, Photo, Sketch. Our goal will be to train our network on images from the Photo domain and then to be able to correctly classify samples from the Art domain.

2 Network architecture

We will use for our task a modified version of Krizhevsky's AlexNet. In particular, we will have a parallel branch of fully connected layers after the convolutional section. The purpose of this branch is to tell whether an image comes from the source or from the target domain. However, in the training phase we will proceed the other way around, trying to confuse the network so that it is not able to tell the difference between the domains: in this way during classification we can indifferently provide images from any domain to the network and expect that they are classified correctly, although we have trained the classification using images from only one domain.
Clearly, the fully connected layers from the original net will output a 7-dimensional vector, corresponding to the classes in the dataset, while the parallel branch will output a 2-dimensional vector, since it plays the role of a binary classifier (telling if an image comes from the source or from the target domain).

3 Baseline model

We will start seeing how a traditional AlexNet performs in case of a domain adaptation problem. This means that we will train the network on images coming from the source domain (*Photo*) and afterwards we will test it using images from our target domain (i.e. Art painting), without giving any clues to the network about the difference between the domains. In order to tune the hyperparameters of the network, we will use the other two domains to perform validation.
The hyperparameters we are going to play with are the step size and the number of epochs. The step size defines the number of epochs after which the learning rate is decreased: the smaller the step size, the faster the learning rate is updated, bringing to smaller optimization steps. The number of epochs defines how many times we will walk through the entire dataset in the course of the training phase. The other hyperparameters are fixed and have the following values:

- Batch size: 256
- Weight decay: $5 \cdot 10^{-5}$
- Gamma: 0.1

They define respectively the number of samples passed through the network at each step before updating the gradients, the regularization parameter and the decreasing factor of the learning rate.

For each of the two validation domains we perform a grid search across two different values of the hyperparameters *step size* and *number of epochs.* The results of the validation phase are shown in tab. 1 for the *cartoon* domain and in tab. 2 for the *sketch* domain.
In order to properly select the best hyperparameter set, we pick the results obtained on the two domains and choose the set that performed at best on average. As it is shown in tab. 3, The best score is obtained with step size=30 and epochs=10. Finally, we test the network on the *Art painting* domain, which delivers an accuracy of 49.31%.

	Epochs=15	Epochs=30
Step size=5	17.7 %	16.6 %
Step size=10	24 %	22.3 %

Table 1: Comparison of different hyperparameter sets on the *cartoon* domain

	Epochs=15	Epochs=30
Step size=5	23.8 %	20.05 %
Step size=10	20.9 %	24.79 %

Table 2: Comparison of different hyperparameter sets on the *sketch* domain

	Epochs=15	Epochs=30
Step size=5	20.75 %	18.32 %
Step size=10	22.45 %	23.54 %

Table 3: Average of the validation results between the *cartoon* and *cartoon* domain

4 Implementation of domain adaptation

In this section we will discuss how it is possible to implement the domain adaptation scheme presented in sec. 2. In the first place we define a class 'DANN' which implements the network, much like the class 'AlexNet' in PyTorch. This class contains three instances of nn.Module, which represent the blocks of the network: the first will contain the convolutional layers, while the other two will have the fully-connected layer. With regards to the original AlexNet, we drop here the pooling layer after the convolutional block.
As a second step, we want to set the initial weights of the network as in the version of the AlexNet pretrained on the ImageNet dataset. For the main body of the network it is sufficient to load the parameters by means of the function *load_state_dict* provided from the superclass nn.Module. To populate the weights and the bias of the parallel branch, we perform a deep copy of the values contained in the original fully connected layers.
Finally, we have to modify the forward function in order to allow to take the parallel branch when needed. Likewise, the backward function has to propagate the gradient from the parallel branch back to the convolutional layers. Furthermore, in this case we have to invert the sign of the gradient in order to 'confuse' the binary classifier, so that the network handles images from both domains in the same way. All this is done by means of an additional class provided with two static methods used from the forward function of the *DANN* class.

5 Training with domain adaptation

After implementing the support for domain adaptation, we are able to execute the task presented in the beginning of the report. In the training phase, our task is to optimize the parameters both of the classifier and of the discriminator (i.e. the binary classifier). In order to do the last, we use the images from the *cartoon* and *sketch* domains. Specifically, each training iteration consists of three step:

- training the classifier using images from the source domain,
- training the discriminator using images from the source domain,
- training the discriminator using images from the target domain.

Clearly, each step will come up with a different loss and a different gradient to update the neurons in the network. Only at the end of each iteration, after all three steps have been executed, the optimizer looks at the gradients obtained and updates the weights accordingly.

Unfortunately, the dataset contains a different number of images for all domains (1670 for *photo*, 2048 for *art painting*, 2344 for *cartoon* and 3929 for *sketch*). Since the batch size is fixed, this means that it takes less iterations to traverse the source domain with reverse to the validation domains. The issue can be solved in (at least) two ways: we can perform the smallest number of iterations between all domains or we can let the network run only on one domain after the other one has run out of images. In order to have more balance in the training of the network, we choose the first alternative: in this way each epoch will consists of 6 batch iterations containing 256 samples.

6 Domain adaptation results

As before, we need to finetune the hyperparameters of the nework. In particular, the gradient propagation from the parallel branch is controlled by a weighting parameter *alpha*, which needs to be set manually. Therefore we perform a grid search between the step size from sec. 3 and three different values of *alpha*, that are 0.2, 0.5 and 0.8.
As before, we validate the network on the domains *cartoon* and *sketch* in order to find the best hyperparameter set. The results for both domains are shown respectively in tab. 4 and tab. 5. The average of these results is indicated in tab. 6

	alpha=.2	alpha=.5	alpha=.8
Step size=5	23.8 %	51.08 %	16.62 %
Step size=10	30.25 %	16.79 %	16.53 %

Table 4: Comparison of different hyperparameter sets on the *cartoon* domain

	alpha=.2	alpha=.5	alpha=.8
Step size=5	17.25 %	16.92 %	29.29 %
Step size=10	17.39 %	26.35 %	19.58 %

Table 5: Comparison of different hyperparameter sets on the *sketch* domain

	alpha=.2	alpha=.5	alpha=.8
Step size=5	20.52 %	34 %	22.95 %
Step size=10	23.82 %	21.57 %	18.05 %

Table 6: Average of the validation results between the *cartoon* and *cartoon* domain