POLITECNICO DI TORINO

Master’s degree in Computer Engineering

Artificial Intelligence and Machine Learning

HOMEWORK 3
DEEP DOMAIN ADAPTATION

Student:
Enrico Loparco, s261072

Professor:
Tatiana Tommasi

ACADEMIC YEAR 2019-2020

Table of Contents

0. Introduction

This homework deals with Deep Domain Adaptation, that is, how to handle a model trained on a certain dataset, but used on another dataset, belonging to a different distribution.
Usually, when this happens, the accuracy drops drastically: here comes the domain adaptation problem, related to the reduction of such performance degradation.
This homework uses PACS dataset and tries to compare the use of traditional techniques and domain adaptation ones.

1. The Dataset

The dataset, published in 2017, consists in 4 domains: Art painting, Cartoon, Photo and Sketch.
Each domain is further divided into 7 categories: ‘dog’, ‘elephant’, ‘giraffe’, ‘guitar’, ‘horse’, ‘house’, ‘person’.
The total number of images is 9991, not equally distributed among the 4 domains.

For the purpose of the homework, the photo domain is used as source dataset, art painting images are part of the target dataset, while the other two domains are intended to be validation datasets.

In the image below, dataset sizes are reported:

```
Train Dataset (photo):          1670
Test Dataset (art_painting):    2048
Validation Dataset 1 (cartoon): 2344
Validation Dataset 2 (sketch):  3929
```

Figure 1: Sizes of the datasets

Images form the source and target datasets are showed (from the notebook).
They are plotted after the transformation defined in the preprocessing phase is applied: each input is resized and centrally cropped to fit the network input size, then normalization is applied, since the model used is pretrained on ImageNet.

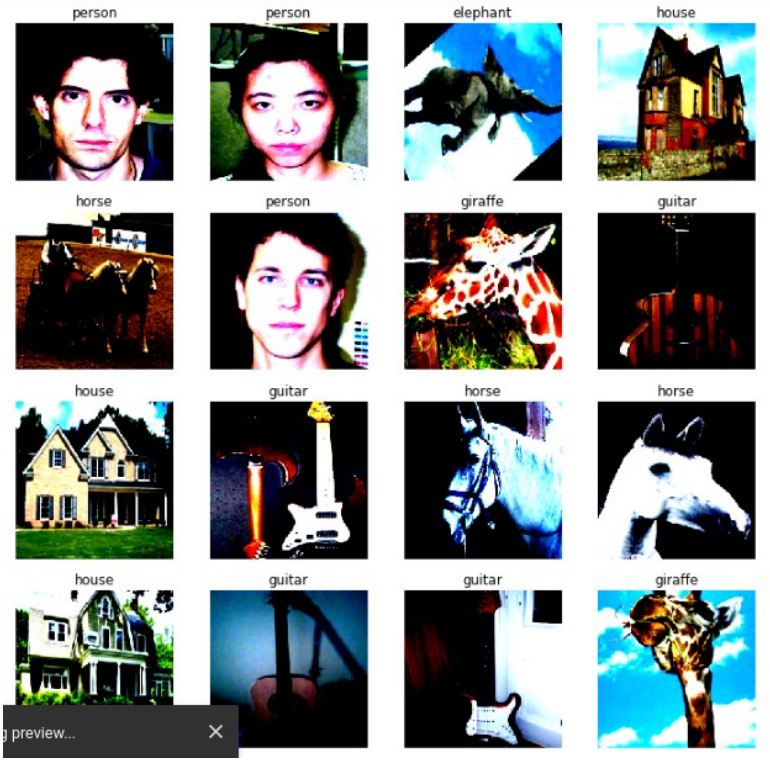

Figure 2: Images from source dataset (photo)

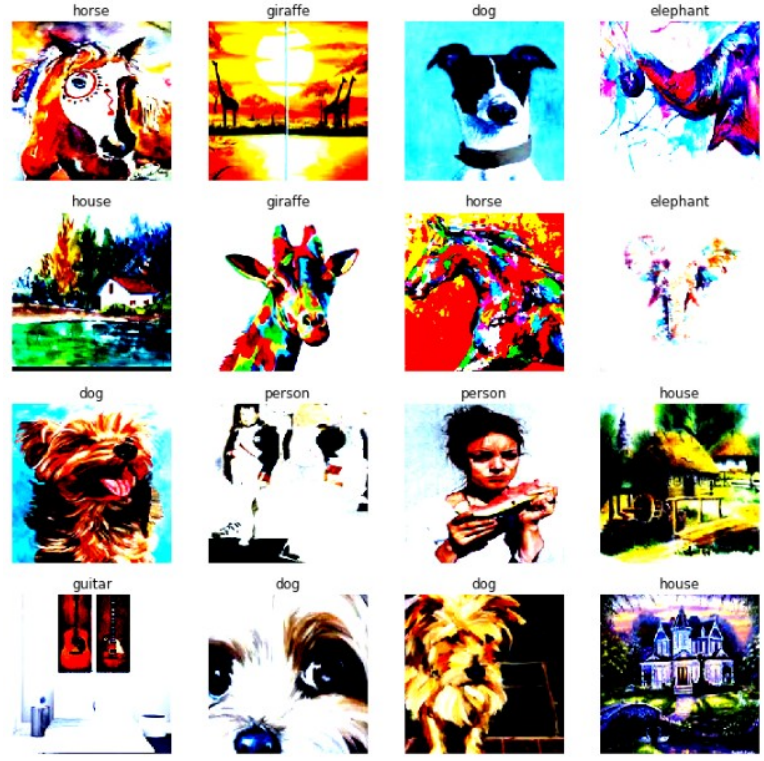

Figure 3: Images from target dataset (art painting)

2. Implementing the Model

The Domain Adaptive Neural Network is implemented starting from AlexNet.
A new branch is added to this basic network, to predict the domain label. The model is preloaded with ImageNet weights, to get a good starting point.
The domain branch is obtained by copying the class branch and modifying the number of outputs with the number of domains (two, one for source and one for target).

The last step consists in modifying the forwarding function, responsible for the forward propagation through the network.
The parameter *alpha* represents how much the gradient obtained from the domain classification is weighted during the back-propagation). In case *alpha* is passed to the forwarding function, an additional operation is performed: the gradient is made negative and is multiplied by alpha.

In this way, if alpha is not passed, the label predictor is used, otherwise the domain classifier is in charge of predicting the domain and propagate the loss back.
The image below shows the final architecture:

```
DANN(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=7, bias=True)
  )
  (dann_classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=2, bias=True)
  )
)
```

Figure 4: DANN architecture

3. Train and Test without Validation

In this part of the homework, validation is not used. As starting parameters, the ones performing well in the previous homework are adopted. A random value is chosen for *alpha*.

3A. Without Adaptation

A learning rate of 0.001 results in a final train loss of about 0.013 (check the figure below for the loss trend).
The procedure to train the network is the usual as in the previous homeworks, since only the convolutional and the fully connected layers for class prediction are exploited (without using the newly added fully connected layers for domain prediction).

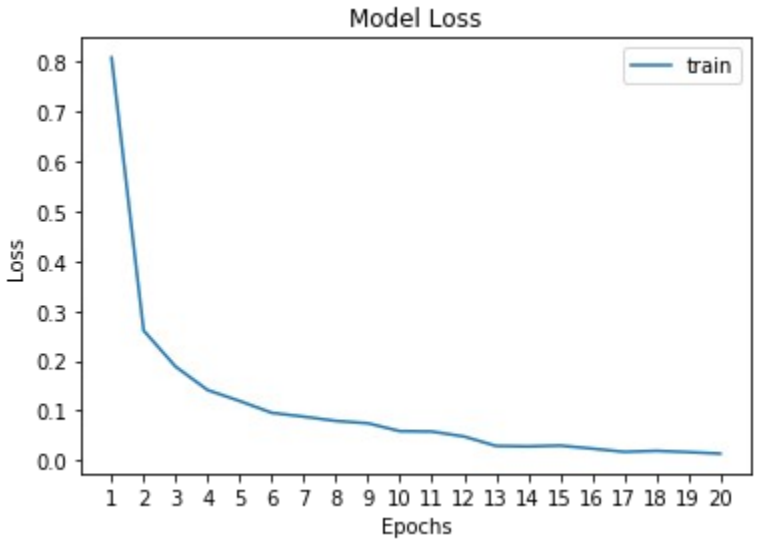

Figure 5: Loss during training without adaptation (and without validation)

The test accuracy reached is about 0.49.

3B. With Adaptation

With adaptation, we expect to get better results, but since no validation has been performed yet, a random value of *learning rate* and *alpha* are used: respectively 0.001 and 0.01

To get the training right, three steps need to be performed: training on source labels by forwarding source data, training of the discriminator by forwarding source data and, finally, training of the discriminator by forwarding the target data.
The objective of the discriminator is to reduce the gap between the two domains (source and target) by training the network in not being able to discriminate well between the two of them.

The three losses are plotted: one for source label prediction, one for source domain prediction and the last one for target domain prediction: the first one uses the label classifier, while the last two use the domain discriminator branch.

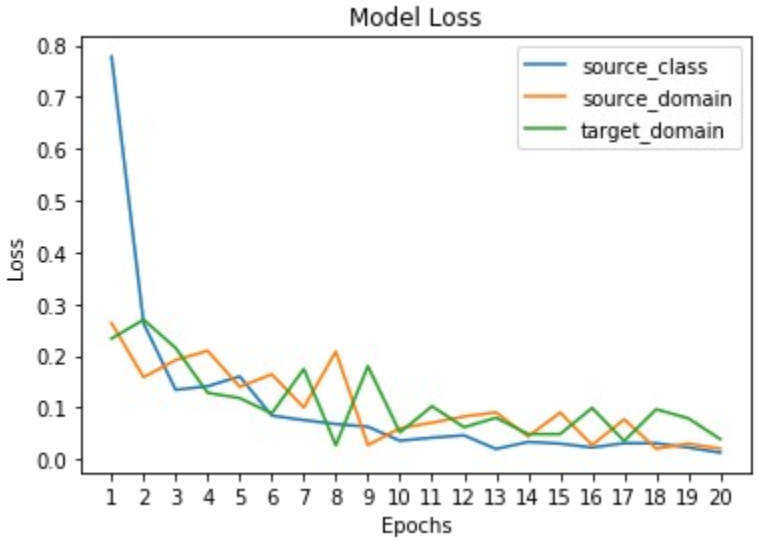

Figure 6: Loss during training with domain adaptation (with validation)

The final accuracy is about 0.49, similar to one obtained during training without domain adaptation, but this is just a coincidence (because no validation to get the best parameters has been performed yet).

4. Train and Test with Validation

This time, validation is used to search for the best parameters, then training is performed again with the best discovered values.

4A-B. Without Adaptation

As opposite to the usual validation process, this time we have two different datasets for validation (cartoon and sketch): they are used separately and then their accuracies are averaged, in order to choose the best parameters.
The values checked for the learning rate are 0.01, 0.005, 0.001, 0.0005, 0.00001.
In the table, the best validation reached with each parameter is reported (more details can be found in the notebook).

	lr = 0.01	**lr = 0.005**	**lr = 0.001**	**lr = 0.005**	**lr = 0.0001**
Best valdation accuracy	0.34	0.3	0.22	0.25	0.32

The best value for the *learning rate* seems to be 0.01.

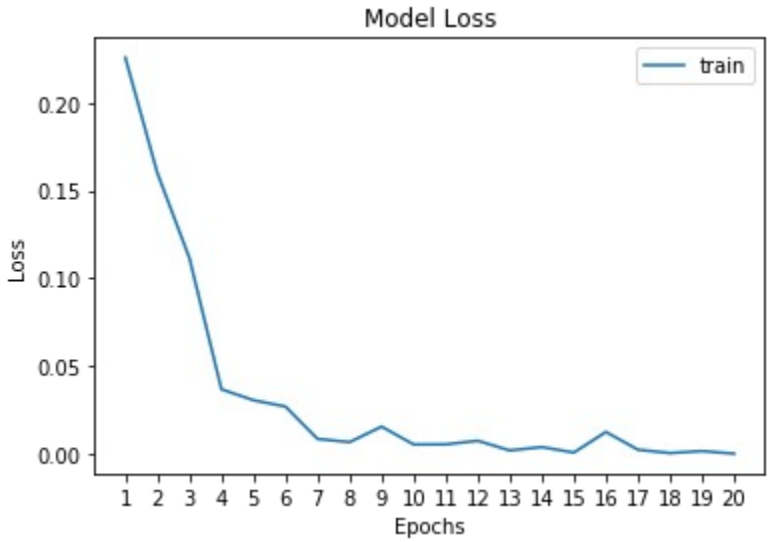

Figure 7: Loss during training without domain adaptation (with validation)

Using it, we obtain the loss curve in the image above.
Validation has increased test accuracy from 0.49 to about 0.51 (a small but significant improvement).

4C-D. With Adaptation

While for the other parts of the homework 20 was used as number of epochs, in this validation step a smaller value is used (8), since the validation is slower/more expensive.

In fact, this time, as in the point 4A, we have to validate against both cartoon and sketch datesets. Since the training phase is also dependent on the target dataset used (our validation sets in case of validation), two models need to be trained in parallel for each combination of hyperparameters: one uses cartoon and the other sketch as target datasets. As in the previous validation phase, accuracies are averaged.

The set of hyperparameters selected (for the *learning rate* and *alpha*) and the related accuracies (in the last epoch) on the validation dataset are reported below:

learning rate / alpha	**0.01**	**0.03**	**0.05**
0.01	0.24	0.31	diverges
0.005	0.24	0.27	0.3
0.001	0.19	0.18	0.2

As can be seen, with *learning rate* = 0.01 and *alpha* = 0.03 we get the best accuracy.
A DANN is trained using these parameters and the image below shows the losses with respect to the epochs:

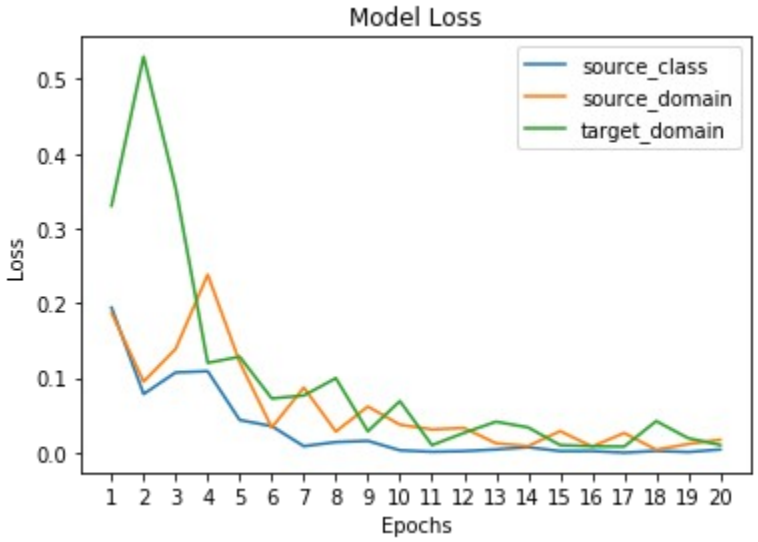

Figure 8: Loss during training with domain adaptation (with validation)

The final accuracy is about 0.5, increasing by 0.01 after the tuning.

5. Conclusion

Accuracy does not improve very much using domain adaptation. Performing many training (with different hyperparameters) the maximum accuracy reached using DANN is about 0.53.

To explain this result, different hypotheses can be made: first of all, the validation for the DANN is not very accurate, since only few epochs were used and only few hyperparameters were tested.
The reason behind it is that DANN validation is very expensive, having to train two separate models: using only 3 values for both the *learning rate* and *alpha* and only 8 epochs, the validation required about 1 hour and half.
Eight epochs are too few to get the best hyperparameters, in fact, using *learning rate* = 0.01 and *alpha* = 0.01 a good accuracy of 0.53 was reached, despite the training phase was telling that (0.01, 0.03) were better parameters.

Another possible explanation for the poor performance can be reconducted to the limited size of the datasets.