MACHINE LEARNING AND DEEP LEARNING

HOMEWORK 3

Sofia Perosin S269748

OVERVIEW

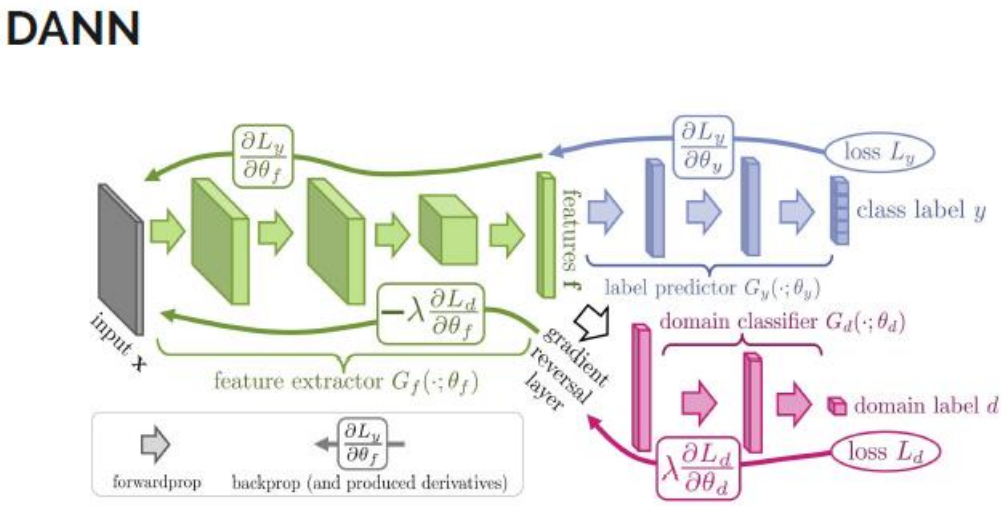
The aim of this homework is to implement a Domain Adaptation algorithm, DANN, on the PACS dataset using AlexNet.

Domain adaptation is useful because in most of the real cases, training data and testing data are from different distributions, and this is the reason why DANN looks for aligning train features and test features.

The general structure of DANN is composed by

- domain classifier, which is trained to discriminate between source and target;
- feature extractor, which tries to fool the domain classifier, so it looks for extracting features more domain invariant as possible;
- label predictor, which thanks to the features extracted by the second component, tries to classify in the correct class each data.

The general structure of DANN is the following one:

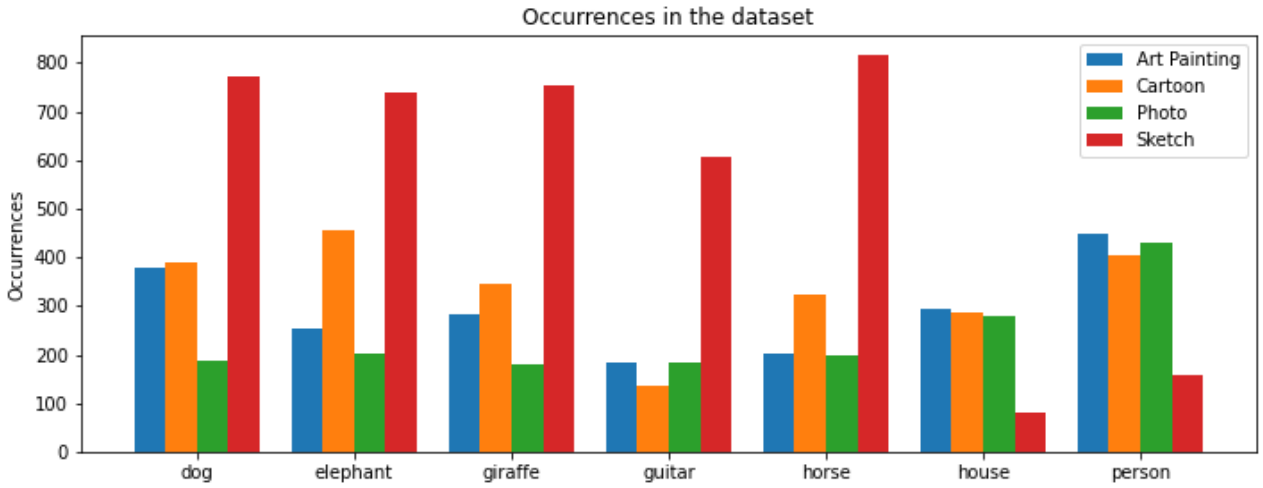


So the domain classifier (the pink one in the figure) is trained in order to be able to understand if the image belongs to the source or to the target domain. Its gradient is inverted by the gradient reversal layer and the feature extractor (the green one) looks for maximizing the error of the domain classifier, and for this reason it tries to build domain invariant features; these features will be also used to train the label predictor (the blue one), which is used for the final classification of each image.

In the implementation proposed, the domain classifier and the label predictor are two AlexNet fully connected layers, instead the feature extractor has the same structure of AlexNet Convolutional layers.

DATA EXPLORATION

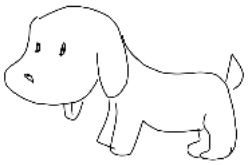
The PACS dataset contains 4 domains (“Art Painting”, “Cartoon”, “Photo”, “Sketch”), each of them contains images belonging to 7 classes (“dog”, “elephant”, “giraffe”, “guitar”, “horse”, “person”).



These are the distributions, among each domain, of the images in each class. It’s possible to see how the distributions are not very similar between the dataset, especially for the “Sketch” where the difference with respect to the other datasets is very huge, and this fact could affect the performances of the final model.

Another consideration that could be done is that since the final goal is to train on “Photo” and test on “Art Painting”, then some images belonging to dog and giraffe classes will be difficulty classified because of the difference of the cardinalities between train and test classes.

Finally it could be interesting to see how images from different domains look like:

			
From Art Painting dataset	**From Cartoon dataset**	**From Photo dataset**	**From Sketch dataset**
			
From Art Painting dataset	**From Cartoon dataset**	**From Photo dataset**	**From Sketch dataset**

Taking into account that images inside each classes of the four domains are various and different from each other, this is an example used to do a consideration: it’s possible to see how, although different, images from the first three datasets are more easily comparable than the one belonging to “Sketch” dataset.

GENERAL

Training without adaptation could be used as baseline to compare the result obtained with adaptation to see if there are some improvements.

So the model will be validate in order to find the best hyperparameters values.

Validation step is an open problem in Domain Adaptation, but something has to be done since in real life problem it's not possible to try random values to assign to the hyperparameters, also because the final results could be a disaster.

For this reason it could be useful to do the validation step on *"cartoon"* and *"sketch"* dataset, and use the best hyperparameters configuration found for the final model, although the limitations seen in the data exploration step.

The model used is AlexNet and also transfer learning is exploited by loading ImageNet weights.

HYPERPARAMETER SELECTION WITHOUT DOMAIN ADAPTATION

The model is trained on *"photo"* dataset and validated both on *"cartoon"* dataset and *"sketch"* dataset.

Many different configuration are tried, and since it's a long process, they are automatically selected according to the highest average accuracy reached on the two validation datasets.

- The configurations tried are the combination of the following values
  - BATCH_SIZE_range = [256, 320]
  - LR_range = [1e-3, 1e-4, 5e-5]
  - NUM_EPOCHS_range = [30, 40]
  - STEP_SIZE_range = [15, 20]
  - GAMMA_range = [0.1, 0.2]
  - SGD_range = [True, False][1]

Summary of some validation accuracies to underline the variation of performances according to configurations:

CONFIGURATION	ACCURACY
{'BATCH_SIZE': 256, 'GAMMA': 0.1, 'LR': 0.001,'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 15}	0.22729064834255128
{'BATCH_SIZE': 256, 'GAMMA': 0.1, 'LR': 0.001,'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.283070034928861
{'BATCH_SIZE': 256, 'GAMMA': 0.1, 'LR': 0.0001,'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 15}	0.24291438606945642
{'BATCH_SIZE': 256, 'GAMMA': 0.1, 'LR': 0.0001,'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.22306320073801444
{'BATCH_SIZE': 256, 'GAMMA': 0.1, 'LR': 0.0001,'NUM_EPOCHS': 30, 'SGD': False, 'STEP_SIZE': 15}	0.2789724239204932
{'BATCH_SIZE': 256, 'GAMMA': 0.1, 'LR': 0.0001,'NUM_EPOCHS': 30, 'SGD': False, 'STEP_SIZE': 20}	0.2462498273536154
{'BATCH_SIZE': 256, 'GAMMA': 0.2, 'LR': 0.001,'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 15}	0.22799480670988545
{'BATCH_SIZE': 256, 'GAMMA': 0.2, 'LR': 0.001,'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.21180747083253343

[1] SGD = False means that Adam optimizer is used

{'BATCH_SIZE': 256, 'GAMMA': 0.2, 'LR': 0.0001,'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 15}	0.24123819598209517
{'BATCH_SIZE': 256, 'GAMMA': 0.2, 'LR': 0.0001,'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.224494536990628
{'BATCH_SIZE': 256, 'GAMMA': 0.2, 'LR': 0.0001,'NUM_EPOCHS': 30, 'SGD': False, 'STEP_SIZE': 15}	0.26932607972397427
{'BATCH_SIZE': 256, 'GAMMA': 0.2, 'LR': 0.0001,'NUM_EPOCHS': 30, 'SGD': False, 'STEP_SIZE': 20}	0.26664636895335897

- BATCH_SIZE = 256, LR = 1e-3, NUM_EPOCHS = 30, STEP_SIZE = 20, GAMMA = 0.1, SGD = True
  This is the hyperparameters configuration which gives the highest average accuracy on the two validation datasets. The accuracy reached on "art painting dataset" is 0.501; but since the final result is subject to variability, in order to estimate the mean and standard deviation the training and test phases are repeated five times: accuracy on the test set = 0.493 ± 0.009.

HYPERPARAMETER SELECTION WITH DOMAIN ADAPTATION

The model in this case is trained on *"photo"* dataset; also *"cartoon"* dataset and *"sketch"* dataset are used to train the domain classifier. Then it is test both on *"cartoon"* and *"sketch"* dataset.

Also in this case the best configuration is selected according to the highest average accuracy reached on the two validation sets.

- The configurations tried are the combination of the following values
  - BATCH_SIZE_range = [128]
  - LR_range = [1e-3, 5e-4, 1e-4, 5e-5]
  - NUM_EPOCHS_range = [30,40]
  - STEP_SIZE_range = [20]
  - GAMMA_range = [0.1, 0.2]
  - SGD_range = [True]
  - ALFA_range=[0.1, 0.5, 0.8, 1]

  Summary of some validation accuracies to underline the variation of performances according to configurations:

CONFIGURATION	ACCURACY
{'ALFA': 0.1, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.2673505273206932
{'ALFA': 0.1, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.0005, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.29692431008767395
{'ALFA': 0.1, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.0001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.2299073268953967
{'ALFA': 0.1, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 5e-05, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.2701053772725259
{'ALFA': 0.5, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'ALFA': 0.5, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.0005,	0.2797470806473609

'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	
{'ALFA': 0.5, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.0001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.3022534913659435
{'ALFA': 0.5, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 5e-05, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.2890803007652035
{'ALFA': 0.8, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'ALFA': 0.8, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.0005, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'ALFA': 0.8, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.0001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.33516516938456233
{'ALFA': 0.8, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 5e-05, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.32390839491416325
{'ALFA': 1, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'ALFA': 1, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.0005, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'ALFA': 1, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 0.0001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.3334144764102061
{'ALFA': 1, 'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LR': 5e-05, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.3192004712843018

- BATCH_SIZE = 128, LR = 1e-4, NUM_EPOCHS = 30, STEP_SIZE = 20, GAMMA = 0.2, ALFA = 0.8, SGD = True

  This is the hyperparameters configuration which gives the highest average accuracy on the two validation datasets. With only one training and test the accuracy reached is 0.493, repeating the process five times the accuracy calculated is 0.481 ± 0.016.

So there is no improvement with the respect to the performance of the model without adaptation, on the contrary it's slightly worse, this can be improved by using a lower value of α at the early stages in order to reduce the negative consequences of the noisy signal.

According to the paper *"Unsupervised Domain Adaptation by Backpropagation"* by Ganin and Lempitsky, the adaptation factor alpha is gradually changed from 0 to 1 (in order to suppress noisy signal from the domain classifier at the early stages of the training procedure) through the formula: $\alpha_p = \frac{2}{1+\exp(-\lambda \cdot p)} - 1$ where "p" indicates the progress of the algorithm and λ is another hyperparameter that has to be tuned.

λ influences how fast the value of α will change during each step: the smaller λ the slower the growth of α.

- The configurations tried are the combination of the following values
  - BATCH_SIZE_range = [128]
  - LR_range = [1e-3, 5e-4, 1e-4, 5e-5]
  - NUM_EPOCHS_range = [30, 40]
  - STEP_SIZE_range = [20]

- GAMMA_range = [0.1, 0.2]
- SGD_range = [True]
- LAMBDA_range=[2, 5, 10, 15]

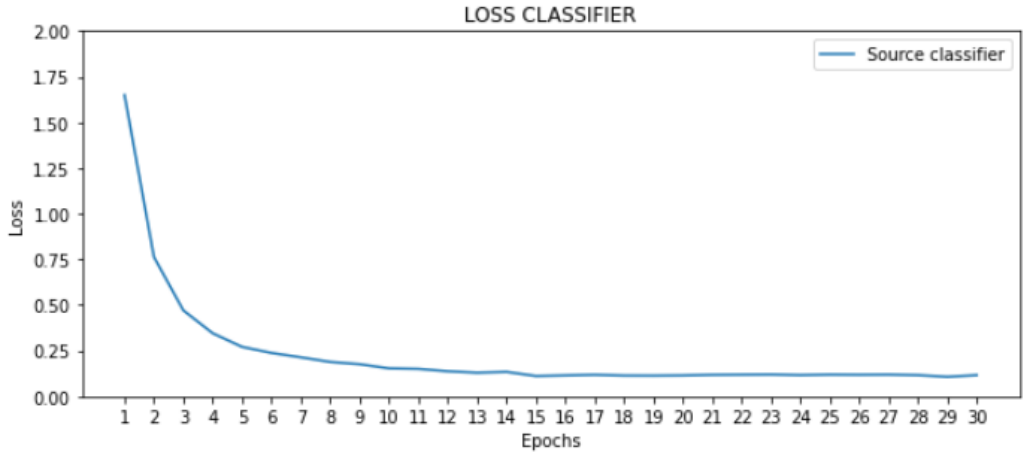
Summary of some validation accuracies to underline the variation of performances according to configurations:

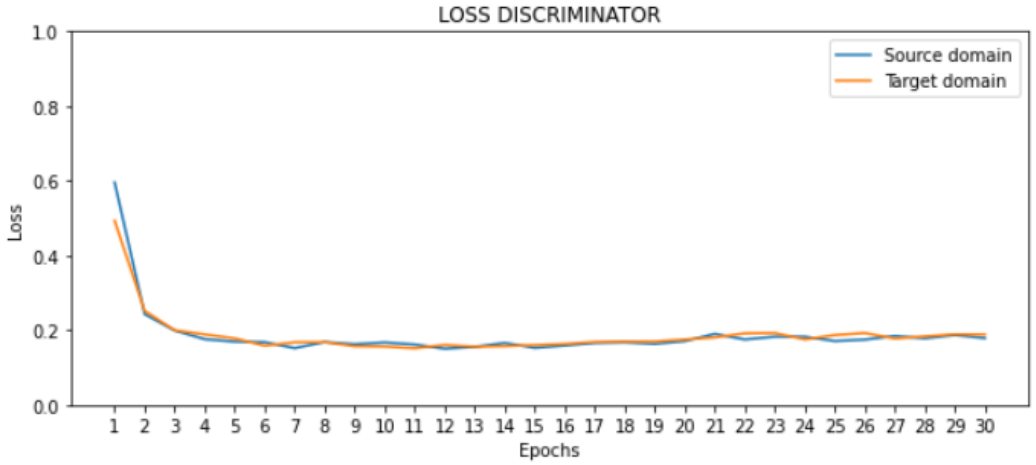
CONFIGURATION	ACCURACY
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 2, 'LR': 0.001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 2, 'LR': 0.0005, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.2296431453521856
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 2, 'LR': 0.0001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.2872465572790756
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 2, 'LR': 5e-05, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.2618590150078 57
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 5, 'LR': 0.001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 5, 'LR': 0.0005, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 5, 'LR': 0.0001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.2586547415429331
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 5, 'LR': 5e-05, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.31818994707248194
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 10, 'LR': 0.001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 10, 'LR': 0.0005, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 10, 'LR': 0.0001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.42314108705981684
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 10, 'LR':5e-05, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.32802644768879696
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 15, 'LR': 0.001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 15, 'LR': 0.0005, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.18122164364570095
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 15, 'LR': 0.0001, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.3413500252346036
{'BATCH_SIZE': 128, 'GAMMA': 0.2, 'LAMBDA': 15, 'LR': 5e-05, 'NUM_EPOCHS': 30, 'SGD': True, 'STEP_SIZE': 20}	0.30013374122760916

- BATCH_SIZE = 128, LR = 1e-4, NUM_EPOCHS = 30, STEP_SIZE = 20, GAMMA = 0.2, LAMBDA = 10, SGD = True

In this case the accuracy reached on the test set *"Art Painting"* is 0.505 with one run, 0.493 ± 0.0099 with five runs, so there is not the big improvement expected.

It could be interesting also to plot how the different losses, calculated during the training phase, varies among the different epochs:





It's possible to see how the loss of classifier decreases during the training. The loss of discriminator doesn't not increase, except for some small peaks, and this is not wrong: it's true that the feature extractor tries to fool the discriminator (and this should causes a growth of discriminator loss), but at the same time the discriminator itself is still trying to minimize its own loss.

Although what has been done, by adopting domain adaptation no significantly improvements have been seen.

FINAL CONSIDERATION

According to what observed, all the modifications adopted didn't change in a significant way the final result. So for this reason, I asked myself if this was due to the fact that *"photo"* and *"art painting*" datasets have two similar distributions and so domain adaption couldn't lead to big improvements.

For this reason I've tried the model built before testing it on *"cartoon"* datasets instead of *"art painting*" (it's still trained on *"photo"* dataset, and in the case of domain adaptation is used *"cartoon"* dataset to train the discriminator):

- WITHOUT ADAPTATION the accuracy reached is 0.311
- WITH ADAPTATION (weight of reversed backpropagation fixed) the accuracy reached is 0.507
- WITH ADAPTATION (weight of reversed backpropagation dynamically scheduled) the accuracy reached is 0.572

So, how it's possible to see, in this case the domain adaptation approach leads to a significant improvement (especially with alpha changed dynamically).