Homework 3: Deep Domain Adaptation

Sanguineti, Francisco Javier (s279265)

Professor: Barbara Caputo

Teaching Assistant: Mirco Planamente

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

Contents

1 Introduction

When developing a new machine learning task, frequently, there is an obstacle that is related to the lack of labeled data. This is what domain adaptation methods intend to solve, it gives the chance of obtaining training sets that are big enough called source, but with the disadvantage, that the data distribution is different from the data targeted. Consequently, for made use of it, the common supervised learning assumption that the training and testing data are drawn from the same distribution is violated, a classifier trained on the source domain will likely experience a drop in performance when tested on the target domain due to the differences between domains. The underlying concept of deep domain adaptation is to bridge the gap between source and target domains so that a supervised classifier trained on labeled source data can be well transferred to the target domain.
The data analysis, implementation choices, results, and conclusions of the third homework of Machine Learning and Deep Learning course are detailed in this report. The assignment consists of implementing a domain-adversarial neural network on the PACS dataset, using as a base a CNN known as AlexNet.
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 Backgroud

2.1 Transfer learning

The focus of this assignment is domain adaptation. Domain adaptation can be viewed as a special case of transfer learning [160], we first review transfer learning to highlight the role of domain adaptation within this topic. Transfer learning is defined as the learning scenario where a model is trained on a source domain or task and evaluated on a different but related target domain or task, where either the tasks or domains (or both) differ.

2.1.1 Notations and definition of transfer learning

Formalising, transfer learning can be formulated in terms of domain and task. Let $\mathcal{D}$ be the domain, which is composed by: a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{x_1, ..., x_n\} \in \mathcal{X}$. In general, if two domains are different, then they may have different feature spaces or different marginal probability distributions. Given a specific domain $\mathcal{D} = \{\mathcal{X}, P(X)\}$, a task $\mathcal{T}$ is composed by two elements: the output or label space $\mathcal{Y}$ and an objective predictive function $f(\cdot)$, and it is denoted by $\mathcal{T} = \{\mathrm{Y}, f(\cdot)\}$. The predictive function is learned from the training labeled data consisting of pairs $\{x_i, y_i\}$, where $x_i \in X$ and $y_i \in Y$, and it is used to make predictions on unseen instances $x \in \mathcal{X}$. From a probabilistic point of view, $f(x)$ can be written as $P(Y \mid X)$, therefore, the task could be also denoted as $\mathcal{T} = \{\mathrm{Y}, P(Y \mid X)\}$.
Considering the case where there is just one source domain $\mathcal{D}_S$ and a learning task $\mathcal{T}_S$, one target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$, transfer learning aims to help improve the learning of the target predictive function $f_T(\cdot)$ in $\mathcal{D}_T$ using the knowledge gained in $\mathcal{D}_S$ and $\mathcal{T}_S$, where $\mathcal{D}_S \neq \mathcal{D}_T$, or $\mathcal{T}_S \neq \mathcal{T}_T$.
There are two main transfer learning tasks: *transfer across domains:* the data distributions between both domains are different, while the tasks are the same; and *transfer across the tasks*: both data distributions and task are different.

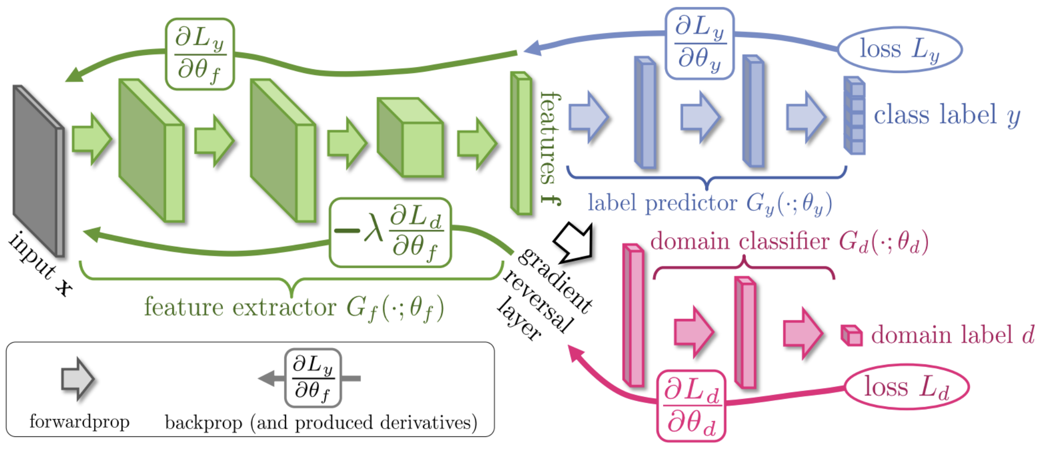
2.1.2 Domain Adaptation

This is one of the transfer learning task cases introduced as transfer across domains, and it is the focus of the assignment. In domain adaptation, the target task remains the same as the source, but the domain marginal probability distributions differ. Domain adaptation can also occur in *one step* (one-step domain adaptation), or through *multiple steps*, traversing one or more domains in the process (multi-step domain adaptation). In this report,

as the assignment stipulates, only one-step domain adaptation will be discussed.
Also, depending on the data that is available from the target domain, domain adaptation can further be classified into *supervised* that refers as the case in which both labeled source and target data are available, *semi-supervised* in the case in which labeled source data in addition to some labeled target data are available, and *unsupervised* as the case in which both labeled source data and unlabeled target data are available. Then, an unsupervised domain adaptation learning algorithm is provided with a labeled source sample $\mathcal{S}$ drawn i.i.d. from $\mathcal{D}_S$, and an unlabeled target sample $\mathcal{T}$ drawn i.i.d. from ${\mathcal{D}_T}^X$, where ${\mathcal{D}_T}^X$ is the marginal distribution of $\mathcal{D}_T$ over X. What this method aims is to build a classifier $h : X \to Y$ with a low target risk without having information about the labels of $\mathcal{D}_T$.

2.1.2.1 Domain-Adversarial Neural Network

DANN was first proposed in [2], it is a neural network that tries to achieve unsupervised domain adaptation by using adversarial training. This type of domain adaptation is known as **Adversarial-based Domain Adaptation**. The network can be divided into the following three parts: a feature extractor G_f, a label predictor G_y, and a domain classifier G_d. The domain classifier is a binary classifier that aims to discriminate between source and target. The adversarial relationship is between the feature extractor and the domain classifier. This type of network contains two losses, the label predictor loss, and the domain confusion loss. It is aimed to minimize the label predictor loss for the source samples and the domain confusion loss for all samples (while maximizing the domain confusion loss for the feature extraction), to be sure that the samples are mutually indistinguishable for the label predictor.



$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features $\mathbf{f}$
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
input $\mathbf{x}$
feature extractor $G_f(\cdot;\theta_f)$
gradient reversal layer
domain classifier $G_d(\cdot;\theta_d)$
domain label d
$\frac{\partial L_d}{\partial \theta_d}$
loss L_d
forwardprop
$\frac{\partial L_y}{\partial \theta_f}$
backprop (and produced derivatives)

Figure 1: The proposed architecture of DANN, includes a deep feature extractor (green), a deep label predictor (blue), and a domain classifier (red).

3 An overview of the data

In this assignment we make use of the PACS dataset [4]. It is a recent domain generalization benchmark, where one domain is an unseen target domain, with a severe distribution shift between domains, making it one of the most challenging tasks. It is composed of four domains: photo, art painting, cartoon, and sketch, each one containing 2048, 2344, 1670, and 3929 images correspondingly. The seven categories found in these domains are dog, elephant, giraffe, guitar, horse, house, and person.
Each image is about 227x227 pixels and is labeled with a single object. Each class contains between 80 and 800 images, giving a total of 9991 images.

3.1 A brief data analysis

A histogram counting the number of pictures belonging to each class of each domain was generated in order to learn more about the PACS dataset.

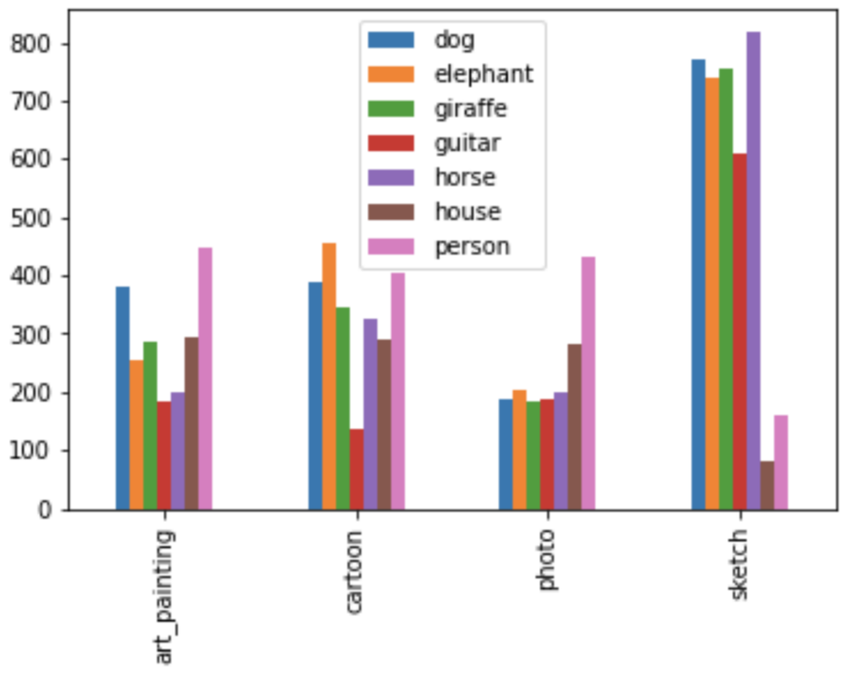


Figure 2: Histogram of the class distribution in each domain in the dataset.

The classes are not equally balanced according to Figure 2. It can also be appreciated that there is one domain, the sketch one, that has several more images that the others.

To compare the number of images of each domain inside each class, Figure 3 was plotted.

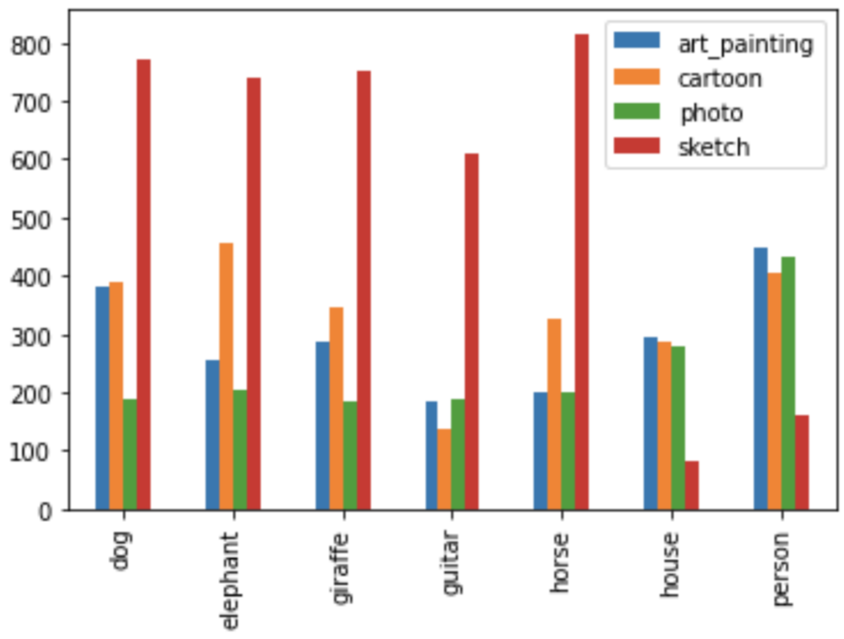


Figure 3: Histogram of domain distribution in each class in the dataset.

The image size was analyzed by a "2D histogram", which's x and y-axis correspond to the width and height of an image correspondingly. It can be noted looking at Figure 4 that there is just one image size and it is 227x227.

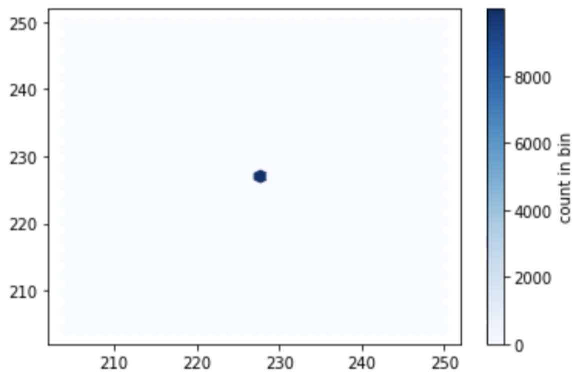


Figure 4: 2D histogram of image size in the dataset.

4 The implemented model

In order to implement the architecture described in 2.1.2.1, we started from the AlexNet [3] source code of *PyTorch*. This network has a convolutional layer that is the feature extractor G_f, and a fully connected layer that is the label predictor G_y. To obtain the DANN a new branch has been created, this new layer is fully connected, in this case, it has the same architecture as the label predictor, and it will work as a domain classifier G_d. The code regarding the new domain classifier is the following:

```
self.domain_classifier = nn.Sequential(
    nn.Dropout(),
    nn.Linear(256 * 6 * 6, 4096),
    nn.ReLU(inplace=True),
    nn.Dropout(),
    nn.Linear(4096, 4096),
    nn.ReLU(inplace=True),
    nn.Linear(4096, num_classes),
)
```

One important change that has been made on the Alexnet source code is the forward function. When recalling it, a sample parameter required and an *alpha* parameter optional are passed. Forwarding consists of extracting the features from the sample, generating the output corresponding to the two classifiers, and then retrieving. It was decided to calculate both results at the same time, instead of the use of each classifier being given by a flag, to achieve greater usability of this network.

```
def forward(self, x, alpha=1.0):
    x = self.features(x)
    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    features = x
    reverse_features = GradientReversalFn.apply(x, alpha)

    class_pred = self.label_classifier(features)
    domain_pred = self.domain_classifier(reverse_features)
    return class_pred, domain_pred
```

In particular when computing the output of the domain classifier, a reverse gradient forward function is called saving the *alpha* parameter to be used in the back-forward step. The code of this reverse gradient function is shown next:

```
class GradientReversalFn(Function):
    @staticmethod
    def forward(ctx, x, alpha):
        # Store context for backprop
        ctx.alpha = alpha

        # Forward pass is a no-op
        return x.view_as(x)

    @staticmethod
    def backward(ctx, grad_output):
        # Backward pass is just to -alpha the gradient
        output = grad_output.neg() * ctx.alpha

        # Must return same number as inputs to forward()
        return output, None
```

As the homework requires, the pre-trained in ImageNet [1] weights of AlexNet were needed, therefore, the weights in the new branch were updated manually with the same weights of the label predictor layer, as follows.

```
model.domain_classifier[1].weight.data = model.label_classifier[1].weight.data
model.domain_classifier[1].bias.data = model.label_classifier[1].bias.data

model.domain_classifier[4].weight.data = model.label_classifier[4].weight.data
model.domain_classifier[4].bias.data = model.label_classifier[4].bias.data
```

5 Experiments and results

The following experiments were stipulated by the assignment. They are intended to train our implemented model, described in section 4, with DANN adaptation and without it in domain adaptation, and thus have a baseline to compare the results of the DANN adaptation.
The goal of the next experiments is to train a model in the photo domain and then use it in the art painting domain. Hence, in the training, the photo domain of the PACS dataset was used as the source domain and depending on the experiment, the target domain varied between art painting, cartoons, and sketches.
In all the experiments the models have been trained with with a *stochastic gradient descent* optimizer and its parameters were: momentum 0.9, weight decay 0.00005, and the learning rate as a hyper-parameter to optimize. The loss function was the *cross entropy* function. And finally, a scheduler was used to update the learning rate every a given number of epochs, its parameters gamma and the step size are the hyper-parameters in the next experiments.

5.1 Train on Photo, and Test on Art painting without adaptation

This is a normal CNN training, the new branch of the implemented model is not needed, therefore, the outputs of the domain classifier are simply discarded.
The hyper-parameters used in this experiment were: initial learning rate 0.001, number of epochs 20, batch size 128, step size 20, and gamma 0.1.

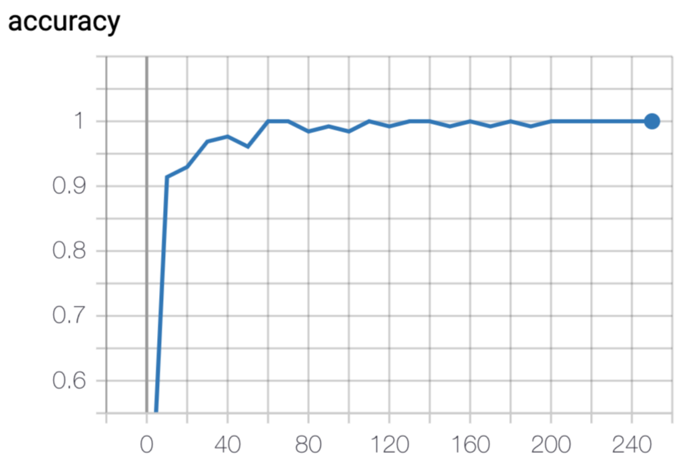


Figure 5: Training accuracy by steps of the experiment in section 5.1

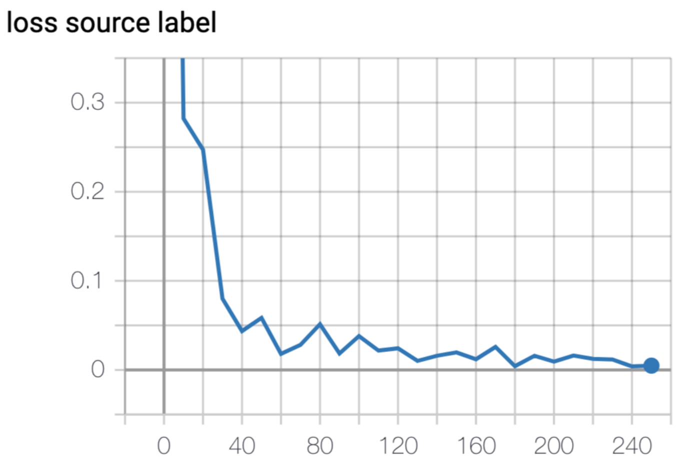


Figure 6: Training loss in source labels by steps of the experiment in section 5.1

From Figures 5 and 6 it is seen that the model was overfitted in the source domain. Then this model was tested on the art painting domain (the target domain) and it got an accuracy of 0.4878. As it was a normal procedure of training, this test accuracy is going to be our baseline to compare further experiments.

5.2 Train DANN on Photo and test on Art painting with DANN adaptation

In this experiment, the training was carried out in the photo domain as a source domain and the art painting as the target domain. The training of the implemented model consists of three main phases:

1. Training the G_y branch with the source data (from photo domain) based on data labels, computing the outputs, and the loss value.
2. Training the G_d branch with the source data but at this time the labels will be at zero. Indeed, our goal is not to classified the source data based on labels but to evaluate how the network discriminates the source and target domain (photo and art painting). Then, the outputs and loss value are computed.
3. Training the G_d branch with target data (from art painting domain) with labels led to 1. Like before, the outputs and loss value are computed

Then, the three losses are then added up and used to update the gradients. This sum is done by the *PyTorch* framework, taking into into consideration the gradient reversal layer within the G_d branch. When the gradients are computed, the optimizer step is called to apply them and update weights.
As we want to compare this procedure with the previous one, the hyper-parameters were maintained constant with the addition of a new hyper-parameter *alpha* for the G_d branch. They were as follows: initial learning rate 0.001, number of epochs 20, batch size 128, step size 20, gamma 0.1, and alpha 0.03.

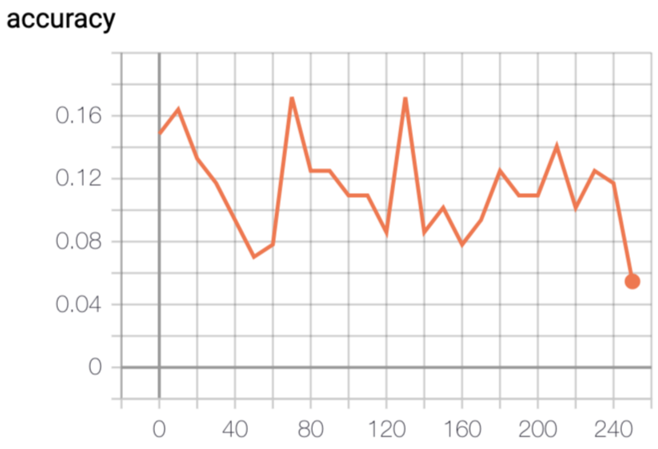


Figure 7: Training accuracy by steps of the experiment in section 5.2

As shown in Figure 7, the training accuracy of the model with the DANN adaptation does not have the usual behavior, and it is probably not a good metric for this model either. As far as we know, it is not over fitting, but we cannot determine if it could learn more about the source domain with more epochs.
To see the different losses in the training, Figure 8 shows them all in one graph. Their tendency is to be minimized, but we can see that the source domain and target domain losses were oscillating, which tells us that they were trying to confuse the feature extractor layers (G_f).

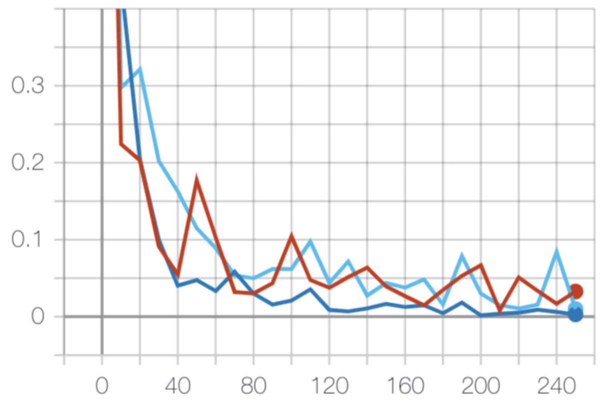


Figure 8: Training losses by steps, source labels loss (blue), source domain loss (red), target domain loss (light blue), in the experiment in section 5.2

This model had a test accuracy on the art painting set of 0.5056. Despite being a difference of ∼2% with the model without DANN adaptation, it is a significant improvement. Just comparing two results it is not enough to make conclusions, so further experiences were made.

5.3 Grid search without domain adaptation

As far as we are concerned, without validation, we are looking at results in the test set, the art painting domain, accordingly, it is like cheating the evaluation. Hence, for getting more reliable results, the use of a validation step was introduced, known as *cross domain validation*.
To obtain the best possible model, from our implementation, in this task without domain adaptation, a grid search was used, but the range of the grid was limited due to the scarce computing resources.
The procedure consists of executing a grid search to optimize the hyper-parameters of our model without domain adaptation while training the model in the photo domain as in the experiment 5.1 and then evaluating it separately in the cartoon and sketch domains for obtaining a cross domain validation. The evaluation metric was the mean accuracy between both target domains.
The learning rate values have been chosen at a very low rate to avoid non-convergent losses, in fact, a value greater than 0.01 turns out to be too large. The sets of the hyper-parameters that were used are the following:

- Learning rate: 0.00005, 0.0001, 0.0005.
- Number of epochs: 10, 15, 20.

- Step size: 5, 10.
- Gamma: 0.01, 0.05.

The batch size was maintained constant with a value of 128.
As it is known, the *GoogleColab* platform has a limited computing time, therefore a persistence system was developed to save the results every time a set of hyper-parameters was evaluated, into a file linked to *GoogleDrive*.
After running the grid search, the best hyper-parameter found were: learning rate 0.00005, number of epochs 20, step size 5, and gamma 0.05. It reached a mean accuracy of 0.3031 when evaluating it. Then these hyper-parameters were used to train a model as the one in experiment 5.1 and it got an accuracy of 0.3630 in the art painting domain. As it seems, it is not a great accuracy compared to that of experience 5.1, but the selection of this model is not biased by the art painting domain.

5.4 Grid search with domain adaptation

In this experience we made use of the same cross validation method as the one in the experiment 5.3. In regards to the grid search, this time two networks were trained, one with photo as the source domain and cartoon as the target domain, and another with photo as the source domain and sketch as the target domain, following the training procedure of the experiment 5.2, and then each one was evaluated on its target domain. The selection of the best hyper-parameter of the grid search was made based on the mean of the accuracy reached by each network in its target domain.
The hyper-parameter sets were the same as in the previous experiment, but adding one more number of possible epochs, that is greater than the others, to see if the model could learn more. Also, the alpha hyper-parameter was introduced:

- Learning rate: 0.00005, 0.0001, 0.0005.
- Number of epochs: 10, 15, 20, 25.
- Step size: 5, 10.
- Gamma: 0.01, 0.05.
- Alpha: 0.01, 0.03, 0.05.

As well as the previous time, the batch size was maintained constant with a value of 128.

It can be noted that the alpha value affects the losses a lot. When it is incremented too much, the losses don't converge.
The results obtained of the grid search were that the best hyper-parameters found for this model with a mean validation accuracy of 0.3446, were: learning rate 0.00005, number of epochs 25, step size 5, gamma 0.05, and alpha 0.03.
A model as the one in the experiment 5.2 was trained with this set of hyper-parameters and reached an accuracy in the art painting domain of 0.3780. Comparing it with the final model of the previous experience we found that its results are not very good. Probably a more intense grid search should be carried out, but due to the computing limitations this was not possible.
Another comparison that could be discussed is the one of this model with the one of the experience 5.2. However, as it was mentioned before, the other model was "cheating" the evaluation procedure while it was looking at results in the art painting domain, therefore, it is not a good baseline for comparison.

6 Conclusion

The results obtained may be sufficient to support the hypothesis that the DANN approach improves model learning when a domain adaptation is needed. By making the correct comparisons between the experiments, it is appreciated that the accuracy in the target domain of the models with DANN adaptation is $sim1$ to 2 percent more than the models without DANN adaptation. For more reliable conclusions, more experiments may be required.

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] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario March, and Victor Lempitsky, *Domain-adversarial training of neural networks*, Journal of Machine Learning Research **17** (2016), no. 59, 1–35.

[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.

[4] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales, *Deeper, broader and artier domain generalization*, International Conference on Computer Vision, 2017.