Homework 3

S274563 Marco Gullotto

May 2020

1 Introduction

As shown in the last homeworks, CNNs are very powerful tools and, wanting to make a classification task, for example, they prove very precise in the forecast if they have been trained with a lot of data, and the test set has a similar distribution. Even if we don't have a large number of photos, pre-trained nets on large ds (like imagenet) still allow us to obtain extraordinary results. However, when using a CNN, we assume first that the train data distribution is representative of the test distribution. If this hypothesis fails, CNN can behave negatively. Recently, many researchers have tried to solve this problem and many different methods have been developed to do it. All of these methods improve performance, but none of them is the ultimately solution to the problem. This homework will be focused on the Domain-Adversarial neural network (Dann) developed by Ganin et al. in the Pacs dataset. As illustrated in Figure 1, the network is divided into two branches: the blue layers, fully connected, which predict the label of the image, while the pink ones provide for the domain of the photo. The idea behind this method is really simple. We want to create domain-independent features (f). This means that, when the network is trained, the domain predictor will no longer be able to tell if a photo belongs to one domain or another. At the same time, the f predictors will be good enough to allow for a better result. This method is developed using two different losses: the classification loss and the domain confusion loss. The first loss is minimized using gradient descent, while to create domain-independent predictors the confusion loss is maximized using gradient ascent and in particular a gradient reversal layer.

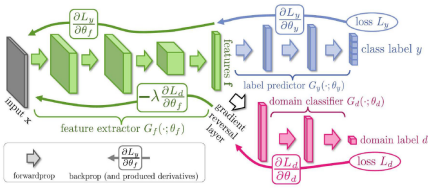


Figure 1: Domain-adversarial neural network architecture by Ganin et al.

One well-known dataset that can be used to try to test this method is the PACS dataset. This ds has four different domains: art painting, cartoon, sketch and photos. For each domain, we have the same seven classes (as shown in figure 2). It is evident from the ds that, while artistic painting and cartoon are often similar, the other two classes are significantly different; the sketch will be the most difficult to predict because, while the other domains use colours, the sketches are in black and white. Moreover, the data set is not large enough because it contains less than ten thousand photos. So to increase the performance we use transfer learning in particular, we will use an AlexNet model trained on the ImageNet dataset.

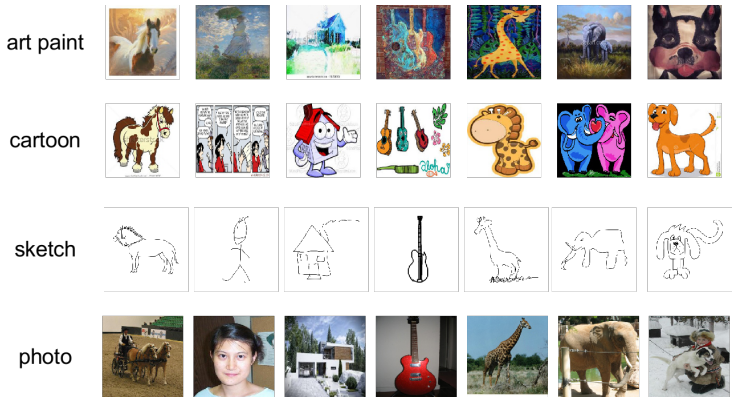


Figure 2: Example of the 4 domains of the pacs dataset

2 Implementing the Model

As previously said, one of the main ingredients for implementing a Dann is the "ReverseLayer". As shown in the following code, nothing changes in the forward step, while in the backward phase we perform gradient ascent weighted of a factor-lambda (we will discuss later on how to set this hyperparameter).

```
class ReverseLayerF(Function):
    # Forwards identity
    # Sends backward reversed gradients
    @staticmethod
    def forward(ctx, x, lambda):
        ctx.lambda = lambda

        return x.view_as(x)

    @staticmethod
    def backward(ctx, grad_output):
        output = grad_output.neg() * ctx.lambda
```

```
        return output, None
```

To implement the two different branches of the fully connected layer, it is necessary to change the structure of the init function. To do this, we add to the "natural" classifier another one for the domain task.

```
self.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),
        )

        self.classifier_target = 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_target),
        )
```

The last step is to change the forward method. If the lambda value is set, we use "ReverseLayer" and the result is produced by the domain classifier, while the other works like AlexNet in its "natural" way.

```
def forward(self, x, lambda = None):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)

        if lambda is None:
          return self.classifier(x)
        else:
          reverse_x = ReverseLayerF.apply(x, lambda)
          return self.classifier_target(reverse_x)
```

It should be noted that the weights of the domain classifier are initialized in the same way as the weights of the label classifier.

3 Train the Cnn without domain adaptation

To understand if we actually achieve improvement using the DANN method, we need to establish a baseline. So, initially, we train our network on Photo

and test on Cartoon. After that, we set up the same train, but we will test it on the sketch domain. In this way, we can also implement a sort of cross-validation using different hyperparameters. For both, the activity that we adopt "CrossEntropyLoss" or "log loss" measures the performance of a classification model whose output is a probability value between 0 and 1. Then the SDG optimizer is used with the predefined hyperparameters, except for the learning rate. So, starting with 30 epochs, we use an optimizer that reduces the learning rate after twenty epochs of factor 0.1. (For more explanations, see the report of homework 2). Let's start with three different learning rate values (1e-5 (pink on the graph), 1e-4 (orange on the graph) and 1e-3 (blue on the graph)). Starting from the lowest value, it is clear that the LR is not sufficient to ensure convergence. The initial loss is very high: it is estimated around 1.2 and the final one is also quite big, about 0.2. The accuracy, although initially increasing rapidly, in a second moment slowly decreases, eventually reaching a score of 0.22. For the values of 1e-3 and 1e-4, the loss and accuracy behaviour is very similar. The loss that occurs in a few iterations is very close to 0 but the accuracy fluctuates significantly continuously. The final score is around 0.25. So let's start testing the net on the sketch part. Since the sketch domain is different from the other photo's, we expect worse performance. But this is only a hypothesis. There are two kinds of graphs: the first one is about Accuracy while the second is about Loss. In both charts x axis represents the number of epochs and the y axis describes respectively accuracy and loss values (the same for all the graphs in this homework).

LR	N. Epochs	Step Size	Gamma	Optimizer	Scheduler	Acc.	Loss	Target domain	Color
10^{-3}	30	20	0.1	SGD	StepLR	0.24	≈ 0.01	Cartoon	Blue
10^{-4}	30	20	0.1	SGD	StepLR	0.25	≈ 0.01	Cartoon	Orange
10^{-5}	30	20	0.1	SGD	StepLR	0.22	≈ 0.17	Cartoon	Pink

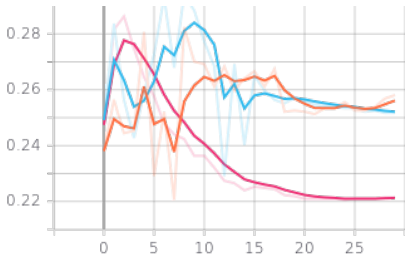


Figure 3: Accuracy AlexNet trained on photo and test on Cartoon

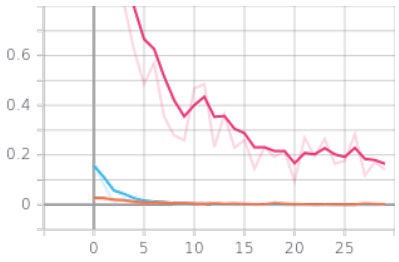


Figure 4: Loss AlexNet trained on photo and test on Cartoon

The result obtained is quite different from the one obtained previously. The trend of the loss functions is similar but the values are always greater. It could depend on the way the data are shuffling. Instead, both the values and the trends in the accuracy graph change radically. With a learning rate of 1e-4 (grey in

the graph) and 1e-5 (orange in the graph), the CNN seems to learn significant features, in fact, the accuracy score increases (although slowly) over time. The behaviour of accuracy, however, when the learning rate is equal to 1e-3, shows that Cnn immediately has good predictors. In fact, the score starts at 0.36, which is the highest seen so far but, although the loss has decreased over the epochs, the score continues to fluctuate until learning changes to 1e-4. Having seen all these graphs so far, the difficulty in obtaining a great result is evident. Loss values do not decrease much after a few epochs and the accuracy score also does not significantly improve after few epochs. So, to reduce training time and try to increment the result, we change the number of epochs from thirty to twenty, and the step-size of the optimizer from twenty to ten.

LR	N. Epochs	Step Size	Gamma	Optimizer	Scheduler	Acc.	Loss	Target domain	Color
10^{-3}	30	20	0.1	SGD	StepLR	0.37	≈ 0.01	Sketch	Blue
10^{-4}	30	20	0.1	SGD	StepLR	0.26	≈ 0.19	Sketch	Grey
10^{-5}	30	20	0.1	SGD	StepLR	0.25	≈ 0.02	Sketch	Orange

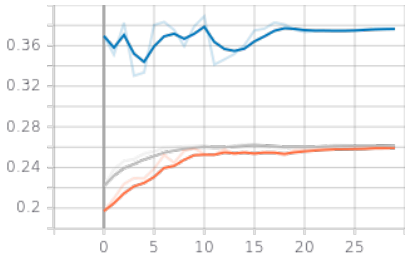


Figure 5: Accuracy AlexNet trained on photo and test on Sketch

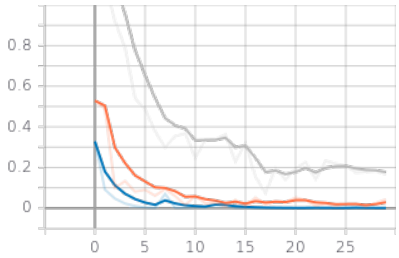


Figure 6: Loss AlexNet trained on photo and test on Sketch

By using this new set of hyperparameters, the loss and accuracy tendencies become smoother. We have fewer spikes than before and the result becomes more easily interpretable. If you use 1e-5 (blue in the graph) as learning rate, at first it seems that the network learns really good features over time, but the final result, in reality, both as regards the accuracy and for that, concerning the loss, it is really poor. With 1e-4 (red in the graph) the accuracy starts from a good value but, even if the loss decreases, the CNN does not learn any useful predictors in order to improve the final result. The network with LR = 1e-3 is the one with the lowest number in the lost functions, but the accuracy graph still shows some peaks and the final result is not so impressive. Finally, to clearly understand which are the best hyperparameters for the network, let's go back to the original settings: train on photos and tests on the cartoon domain.

LR	N. Epochs	Step Size	Gamma	Optimizer	Scheduler	Acc.	Loss	Target domain	Color
10^{-3}	20	10	0.1	SGD	StepLR	0.26	≈ 0.01	Sketch	Orange
10^{-4}	20	10	0.1	SGD	StepLR	0.3	≈ 0.015	Sketch	Red
10^{-5}	20	10	0.1	SGD	StepLR	0.28	≈ 0.3	Sketch	Blue

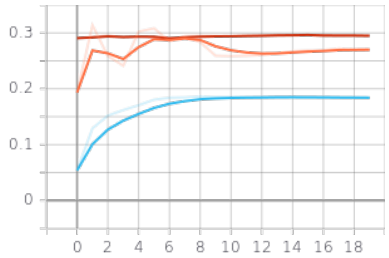


Figure 7: Accuracy AlexNet trained on photo and test on Sketch

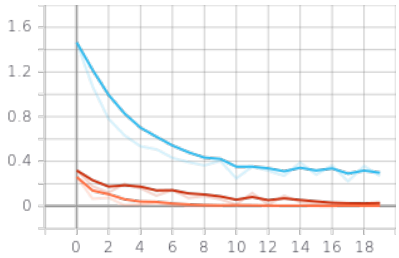


Figure 8: Loss AlexNet trained on photo and test on Sketch

If we compare figure 9 with figure 3 it is immediately clear that this new set of hyperparameters greatly helps the behavior of the convergences. The values of the three losses are drastically lower than before and the accuracy also increases significantly. The learning rate equal to 1e-5 seems to be low in all contexts. Loss functions values are higher than the others and accuracy is often poor. Furthermore, this time, as illustrated by the graph (pink line), the CNN does not learn any significant predictors and this is clear in the accuracy graph. A similar analysis can be made for 1e-4 (green line). In fact, accuracy is often worse than 1e-5 and the loss function is often greater than 1e-3. So the final choice is to use 1e-3 as a learning rate, to make the network train for twenty epochs with a step size of ten.

LR	N. Epochs	Step Size	Gamma	Optimizer	Scheduler	Acc.	Loss	Target domain	Color
10^{-3}	30	20	0.1	SGD	StepLR	0.28	≈ 0.005	Cartoon	Gray
10^{-4}	30	20	0.1	SGD	StepLR	0.23	≈ 0.015	Cartoon	Green
10^{-5}	30	20	0.1	SGD	StepLR	0.19	≈ 0.22	Cartoon	Pink

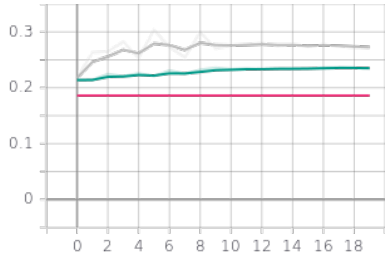


Figure 9: Accuracy AlexNet trained on photo and test on Cartoon

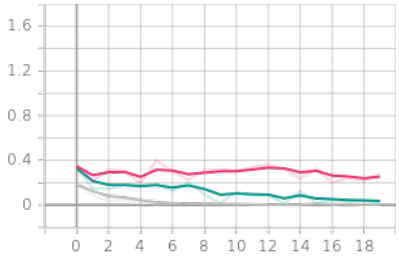


Figure 10: Loss AlexNet trained on photo and test on Cartoon

Using these best hyperparameters it is possible to train our CNN on Photo and use as test set the images of the art painting domain. The result is quite good. As shown in Figure 12, the loss reaches a value very close to 0 after only eight epochs. Moreover, the accuracy increases, and from an initial value of

0.45, the final result is about 0.5. Not bad, if we think that the random guess is equal to about 0.14.

LR	N. Epochs	Step Size	Gamma	Optimizer	Scheduler	Acc.	Loss	Target domain
10^{-3}	30	20	0.1	SGD	StepLR	0.49	≈ 0.01	Art Painting

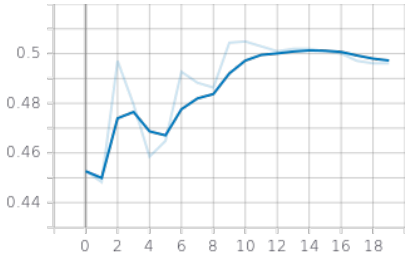


Figure 11: Accuracy AlexNet trained on photo and test on Art painting with best hyperparameters

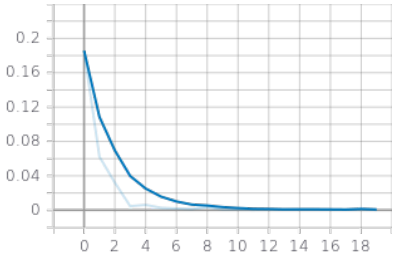


Figure 12: Loss AlexNet trained on photo and test on Art painting with best hyperparameters

4 Train the Cnn with domain adaptation

From now on we want to ignore the distribution between the different domains. Therefore, we focus on extracting features that allow us to understand the main characteristics of each class regardless of colours, exact shape, and so on. So, when we use the Dann method our input is divided into two groups:

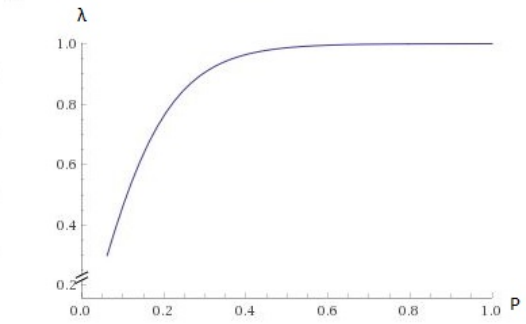
1. Images with class labels are called "source". For example, we can assign domain 0 to that.

2. Images without class labels are called "targets". To target images, we assign domain 1.

When we go through the network a source image, we obtain two different results: the first is represented by the class of the image, the second is instead the domain of the image. Conversely, when we use a target image, we only receive the expected domain. Initially, the features created by the network allow the domain classifier to distinguish well between the two domains. But, using a reverse layer (training to maximize domain loss) the network will learn features that will not depend on the domains. This implies that during training, the loss of label predictors decreases while the domain confusion loss increases. This is not always true, because the domain discriminator is still trying to minimize his loss. λ is another important hyperparameter, when we are dealing with Dann. In fact, when we invert the gradient, we also weight the gradient itself by this value. Therefore, there are several ways to set this hyperparameter: we can keep λ fixed throughout the training phase, or we can try to change the value

at each iteration. To try to understand which is the best solution, we tried to apply them both. A formula commonly used to update the value of λ is the following:

$$\lambda_p = \frac{2}{1 + e^{-10p}} - 1$$



Where p is a number in the range [0,1] and represents the progress of the training process. As illustrated by the graph, the lambda value increases rapidly and reaches the value of 1 in a few times. This type of behaviour is exactly what we are looking for because, during the first epochs, the domain loss is noisy since the convolutional features are not yet trained, therefore we prefer to use a low lambda value. Conversely, when the features are well initialized, we begin to increase lambda and, consequently, the confusion domain loss. So, starting from the old hyperparameters, we set LR = 1e-3, Epochs = 20 and step-size = 10. It is immediately evident that, using the lambda function mentioned above, the network diverges in a few steps. This could mean that the network needs more time to learn convolutional features. So we replace the factor 10 in formula with 1.5 and then with 1.25. Finally, let's try two common stable values for lambda, namely 0.25 and 0.15. Let's start by training the network again on the photo domain (as source) and the sketch domain as target. The result is the following:

LR	N. Epochs	Step Size	Lambda	Conf. l. sour.	Conf. l. tar.	Label Loss	Acc.	Target domain	Color
10^{-3}	30	20	function(1.5)	≈ 0.8	≈ 0.2	≈ 0.7	0.25	Sketch	Green
10^{-3}	30	20	function(1.25)	≈ 1.5	≈ 0.2	≈ 0.5	0.37	Sketch	Grey
10^{-3}	30	20	0.15	≈ 0.01	≈ 0.01	≈ 0.22	0.27	Sketch	Orange
10^{-3}	30	20	0.25	≈ 0.01	≈ 0.01	≈ 0.01	0.55	Sketch	Pink

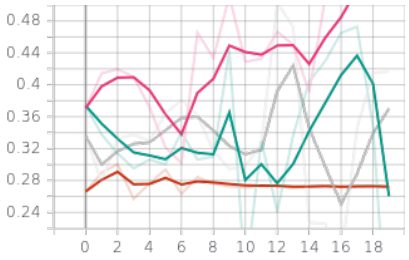


Figure 13: Accuracy on Sketch using Dann

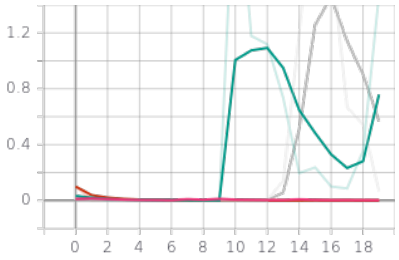


Figure 14: Label loss on sketch using Dann

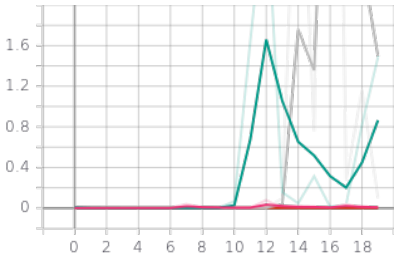


Figure 15: Confusion domain loss on Sketch for source

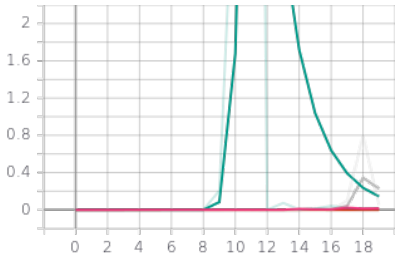


Figure 16: Confusion domain loss on Sketch for target

As shown in the graph, the result obtained is quite relevant. We must remember, from figure 5, that the best accuracy score previously achieved was around 0.36. So, we immediately notice that the pink line (the one with lambda constant equal to 0.25) shows a remarkable improvement. Accuracy starts at 0.36 and ends at 0.55. This may suggest that we could use a fairly large lambda value from the start without problems. This may be due to the fact that we use a pre-trained network so that the convolutional layer immediately extracts good features. The only problem is that the domain confusion loss is always very close to 0 and does not increase over time. Now let's look at the orange line, that is, the one with lambda equal to 0.15. In this case, virtually no improvement has been recorded since its initialization. This may suggest that this lambda value is too low to develop domain-invariant predictors. If we consider functions with a non-fixed lambda value, we notice that they both behave similarly. The one with exp (1.5) (in green) has a low domain confusion loss in the early epochs and a high value in the latter. This is exactly what we were looking for. Accuracy is also good, but in the last epoch, it has been somewhat reduced. We have very similar behaviour for the grey line (exp (1.25)). To confirm this result, we replace the same tests also using the Cartoon domain as target. The result is the following:

LR	N. Epochs	Step Size	Lambda	Conf. l. sour.	Conf. l. tar.	Label Loss	Acc.	Target domain	Color
10^{-3}	30	20	function(1.5)	≈ 0.31	≈ 0.5	≈ 0.3	0.52	Cartoon	Pink
10^{-3}	30	20	function(1.25)	≈ 0.1	≈ 0.18	≈ 0.01	0.5	Cartoon	Light-blue
10^{-3}	30	20	0.15	≈ 0.52	≈ 0.42	≈ 0.5	0.38	Cartoon	Green
10^{-3}	30	20	0.25	≈ 0.2	≈ 0.31	≈ 0.05	0.51	Cartoon	Blue

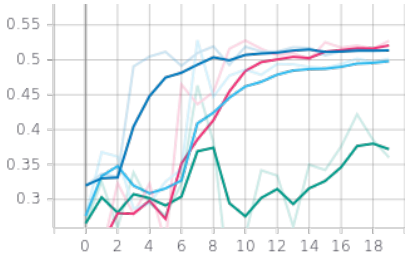


Figure 17: Accuracy on cartoon using Dann

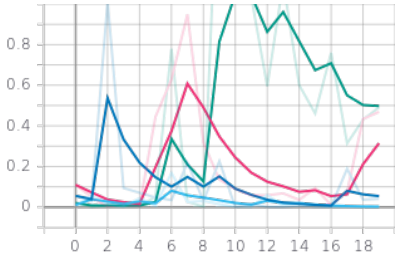


Figure 18: Label loss on cartoon using Dann

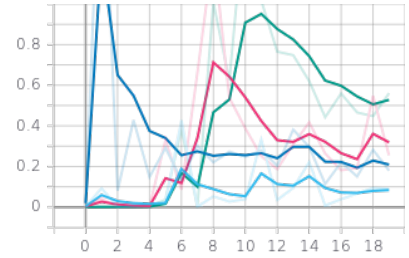


Figure 19: Confusion domain loss on cartoon for source

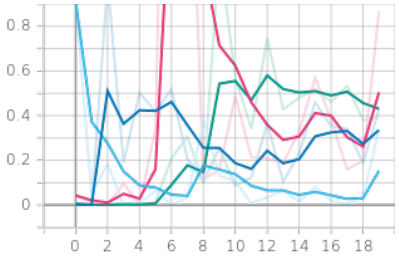


Figure 20: Confusion domain loss on cartoon for target

The blue line (lambda = 0.25), the pink one (exp (1.5)) and the light blue one (exp (1.25)) have a very similar result as regards the accuracy score, while for the green line (lambda = 0.15), it is observed that the final score is really poor. So, considering only the first three lines, we can see that domain confusion loss works quite well. In fact, over time this loss tends to increase, and as regards the pink line, for some epoch it also exceeds threshold 1. Summing up, it is evident that using lambda = 0.25 as a hyperparameter represents the most suitable choice. The accuracy and the domain confusion loss work quite well. The end result is always better than that obtained without adaptation of the domain.

LR	N. Epochs	Step Size	Lambda	Conf. l. sour.	Conf. l. tar.	Label Loss	Acc.	Target domain
10^{-3}	30	20	0.25	≈ 0.27	≈ 0.21	≈ 0.01	0.25	Art Painting

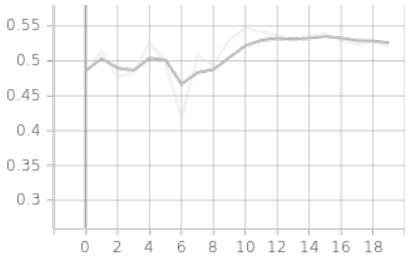


Figure 21: Accuracy on art painting using Dann

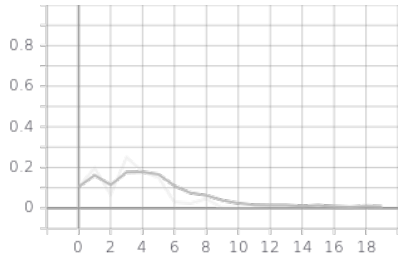


Figure 22: Label loss on art painting using Dann

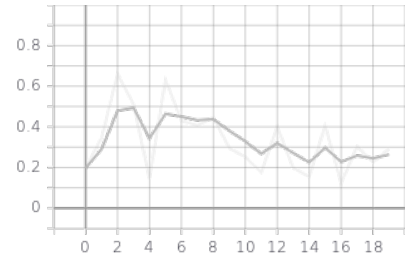


Figure 23: Confusion domain loss on art painting for source

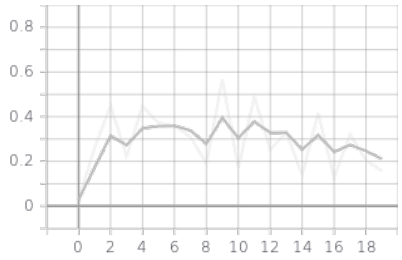


Figure 24: Confusion domain loss on art painting for target

So, using LR = 0.001, lambda = 0.25 (constant), epochs = 20 and step size = 10, we train the net using art painting as target domain. The end result is quite relevant. The accuracy exceeds 0.5 and also reaches values of 0.55. For the Pacs dataset, an increase of 0.01 in accuracy is also a good result. Having said that, we must also point out that the domain confusion loss works quite well. In fact, during the whole workout, it never decreases rapidly and often also tends to increase. The labels loss, however, after a few epochs gradually decreases until it reaches 0. This is exactly the behaviour we expect.

5 Cnn as features exctractor

Final step: I tried to use Cnn as a features extractor. Therefore, 4096 new predictors can be obtained for each image. To try to understand if these features are representative of my sample distribution, I use Tsne to switch from a space of 4096 dimensions to a space of 2 dimensions and trace it on a two-dimensional Cartesian graph. Unlike the PCA, Tsne does not constitute a linear projection. Conversely, use local relationships between points to create a low-dimensional mapping. There is also another technique that implies PCA withening but the Tsne algorithm is often recommended to represent the convolutional predictors. Using photos as source and sketch as destination, this is the result:

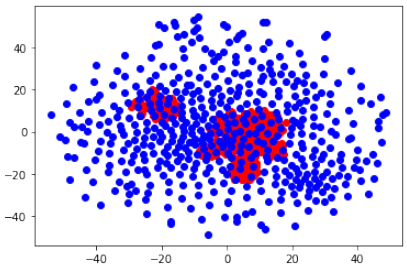


Figure 25: Non-adpted domain

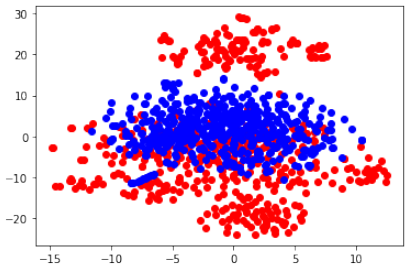


Figure 26: Adpted domain

As we can see, the blue points (belonging to the sketch domain) in figure 25 have a very widespread distribution, while the red points (the domain of the photo) are more compact. Using the features obtained with domain adaptation, it is clear that these predictors have a much more similar distribution. This is exactly what we are looking for. Looking at this graph, we can understand why it was possible to obtain an improvement in accuracy, and why the accuracy of the final score does not exceed 0.55. This not only underlines that everything we have done since now works correctly, but also that there is further room for improvement.