Machine learning and Deep Learning

Homework 3

Introduction

The aim of "Machine learning and Deep learning" third homework is to implement a *Domain Adaptation* (DA) algorithm called *Domain-Adversarial Neural Network* (DANN), starting from a provided template code.

Domain Adaptation is a sub-discipline of machine learning which deals with scenarios in which a model trained on a source distribution is used in the context of a different (but related) target distribution. So, the problem of *Domain Adaptation* is to learn from multiple training domains, and extract a domain-agnostic model that can then be applied to an unseen domain.

Mathematically, a domain D consists of a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$, where $X = \{x_1, ..., x_n\} \in \mathcal{X}$. Given a specific domain $D = \{ \mathcal{X}, P(X) \}$, a task $\mathcal{T}$ consists of a feature space $\mathcal{Y}$ and an objective predictive function, which can also be viewed as a conditional probability distribution $P(Y \mid X)$ from a probabilistic perspective.

Assume that we have two domains: the training dataset with sufficient labeled data is the source domain $D_s = \{ \mathcal{X}_s, P(X^s)\}$, and the test dataset with a small amount of labeled data or no labeled data is the target domain $D_t = \{ \mathcal{X}_t, P(X^t) \}$. Each domain is together with its task: the former is $\mathcal{T}_s = \{ \mathcal{Y}_s, P(Y_s \mid X_s)\}$, and the latter is $\mathcal{T}_t = \{ \mathcal{Y}_t, P(Y_t \mid X_t)\}$.

So, *Domain Adaptation* problem consists of having $D_s \neq D_t$ and $\mathcal{T}_s = \mathcal{T}_t$.

In addition, *Domain Adaptation* problem described in the report can be categorized as **unsupervised**, because here the network is trained on labeled data from a source domain and unlabeled data from a related but different target domain.

Further, in this case, as the source and target domains are directly related, transferring knowledge can be accomplished in one step. This is called **one-step Domain Adaptation**.

So, from one-step approaches the adversarial-based one that exploits domain-confusion loss is considered to carry out the homework. It is performed by a neural network on domains from PACS dataset. Neural network used is precisely the DANN and it is illustrated in the next section.

Domain-Adversarial Neural Network (DANN)

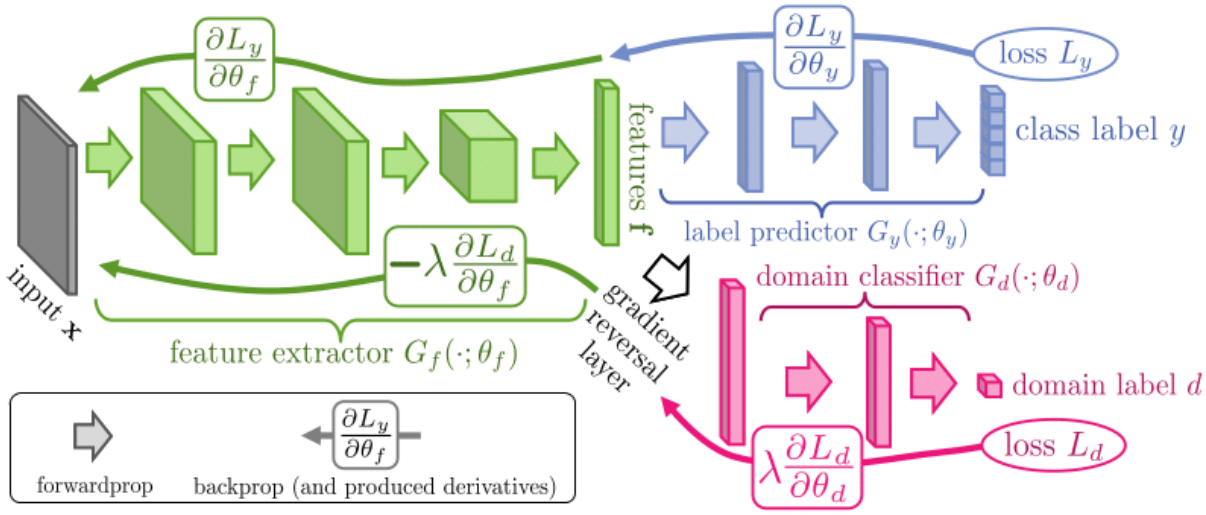


Figure 1 *- The Domain-Adversarial Neural Network (DANN) architecture*

The Domain-Adversarial Neural Network (DANN) [1] architecture includes a deep feature extractor and a deep label predictor which together form a standard feed-forward architecture.

Unsupervised *Domain Adaptation* is achieved by adding a domain classifier connected to the feature extractor through a gradient reversal layer (GRL) that multiplies the gradient by a certain negative constant during the backpropagation-based training. Otherwise, the training proceeds standardly and minimizes the label prediction loss (for source examples) and the domain classification loss (for all samples), maximizing domain confusion loss via the use of the GRL. Gradient reversal ensures that the feature distributions over the two domains are made similar (as indistinguishable as possible for the domain classifier), thus resulting in the domain-invariant features.

It is important to notice that gradient reversal layer has no parameters associated with it. During the forward propagation, the GRL acts as an identity transformation, while during the backpropagation the GRL takes the gradient from the subsequent level and changes its sign, multiplying it by−1, before passing it to the feature extractor G_f. So, as the backpropagation process passes through the GRL, the partial derivatives of the L_d loss become negatives.

1 – The dataset

To perform DANN, **PACS** dataset is used.

As described in [2], **PACS** dataset is created by intersecting the classes found in Caltech256, Sketchy, TU-Berlin and Google Images databases and includes 4 domains comprising different stylistic depictions (*Photo*, *Art painting*, *Cartoon*, *Sketch*), with objects from 7 classes that are dog, elephant, giraffe, guitar, horse, house and person.

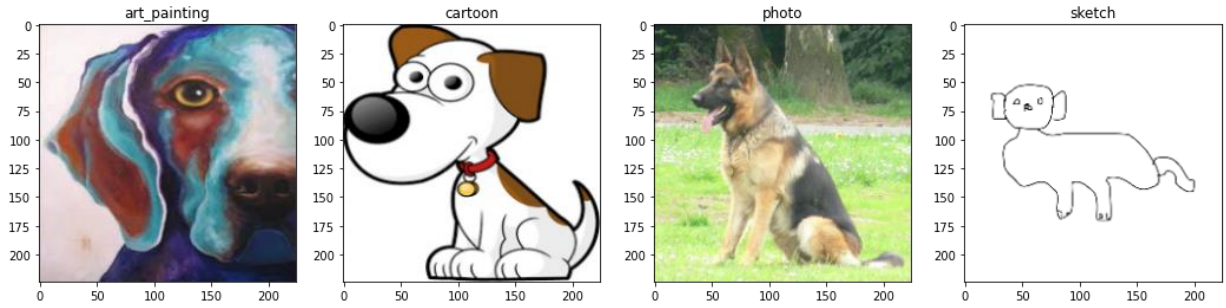


Figure 2 *- Dog pictures examples from each domain*

The total number of items is **9991**, each of them is a squared image having a size equal to 227x227 pixels.

To provide a qualitative summarization, it is shown the distribution of domain features, by the following 2-Dimensional t-SNE (t-distributed Stochastic Neighbor Embedding) plot[1] coming from [1], where the features are categorized and colored by their associated domain.

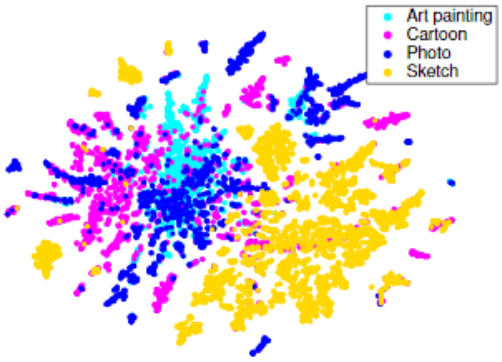
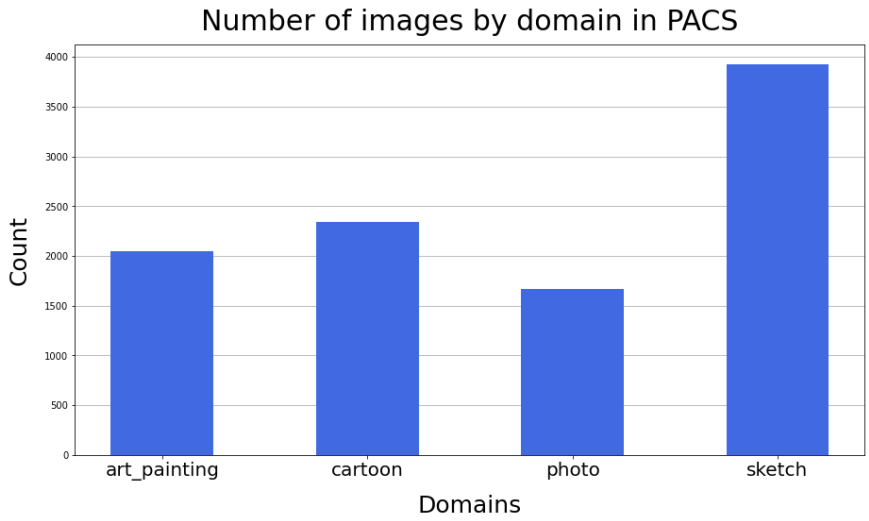


Figure 3 *- PACS t-SNE*

[1] t-SNE is a nonlinear dimensionality reduction technique well-suited for visualizing high-dimensional data in a low-dimensional space, by converting similarities between data points to joint probabilities and by trying to minimize the *Kullback-Leibler* divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data.

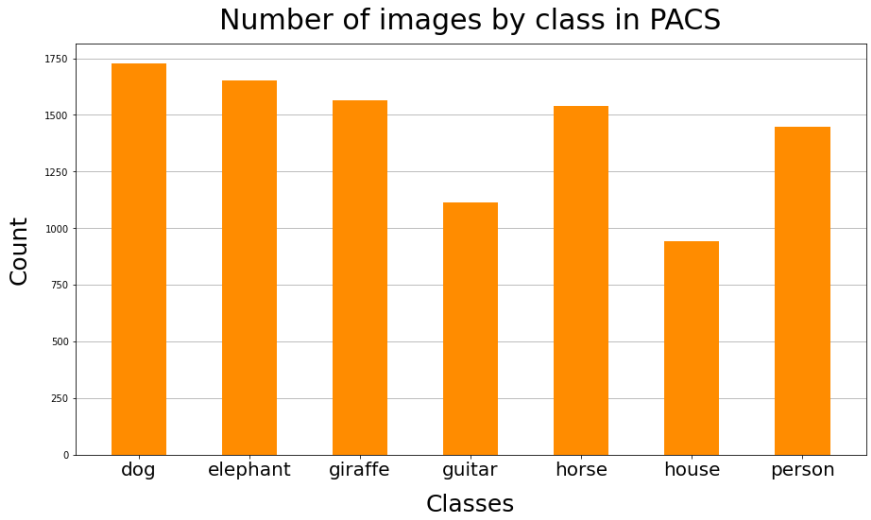
As we can see, PACS data are quiet well separated by domain, so this illustrates a sufficient degree of shift between domains. *Art Painting* and *Photo* however have more affinities as shown by their feature areas that are partially overlapped.

Moreover, it is possible to extract some further interesting statistics about domain and class distribution of images from dataset. These are summarized in the following plots and tables.



Domain	*Images*
Art painting	2048
Cartoon	2344
Photo	**1670**
Sketch	**3929**

Figure 4 *- Number of images by domain in PACS*



Class	*Images*
Dog	**1729**
Elephant	1654
Giraffe	1566
Guitar	1113
Horse	1540
House	**943**
Person	1440

Figure 5 *- Number of images by class in PACS*

So, it is possible to notice that most represented domain is *Sketch* and most relevant class is composed by *dog* pictures. On the other hand, the least represented domain is *Photo* and the least relevant class is composed by *house* images. It is also important to pay attention to different distribution of domains across classes and classes across domains, as it is shown in the table below.

Domain / class	***dog***	*elephant*	*giraffe*	*guitar*	*horse*	*house*	*person*	*Total (by domain)*
Art painting	379	255	285	184	201	295	449	2048
Cartoon	389	457	346	135	324	288	405	2344
Photo	189	202	182	186	199	280	432	1670
Sketch	772	740	753	608	816	80	160	**3929**
Total (by class)	**1729**	1654	1566	1113	1540	943	1446	9991

Table 1 *- Number of images for each class and domain*

As we see, PACS dataset seems to have domain-class dependency probably owing to the data characteristics. For example, $P(y = \text{person} \mid d = \text{Photo})$ is much higher than $P(y = \text{person} \mid d = \text{Sketch})$, indicating that photos of a person are easier to obtain than those of animals, but sketches of persons are more difficult to obtain than those of animals in the wild.

1.1 - Data preprocessing

After a first analysis, training and test data considered are chosen from two of PACS domains.

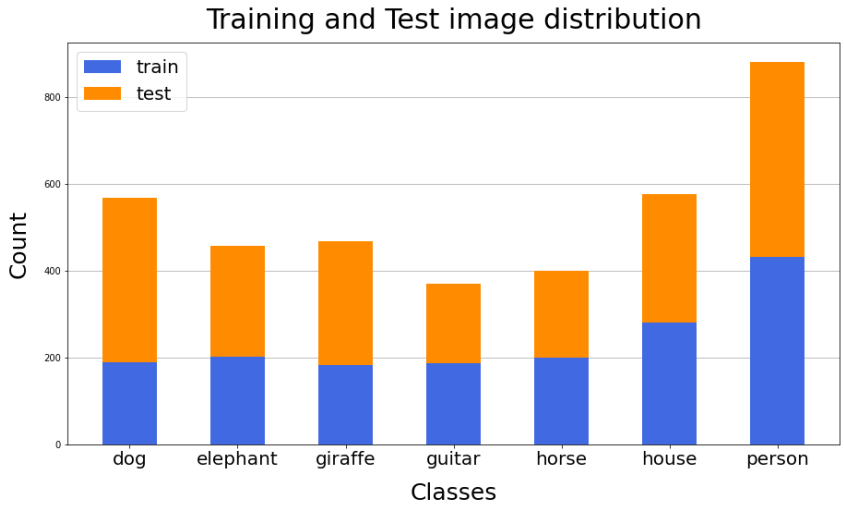
The source dataset will be composed by pictures coming from *Photo* domain, while target dataset will include images belonging to *Art painting* domain.

Then, we have the following configuration:

- Training dataset: **1670** images
- Test dataset: **2048** images

So, it is considered only the **37%** of the original dataset size: indeed, the **17%** of the whole dataset is regarded for training set, while the **20%** is reserved for test.

Distribution among classes is shown in the following bar plot and table.



	Train	*Test*	*Tot*
Dog	189	379	568
Elephant	202	255	457
Giraffe	182	285	467
Guitar	**186**	**184**	**370**
Horse	199	201	400
House	280	295	575
Person	**432**	**449**	**881**

Figure 6 *- Training and test image distribution*

As we can see, majority of images in the portion of PACS dataset considered belongs to *person* class, while the class least represented is the *guitar* one.

During dataset splitting, images are processed by some transformations.

Transformations are taken from *torchvision.transforms* PyTorch library and perform following sequential operations:

- **Center Crop**: images are cropped into a 224 by 224 pixels square image taken from the center of the original ones.

- **Conversion to tensor**: PIL (Python Imaging Library) Images are converted to tensors, so each picture is coded into numbers. An appropriate function separates the three colors that every pixel of the pictures is comprised of: red, green and blue. This essentially turns one image into three images (one tinted red, one green, one blue). Then, the pixels of each tinted image are converted into the brightness of their color, from 0 to 255. These values are divided by 255, so they can be in a range from 0 to 1. Images are now Torch Tensors (data structure that store lots of numbers).

- **Normalization**: lastly, tensor images are normalized with mean and standard deviation so that *DANN* can perform better. There are three values in the mean and standard deviation to match each RGB picture. Due to network pre-training from ImageNet, values used are:

    - *Means* = 0.485, 0.456, 0.406
    - *Standard deviations* = 0.229, 0.224, 0.225

2 – Implementing the model

Performed DANN implementation is based on pre-trained *AlexNet* [3].

Architecture previously described and shown in Figure 1 has been built in this way:

- **Feature extractor G_f** : it is composed by the five *AlexNet* convolutional layers. First, second and last of these are followed by a max-pooling layer, while the third, fourth and fifth are directly connected.
- **Label predictor G_y** : it is made by *AlexNet* three fully-connected layers. The last of these has 7 outputs, each of them corresponding to a single PACS class.
- **Domain classifier G_d** : it is made by the same *AlexNet* three fully-connected layers, but now the last one has only two outputs neurons. These two outputs denote the binary variable that points out if input items come from the source or the target distribution.

ReLU nonlinearity is applied after all the convolution and fully connected layers. Between G_f and G_y and between G_f and G_d is applied a 2D adaptive average pooling over the output features extracted by G_f. The output of this layer is of size 6x6 and the number of output features is equal to the number of input planes.

Both G_y and G_d distributions are initialized by weights from *ImageNet*, a large visual database of human annotated photographs which contains over 14 million images divided into more than 20.000 categories. This transfer learning operation allow us to get better results rather than training the network from scratch.

It is important to notice that gradient reversal is multiplied by a factor α. This is the domain regularization parameter that weights the loss provided by domain classifier G_d.

DANN so built will effectively attempt to train G_f to map an example (either source or target) into a representation allowing the final output layer of G_y to accurately classify source samples, but crippling the ability of G_d to detect whether each example belongs to the source or target domains.

3 – Domain Adaptation

Once implemented DANN, it is possible to train the network on images belonging to *Photo* domain and then testing it on pictures from *Art Painting* domain.

First step consists of considering only feature extractor G_f and label predictor G_y from DANN and so it is not being applied the domain adaptation phase. In this case, all trials are performed by using the same optimization algorithm based on configuration shown below:

Optimizer	SGD
Gamma	0.1
Momentum	0.9
Weight decay	5e-5

While previous parameters are steady, other ones are changing during experiments. These are batch size, learning rate, number of epochs and step size.

Results obtained are shown in the following table, where worst and best outcomes are highlighted respectively by red and green color:

Batch Size	*Learning Rate*	*Number of Epochs*	*Step Size*	*Accuracy on Art Painting Domain*
64	0,0001	30	20	(50,67 ± 1,35) %
64	0,001	30	20	(52,68 ± 0,14) %
64	0,01	30	20	(46,49 ± 2,04) %
64	**0,1**	**20**	**15**	**(18,51 ± 0,00) %**
128	0,0001	30	20	(49,25 ± 0,61) %
128	0,001	30	20	(51,76 ± 0,35) %
128	**0,01**	**30**	**20**	**(53,05 ± 1,71) %**
128	0,1	20	15	(18,51 ± 0,00) %
256	0,0001	30	20	(46,13 ± 2,29) %
256	0,001	30	20	(52,79 ± 0,21) %
256	0,01	30	20	(50,97 ± 1,40) %
256	0,1	20	15	(18,51 ± 0,00) %

Table 2 *– Results by using DANN without domain adaptation*

Accuracy on target domain is pointed out as mean and standard deviation calculated by three different runs of code. This criterion will be applied for all representations in section 3.

As we can see, although the network is pretrained, scores are not so good: this underlines the difficulties about domain adaptation problem. Indeed, highest result got is just equal to **53,05%**, so in the best case the model trained on *Photo* domain is able to classify correctly about half of *Art Painting* picture.

From Table 2 it is possible to notice that a too large learning rate value makes the model unable to converge to an optimal solution, as witnessed by divergent trend of cross entropy loss evaluated during training phase on source domain set.

Thus, in order to improve accuracy, it is tried to adjust learning rate during the stochastic gradient descent using the same formula applied in DANN paper experiments [1].

Such learning rate is defined as:

$$LR_p = \frac{LR_0}{(1 + a \cdot p)^b}$$

Where p is the training progress linearly changing from 0 to 1, $LR_0 = 0.01$, $a = 10$ and $b = 0.75$. In such way, the schedule should be optimized to promote convergence and low error on the source domain.

Hyperparameters used and results obtained are summarized in the following table.

Batch Size	*Learning Rate*	*Number of Epochs*	*Accuracy on Target Domain*
128	*LRp*	30	(53,11 ± 0,63) %
128	*LRp*	50	(53,29 ± 0,84) %

Table 3 *– Results of DANN without domain adaptation and decaying learning rate LRp*

Due to updating learning rate every epoch, mean values of accuracy got by testing the model on target domain is slightly better than best one got before. Moreover, now the result is more reliable because standard deviation is smaller.

Once completed last learning rate test, it is possible to proceed to train DANN applying domain adaptation technique through domain classifier branch. Configurations considered are composed by choosing previous parameter sets achieving best results for each batch size, evaluating with more trials the case in which batch size is equal to 128.

By these it is possible to see small performance improvements on *Art Painting* images evaluation.

Batch Size	*Learning Rate*	*Number of Epochs*	*Step Size*	*Alpha*	*Accuracy on Target Domain*
64	**0,001**	**30**	**20**	**0,005**	**(53,45 ± 0,45) %**
64	0,001	30	20	0,05	(53,20 ± 0,66) %
64	0,001	30	20	0,5	(20,22 ± 2,41) %
128	0,01	30	20	0,005	(52,62 ± 0,04) %
128	0,01	30	20	0,05	(53,14 ± 1,63) %
128	0,01	30	20	0,5	(18,51 ± 0,00) %
128	0,001	30	20	0,005	(52,20 ± 0,21) %
128	0,001	30	20	0,05	(52,44 ± 1,32) %
128	0,001	30	20	0,5	(23,95 ± 2,87) %
128	0,0001	50	30	0,005	(49,74 ± 0,59) %
128	0,0001	50	30	0,05	(48,95 ± 1,34) %
128	0,0001	50	30	0,5	(52,22 ± 1,00) %
256	0,001	30	20	0,005	(52,03 ± 0,59) %
256	0,001	30	20	0,05	(53,08 ± 0,07) %
256	**0,001**	**30**	**20**	**0,5**	**(15,72 ± 8,36) %**

Table 4 *– Results of DANN with domain adaptation*

These experiments show that starting from images belonging to *Photo* domain best results are obtained slightly weighting the domain regularizer by tuning low the adaptation parameter and using a learning rate equal to 0,001. In this way learning rate on domain target quite slows down and less importance is given to domain losses during back-propagation phase. On the other hand, with increasing number of epochs and decreasing learning rate, also good results on 128 batch size experiments have been obtained also by using $\alpha = 0,5$.

Below are analyzed trends of losses during training phase in the best case. Losses are computed averaging values over all batches for each epoch.

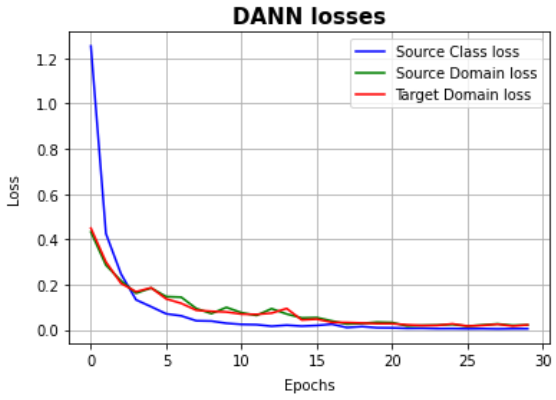


Figure 7

From the Figure 7 it is possible to see that all three losses converge close to zero. Trying to plot losses in the worst case nothing has been appeared because losses diverge within the first epoch and then the network will not be able to recognize many *Art Painting* pictures correctly.

It is also interesting the analysis of losses with lowest learning rate and highest α tried.

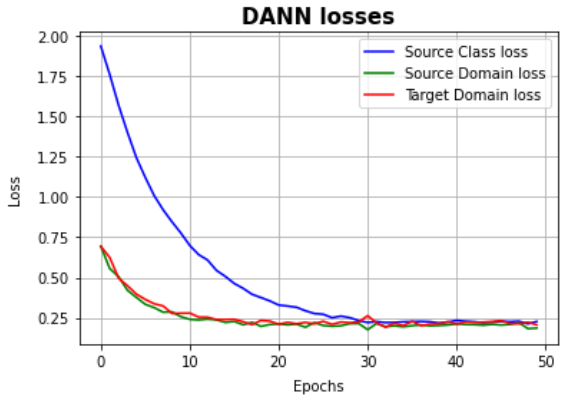


Figure 8

Now classifier loss is slower to converge, while domain losses are faster with respect to best case shown before. However, convergence value is about 0,25, that is larger than one found before so the model cannot provide the same accuracy results.

Further trials are made performing domain adaptation with 128 as batch size and decaying learning rate illustrated before, but now initiating parameter α at 0 and gradually changing to 1 using the following schedule [1]:

$$\alpha_p = \frac{2}{1 + \exp(-\gamma \cdot p)} - 1$$

Here, γ is set to 10 and p is ever considered as the training progress (current epoch / total epoch). This strategy should allow the domain classifier to be less sensitive to noisy signal at the early stages of the training procedure.

Batch Size	*Learning Rate*	*Number of Epochs*	*Alpha*	*Accuracy on Target Domain*
128	*LRp*	30	α_p	(21,92 ± 0.00) %
128	*LRp*	50	α_p	(33,47 ± 16,82) %
128	*LRp*	90	α_p	(36,5 ± 20,61) %

***Table 5** – Results of DANN with domain adaptation, adjusting learning rate and α_p*

These trials do not improve accuracy on target domain as mean values obtained are not good. However, during last two experiments it has been possible to notice great variations from one run to

other (as highlighted by huge standard deviation). Cause of this behavior is due to the high learning rate (0.01) and contemporarily low α (0) from training beginning. These factors bring a very fast convergence to a suboptimal solution, that can be very different at each run.

This behavior is also emphasized by oscillating trend of losses, as shown in Figure 9.

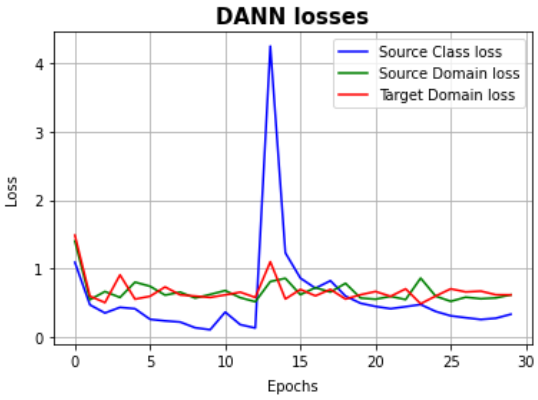


Figure 9

So, in order to avoid this inconstant progress of losses, previous experiments are repeated keeping fixed learning rate value, decreasing it only after a certain number of epochs determined by the step size.

Batch Size	*Learning Rate*	*Number of Epochs*	*Step Size*	*Alpha*	*Accuracy on Target Domain*
128	0,0001	30	20	α_p	(49,54 ± 1,69) %
128	0,0001	50	30	α_p	(44,83 ± 0,06) %
128	0,0001	90	40	α_p	(38,31 ± 1,28) %

***Table 6** – Results of DANN with domain adaptation, with constant learning rate and α_p*

Better accuracy scores are achieved but this time, increasing number of epochs, DANN performances get worse and losses diverge. Following two plots represent the trend of losses when number of epochs is equal to 50 (Figure 10) and 90 (Figure 11).

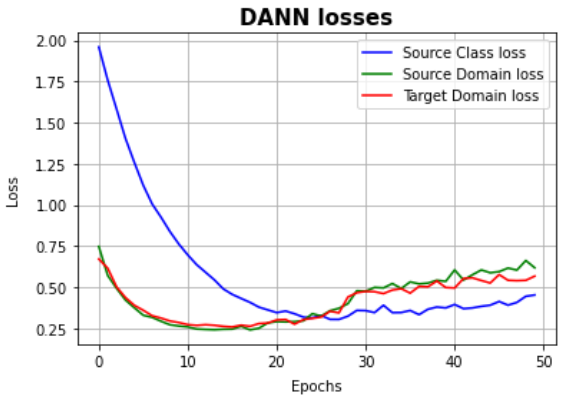


Figure 10

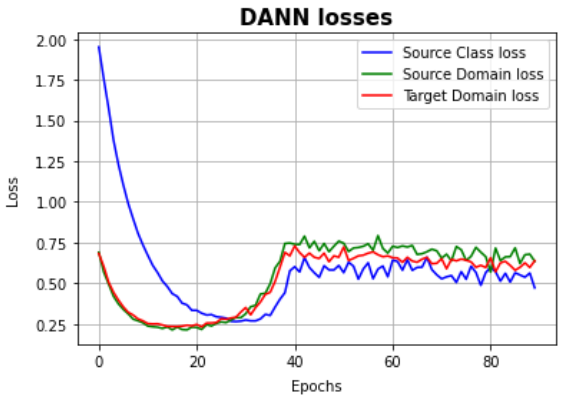


Figure 11

Lastly, common parameter sets used during trials are grouped and respective accuracy scores on target domain are compared. As *Domain Adaptation* results, ones achieved by best α are considered.

Batch Size	*Learning Rate*	*Number of Epochs*	*Step Size*	*Accuracy on Target Domain without DA*	*Accuracy on Target Domain with DA*	*Mean value improvements*
64	0,001	30	20	(52,68 ± 0,14) %	(53,45 ± 0,45) %	+ 0,77 %
128	0,01	30	20	(53,05 ± 1,71) %	(53,14 ± 1,63) %	+ 0,09 %
256	0,001	30	20	(52,79 ± 0,21) %	(53,08 ± 0,07) %	+ 0,29 %

Table 7 *- DANN results comparison*

As we can see, slight improvements are noticed by applying *Domain Adaptation* to the network. This is an effect of domain loss usage during training phase that allows DANN to be less skilled to discern if a picture belongs to *Photo* or *Art Painting* domain and so slightly improving class recognition.

4 – Cross Domain Validation

As said before, previous experiments are performed exploiting target domain during training phase in order to address the domain confusion. But performing hyperparameter search directly on test data (and so without any validation step) it is not a common pattern because the model should not know that data previously. So, fine-tuning a model for the identical specific target to test in deep learning field is considered as cheating.

Then, it is tried to apply the validation phase by training the network ever using images from *Photo* domain but now testing it by exploiting *Cartoon* domain in first place and *Sketch* one later. Goodness of model obtained by different hyperparameter sets is evaluated on these domains. Configuration that will provide best results in term of average accuracy over *Cartoon* and *Sketch* domain will be used to test the network on *Art Painting* domain. Clearly will be very hard getting results as good as before.

First grid search involves use of DANN without applying domain adaptation technique. Parameter search is carried out changing batch size and learning rate, performing SGD optimizer and considering as constant following values:

Optimizer	SGD
Gamma	0.1
Momentum	0.9
Weight decay	5e-5
Number of Epochs	30
Step Size	20

Results of grid search are represented by means and standard deviations calculated over two runs of code for each configuration applied on each domain and then are collected in the table below. This criterion will be applied over all experiments developed on this section of the report.

Batch Size	*Learning Rate*	*Accuracy on Cartoon domain*	*Accuracy on Sketch domain*	*Final Accuracy*
64	0,0001	(30,51 ± 1,45) %	(31,07 ± 3,19) %	(30,79 ± 0,40) %
64	0,001	(29,52 ± 0,66) %	(41,17 ± 0,63) %	(35,34 ± 8,23) %
64	**0,01**	**(34,32 ± 0,93) %**	**(38,22 ± 2,82) %**	**(36,27 ± 2,75) %**
64	0,1	(16,60 ± 0,00) %	(19,65 ± 0.00) %	(18,13 ± 2,16) %
128	0,0001	(29,74 ± 0,12) %	(23,15 ± 1,24) %	(26,44 ± 4,66) %
128	0,001	(31,36 ± 1,39) %	(38,88 ± 3,44) %	(35,12 ± 5,32) %
128	0,01	(28,78 ± 1,53) %	(33,18 ± 2,54) %	(30,98 ± 3,11) %
128	0,1	(16,94 ± 0,48) %	(19,65 ± 0.00) %	(18,30 ± 1,92) %
256	0,0001	(29,08 ± 1,48) %	(20,27 ± 1,92) %	(24,67 ± 6,22) %
256	0,001	(31,06 ± 3,08) %	(34,80 ± 4,86) %	(32,93 ± 2,64) %
256	0,01	(27,88 ± 0,69) %	(33,65 ± 1,51) %	(30,77 ± 4,08) %
256	**0,1**	**(16,60 ± 0,00) %**	**(11,86 ± 11,02)%**	**(14,23 ± 3,35) %**

Table 8 *- Cross Validation without domain adaptation*

Best configuration is highlighted by green color and shows that a batch size equal to 64 and a quite large starting learning rate equal to 0,01 provide best average accuracy result. So, also on *Cartoon* and *Sketch* domains the same relatively high learning rate allows to obtain best results without applying *Domain Adaptation*.

Details about loss and accuracy trend are represented by following plots.

Figure 12 *- Loss and accuracy from training on Cartoon domain*

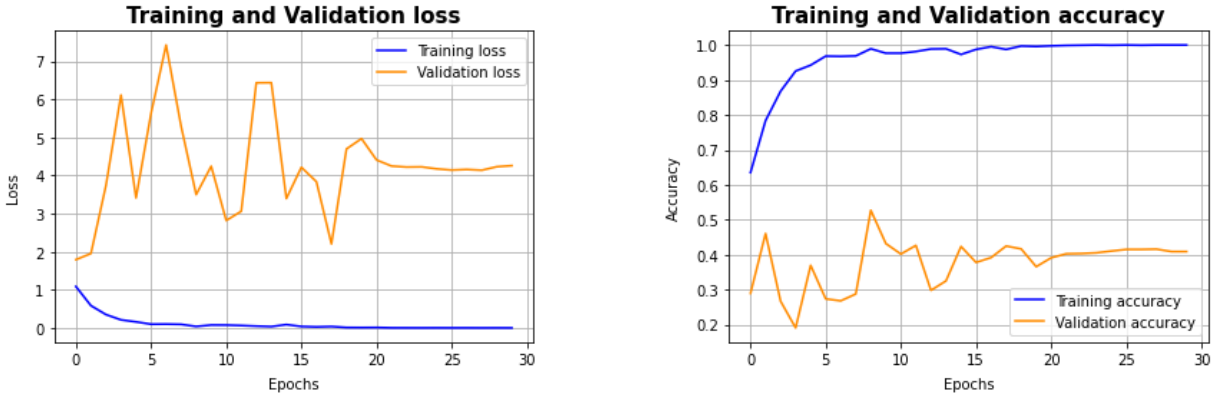


Figure 13 *- Loss and accuracy from training on Sketch domain*

As it is possible to see, trend of validation loss and validation accuracy is highly oscillating, and begins to stabilize after 20th epoch, when learning rate decreases. The gap between train and validation loss reveals overfitting on training data. On the other hand, worst results are obtained increasing learning rate to 0,1 and considering 256 as batch size. In this case both training and validation losses diverge, and accuracy of model is not improving during training epochs.

Later, decaying learning rate LR_p described in the previous section of the report is applied on cross domain validation. Results achieved are summarized in the table below.

Batch Size	*Learning Rate*	*Number of Epochs*	*Accuracy on Cartoon domain*	*Accuracy on Sketch domain*	*Final Accuracy*
128	LR_p	30	(30,29 ± 0,06) %	(39,68 ± 0,47) %	(34,99 ± 6,64) %
128	LR_p	50	(29,25 ± 1,18) %	(39,15 ± 3,74) %	(34,20 ± 7,00) %

Table 9 *- Results obtained by applying cross domain validation by using decaying learning rate* LR_p

Data shows as LR_p allows DANN to obtain better results on *Sketch* domain, although ones on *Cartoon* domain are not improved. Then, final average accuracy is not increased from this choice.

Finally, the network is trained by using best hyperparameters found during previous validation steps and tested on *Art Painting* images. DANN so tuned gets an accuracy score equal to **(46,45 ± 1,61)%**. As expected, evaluating hyperparameters on other domains cannot provide better results than ones assessed by a grid search on the final target domain because the model parameter choice is optimized toward other domains.

Afterward, DANN is trained by applying domain adaptation technique through domain classifier branch. Grid search is now performed considering following variable values:

- *Batch size* = **64**, **128**
- *Learning rate* = **0.01**, **0.001**, **0.0001**
- *Alpha* = **0.05**, **0.1**, **0.5**, **1**

Results are summarized in the table below.

Batch Size	*Learning Rate*	*Alpha*	*Accuracy on Cartoon domain*	*Accuracy on Sketch domain*	*Final Accuracy*
64	0,01	0,05	(49,04 ± 1,41) %	(19,65 ± 0,00) %	(34,35 ± 16,99) %
64	0,01	0,1	(16,94 ± 0,48) %	(19,65 ± 0,00) %	(18,30 ± 1,59) %
64	**0,01**	**0,5**	**(16,6 ± 0,00) %**	**(19,65 ± 0,00) %**	**(18,13 ± 1,76) %**
64	0,01	1	(16,6 ± 0,00) %	(19,65 ± 0,00) %	(18,13 ± 1,76) %
64	0,001	0,05	(31,10 ± 0,06) %	(40,28 ± 2,57) %	(35,69 ± 5,50) %
64	0,001	0,1	(36,48 ± 5,01) %	(38,63 ± 2,35) %	(37,55 ± 3,43) %
64	0,001	0,5	(16,6 ± 0,00) %	(19,65 ± 0,00) %	(18,13 ± 1,76) %
64	0,001	1	(16,6 ± 0,00) %	(19,65 ± 0,00) %	(18,13 ± 1,76) %
64	0,0001	0,05	(30,57 ± 0,81) %	(32,32 ± 1,12) %	(31,44 ± 1,29) %
64	0,0001	0,1	(31,64 ± 0,39) %	(30,46 ± 0,77) %	(31,05 ± 0,84) %
64	**0,0001**	**0,5**	**(45,76 ± 1,18) %**	**(32,71 ± 4,86) %**	**(39,23 ± 8,07) %**
64	0,0001	1	(12,99 ± 1,17) %	(18,94 ± 5,30) %	(15,96 ± 4,65) %
128	0,01	0,05	(45,82 ± 0,78) %	(19,65 ± 0,00) %	(32,73 ± 15,11) %
128	0,01	0,1	(35,59 ± 26,78) %	(19,65 ± 0,00) %	(27,62 ± 17,99) %
128	0,01	0,5	(16,6 ± 0,00) %	(19,65 ± 0,00) %	(18,13 ± 1,76) %
128	0,01	1	(16,6 ± 0,00) %	(19,65 ± 0,00) %	(18,13 ± 1,76) %
128	0,001	0,05	(31,79 ± 0,18) %	(37,91 ± 0,92) %	(34,85 ± 3,58) %
128	0,001	0,1	(36,54 ± 7,03) %	(38,98 ± 2,21) %	(37,76 ± 4,48) %
128	0,001	0,5	(16,6 ± 0,00) %	(19,65 ± 0,00) %	(18,13 ± 1,76) %
128	0,001	1	(16,6 ± 0,00) %	(19,65 ± 0,00) %	(18,13 ± 1,76) %
128	0,0001	0,05	(30,70 ± 2,02) %	(24,48 ± 2,07) %	(27,59 ± 3,96) %
128	0,0001	0,1	(32,27 ± 2,26) %	(26,89 ± 2,53) %	(29,58 ± 3,67) %
128	0,0001	0,5	(39,49 ± 0,15) %	(23,14 ± 2,55) %	(31,31 ± 9,55) %
128	0,0001	1	(43,9 ± 5,06) %	(19,69 ± 2,57) %	(31,80 ± 14,36) %

Table 10 *- Cross Validation with Domain Adaptation*

Now, best result is provided by tuning learning rate (that tunes class classifier speed) lower and domain regularization parameter α (that tunes domain classifier speed) larger with respect to the application of *Domain Adaptation* without validation phase described at point 2. It is possible to notice that higher values of α perform better on *Cartoon* domain, while best results on *Sketch* are achieved by lowest α tried together with a larger learning rate (0,001). This behavior could be caused by a greater proximity between *Photo* domain and *Sketch* domain.

Performing best configuration found, domain losses converge approximately to zero during the training phase on *Cartoon* domain, while on *Sketch* domain the source losses have a similar trend, but target domain loss diverges around 10^{th} epoch and then converge after 20^{th} epoch to a larger value. Respective plots are represented on the next page by Figure 14 and Figure 15.

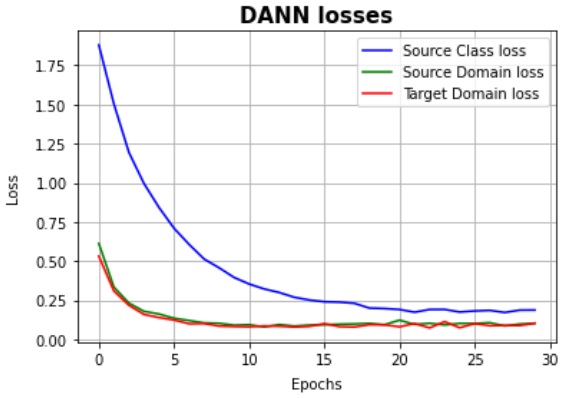


Figure 14 *- DANN losses on Cartoon domain*

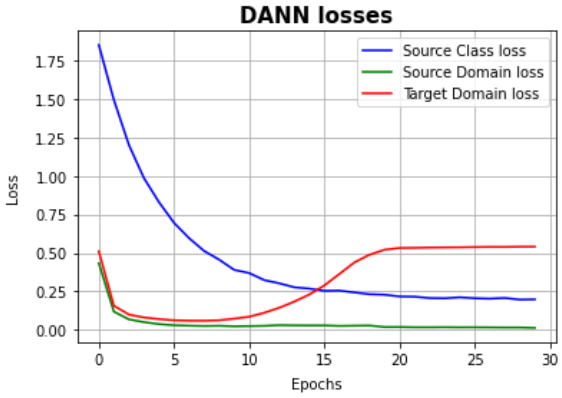


Figure 15 *- DANN losses on Sketch domain*

On the other hand, it is possible to see less overfitting as the gap between validation and training Cross-Entropy losses is being decreased that before, this time also without an oscillating trend, as shown below.

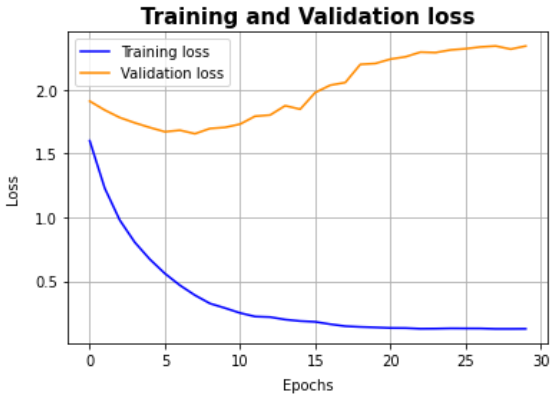


Figure 16 *- Training and validation loss on Cartoon domain*

Figure 17 *- Training and validation loss on Sketch domain*

Below are shown plots about training and validation accuracies. These are very good both on *Cartoon* and quietly good on *Sketch*, compatibly with other results obtained performing the grid search.

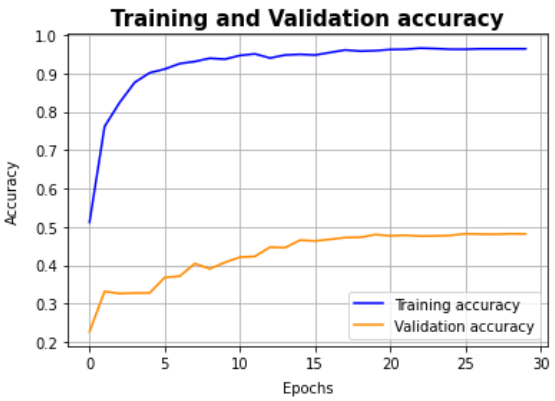


Figure 18 *- Training and validation accuracy on Cartoon domain*

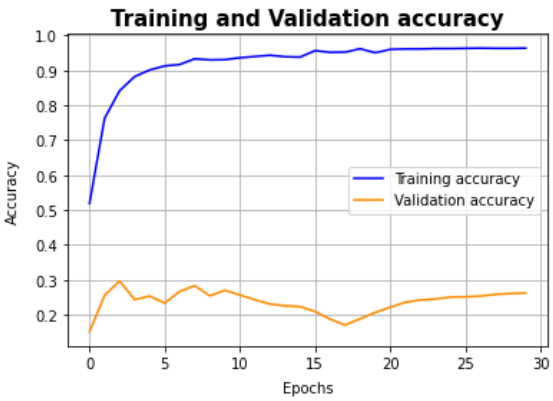


Figure 19 *- Training and validation accuracy on Sketch domain*

Improvements brought by *Domain Adaptation* on validation phase are summarized by the table below, where are considered the accuracy averages achieved by common parameters used in this section of the report. About DA accuracies, ones obtained with best α for each configuration are regarded.

Batch Size	*Learning Rate*	*Number of Epochs*	*Step Size*	*Average accuracy on Cartoon and Sketch Domains without DA*	*Average accuracy on Cartoon and Sketch Domains with DA*	*Mean value improvements*
64	0,001	30	20	(35,34 ± 8,23) %	(37,55 ± 3,43) %	+2,21 %
64	0,0001	30	20	(30,79 ± 0,40) %	(39,23 ± 8,07) %	+8,44 %
128	0,01	30	20	(30,98 ± 3,11) %	(32,74 ± 15,11) %	+1,76 %
128	0,001	30	20	(35,12 ± 5,32) %	(37,76 ± 4,48) %	+2,64 %
128	0,0001	30	20	(26,44 ± 4,66) %	(31,80 ± 14,36) %	+5,36 %

Table 11 *- DANN results comparison on domain cross validation*

Such as point 2, further trials are performed adopting a batch size equal to 128, a learning rate equal to 0,0001 and the α_p parameter.

Batch Size	*Learning Rate*	*Number of Epochs*	*Step Size*	*Alpha*	*Accuracy on Cartoon Domain*	*Accuracy on Sketch Domain*	*Final Accuracy*
128	0,0001	30	20	α_p	(49,62±0,91)%	(24,9±0,20)%	(37,26±14,28)%
128	0,0001	50	30	α_p	(51,35±0,94)%	(42,88±5,10)%	(47,11±5,73)%
128	0,0001	90	40	α_p	(43,67±2,02)%	(32,12±2,02)%	(37,89±6,87)%

Table 12 *- Results of DANN cross domain validation with domain adaptation and* α_p

Observing achieved results in Table 12, great improvements are possible to be seen if the number of epochs is increased to 50.

After these attempts, the network is trained on *Photo* domain images and tested on *Art Painting* ones by using best hyperparameters found during very last step, so using α_p, 128 as batch size, 0,0001 as learning rate, 50 as number of epochs and 30 as step size. DANN so tuned gets an accuracy score equal to **(47,19 ± 0,57)%**. So, applying *Domain Adaptation* with hyperparameters found through cross validation allows our model to get a better accuracy on our original task with respect to preceding configuration obtained without performing DA, that was **(46,45 ± 1,61) %**.

Conclusions

Once finished last attempts, it is possible to compare best outcomes obtained from point 3 and 4 about training DANN on *Photo* domain and testing it on *Art Painting* one. These are summarized in the following table.

	No Cross-Validation Domain	*Cross-Validation Domain*
No Domain Adaptation	(53,29 ± 0,84) %	(46,45 ± 1,61) %
Domain Adaptation	(53,45 ± 0,45) %	(47,19 ± 0,57) %

Table 13 *- Final comparison of results*

As speculated before applying Cross domain validation, hyperparameters found performing a grid search on *Cartoon* and *Sketch* domain have provided decreasing performances, although *Domain Adaptation* has however improved the result.

As a final consideration, it can be interesting to observe the different t-SNE plots about features extracted by G_y on source and target domain after one of DANN trainings with the same set of hyperparameters, before without applying and then applying *Domain Adaptation* tuning α equal to 0,5.

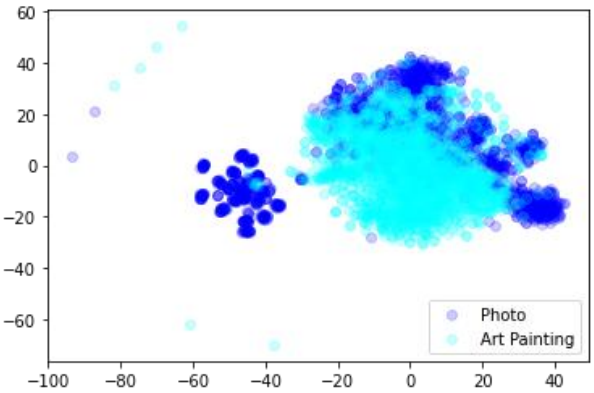


Figure 20 *- t-SNE of features without DA*

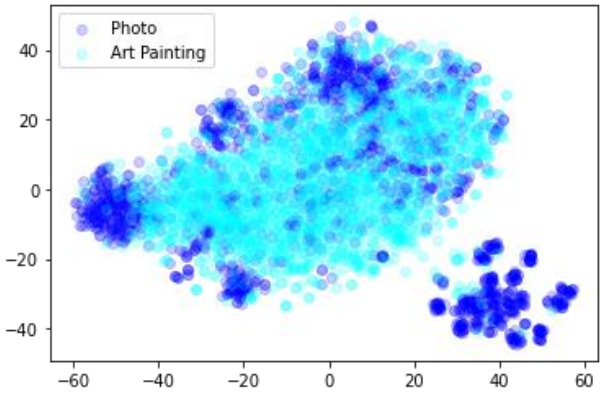


Figure 21 *- t-SNE of features with DA*

Pictures above show the 2-Dimensional distribution of features from *Photo* and *Art Painting* domain, firstly applying *Domain Adaptation* to the network (Figure 20) and then without DA application (Figure 21). As we can see, in Figure 21 feature areas are slightly most overlapped and so is graphically revealed that DANN is less able to discern between the two different domains.

References

[1] Domain-Adversarial Training of Neural Networks

https://arxiv.org/abs/1505.07818

[2] Deeper, Broader and Artier Domain Generalization

http://www.eecs.qmul.ac.uk/~dl307/Doc/Publication/2017/deeper-broader-artier.pdf

[3] ImageNet Classification with Deep Convolutional Neural Networks

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf