Machine Learning and Artificial Intelligence
Homework 3

Ilio Di Pietro, s266393

Prof. Tatiana Tommasi, A.Y. 2019/2020

1. Introduction

Deep learning architectures have brought impressive advances to the state-of-the-art across a wide variety of machine-learning tasks and applications. At the moment, however, these performance come only when a large amount of labeled training data is available.
One of the main problems of these models is that in many real world applications of machine learning, the distribution of the training data (on which the machine learning model is trained) is different from the distribution of the test data (where the learnt model is actually deployed). This is known as the problem of **Domain Adaptation**.
The goal of this homework is to implement a Domain Adaptation algorithm based on **AlexNet**, a network already used in the last homework.

1.1. Homework Sub-tasks

- Prepare the dataset by dividing the different domains;
- Implement a Domain Adversarial Neural Network (DANN) adding a separate branch inside of AlexNet;
- Validate hyperparameters of both original AlexNet and DANN by measuring performances on different domains (cross domain validation);
- Final test the two models (with and without adaptation) to make a comparison.

2. Data Preparation

The dataset used for this homework is **PACS**, which is a recently proposed benchmark dataset for domain generalization. It contains images from four domains (*Photo*, *Art painting*, *Cartoon* and *Sketch*) and 7 different classes of objects.
We divided it using *Photo* as **source domain** and *Art painting* as **target domain**, using the ImageFolder class to extract them from a GitHub directory.
We also took *Cartoon* and *Sketch* as validation sets.

2.1. Programming Environment

The proposed solution is obtained using Python 3.7 programming language. Google Colab framework is used to compute the requested data. Moreover, the most used libraries and packages are:

- PyTorch, an open-source machine learning library and deep learning framework for Python, based on Torch. The used modules and classes are nn and optim, specialized on creation and training of neural networks;
- matplotlib.pyplot: matplotlib is a Python 2D plotting library; pyplot provides a MATLAB-like interface;
- numpy, a package for scientific computing with Python;
- torchvision, a package made of popular datatsets and common image transformations for computer vision.

3. Implementation of DANN

As already seen, 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 distribution of data.
The first part of the homework concerns in modifying a Convolutional Neural Network already studied (AlexNet) introducing a binary classifier which goal is to discriminate between source and target domain. In this way, while the first part of the net continues to extract features, the discriminator part will be increasingly confused in distinguish source from target domain.
More in particular, our DANN architecture(shown in Figure[1]) is made up of three different groups of layers:

- **Gf**: convolutional layers of AlexNet, which are feature extractors;
- **Gy**: fully connected layers of AlexNet with 7 final output neurons which goal is to predict the label of the input image;
- **Gd**: fully connected layers with 2 final output neurons which implements the binary classifier.

Parameters of the first branch are updated with backpropagation, which takes in account both the losses on Gy and Gd.
Gradient reversal layer is applied to connect feature extractor Gf to domain discriminator Gd. It multiplies the gradient by a negative quantity (-alpha).

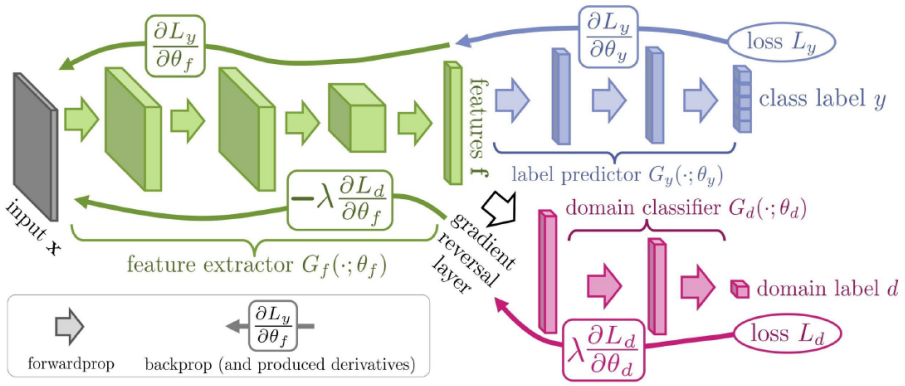


Figure 1: DANN architecture

To implement this net we added a new class called *DANN* which initially contained only the source code of AlexNet(i.e. Gf and Gy). To add the new branch Gd we created a new classifier in the *init* function as a copy of AlexNet classifier, which contains the fully connected layers.
To implement the Gradient Reversal Layer we also modified the *forward* function, using the parameter ALPHA as a flag to indicate if the current data must go to Gy or Gd: if ALPHA is specified when the *forward* function is called, then it is called the *ReverseLayerF* function which reverts the gradient and computes the loss for the discriminator part; otherwise a standard loss for the object classifier is computed. The starting code to implement both AlexNet and Gradient Reversal Layer was taken from GitHub.

4. Domain Adaptation

To better understand the differences between a simple CNN and a DANN beyond the theoretical aspects, we have trained and tested them both.
From the beginning, with the aim of getting more precise results, we have decided to apply the validation step. In this way we have been able to tune in a proper way a set of hyperparameters, i.e. all of those parameters dealing with the structure of the net (as we have seen in the previous homework); in particular we focused on:

- **Learning rate**: hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated;
- **Batch size**: number of samples processed before the model is updated;

- **Number of epochs**: total number of iterations over dataset;
- **Step size**: indicates how many epochs must pass before decreasing learning rate.

Only for DANN we also considered **ALPHA** as a parameter to tune, which represent a domain adaptation parameter.
For both the proposed models we have imported weights pre-trained on ImageNet dataset, to allow us to achieve better result on classification, otherwise impossible due to the small size of PACS dataset.
After some runs we have noticed that, thanks to transfer learning from ImageNet, in a few epochs an high value of accuracy was reached by both of our models, so we have decided to take as a reasonable value for the number of epochs 15, and 10 for step size. We left these values as constants to decrease the computational cost of the best set of hyperparameters.

4.1. AlexNet

Although we were using a normal convolutional neural network, the proposed dataset forced us to make different choices for the evaluation of the hyperparameters.
Due to the fact that training set(*Photo*) and test(*Art painting*) set belongs to different domains, we could not do validation as the accuracy computation on a subset of the train set as we are used to doing. Indeed, using only one domain to train and validate our model would have led the network to adapt to a single distribution of data and consequently to a misleading accuracy.
A more reasonable solution was to perform validation using the remaining two domains, to better adapt our net to different data distributions. Eventually, the average value between the value of accuracy obtained in the last epoch on *Cartoon* and on *Sketch* set was chosen as comparison measure to choose the best hyperparameters and consequently the best net.
Various sets of hyperparameters have been taken into account and they are shown in Table[1]. Their accuracy results are shown in Table[2].

Number of set	Batch size	Learning rate
1	128	0.001
2	128	0.002
3	128	0.005
4	**128**	**0.01**
5	256	0.001
6	256	0.002
7	256	0.005
8	256	0.01

Table 1: Different set of hyperparameters proposed for AlexNet

Number of set	Accuracy on Cartoon(%)	Accuracy on Sketch(%)	Average(%)
1	22.0	30.7	26.3
2	25.7	22.4	24.0
3	26.0	38.4	27.2
4	27.4	37.6	**32.5**
5	22.3	20.1	21.2
6	25.5	23.0	24.3
7	25.7	23.7	24.7
8	25.9	27.6	26.8

Table 2: Accuracy results for each previous set

The higher value of average accuracy resulted 32.5%, corresponding to the following set of hyperparameters:

- **Learning rate**: 10^{-2};
- **Batch size**: 128;
- **Number of epochs**: 15;
- **Step size**: 10.

These values have allowed us to reach an accuracy of 47% on test set.
Figure[2] and Figure[3] shows the obtained results.

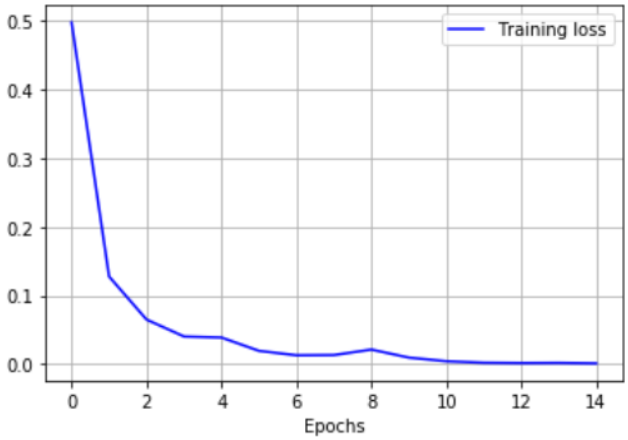
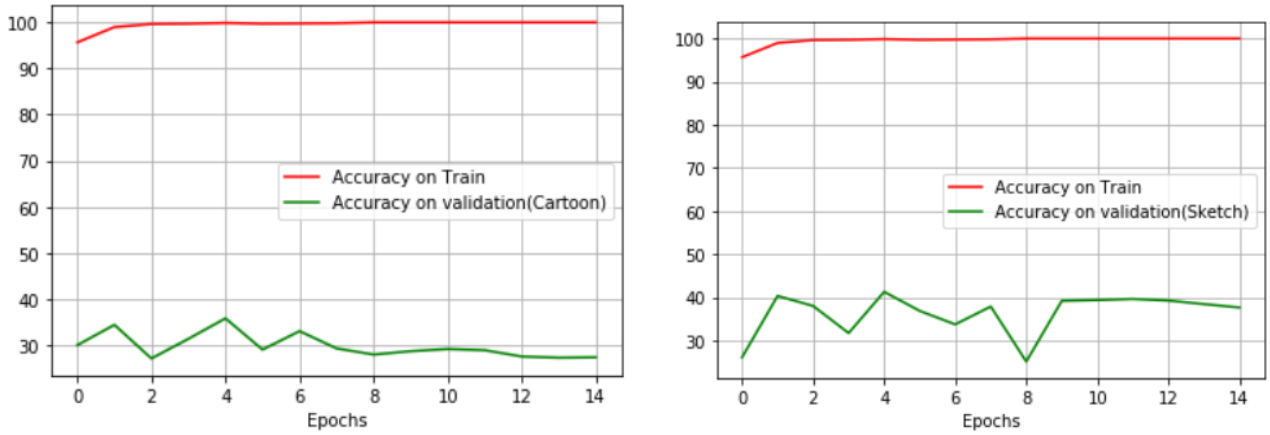


Figure 2: Loss on Training set (Photo)



((a)) *Validation: Cartoon* ((b)) *Validation: Sketch*

Figure 3: Accuracies on Training set (Photo) and on different Validation sets

4.2. DANN

Train and validate a net with domain adaptation turned out to be the most challenging part of the homework. Practically, our goal is: given a fully labeled source dataset (*Photo*) and an unlabeled target dataset (*Art painting*), to learn a model which can generalize to the target domain while taking the domain shift across the datasets into account.
The training part has been made applying a technique called *Transductive Learning*, in which the model has access also to unlabelled target samples. We have indeed split the single training iteration in three steps:

1. Training on source labels by forwarding data to Gy to compute loss and to update gradient. this part represent a standard training procedure;
2. Training the discriminator by forwarding source data to Gd;
3. Training the discriminator by forwarding target data to Gd.

To look for a more accurate solution, validation part has been performed and also in this case, with the aim of taking into account the different data distributions, we have used both *Cartoon* and on *Sketch* set.
Different set of hyperparamenters (listed in Table[3]) are tested, performing a 'manual' grid search. Since in training phase it is still involved the target set(which in case of validation it is the validation set), we could not perform a single validation with both *Cartoon* and *Sketch* as it was for AlexNet.
In practice, for each set of hyperparameters we have defined two different DANN, one trained and validated on *Photo-Cartoon* and the other to *Photo-Sketch*, to compute in a separate way accuracy and loss.
Eventually, to have a global evaluation of how good is a particular model, we have considered the lasts value of accuracy for both of the two nets and we averaged it.
Table[4] shows average accuracy results for each model proposed in Table[3].

Number of set	Batch size	Learning rate	ALPHA
1	128	0.001	0.01
2	128	0.001	0.05
3	128	0.001	0.1
4	128	0.005	0.01
5	128	0.005	0.05
6	**128**	**0.005**	**0.1**
7	128	0.01	0.01
8	128	0.01	0.05
9	128	0.01	0.1
10	256	0.001	0.01
11	256	0.001	0.05
12	256	0.001	0.1
13	256	0.005	0.01
14	256	0.005	0.05
15	256	0.005	0.1
16	256	0.01	0.01
17	256	0.01	0.05
18	256	0.01	0.1

Table 3: Different set of hyperparameters proposed

Number of set	Accuracy on Cartoon(%)	Accuracy on Sketch(%)	Average(%)
1	30.2	28.5	29.3
2	30.5	36.4	33.5
3	33.1	33.8	33.5
4	32.6	37.9	35.3
5	36.2	40.8	38.5
6	52.8	39.9	**46.3**
7	34.2	37.8	36.0
8	48.3	4.1	26.2
9	54.9	19.6	37.3
10	26.8	23.6	25.2
11	33.6	29.5	31.6
12	33.1	24.0	28.6
13	32.3	31.5	31.9
14	34.4	35.6	35.0
15	37.2	34.9	36.0
16	36.2	30.0	33.1
17	34.6	19.6	27.1
18	43.6	19.6	31.6

Table 4: Accuracy results for each previous set

As we can see from the last table, the best value of accuracy was reached considering the sixth net, which have:

- **Learning rate**: $5 * 10^{-3}$;
- **Batch size**: 128;
- **Number of epochs**: 15;
- **Step size**: 10;
- **ALPHA**: 10^{-1}.

The results in terms of loss are the following:

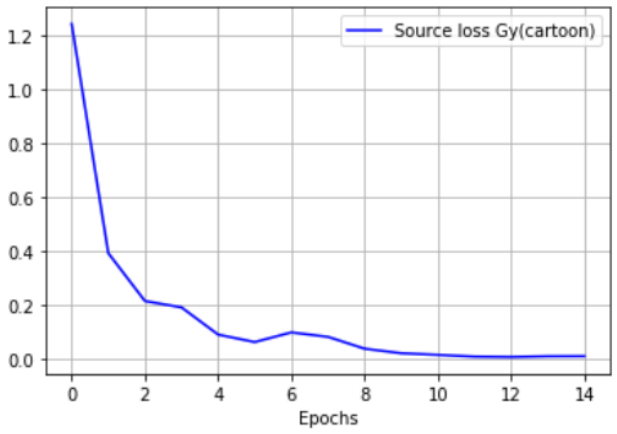
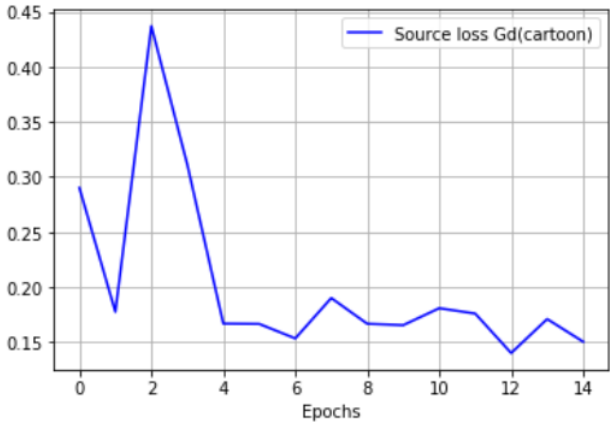
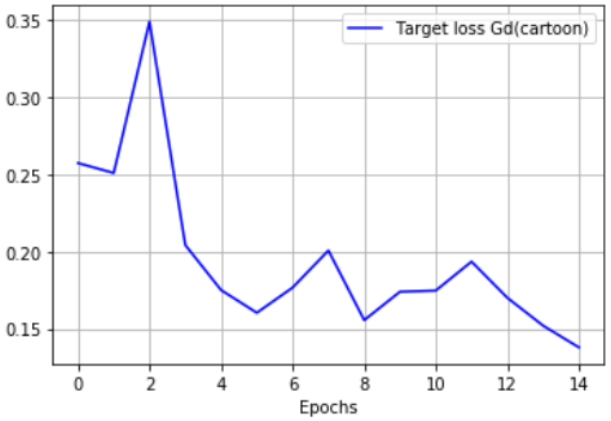


Figure 4: Loss trend for branch Gy (when the validation set is Cartoon)



((a)) *Loss on source data (Photo)*



((b)) *Loss on target data (Cartoon)*

Figure 5: Loss trend for branch Gd

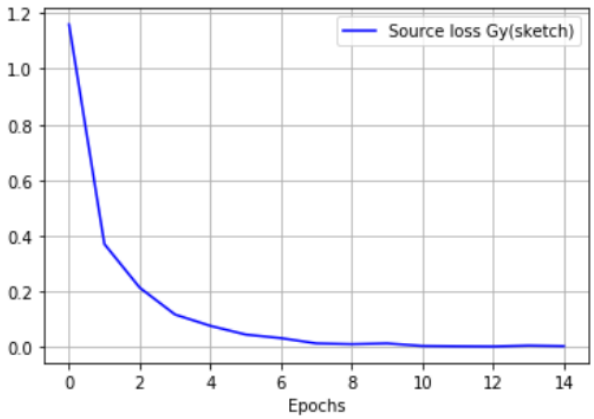
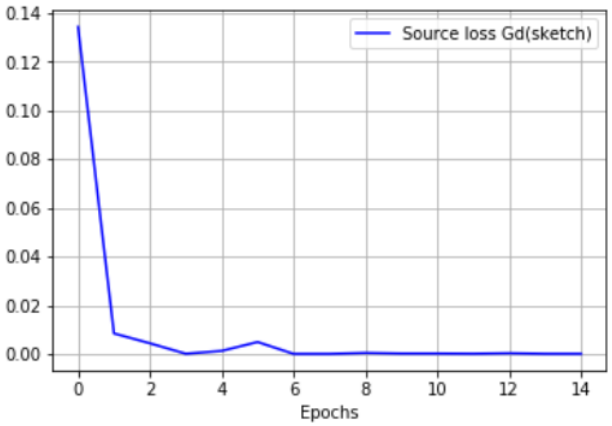
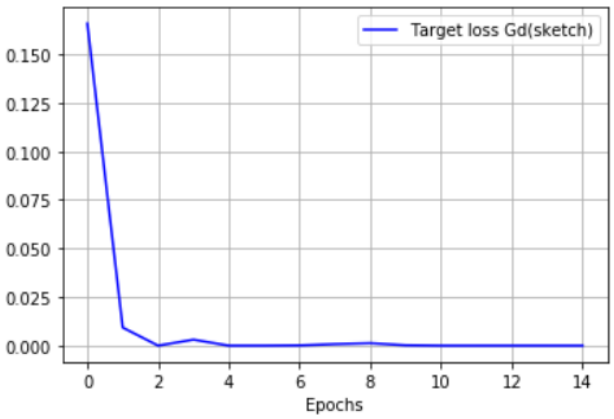


Figure 6: Loss trend for branch Gy (when the validation set is Sketch)



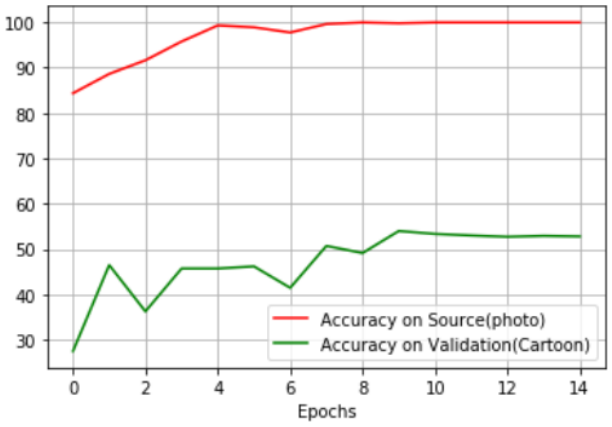
((a)) *Loss on source data (Photo)*



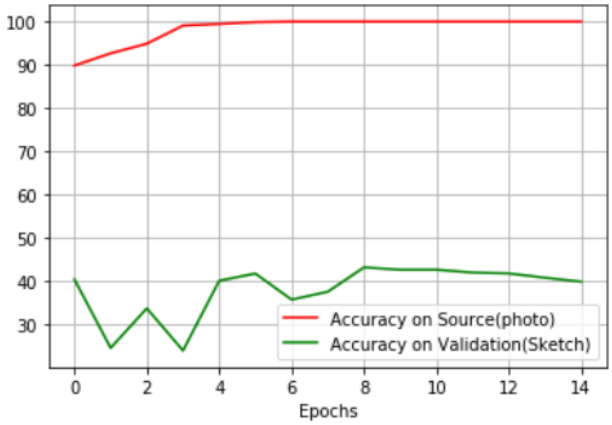
((b)) *Loss on target data (Sketch)*

Figure 7: Loss trend for branch Gd

As we can see, it would seem that there is an anomalous trend for the loss of the discriminator part. The practical goal of domain adaptation is to minimize the object classifier loss while maximizing the discriminator loss, to not make the net understand which domain belongs to the image and so to better generalize. In this terms we would expect contrary behavior on its trend, but this does not happen because what the reverse layer actually does is not to confuse the domain discrimination, but only the features extraction; the domain classifier indeed still tries to discriminate correctly between the two domains and, since in PACS dataset the domain are very distinguishable, the DANN net is very successful in its classification, so our loss trend is always decreasing. In terms of accuracy our results are:



((a)) *Accuracy Photo-Cartoon*



((b)) *Accuracy Photo-Sketch*

Figure 8: Accuracy trend on source and both validation sets

These best hyperparameters were finally used to train a new DANN network on *Photo* and *Art painting*. Figure[1] and Figure[1] show the loss obtained for each branch of the net.

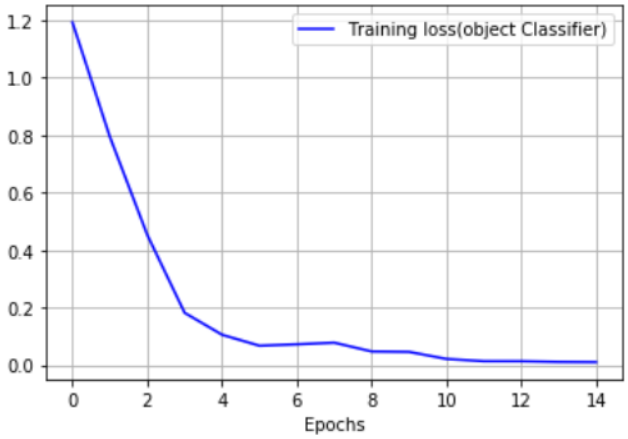
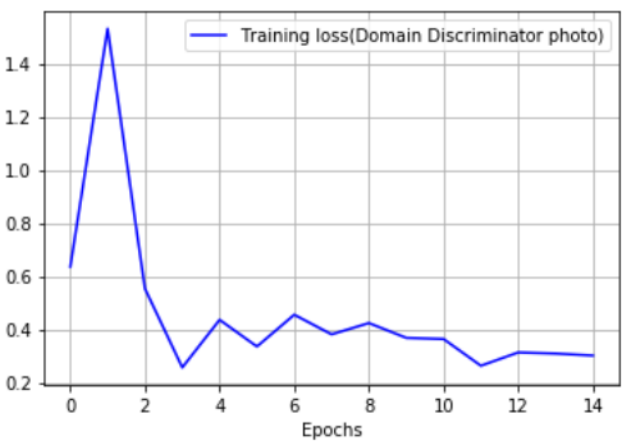
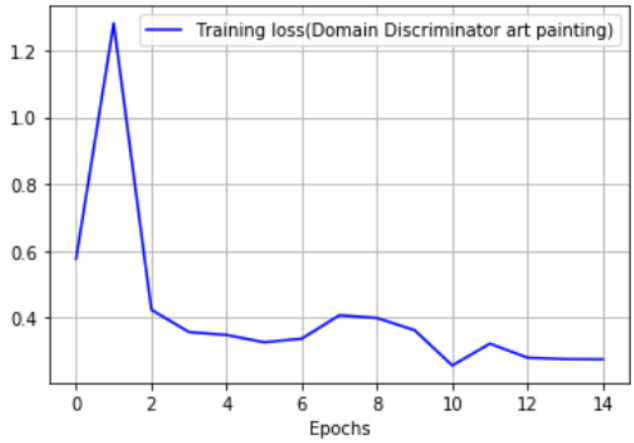


Figure 9: Loss trend for branch Gy with the best model



((a)) *Loss on source data (Photo)*



((b)) *Loss on target data (Art painting)*

Figure 10: Loss trend for branch Gd with the best model

The obtained accuracy on test set is 55%.

5. Conclusions

The purpose of this homework was to understand the difficulties that arise when we move from the ideal world to a more real situation, where data are does not always have an equal distribution. We have to change our way to learn from data and apply new models.
A Domain Adversarial Neural Network has been implemented starting to the code belongs to a particular Convolutional Neural Network: AlexNet. Both the models have been made into consideration and an hyperparameter tuning was performed to better improve the quality of the predictions.
A significant improvement was reached on the final test:

- **Accuracy on test set with DANN**: 55%;
- **Accuracy on test set with AlexNet**: 47%.

More in particular, DANN has performed well on validation on *Cartoon* domain with respect to ALexNet, with a gap of 25% (55% of accuracy with DANN and 30% for AlexNet), while on *Sketch* domain there was no significant improvement (42% vs 40%). Notice that AlexNet has achieved better level of accuracy on *Art painting* domains than the other two, maybe because the features of this domain were more similar with the distribution of data of ImageNet dataset, on which AlexNet was pre-trained.