POLITECNICO DI TORINO

Master's Degree in Data Science and Engineering

Machine Learning and Deep Learning

Homework 3:
Deep Domain Adaptation

Giovanni Cioffi

S277590

Homework 3 - Report

PACS dataset

PACS is a dataset of images belonging to 7 different semantic micro-categories: ‘dog’, ‘elephant’, ‘giraffe’, ‘guitar’, ‘horse’, ‘house’, ‘person’. Every category is depicted with 4 different macro-styles, which are called ‘domains’: Photo, Cartoon, Sketch, Art_painting. An example of how the ‘dog’ category is depicted in the four different domains is showed below in Figure 1.

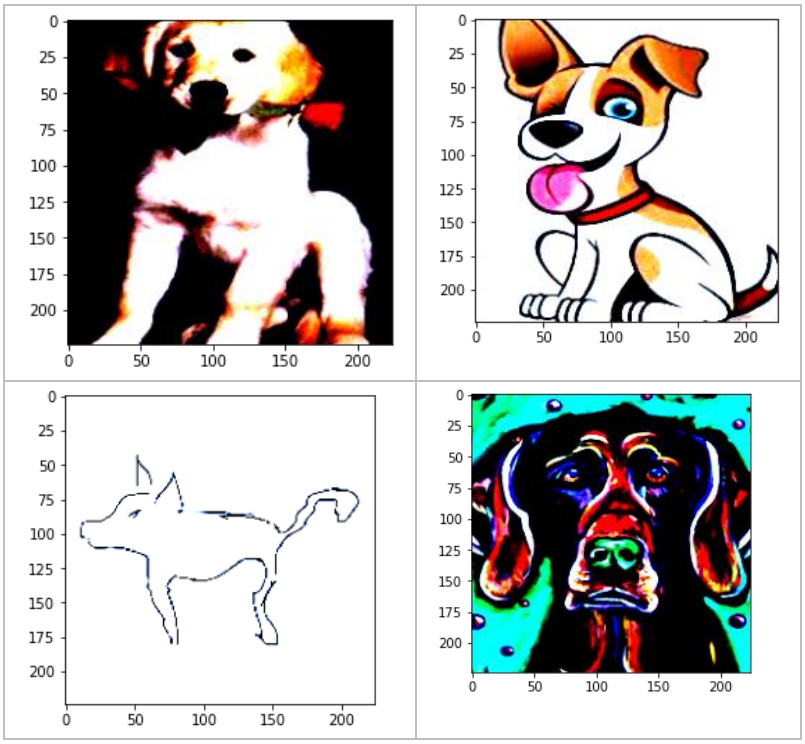

Figure 1 – dog in Photo (upper-left), Cartoon (upper-right), Sketch (bottom-left), Art Painting (bottom-right)

PACS dataset cardinality of data is not balanced among different domains: Photo has 1670 pictures, Art Painting 2048, Cartoon 2344, Sketch 3929 (Figure 2).

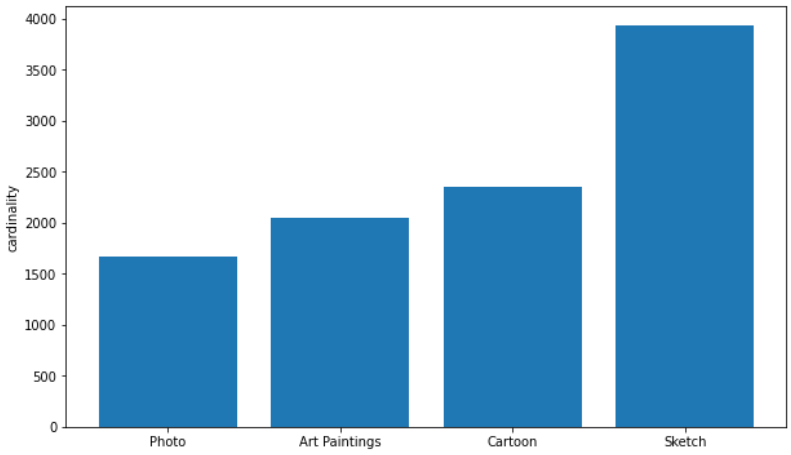

Figure 2 – Cardinality of domain datasets

The aim of this homework was to achieve an image classification task on PACS dataset training a model by using DANN adaptation (Domain-Adversarial Training of Neural Networks). The resulting model, trained on a customized version of AlexNet with a domain classifier and a gradient reversal layer, should be capable to

classify images depending on their category and to discriminate among different domains. For this homework, the source domain is Photo, while the target domain is Art painting.

AlexNet customized architecture

In order to implement DANN, a new densely connected layer with 2 output neurons (called 'domain_classifier') was added to the original AlexNet class infrastructure. It is separate branch with the same shape of AlexNet fully connected layers, with the difference that the classification is binary, in fact the last layer is a nn.Linear with out_features = 2. The picture below shows the implementation of the original AlexNet classifier and the new 'domain_classifier' branch.

```
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.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, 2),
)
```

Figure 2 - Implementation of the AlexNet classifiers

Both the original classifier and the new domain classifier were initialized with the weights of ImageNet; in other words, a pretrained net was created and weights and biases of the domain classifier Linear layers (except for the last one) were also initialized in the same way. Moreover, regarding the original classifier, his last layer was adapter to output 7 class weights, as the number of PACS dataset categories.

```
def alexnet(pretrained=False, progress=True, **kwargs):
    model = AlexNet(**kwargs)

    if pretrained:
        state_dict = load_state_dict_from_url(model_urls['alexnet'],
                                              progress=progress)
        model.load_state_dict(state_dict, strict=False)
        model.domain_classifier[1].weight.data = model.classifier[1].weight.data
        model.domain_classifier[1].bias.data = model.classifier[1].bias.data

        model.domain_classifier[4].weight.data = model.classifier[4].weight.data
        model.domain_classifier[4].bias.data = model.classifier[4].bias.data

    model.classifier[6] = nn.Linear(4096, 7)
    return model
```

Figure 3 – Preloading ImageNet weight on the domain_classifier

Then, the forward function was modified putting an alpha flag (alpha is the weight of the reversed backpropagation that multiplies the reversal), that if passed along with data in the training loop, tells the model to send data to domain classifier (the discriminator or Gd) and to revert the gradient applying Gradient

Reversal layer, otherwise data would be sent to the usual label predictor (Gy) and the training would proceeds standardly minimizing the label prediction loss (for source data) and the domain classification loss (for all data). Gradient reversal ensures that the feature distributions over the two domains are less distinguishable as possible for domain classifier (domain-invariant features).

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

    if (alpha is not None):
      reverse_feature = ReverseLayerF.apply(x, alpha)
      x = self.domain_classifier(reverse_feature)
      return x
    else:
      x = self.classifier(x)
      return x
```

Figure 3 – Implementation of the forward function with Gradient Reversal Layer

Train on source, test on target without adaptation

The first experiment carried out was training a model on Photos subset without adaptation and testing it directly on the Art Painting subset. This was equivalent to the classic training of a deep learning model, with the only difference that the training and evaluation phases were carried out on different domain datasets (even if they are composed of the same sub-classes). Furthermore, no validation set was used for tuning hyperparameters.

The datasets were loaded using ImageFolder from Pytorch, which takes as inputs the root (one of the four domains roots) and the transformations to be applied on the pictures. As AlexNet requires for pre-trained models, input images were reshaped with size 256, cropped with size 224 and normalized using mean = [0.485, 0.456, 0.406] and std = [0.229, 0.224, 0.225], as 3-channel RGB images of shape 3 x H x W. Then, Dataloaders are created for both source and target data in order to organize data in batches of size BATCH_SIZE.

Cross Entropy Loss was used as loss function of the weights with Stochastic Gradient Descent optimizer, parameters where manually selected: BATCH_SIZE = 128, LR = 0.001, MOMENTUM = 0.9, WEIGHT_DECAY = 5e-5, NUM_EPOCHS = 30, STEP_SIZE = 20, GAMMA = 0.1

Train on source, test on target with adaptation

Unsupervised domain adaptation process was applied in transductive way: the model was trained on both source and target datasets, the former with his own labels, the latter unlabelled. The model should be able to learn how to classify images depending on the category and how to distinguish between different domains (binary classification). Then, the so-called adaptation set coincide with the test set which is Art paining dataset. Same pre-processing steps mentioned in the previous paragraph were applied here, while the training loop is organized in a different way in order properly exploit all the data: since the target dataset is bigger than the source (2048 pictures against 1670, as mentioned in the bar-chart above), for each training epoch, a training loop iterates over the target_datatoader while the source_dataloader is iterated through an *iterator* with *next*. When batches of source_dataloader reach their limit because of the lower cardinality of source dataset, a new iterator is created, and the training is carried out again on the initial batches. This allows to use all the target data even if the source ones have a lower cardinality. The training phase is the divided in three main steps, as suggested in DANN paper:

1. **STEP 1.** optimizer.zero_grad() clears the gradients for every parameter in the optimizer. A batch of source data (Photos dataset) is sent to the label classifier Gy (without any alpha passed to the net, None is passed), loss is computed between outputs and labels, gradients are computed with loss.backwords().

2. **STEP 2.** A batch of source data (the same batch of point 1) is sent to the domain classifier Gd (without any alpha passed to the net), loss is computed between the outputs and a Pytorch tensor of size BATCH_SIZE of zeros (since here the classification is binary). Then, gradients are updated with loss.backwords()

3. **STEP 3.** Target data (Art Painting dataset) are sent to the domain classifier Gd with ALPHA parameterpassed along with data, loss is computed between the outputs and a tensor of ones. Finally, gradients are updated with loss.backwords(). Finally, optimizer.step() updates the value of the weights using the gradients

```
for epoch in range(NUM_EPOCHS):

  source_dataloader_iterator = iter(source_dataloader)

  for target_images, _ in target_dataloader:
    try:
      images, labels = next(source_dataloader_iterator)
    except StopIteration:
        source_dataloader_iterator = iter(source_dataloader)
        images, labels = next(source_dataloader_iterator)

    target_images = target_images.to(DEVICE)
    images = images.to(DEVICE)
    labels = labels.to(DEVICE)
    net.train()
    optimizer.zero_grad()

    # STEP 1
    outputs = net(images, None)
    loss_1 = criterion(outputs, labels)
    loss_1.backward()

    # STEP 2
    outputs = net(images, ALPHA)
    step2_labels = torch.zeros(BATCH_SIZE, dtype=torch.long)
    step2_labels = step2_labels.to(DEVICE)
    loss_2 = criterion(outputs, step2_labels)
    loss_2.backward()

    # STEP 3
    outputs = net(target_images, ALPHA)
    step3_labels = torch.ones(BATCH_SIZE, dtype=torch.long)
    step3_labels = step3_labels.to(DEVICE)
    loss_3 = criterion(outputs, step3_labels)
    loss_3.backward()

    optimizer.step()
    current_step += 1
```

Figure 4 - DANN implementation training steps

Comparison of the results

Since no validation set was available, all parameters were manually set depending on the actual score on the test set: the target Art painting dataset. This procedure is considered 'cheating', the hyperparameters' tuning must not be done on the test set because in the reality you are not supposed to have it available. The hyperparameter tuning must be rather done on the validation set in order to train a robust model, which is capable to generalize. Same learning rate, number of epochs, step size and gamma were initially used with and without adaptation. The result is a slight improvement using DANN (with alpha set to 0.05) in terms of accuracy on the test set, which increases from 48.5% to 49.8% as plotted in figure 5.

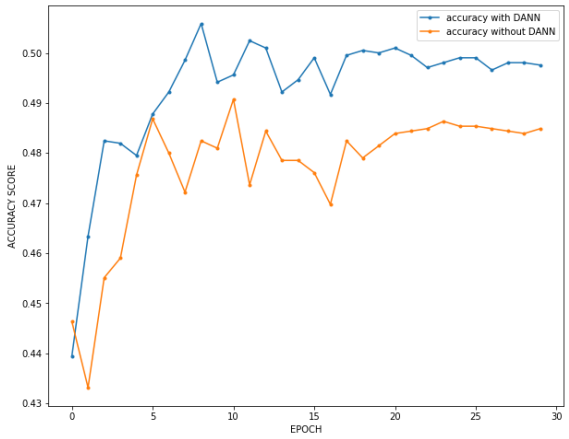

Figure 5 – Accuracy on target with and without DANN (vs epoch)

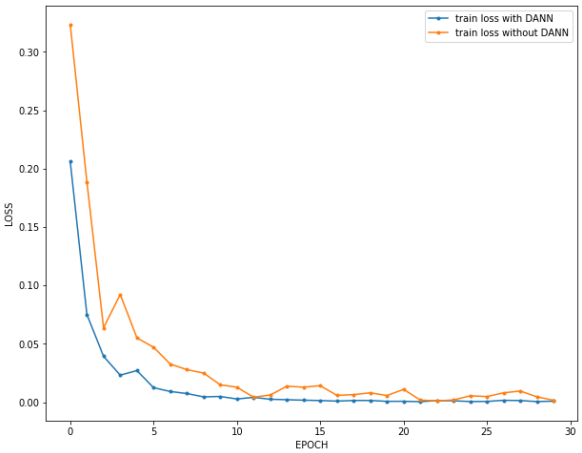

Figure 6 – Train loss with and without DANN (vs epoch)

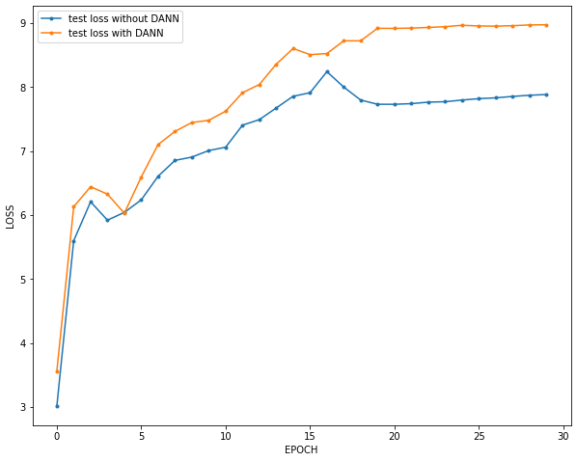

Figure 7 – Test loss with and without DANN (vs epoch)

Moreover, t-SNE was applied in order to obtain a graphical 2D representation of the features for a given class label. The feature representation for source and target datasets of the net trained with DANN and without DANN for class 2 is shown below.

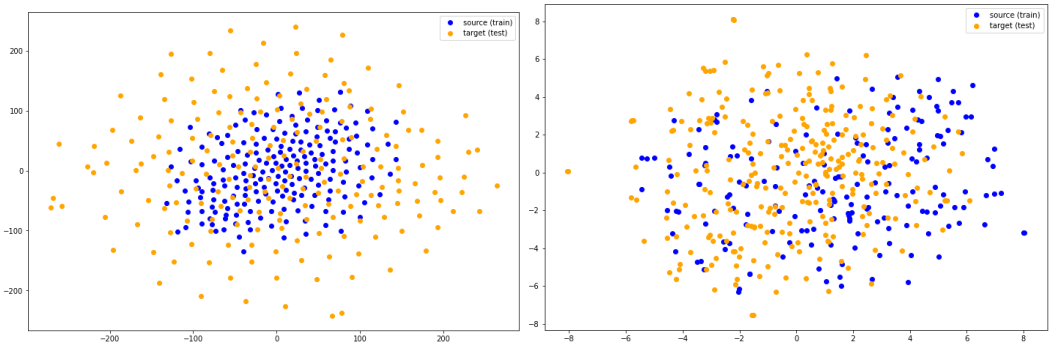

Figure 8 – Feature representation without DANN (left) and with DANN (right) of class 2 using t-SNE

The plots confirm that with DANN a more heterogeneous feature representation is obtained, because features appear mixed, while without DANN we get homogenous features for each dataset, in the sense that their appear separated for source and target.

Cross Domain Validation

Cross domain validation with and without DANN was exploited in order to find the best hyperparameters for the model.

Cross Domain Validation without DANN

Photo dataset was used as training set, Cartoon and Sketch datasets were used as validation sets. Cross domain validation without DANN adaptation was carried out in two steps: tuning of the learning rate with a fixed configuration of the other parameters and then tuning of the other hyperparameters by fixing the best-found learning rate. Five models with different learning rates were trained on Photo dataset, fixing other parameters in order to have a fair comparison between different learning rates, highlighting their effect on the model performances. The other fixed parameters were the following: NUM_EPOCHS=30, STEP_SIZE=20 with StepLR decaying policy and GAMMA=0.1, WEIGHT_DECAY=5e-05. Those models were all evaluated on Cartoon and Sketch datasets, the learning rate of the better performing model in terms of average accuracy score on them was chosen. The results are showed in the table below:

LR	ACCURACY ON CARTOON	ACCURACY ON SKETCH	AVERAGE ACCURACY
0.0001	0.25213	0.26562	0.25887
0.0005	0.24317	0.27946	0.26131
0.005	0.25810	0.31127	0.28469
0.001	0.24317	0.28124	0.26220
0.01	0.25554	0.29931	0.27742

Table 1 – Experiments tuning learning rate without DANN

Especially on Cartoon domain, the performances are quite low, the average accuracy do not overcome 30%. However, chosen the best learning rate (LR=0.005), other different sets of hyperparameters were tuned with a manual grid search. The best set of parameters was the one which resulted in the best average accuracy on Cartoon and Sketch datasets.

LR	NUM_EPOCHS	STEP_SIZE	GAMMA	WEIGHT_DECAY	ACCURACY ON CARTOON	ACCURACY ON SKETCH	AVG ACCURACY
0.005	30	20	0.5	0.00005	0.25426	0.24382	0.24904
0.005	30	20	0.1	0.00005	0.25554	0.30567	0.28061
0.005	30	10	0.5	0.00005	0.22610	0.25120	0.23865
0.005	30	10	0.1	0.00005	0.24189	0.31763	0.27976
0.005	30	20	0.1	0.005	0.31527	0.33469	0.32498

0.005	30	10	0.1	0.005	0.27005	0.30847	0.28926
0.005	30	20	0.5	0.005	0.26706	0.30771	0.28738
0.005	30	10	0.5	0.005	0.27218	0.30847	0.29032
0.005	50	20	0.1	0.005	0.27261	0.31941	0.29601
0.005	50	25	0.1	0.005	0.27986	0.32832	0.30409
0.005	50	30	0.1	0.005	0.26450	0.31305	0.28887

Table 2 – Experiments tuning other hyperparameters without DANN

The performances seem to be slightly better on sketch dataset, probably because of the higher cardinality of training data as mentioned above; however, the maximum average accuracy was about 0.325%. A model without adaptation was trained with the configuration highlighted in the table above without adaptation and evaluated on the Art Painting dataset. The resulting accuracy score on the target dataset is 50.56%.

Cross Domain Validation with DANN

For each configuration of the parameters (included alpha, the weight of the reverse backpropagation), two models were trained in transductive way with respectively Cartoon and Sketch as targets:

1. Training on labelled Photo dataset as source and Cartoon dataset as unlabelled target using DANN architecture with Gradient Reversal Layer. Validation phase on labelled Cartoon dataset.
2. Training on labelled Photo dataset as source and Sketch dataset as unlabelled target using Gradient Reversal Layer. Validation phase on labelled Sketch dataset.
3. Computation of the average accuracy score on Cartoon and Sketch

As in the previous cross domain validation, the accuracy scores on the target datasets were averaged and the model with the highest score was chosen.

LR	NUM_EPOCHS	STEP_SIZE	GAMMA	ALPHA	ACCURACY ON CARTOON	ACCURACY ON SKETCH	AVG ACCURACY
0.005	30	20	0.1	0.001	0.28114	0.30669	0.29391
0.005	30	20	0.1	0.005	0.29266	0.31585	0.30425
0.005	30	20	0.1	0.01	0.28370	0.26291	0.27330
0.005	30	20	0.1	0.05	0.41126	0.31891	0.36508
0.005	30	20	0.1	0.07	0.44283	0.24433	0.34358
0.005	50	20	0.1	0.05	0.47909	0.29931	0.38920

Table 3 – Experiments tuning other hyperparameters (including ALPHA) with DANN

The model trained with the highlighted configuration of Table 3 achieved an accuracy score on Art Painting of 53.27%, which is the highest accuracy seen so far. The behaviour of the three losses is plotted below in Figure 6, they all reach values close to zero after the 10^{th} epoch, nevertheless the model is hardly good at generalizing this knowledge on the Art Painting dataset, since the accuracy score on the target dataset is still quite low.

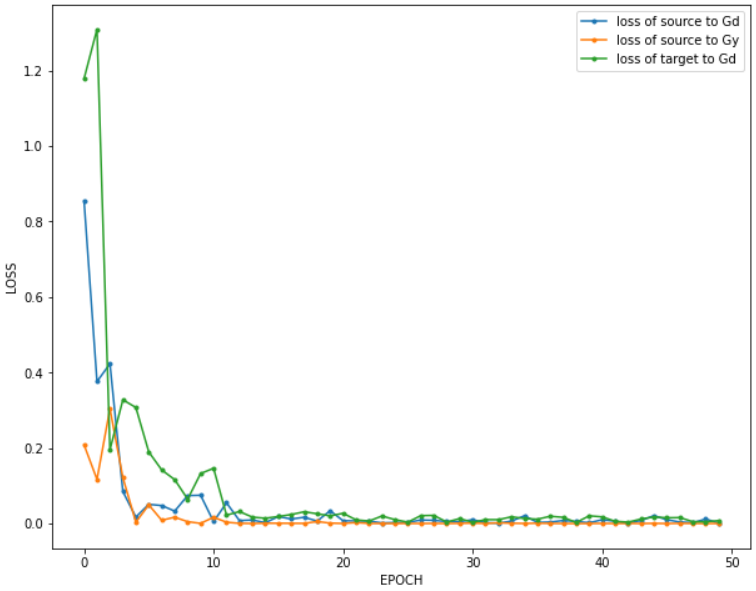

Figure 9 – Behaviour of the tree losses with DANN

This states that the domain adaptation problem is still an open problem, even if DANN architecture and Cross Domain Validation bring some improvements. Below in Table 4, the type of the trained models and the final scores and the target dataset are summarized.

TYPE OF MODEL	ACCURACY SCORE ON TARGET DATASET
Training on source dataset without DANN adaptation (no Cross Domain Validation)	48.48%
Training on source dataset with DANN adaptation after (no Cross Domain Validation)	49.75%
Training on source dataset without DANN adaptation after Cross Domain Validation	50.56%
Training on source dataset with DANN adaptation after Cross Domain Validation	53.27%

Table 4 – Summary of experiments and scores