Machine Learning and Deep Learning

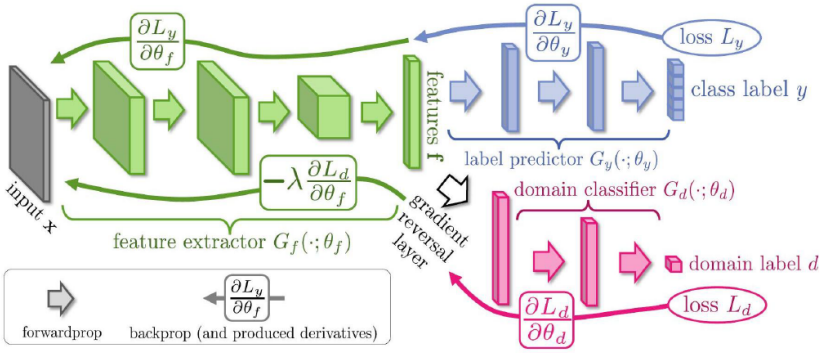
Politecnico di Torino

Homework 3 report

Student ID: s275090

Domain Adaptation

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 target distribution. The goal of this method is to align training features with test features.



The most famous example is the Domain-Adversarial Neural Network. It consists of two losses, the classification loss and the domain confusion loss. It contains a gradient reversal layer to match the feature distributions. By minimizing the classification loss for the source samples and the domain confusion loss for all samples.

The dataset

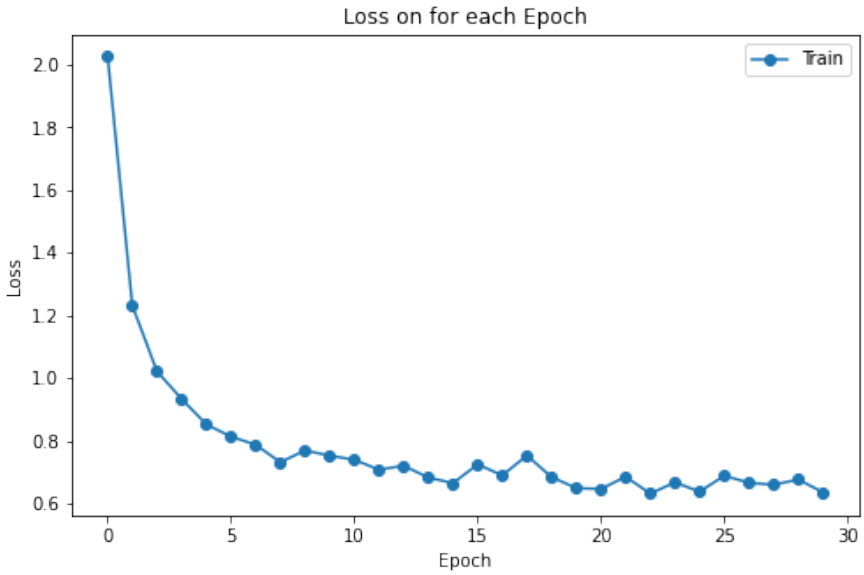
PACS is an image classification Dataset. The dataset contains 1776 images, divides in 4 domains: Photo, Art painting, Carton, Sketch. Each domain contains 7 classes.

Domain	**samples**
Art_painting	410
cartoon	1875
photo	1336
sketch	786

Initially for this work we train on Photo (1336 samples) and test on Art painting (410 samples). We change the Normalize function of Data Preprocessing to Normalize using ImageNet's mean and standard deviation *[(0.485, 0.456, 0.406), (0.229, 0.224, 0.225)].*

Implementing the Model

For a first test we are going to use a convolutional neural network: AlexNet, pretrained on ImageNet, and SGD with momentum as an optimizer. The initial hyperparameters are LR = 0.001, STEP_SIZE = 20, NUM_EPOCHS = 30.



We test the model on the test set, and we get an accuracy of *0,4268*.

Implementing training with DANN

To implement DANN in AlexNet we have to add a new densely connected branch with 2 output neurons. The network is composed by:

- An initial common path that extract feature from input and this is the AlexNet convolutional layer
- A branch that runs the label predictor, as the AlexNet fully connected layers
- A new separate branch with the same architecture of AlexNet fully connected layers where the gradient has to be reverted with the Gradient Reversal layer. The reversal is multiplied by an alpha factor, the weight of the reversed backpropagation.

```
self.dann = 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),
        )
```

Figure 1 - Architetture Dann

Both the original network and the new classifier are initialized with the weights of ImageNet, the last layers have an output of 7 and 2 neurons. To use the new branch, we implement a flag (if alpha is None) in the forward function to indicate where the batch of data needs to go.

```
if alpha is not None:
# gradient reversal layer (backward gradients will be reversed)
      reverse_feature = ReverseLayerF.apply(features, alpha)
      discriminator_output = self.dann(reverse_feature)
      return discriminator_output
# If we don't pass alpha, we assume we are training with
supervision
else:
      class_outputs = self.classifier(features)
      return class_outputs
```

Figure 2 – Flag in the forward function

So, the network is divided in three steps where trains jointly on the labelled task Photo and the unsupervised task and then test on Art Painting. In a single training iteration, first the network trains on source labels (Photo), gets the loss and updates gradient. Then it trains the discriminator by forwarding source data, gets the loss and updates gradient. Finally, it trains the discriminator by forwarding target data (Art painting), gets the loss and updates the loss.

After implementing DANN, we test the network with the same hyperparameters used before with a first ALPHA=0.05.

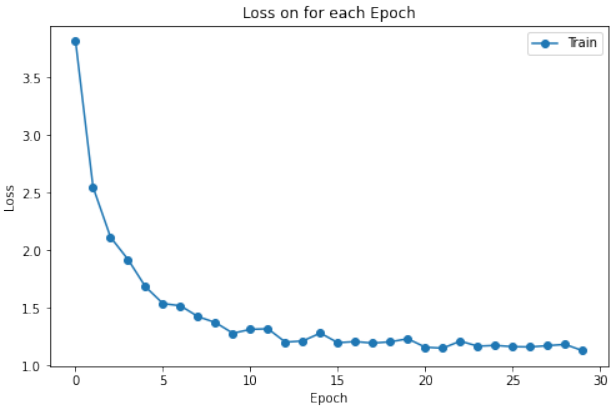


Figure 3 - Total loss network

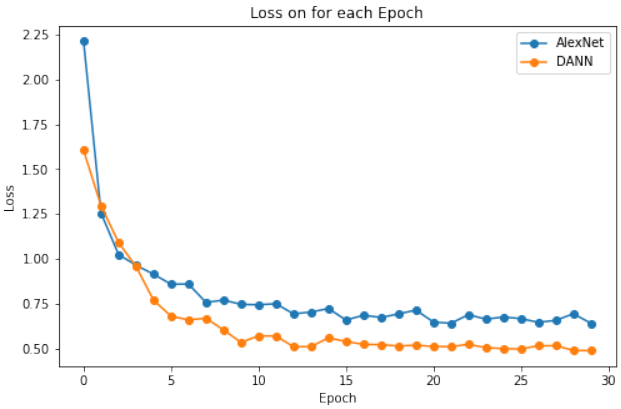


Figure 4 - Loss by branch

The trend of the graph (3) is similar to the previous one. We can see that there is a small contribution from the DANN compared to before. We get a better accuracy of *0.4512* on test set.

Cross Domain Validation

In this homework we don't use any validation step to validate the model, but we are looking at results in the test set and this is essentially cheating. Now we want to validate hyperparameters by measuring performances on Photo to Cartoon transfer and Photo to Sketch transfer.

To find the most appropriate hyperparameters we use a grid search, with and without Domain Adaptation, and average results for each set of hyperparameters. We use ParameterGrid, from sklearn library, to define the combination of hyperparameters to test.

```
hyperparameters = {'NUM_EPOCHS':[25,30],
                   'LR':[1e-3,1e-2],
                    'STEP_SIZE':[10,20],
                    'ALPHA':[0.5,0.3,0.1]}
```

In summary, for each epoch we evaluate the model, trained on photo, on Cartoon and Sketch, we calculate the accuracy and, in the end, we calculate the average accuracy for epoch. We will choose the model that gets the best result.

- Without DANN

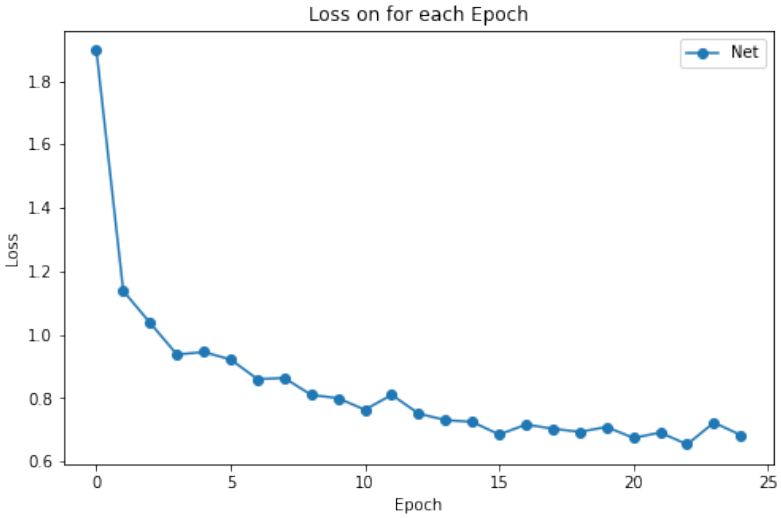


Figure 5 - Loss on train

After applying a grid search without DANN, we get an average accuracy of 0.42 on the validation sets, 0.51 on cartoon set and 0.30 on sketch set, with hyperparameters {'LR': 0.001, 'NUM_EPOCHS': 25, 'STEP_SIZE': 10}. Finally, we test the model on test set with the best hyper and we get an accuracy of 0.4341.

- With DANN

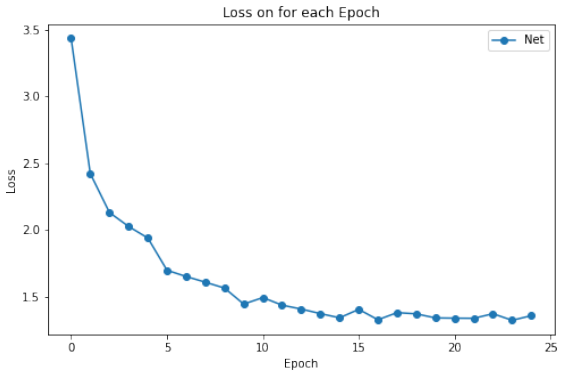


Figure 6 - Total loss network

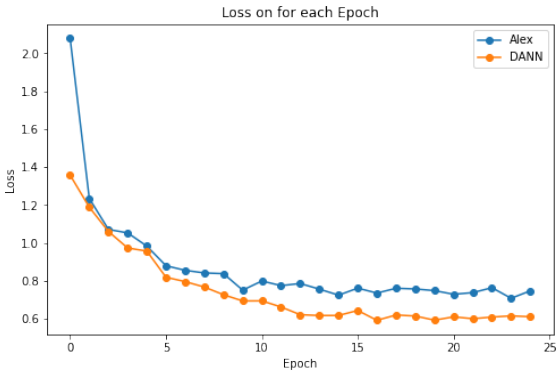


Figure 7 - Loss by branch

As before we apply a grid search with the same parameters. The best hyperparameters found are {'ALPHA': 0.5, 'LR': 0.01, 'NUM_EPOCHS': 25, 'STEP_SIZE': 10} where we get an average accuracy of 0.46. On test set we get 0.4317.

From the two results we don't get any improvement, the sketch set usually gives a bad accuracy probably because too different from photo set. However, by testing on multiple datasets we obtain hyperparameters that allow us to obtain more robust results and model.