



AML - HOMEWORK 3

Unsupervised Domain Adaptation

TA: Antonio Alliegro (name.surname at polito.it)

STEP 0 - OVERVIEW



HOMework PURPOSE

- The task is to implement **DANN**, a **Domain Adaptation** algorithm, on the **PACS** dataset using **AlexNet**.
- You are not provided with a code template, but you can re-cycle some snippets you used in HW2.

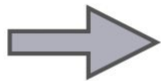
Data and resources: <https://github.com/antoalli/AML-Homework3-DA>
Fork this repo to download the data in your Colab environment.

WHY DOMAIN ADAPTATION

Generalization: Source (Train) = Target (Test)



Source

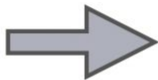


Target

Domain adaptation: Source (Train) \neq Target (Test)



Source (with Labels)



Target (No Labels)

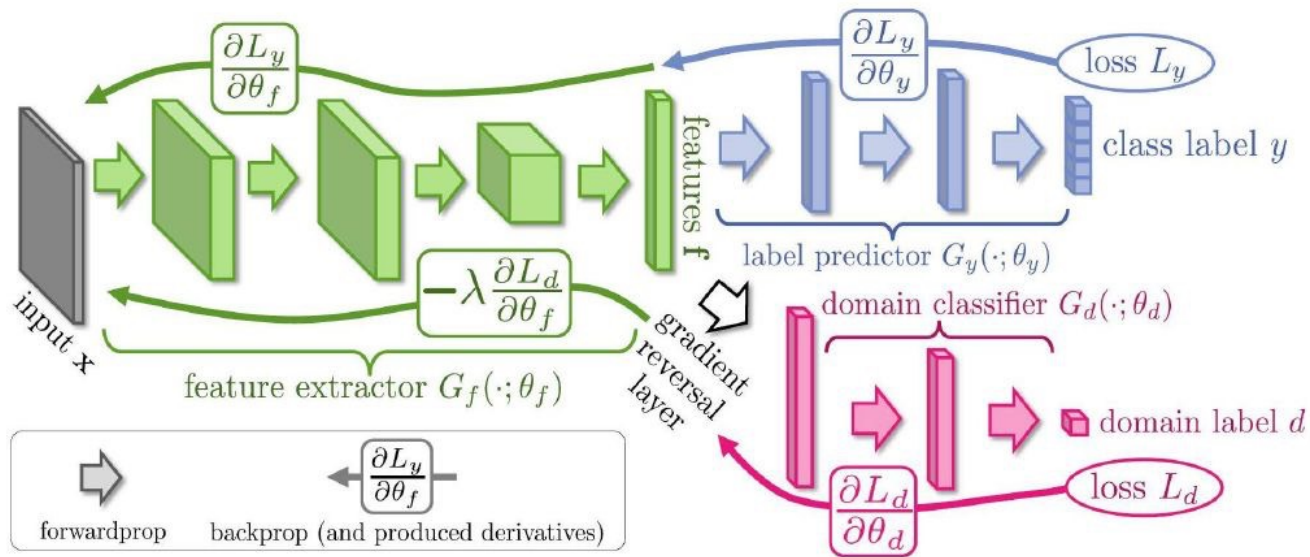
- Data has been collected and annotated in ideal conditions
 - Test data is from the same distribution of training data
- This may not always be the case, with test data coming from different distribution(s)
- **Domain Adaptation:** align train and test extracted features, make them live in the same space



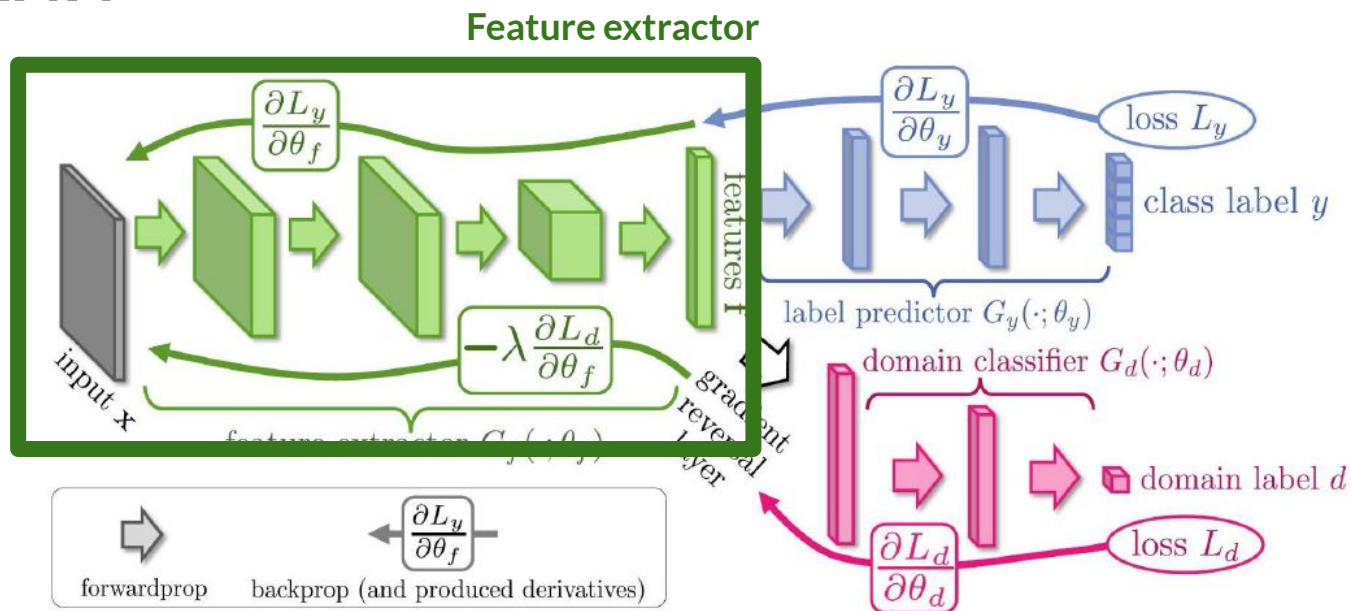
DANN

- Why do we need the features to be aligned?
 - Otherwise, the classifier will not work properly at test time, since it has been trained with features living in a different part of the space.
- How do we align them?
 - The goal is to train the feature extractor such that he is able both to produce features:
 - meaningful for the classifier to tell classes apart
 - **domain-wise indistinguishable**: they must capture characteristics of the image regardless its specific domain
 - DANN is a specific implementation designed to satisfy both requirements.

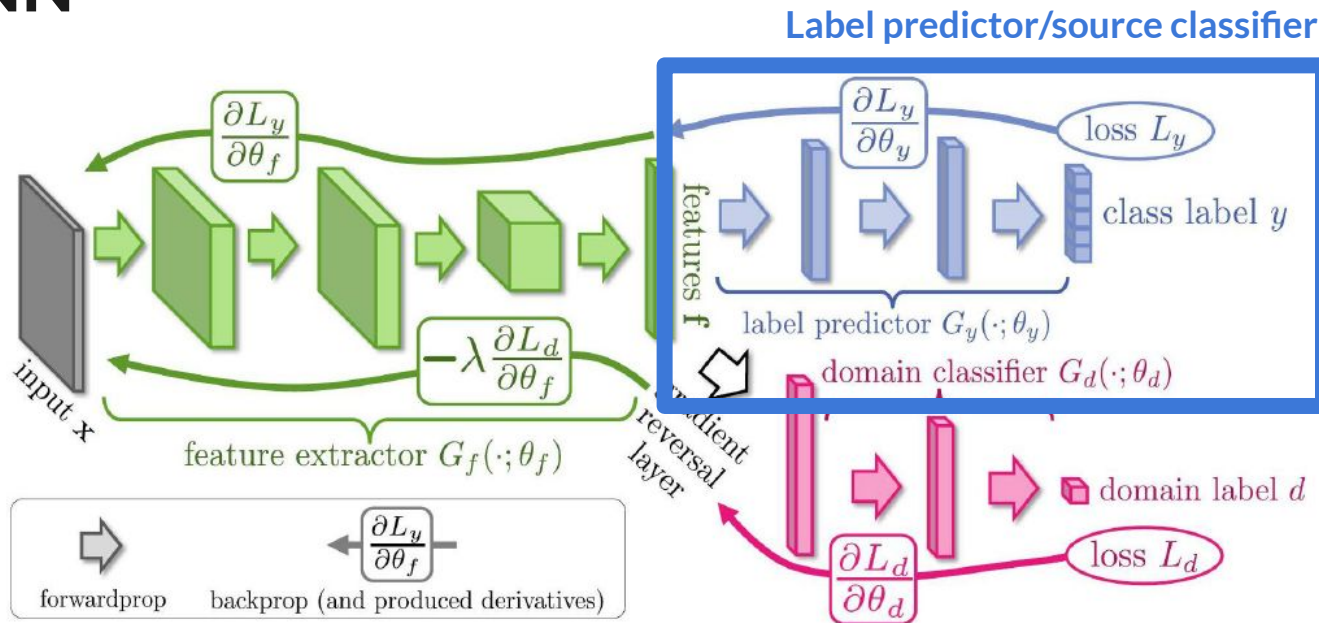
DANN



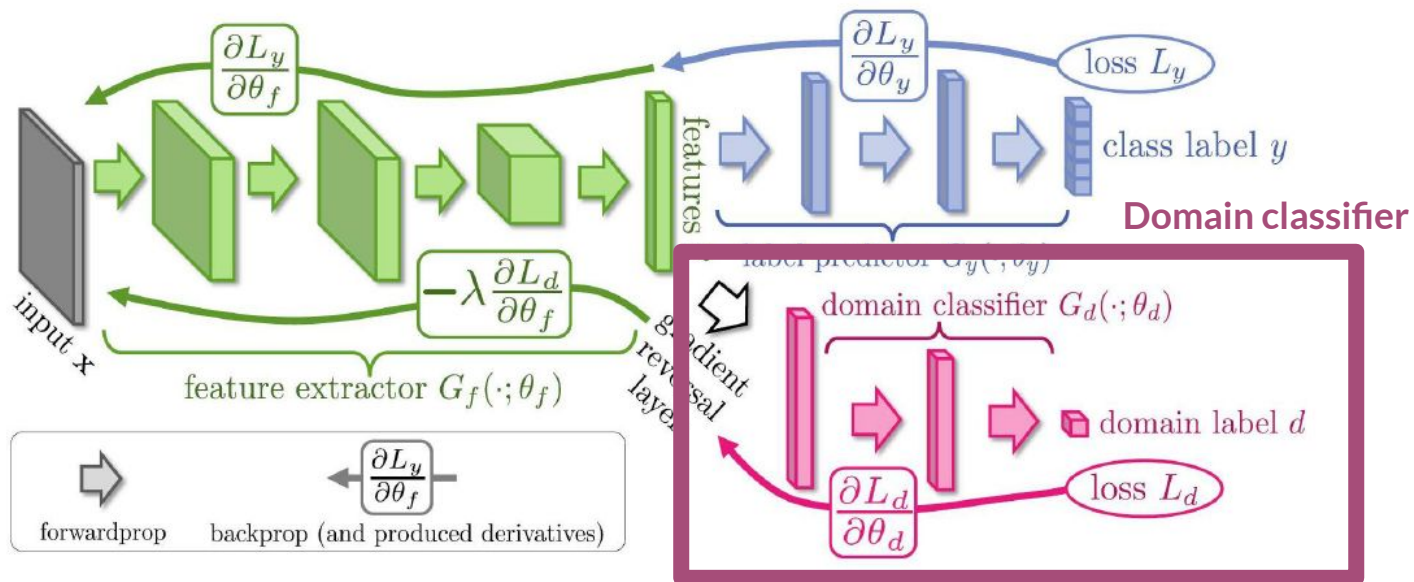
DANN



DANN



DANN





DANN - DOMAIN CLASSIFIER

- It is a classifier with as many output nodes as domains involved (in our case, we only have source and target, thus 2)
 - source domain is labelled with 0
 - target domain is labelled with 1
- Its standalone goal is to discriminate the features extracted between the two domains, thus the loss is normally backpropagated through that branch
- On the other hand, the **feature extractor** tries to *fool* him, by generating features as much **domain invariant** as possible
 - to achieve that goal, he tries to **MAXIMIZE** the **Loss of the domain classifier**
 - this is where the **GRADIENT REVERSAL LAYER** comes in, **negatively** backpropagating the **Loss of the domain classifier**

STEP 0 - THE DATASET

PACS

PACS is an image classification dataset

- 7 classes
- 4 domains
- **P**hoto, **A**rt painting, **C**artoon, **S**ketch





PACS

- Dataset provided here:
 - <https://github.com/antoalli/AML-Homework3-DA>
- Images are organized in folders: read each domain and build a PyTorch dataset for each using the `ImageFolder` class

STEP 1 - BUILD THE NET





WHAT YOU SHOULD DO

- The DANN we will build is based on the AlexNet skeleton
 - the **feature extractor** will be the AlexNet convolutional layers
 - The **label predictor** will be the AlexNet fully connected layers
 - The **domain classifier** is a separate branch with the same architecture of AlexNet fully connected layers, except for the number of output neurons




WHAT YOU SHOULD DO

Starting from the AlexNet class definition.

1. modify the *init* function to add a new classifier for the domain discrimination
 - a. all the branches must be initialized with the weights of AlexNet pre-trained on ImageNet
 - b. If you create a new branch in the init function, and try to preload weights into the original branches, it gives you an error 
 - i. Use the flag *strict=False* in the *load_state_dict* function to avoid this error
 - ii. After you preload ImageNet weights in the original branches, copy weights of the original classifier into the new classifier 



WHAT YOU SHOULD DO

2. implement a flag in the *forward* function that you pass along with data to indicate if a batch of data must go to the **domain classifier** or to the **label predictor**
 - a. if data goes to the domain classifier, the gradient has to be reverted with a Gradient Reversal layer (find an example of such layer [here](#))
 - i. as you may notice, the **reversal is multiplied by an α factor** in the forward: this is the weight of the reversed backpropagation and must be optimized as an hyperparameter

STEP 2 - DOMAIN ADAPTATION



WHAT YOU SHOULD DO

For this homework, the **source domain** is “**PHOTO**” and the **target domain** is “**ART-PAINTING**”. You are required, for each step, to run an hyperparameter optimization (beware that when using Domain Adaptation also the ***alpha*** parameter must be taken into account)

1. Train on “PHOTO” and test on “ART-PAINTING” **without** Domain Adaptation



WHAT YOU SHOULD DO

2. Train DANN on “PHOTO” and test on “ART-PAINTING” **with** Domain Adaptation


- a. the net must be trained jointly on the labeled task (Photo) and the unsupervised task (discriminate between Photo and Art-Painting)
- b. each training iteration is divided in 3 steps before calling `optimizer.step()`
 - i. forward **source** data along with the labels to **label predictor**, get the loss, and update gradients with `loss.backward()`
 - ii. forward **source** data to **domain classifier**, get the loss (the label is 0 for all data), and update gradients with `loss.backward()`
 - iii. forward **target** data to **domain classifier**, get the loss (the label is 1), and update gradients with `loss.backward()`

i.)
Obj.
Classifier
Training

ii + iii)
Domain
Classifier
Training



HINTS

- the second and the third are binary classification steps (the discriminator must tell if the images comes from source or target), so you can use the `nn.CrossEntropyLoss()` function as your criterion
- For the second step you can use the same data you used for the first one, but you will not use the category labels since you're classifying only the domain (source domain specifically)
- For the third step, you will need to iterate over the target data while you are already iterating on the source data: you can find a possible solution to this [here](#) 
- Use the entire “ART-PAINTING” dataset both for training and testing
 - the adaptation and test set coincide (basically cheating, see more on validation in next step)

STEP 3 - CROSS VALIDATION



WHAT YOU SHOULD DO

- So far, we have basically tuned the parameter directly on the test set (Art-Painting). How can we validate the transfer from Photo to Art-Painting **if you do not have Art-Painting labels at training time?**
 - We validate hyperparameters by measuring performances on *Photo to Cartoon* and *Photo to Sketch* transfer!
 - We take the average of the accuracy results on them to choose the best set of hyperparameters



WHAT YOU SHOULD DO

1. Run a grid search on Photo to Cartoon and Photo to Sketch, with Domain Adaptation, and **average** results for each set of hyperparameters
2. Use only such best set to train on Photo to Art-Painting, and compare the results obtained with respect to the previous step, where you were basically cheating.



LR	ALPHA	CARTOON_ACC	SKETCH_ACC	AVG_ACC
0.1	None	0.163	0.191	0.177
0.01	None	0.313	0.310	<u>0.312</u>
0.005	None	0.262	0.261	0.261
0.001	None	0.250	0.199	0.224
0.1	0.5	0.163	0.191	0.177
0.01	0.5	0.162	0.191	0.176
0.005	0.5	0.165	0.192	0.178
0.001	0.5	0.162	0.192	0.177
0.1	0.1	0.163	0.192	0.177
0.01	0.1	0.520	0.191	0.356
0.005	0.1	0.468	0.191	0.330
0.001	0.1	0.277	0.198	0.237
0.1	0.05	0.164	0.191	0.177
0.01	0.05	0.507	0.171	0.339
0.005	0.05	0.339	0.583	<u>0.461</u>
0.001	0.05	0.253	0.362	0.308

STEP 4 (EXTRA) - FEATURE REPRESENTATION



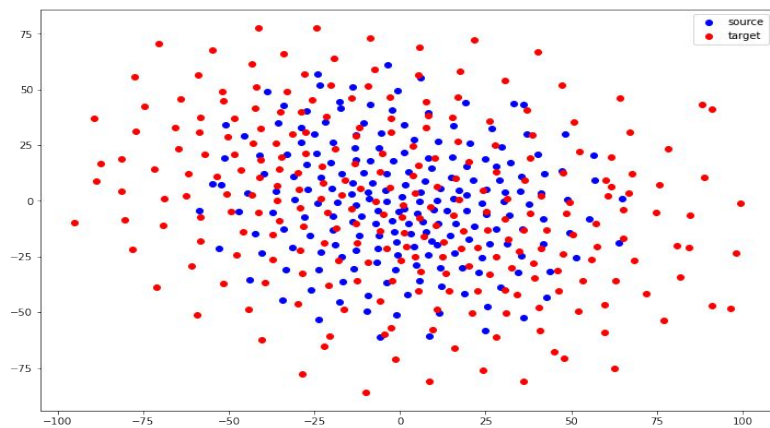
WHAT YOU SHOULD DO

Take a look at the feature representation of source and target images once the network has been trained, with and without Domain Adaptation.

1. Train the net with and without domain adaptation
2. Extract the features for some images belonging to different domains
3. Plot them using some dimensionality reduction technique, such as PCA or t-SNE



NO ADAPTATION



ADAPTATION

