Homework 3:
Deep Domain Adaptation

Mirco Planamente
mirco.planamente@polito.it

Originally made by
Antonio D’Innocente

Overview

- The task is to implement **DANN**, a **Domain Adaptation** algorithm, on the PACS dataset using AlexNet

- Let's have a brief recap at **Domain Adaptation** and **DANN**

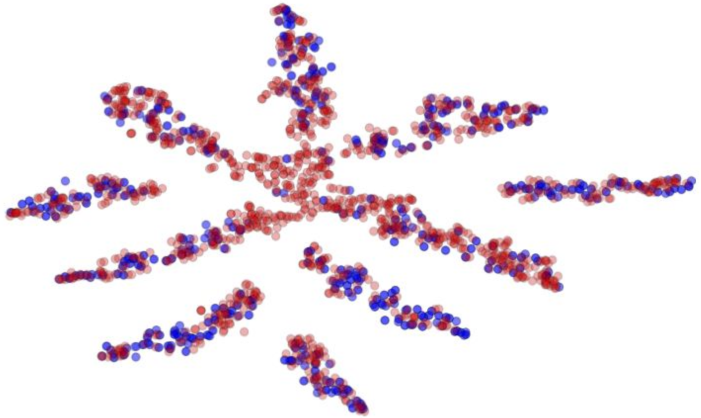
The ideal case

Train

Test

The ideal case

- Data has been collected and annotated in ideal conditions
  - Test data is from the same distribution of training data



- Accuracy: **HIGH**

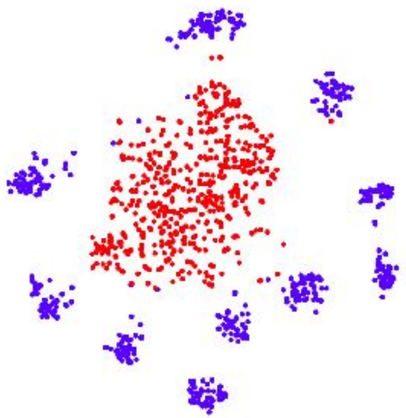
The not ideal case

Train

Test

The not ideal case

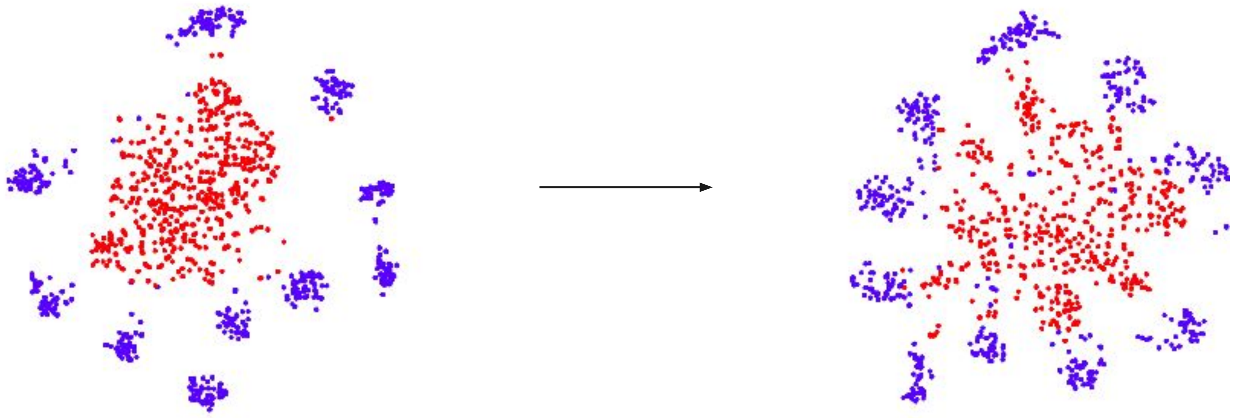
- Annotated data and unlabeled data don’t match
  - Test data is from a different distribution of training data



- Accuracy: **LOW**

Domain Adaptation

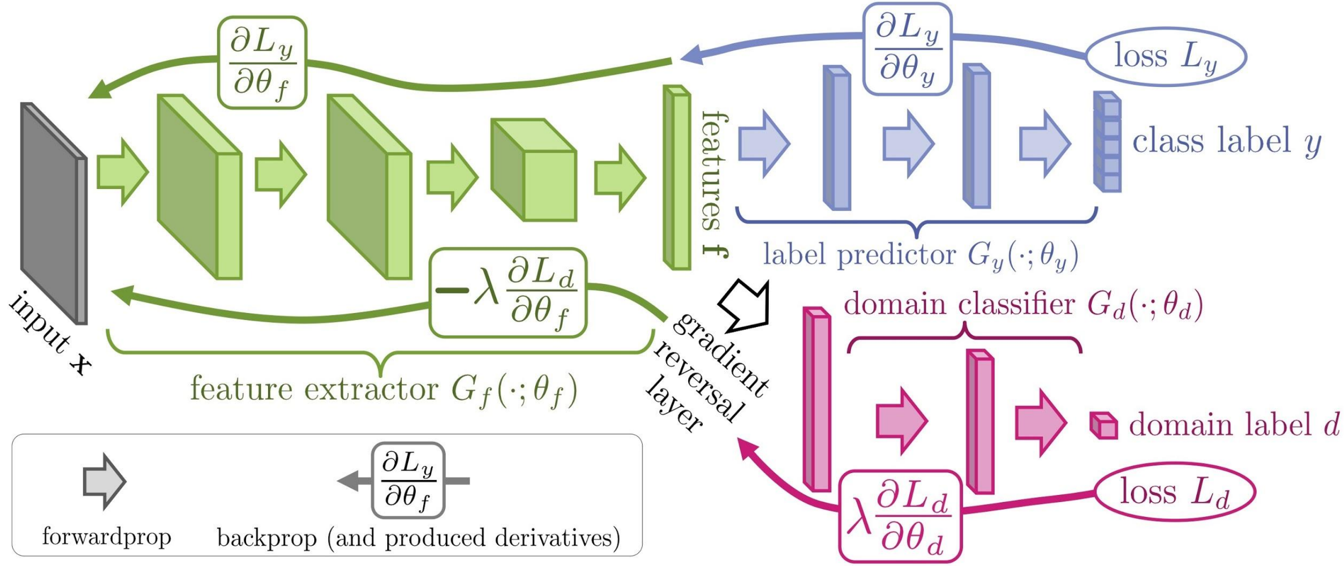
- Align training features with test features



DANN

- Align training features with test features
- Train a binary classifier to discriminate between source and target
- Train a feature extractor to confuse the features such that it is hard for the binary classifier to distinguish source from target

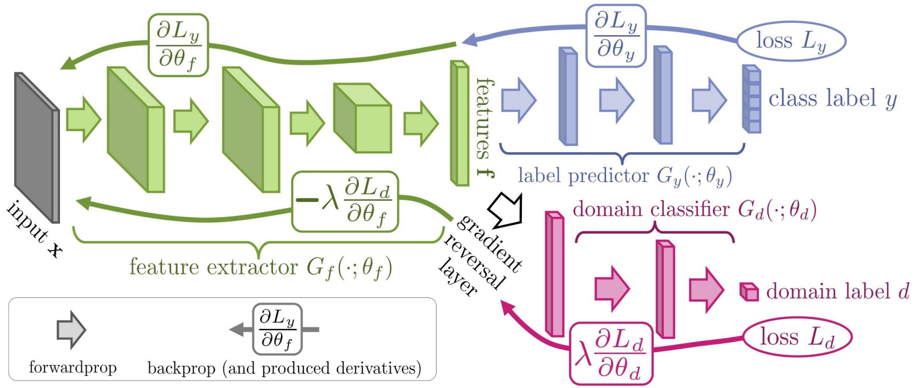
DANN



$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
features $\mathbf{f}$
class label y
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
input $\mathbf{x}$
feature extractor $G_f(\cdot;\theta_f)$
gradient reversal layer
domain classifier $G_d(\cdot;\theta_d)$
domain label d
$\lambda \frac{\partial L_d}{\partial \theta_d}$
loss L_d
forwardprop
backprop (and produced derivatives)

DANN: architecture

- Feature extractor **Gf**
  - Fully Convolutional
- Label predictor **Gy**
  - Trained on source labels
- Domain classifier **Gd**
  - Must discriminate if an image comes from the source domain or the target domain
- gradient reversal layer
  - Inverts the gradients of **Gd**

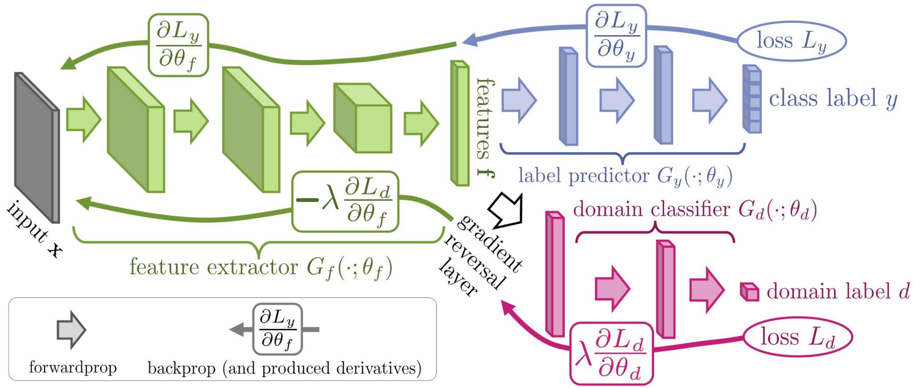


DANN: architecture

- Gd is trained to discriminate between source and target
- The **gradient reversal layer** inverts the gradient of Gd
  - Gf now wants to maximize error of Gd

↓

- The feature extractor Gf tries to fool the domain classifier Gd
  - By generating domain invariant features
  - Gy is trained on these features



Homework 3

0 - Before starting

1 - The Dataset

2 - Implementing the Model

3 - Domain Adaptation

4 - (Extra) Cross Domain Validation

0 - Before Starting

- As with Homework 2, the assignment is in Colab
- You are not provided a new starting template
- However, you can start from the Homework 2 template
- https://colab.research.google.com/drive/1PhNPpklp9FbxJEtsZ8Jp9qXQa4aZDK5Y

1 - The dataset

PACS is an image classification Dataset

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

The Dataset is provided here (you have to integrate it in the template):

https://github.com/MachineLearning2020/Homework3-PACS

1 - The dataset

Navigate https://github.com/MachineLearning2020/Homework3-PACS/tree/master/PACS and explore the dataset structure

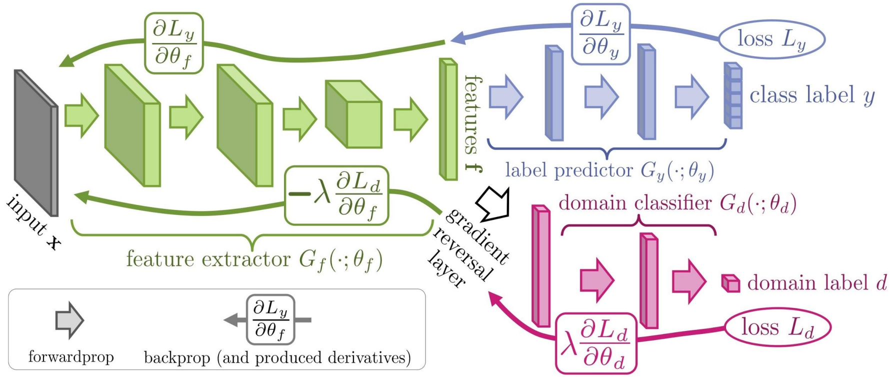
As you will see, images are organized in folders

Hint: *You can easily read each domain as a PyTorch dataset using the ImageFolder class*

2 - Implementing the Model

The original implementation of DANN is on small networks for digits datasets, we will try to implement it using AlexNet

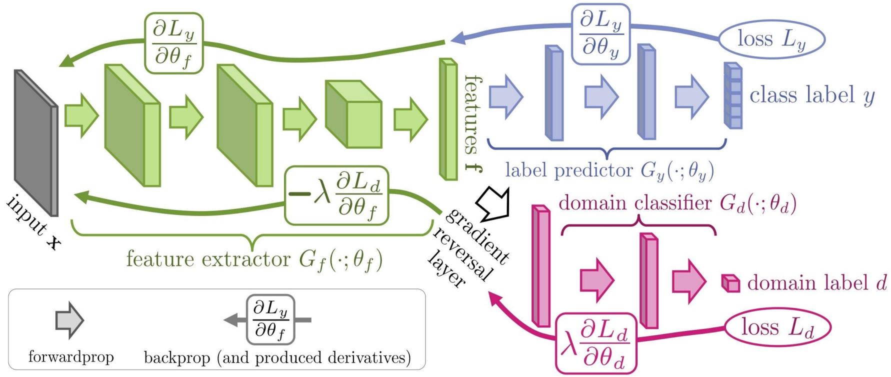
***Hint**: Original implementations of DANN already exists in public PyTorch repositories*



2 - Implementing the Model

You have to implement DANN in PyTorch in this way:

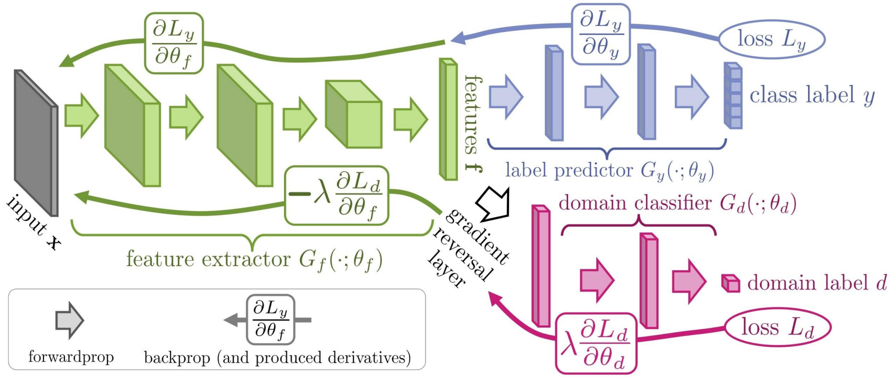
- **Gf** as the AlexNet convolutional layers
- **Gy** as the AlexNet fully connected layers
  - (up to this point it is a standard AlexNet)
- **Gd** as a separate branch with the same architecture of AlexNet fully connected layers (classifier)
  - Basically, you have to add a new densely connected branch with 2 output neurons



2 - Implementing the Model

To implement Gd, you have to modify the init function of AlexNet, and add the new branch

A. Modify the AlexNet's init function and add a new classifier, Gd identical to the AlexNet classifier
B. Both the original network and the new classifier must be initialized with the weights of ImageNet (Gy and Gd will have the same starting weights)

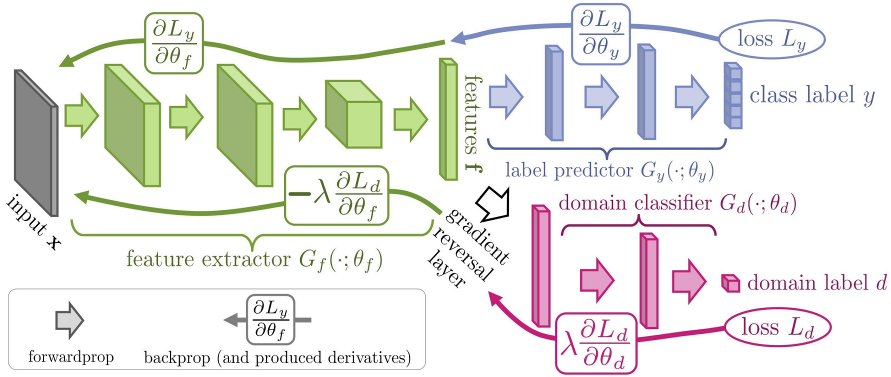


2 - Implementing the Model

Hint: *Start from the AlexNet source code in* *https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py* (you might need to adjust some imports)

If you create a new branch in the init function, and try to preload weights into the original branches, it gives you an error

- Use the flag *strict=False* in the load_state_dict function to avoid this error

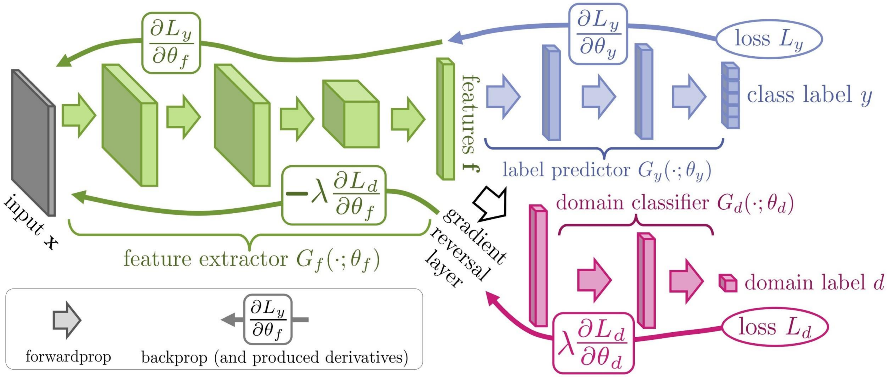


2 - Implementing the Model

***Hint**: Start from the AlexNet source code in https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py* (you might need to adjust some imports)

- After you preload ImageNet weights in the original branches, copy weights of the original classifier into the new classifier

***Hint**: You can access fc6 weights and biases with model.classifier[1].weight.data and model.classifier[1].bias.data*

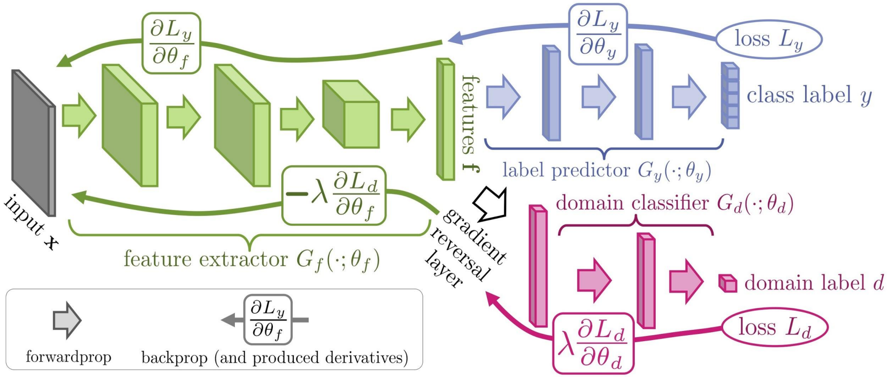


2 - Implementing the Model

To implement the DANN logic you will have to change the forward function

C. Implement a flag in the forward function (a boolean or something else) that you pass along with data to indicate if a batch of data must go to **Gd** or **Gy**

   a. If data goes to Gd, the gradient has to be reverted with the Gradient Reversal layer



***Hint**: you find an example of gradient reversal in:*

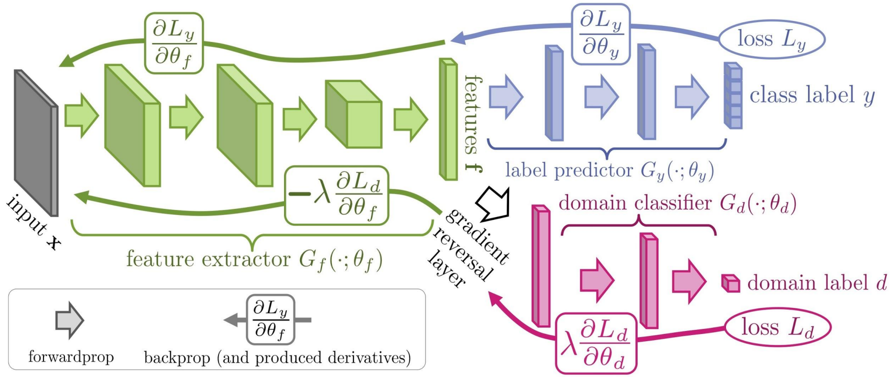
[*https://github.com/MachineLearning2020/Homework3-PACS/blob/master/gradient_reversal_example.py*](https://github.com/MachineLearning2020/Homework3-PACS/blob/master/gradient_reversal_example.py)

2 - Implementing the Model

***Hint**: you find an example of gradient reversal in:*
https://github.com/MachineLearning2020/Homework3-PACS/blob/master/gradient_reversal_example.py

As you may notice from the example, the reversal is multiplied by an alpha factor in the forward

<u>This is the weight of the reversed backpropagation, and must be optimized as an hyperparameter of the algorithm</u>



3 - Domain Adaptation

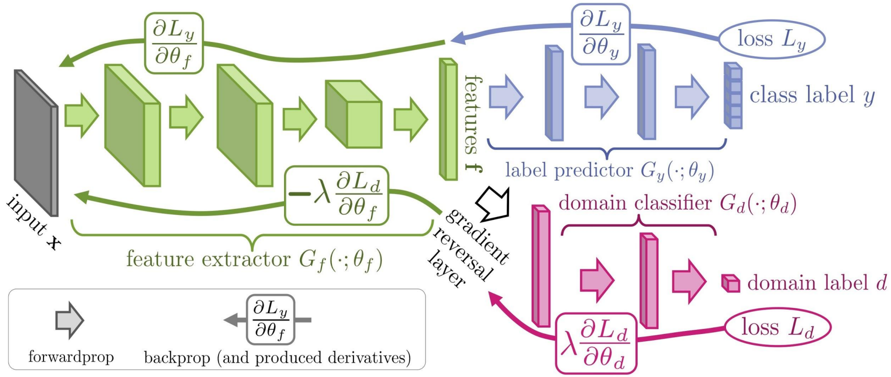
For this Homework, the source domain is Photo, while the target domain is Art painting

A. Train on Photo, and Test on Art painting without adaptation
B. Train DANN on Photo and test on Art painting with DANN adaptation
C. Compare results

3B - Implementing training with DANN

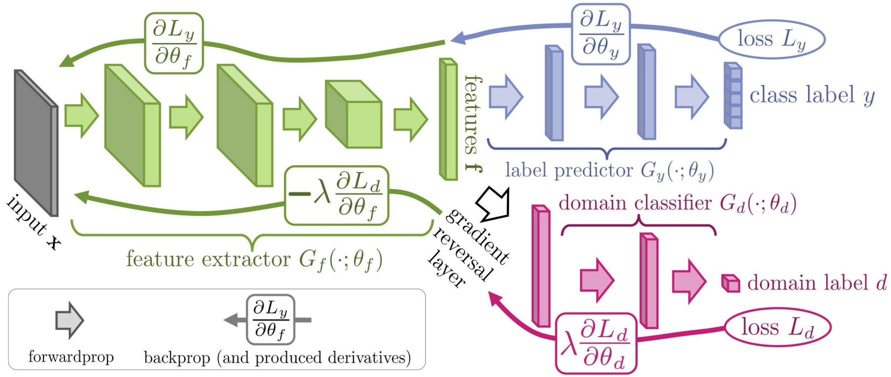
The network must be trained jointly on the labeled task (Photo) and the unsupervised task (discriminating between Photo and Art painting), and then tested on Art painting

- Divide a single training iteration in three steps that you execute sequentially before calling optimizer.step()



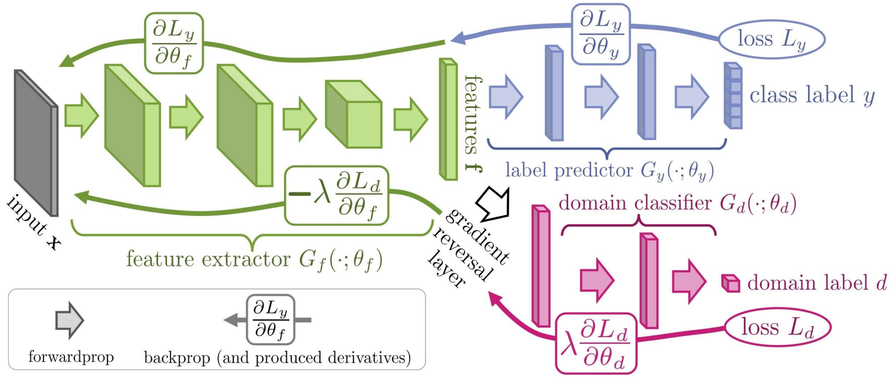
3B - Implementing training with DANN

1. train on source labels by forwarding source data to **Gy**, get the loss, and update gradients with loss.backward()
2. train the discriminator by forwarding source data to **Gd**, get the loss (the label is 0 for all data), and update gradients with loss.backward()
3. train the discriminator by forwarding target data to **Gd**, get the loss (the label is 1), and update gradients with loss.backward()



3B - Implementing training with DANN

***Hint**: 2. and 3. 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*

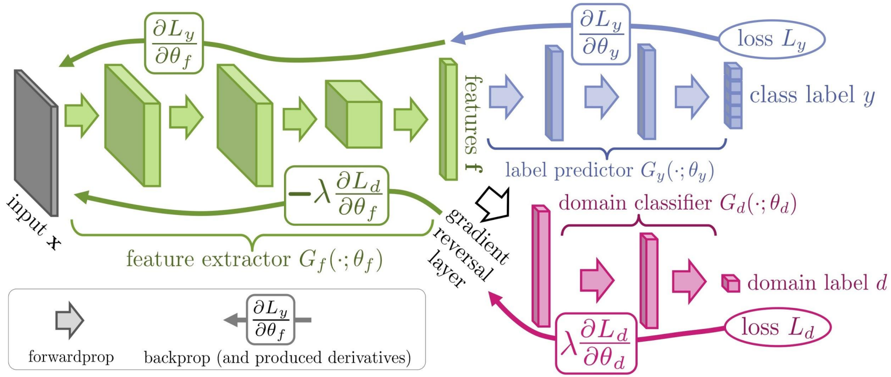


3B - Implementing training with DANN

After doing the 3 steps you call optimizer.step() to apply gradients

Hint: For the second step you can use the same data you used for **Gy**

Hint: For the third step, you have to use a separate dataloader that iterates over the target dataset https://github.com/pytorch/pytorch/issues/1917



3 - Additional informations

- For this task, use the entire Photo dataset for training, and the entire Art painting dataset for both training (without labels) and testing
    - This means that the adaptation set and the test set coincide (transductive)
    - we don't do validation (more on this later)

3 - Additional informations

If you are not going to do the extra point 4, run hyperparameter optimization for both 3A and 3B to optimize performances with some hyperparameter search algorithm

- Besides usual hyperparameters, you have to optimize alpha
- (This is essentially cheating)
- You can use the same batch size for the Photo dataloader and the Art painting dataloader

4 - (Extra) - Cross Domain Validation

As you noticed, we didn't do the validation step

- Validation is an open problem in Domain Adaptation
- However, without validation, we are looking at results in the test set (cheating)
- If you want to complete the extra assignment: do 3A and 3B by validating hyperparameters
- How can you validate Photo to Art painting transfer if we don't have Art painting labels?

4 - (Extra) - Cross Domain Validation

- We validate hyperparameters by measuring performances on Photo to Cartoon transfer and Photo to Sketch transfer

A. Run a grid search (or other hyperparameter search method) on Photo to Cartoon and Photo to Sketch, without Domain Adaptation, and average results for each set of hyperparameters

B. Implement 3A with the best hyperparameters found in 4A

4 - (Extra) - Cross Domain Validation

- We validate hyperparameters by measuring performances on Photo to Cartoon transfer and Photo to Sketch transfer

C. Run a grid search (or other hyperparameter search method) on Photo to Cartoon and Photo to Sketch, with Domain Adaptation, and average results for each set of hyperparameters

D. Implement 3B with the best hyperparameters found in 4C

4 - (Extra) - Cross Domain Validation

If you do the validation step, you don't have to optimize hyperparameters of 3A and 3B on the test set

Submission rules

- Deadline: **Two week before the first round (NOT the first you enrol in).**
- Uploading: through “Portale della didattica”

Submit a zip named <YOUR_ID>_homework3.zip. The zip should contain two items:

- A pdf report describing data, your implementation choices, results and discussions
- Code

FAQ

1. *Should I implement hyperparameter search in the code or it is fine to test manually?*
   - Due to Colab limitations, it is ok to test hyperparameters by manually changing them in the code
2. *The Discriminator loss doesn't increase*
   - The feature extractor tries to maximize the loss of the discriminator
   - However, the Discriminator is still trying to minimize its own loss
3. *Results with DANN have small or no improvements*
   - Even 1-2% increases on PACS are significative
   - The implementation might not yield adaptation results in this setting
   - Optional things that you can try:
     i. Advanced exponential scheduling for alpha (check DANN paper)
     ii. Different implementations for the discriminator (experiment, check DANN paper)
     iii. Implement the exact AlexNet architecture in pytorch, load pre-trained weights from the original caffe model, adjust inputs to fit pytorch preprocessing ([0-255] -> [0-1], BGR -> RGB)