Deep Learning Assignment 2022
Unsupervised Domain Adaptation

May 16, 2022

1 Introduction

In this assignment you are expected to build, train and evaluate a deep learning framework on a standard setting of **Unsupervised Domain Adaptation (UDA)**. As you have studied during the theoretical course, UDA is a generic expression that covers any possible technique designed to address the issue of the **domain shift** that may exist between two or more data distributions. In particular, for this assignment you will be provided with a UDA benchmark consisting of two datasets. You will be required to treat one of the two datasets as *source domain* and the other as *target domain*, and to propose a UDA technique to counteract the negative impact of the domain gap when training your model on the source distribution and evaluating it on the target distribution. As it holds for any standard UDA framework, the quality of the domain alignment strategy shall essentially be assessed by looking at the gain obtained with your proposed framework over the so called source-only baseline. Details will follow in Section 3.

2 Dataset

As a UDA benchmark, in this assignment you will be using the Adaptiope [3] object recognition dataset. This dataset comprises images from 3 distinct domains, referring to *synthetic*, *product* and *real world* data. A visual example is provided in Figure 1.

Figure 1: Adaptiope Dataset

The original dataset comprises 123 object categories in each domain. For the sake of this assignment, you are going to extract 20 randomly chosen categories from the *product* and *real world* domains. The chosen categories are the following: *backpack*, *bookcase*, *car jack*, *comb*, *crown*, *file cabinet*, *flat iron*, *game controller*, *glasses*, *helicopter*, *ice skates*, *letter tray*, *monitor*, *mug*, *network switch*, *over-ear headphones*, *pen*, *purse*,

stand mixer and *stroller*. This dataset (and thus its subset defined for this assignment) already follows the file system structure expected from the `torchvision.datasets.ImageFolder` utility, thus it should be easy for you to load the data. We suggest you to consider resizing and cropping the images at dataloading time. The dataset is publicly available for download[1]; since the file is hosted on Google Drive, we suggest you to simply add the shortcut in your own Drive folder, so that you can easily access it from Colab without having to download anything to your local storage. If you have troubles quickly recalling how to access files in Google Drive from Colab, and/or extracting the desired part of the dataset, you will find in the moodle a short notebook which contains the code to do exactly this, assuming the shortcut to the zip file to be in the right path. For the assignment, you are required to train and evaluate your UDA framework in **both directions** between the two selected domains, and provide an interpretation in the discussion section about the comparison between the two settings. For this reason, please use the PyTorch utility that was already shown to you during the lab[2] in order to split both domains into training and test sets: to avoid additional degrees of confusion, please use a standard 80%/20% split.

3 Task

In this section we will outline the details of the UDA task for this assignment. The base task consists of a pretty straightforward **object recognition** challenge, i.e., given as input a image depicting an object, you want to produce the label associated with this object. Your specific task differs from its standard version for the fact that you are going to address it in a setting where domain shift is present. Formally, you are going to consider a source dataset $\mathcal{S} = \{X_i^S, y_i^S\}_{i=1}^{N_S}$ of images and associated labels, and an **unlabelled** target dataset $\mathcal{T} = \{X_i^T\}_{i=1}^{N_T}$, where $X_i \in \mathcal{X}$ and $y \in \mathcal{Y}$, $\mathcal{Y} = \{1, 2, \ldots, K\}$ (K denotes the number of object categories). You are going to aim to learn a function $F_\theta : \mathcal{X} \to \mathcal{Y}$ with parameters θ that maps an input image X to a class label y and perform well on target data. In other words, you are required to

- train your model **supervisedly** on the source domain, i.e. having access to the images as well as the object labels
- train your model **unsupervisedly** on the training set of the target domain, i.e. having access to the images, but not to the labels
- evaluate your model on the test set of your target domain

As briefly mentioned in Section 1, the adopted domain alignment strategy is generally evaluated with respect to the *gain* that it enables over the source-only performance. As the name suggests, the source-only baseline that should be taken into account when evaluating any UDA framework simply consists of the score obtained when training your model on the source domain and evaluating it on the target domain, **without** any domain alignment strategy involved. The more effective your chosen domain alignment strategy is, the higher your gain will be over the source-only version. Practically, your task should go through the following steps:

1. implement a deep learning framework for object recognition
2. train your framework supervisedly on the source domain and evaluate it, as it is, on the target domain (source only version)
3. implement and plug into your framework a domain alignment strategy of your own choice, which should allow to benefit from learning unsupervisedly from the target training set
4. train the final model and evaluate it on the target domain

As evaluation metric, we can simply rely on the **validation accuracy**, as defined in Eq. 1

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

[1] **Adaptiope download**: https://drive.google.com/file/d/1FmdsvetCOoVyrFJ9ER7fcN-cXPOWx2gq/view
[2] **PyTorch tool**: https://pytorch.org/docs/stable/data.html#torch.utils.data.random_split

where TP stands for the number of true positives, TN true negatives, FP false positives and FN false negatives. Your gain, therefore, will be defined as the difference between the accuracy of your full model and the accuracy of your source-only model.

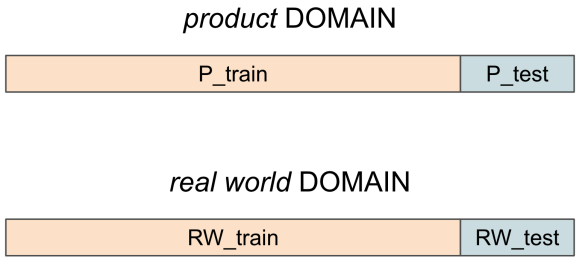

Figure 2: Split visual representation

For a better understanding, let's take Figure 2 as reference. Suppose you're working on the direction *product* $\rightarrow$ *real world*. Then the first thing you will do is train your model on P_{train}. Since this is your source domain, **you are allowed to use label information** (e.g. use a cross entropy loss in your training step). In your test step, you are going to evaluate the model on RW_{test}. This will achieve a certain accuracy; since we only trained on the source domain, and not on the target domain, this accuracy refers to the *source only* scenario. We call it acc_{so}. Now you want to evaluate your UDA component which, differently from the former case, implies training on the target domain. Since **you are not allowed to use labels** there, here you will use any UDA device of your choice. So, in this case, in your training step, you will train supervisedly on P_{train} (like you did before) and simultaneously train unsupervisedly on RW_{train}. In your test step, once again, you want to evaluate on RW_{test}. This will achieve a new accuracy acc_{uda}, which hopefully will be higher than acc_{so} since this time you also trained on the target domain, even if without label information. At this point you can compute your *gain* G:

$$G = acc_{uda} - acc_{so} \tag{2}$$

It goes without saying that the same exact pipeline holds for the opposite direction *real world* $\rightarrow$ *product*, by simply switching P and RW. Note that the two directions are tested in **two separate and independent experiments**, which in general will differ only for the datasets treated respectively as source and target (i.e. switching the roots of the dataloaders). Should you decide to make changes when running different directions, please provide a motivation for that. A useful thing that you may want to do is computing some sort of *upper bound* for your task. The upper bound consists in the value of your evaluation metric (in this case the validation accuracy) that is obtained by enabling supervision on the target domain. In the example that we just used for *product* $\rightarrow$ *real world*, in order to compute the upper bound you would (i) train supervisedly on RW_{train} (like you would on the source domain) and (ii) evaluate on RW_{test}. The obtained accuracy acc_{ub} can be useful for you, because you are going to assume that the performance of your UDA framework will lie somewhere in between acc_{so} and acc_{ub}, and thus have an idea of what to reasonably expect.

Reference for target accuracy We have performed a single experiment using a very simple baseline based on adversarial learning, on top of a pretrained resnet18 backbone. We report the accuracy obtained on the target test set in Table 1. This is meant to provide some reference values that may help you assess your own performance. Should your target accuracy be lower than the one reported here, chances are you need to improve your method if you want to obtain a good score with respect to one of the evaluation metrics listed ahead (*performance with respect to validation accuracy*). **Note**: the values reported here are not necessarily the best that you would obtain with an adversarial learning based method. They have been obtained without tuning any parameter and they should be treated as nothing more than a generic reference.

Note Domain adaptation is an inherently ill-posed problem, since it may not be clear how to properly validate and assess the quality of the approach during the development without a validation set extracted from the target domain. For the scope of this assignment, it is not requested that you use such validation set: you are allowed to train your model, test the performance on the target test set and then apply modifications

Version	P→RW	RW→P
Source only	76%	90%
DA	80%	93%

Table 1: Accuracy on the target test set

accordingly. The only important thing to remember to **never involve target ground truth labels** in any of your training losses.

Architecture We have been going through some possible architectural choices during the lab sessions held so far, which you might decide to start from when implementing your model. Nevertheless, you are free to design and build any architecture, provided you are able to run it with the resources provided by Google Colab.

Domain alignment module You have quickly been going through some standard and well-known domain alignment techniques during the theoretical course. Also in this case, you are free to either pick one of those or any other technique of your own liking. With these respect, we suggest you at least take a look at some of the existing surveys that have been published about the most common domain alignment techniques [4, 1, 2].

4 Evaluation

Your delivery will be **self-contained within a single Jupyter Notebook** hosted on Google Colab. The notebook will contain your code, properly divided into multiple executable cells in a clean, modularized and readable fashion. In addition, you are required to exploit the text cells provided in Google Colab in order to integrate comments into your notebook, which should come together as an actual project report. In the report you are required to provide:

- a description of the **solution** that you adopted. This must include a detailed overview of the developed architecture, as well as the losses that govern the training. Please make sure to highlight all your original contributions as well as to correctly cite the literature. Pay attention to properly motivating your design choices and all the components you decide to include, especially when it is not obvious to simply infer it. Keep in mind that the text cells also allow you to include pictures: diagrams and charts are always very helpful!
- a description of the **details** of the training and evaluation strategy employed. Provide a detailed overview of how your model was built and trained, extensively motivating your methodological choices with respect to hyperparameters choice and other variables
- an extensive presentation and **discussion** of the results obtained with your framework. Organize the scores you obtained in tables and consider including charts depicting learning curves, confusion matrices and any other useful visual representation. Make an effort in order to structure this section in such as a way as to emphasize the take-home message deriving from your findings, e.g., put together the source-only scores and the full model ones so that the comparison is straight-forward and the gain is properly highlighted
- **comments** to the code. It is important to employ the utility of the Jupyter notebook in order to provide a step-by-step description of your code. Try to be clear but concise, proving that you are aware of all the steps that you go through during the development and the training/evaluation pipeline
- **instructions** for correctly running your code, if needed

Your project will be evaluated according to the following metrics:

- originality of the solution
- methodological thoroughness
- clarity of the report

- performance with respect to validation accuracy
- quality of the code

In order to deliver the solution, send an **e-mail** to giacomo.zara@unitn.it, putting e.ricci@unitn.it as cc. The email should have object `[DL2022] - Assignment delivery`. Attach to the email your self-contained notebook in `.ipynb` format, with name `[student_id_1]_[student_id_2].ipynb`. For example, the delivery of students with IDs (matricola) 123456 and 987654 will be `123456_987654.ipynb`. **IMPORTANT**: make sure that the file can be seamlessly loaded and executed into Google Colab. The solution must be delivered strictly **not later than 10 days before the date in which you decide to take the oral exam**.

5 Group registration

The project is intended for groups of 2 people. You can register your group by filling the Google Form[3], strictly within the end of April. If you have troubles in finding a group, you may use the provided telegram channel[4].

6 Policies

For the whole architecture of your model, we require that you **don't start from an existing GitHub repository** containing code developed by other people: your architecture doesn't have to be overly complex, but you are expected to either build it from scratch or to start from the code used in the lab sessions. If needed, only small specific snippets of code may be borrowed from existing GitHub repositories. This policy derives from the purpose of having you practically familiarize with the tools involved in building a deep learning framework rather than just assemble existing components. We should specify that this does not refer to the choice of a **feature extractor** for your model: you are allowed to choose any existing architecture (e.g. ResNet) - possibly with pretrained weights, if available - as a model *backbone*. You may generically discuss the project with other groups, but it is strictly forbidden to share the source code. Keep also in mind that your delivery may be checked against an automatic utility for plagiarism detection. Any violation of these policies will result in actions being taken towards all involved groups, including those potentially lending their code: it is the responsibility of each group to comply to the policy.

References

[1] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey, 2017.

[2] Vishal M Patel, Raghuraman Gopalan, Ruonan Li, and Rama Chellappa. Visual domain adaptation: A survey of recent advances. *IEEE Signal Processing Magazine*, 32(3):53–69, 2015.

[3] Tobias Ringwald and Rainer Stiefelhagen. Adaptiope: A modern benchmark for unsupervised domain adaptation. In *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 101–110, 2021.

[4] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, oct 2018.

[3] **Google form**: https://forms.gle/rDeoemyxNjwLr9u77
[4] **Telegram channel**: https://t.me/+5jvTgRd2b-Q0OGE0