POLITECNICO DI TORINO

Corso di Laurea Magistrale
in Data Science and Engineering

Machine Learning and Deep Learning Report

Exam session: Winter 2021

Domain Adaptation Neural Network on Python using Google Colab

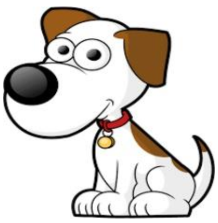
Emanuele Fasce

1. INTRODUCTION

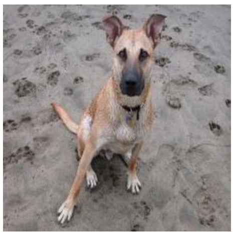
Domain adaptation is an important technique in deep learning. It is the ability to apply an algorithm trained in one or more "source domains" to a different (but related) "target domain".
The network used is the Alexnet network enriched with a fully connected branch for domain classification.
In the first part of the report, the source dataset is composed of photos and the target dataset is composed of art paintings.
A manual hyperparameter tuning follows, both without and with the domain adaptation technique.
In the second part of the report the networks are validated on sketches and photos, and the best hyperparameters found are used for testing on the art paintings domain.
The code is available at: https://github.com/emanueleing/MLDL_Homework3

2. THE DATASET

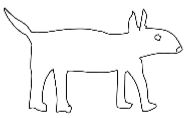
The PACS dataset is composed of 9991 pictures of 7 different entities (dogs, elephants, guitars, giraffes, horses, houses and people) in 4 different domains: art paintings, cartoons, real photos and sketch.
To give an intuition of the complexity of this task, 4 pictures representing a dog in different domains are shown.



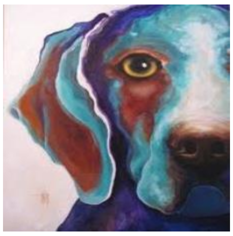
A cartoon



A photo



A sketch



A artwork

3. DANN IMPLEMENTATION

The standard Alexnet network is augmented with a new densely connected branch with 2 output neurons, that is used as a domain classifier (Figure 2). The *num_classes* parameter of the standard classifier in this case is set to 7 (Figure 1). The weights of this domain classifier, when pretrained on the ImageNet dataset, are copied from the weights of the standard classifier, as seen in the code (Figure 3).

Feeding the domain classifier with training pictures with 0 as label and test pictures with 1 as label (different labels since they belong to different domains) could be useful in order to recognize the features that make the two sets different.
However, here the goal is the inverse, as we want to learn the features in common with the two sets, so instead of the standard gradient a gradient reversal layer is used, that simply takes the inverse gradient.
This procedure is performed along with the standard classification training.
The reversal is multiplied by the alpha parameter, and this will be an hyperparameter to optimize.
(Figure 5).
In the forward function of the network, a simple flag (*training*) discriminates if the inputs should be sent to the standard classifier or the domain classifier (Figure 4).

```
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),
)
```

1.Classifier for supervised task

```
self.classifier_d = 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))
```

2.Classifier for domain adaptation

```
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.classifier_d[1].weight.data = model.classifier[1].weight.data
        model.classifier_d[1].bias.data = model.classifier[1].bias.data
        model.classifier_d[4].weight.data = model.classifier[4].weight.data
        model.classifier_d[4].bias.data = model.classifier[4].bias.data
        model.classifier_d[6].weight.data = model.classifier[6].weight.data
        model.classifier_d[6].bias.data = model.classifier[6].bias.data
    return model
```

3.Copying the weights

```
def forward(self, x, training, alpha=None):
    x = self.features(x)
    x = self.avgpool(x)
    x = torch.flatten(x, 1)
    if training=='s':
        x = self.classifier(x)
    else:
        if training=='d':
            x = ReverseLayerF.apply(x, alpha)
            x = self.classifier_d(x)
    return x
```

4. Forward function

```
class ReverseLayerF(Function):
    @staticmethod
    def forward(ctx, x, alpha):
        ctx.alpha = alpha
        return x.view_as(x)
    @staticmethod
    def backward(ctx, grad_output):
        output = grad_output.neg() * ctx.alpha
        return output, None
```

5.Reversed layer function

3A-3B. TRAINING ON PHOTOS, TESTING ON ART WORKS

The standard Alexnet and the domain adaptation network are trained on the photos of the PACS dataset and tested on art works.
Due to hardware limitations the number of trials is limited, however some patterns can be observed by the tables below.

In table 1 (results without domain adaptation) different values for the learning rate and the number of epochs are used, while in Table 2 (results with domain adaptation) only a learning rate is selected, and different values for alpha are used. Too high values for the learning rate or alpha lead to Nan losses.
The best accuracy is reached without domain adaptation (Table 1 model 6, 54.15%), but the difference with the best accuracy found with domain adaptation (Table 2 model 6, 52.83%) is not that high.
It can be noticed however that a lot of the models with domain adaptation (5,6,8) lead to very low losses on the target domain.

Number	LR	Epochs	Momentum	Weight decay	Stepsize	Gamma	Acc Test	Acc Train	Loss Test	Loss Train
1	1,00E-04	20	0.9	5,00E-05	20	0.1	47.41	99.52	2.28	0.02
2	1,00E-04	35	0.9	5,00E-05	20	0.1	48.92	99.82	2.47	0.01
3	1,00E-04	50	0.9	5,00E-05	20	0.1	49.51	99.94	2.31	0.005
4	1,00E-03	20	0.9	5,00E-05	20	0.1	47.11	100	3.19	0.0004
5	1,00E-03	35	0.9	5,00E-05	20	0.1	50.34	100	3.11	0.0006
6	1,00E-03	50	0.9	5,00E-05	20	0.1	54.15	100	2.86	0.0006
7	1,00E-02	20	0.9	5,00E-05	20	0.1	13.91	10.89	Nan	Nan
8	1,00E-02	35	0.9	5,00E-05	20	0.1	33.44	98.68	4.20	0.06
9	1,00E-02	50	0.9	5,00E-05	20	0.1	13.91	10.89	Nan	Nan

Table 1: Results without domain adaptation

Number	Alpha	LR	Epochs	Momentum	Weight decay	Stepsize	Gamma	Acc Target	Acc Source	Loss Target	Loss Source	Loss discriminator
1	0.3	1,00E-04	20	0.9	5,00E-05	20	0.1	0.29	0.78	5.75	1.19	0.13
2	0.2	1,00E-04	20	0.9	5,00E-05	20	0.1	45.65	96.40	2.34	0.10	0.02
3	0.1	1,00E-04	20	0.9	5,00E-05	20	0.1	46.19	99.04	2.38	0.03	0.008
4	0.3	1,00E-04	35	0.9	5,00E-05	20	0.1	10.89	13.91	Nan	Nan	Nan
5	0.2	1,00E-04	35	0.9	5,00E-05	20	0.1	44.23	93.77	1.90	0.16	0.02
6	0.1	1,00E-04	35	0.9	5,00E-05	20	0.1	52.83	98.86	2.11	0.03	0.009
7	0.3	1,00E-04	50	0.9	5,00E-05	20	0.1	10.89	13.91	Nan	Nan	Nan
8	0.2	1,00E-04	50	0.9	5,00E-05	20	0.1	50.53	96.52	2.03	0.11	0.02
9	0.1	1,00E-04	50	0.9	5,00E-05	20	0.1	47.99	99.52	3.88	0.01	0.01

Table 2: Results with domain adaptation

However, the domain adaptation does not bring improvements for what concerns the accuracy. A hypothesis could be that choosing a constant value for alpha is not an optimal choice. This problem will be addressed in the next section.

4A-4B. VALIDATION ON SKETCH AND CARTOON

In this section a validation procedure is performed by choosing a set of hyperparameters and training the networks both with and without the domain adaptation on sketches and cartoons. The hyperparameters with which the network reaches the highest accuracies on both the domains are then used to test the model on art works.

As discussed in the previous section, another strategy introduced by the DANN paper for finding a useful value for alpha is now implemented. The authors suggest an updating for alpha such that for every epoch is updated according to the following formula (where alpha is named λp), so that is starts at 0 and is gradually changed towards 1.

$$\lambda_p = \frac{2}{1+\exp(-\gamma \cdot p)} - 1 ,$$

The parameter p in the formula is the learning progress (in my implementation p is the ratio of the current number of epoch and the total number of epochs) and the optimal value for γ (called Gammap in the table) is searched through the values [0.1,0.25,0.4]. Higher values of γ lead to quicker increases of the parameter λp.

Therefore, in the grid-search without domain adaptation (Table 3) optimal values for the learning rate and the number of epochs are looked for, while in the grid-search with domain adaptation (Table 4) also the best value for γ is looked for.

Number	Batch Size	Optimizer	LR	Epochs	Momentum	Weight decay	Stepsize	Gamma	Target Dataset	Target Acc	Source Acc	Target Loss	Source loss
1	32	SGD	1,00E-04	20	0.9	5,00E-05	20	0.1	Cartoon	44.82	99.28	2.49	0.02
2	32	SGD	1,00E-04	35	0.9	5,00E-05	20	0.1	Cartoon	52.78	99.70	2.62	0.01
3	32	SGD	1,00E-05	20	0.9	5,00E-05	20	0.1	Cartoon	47.26	94.79	2.71	0.18
4	32	SGD	1,00E-05	35	0.9	5,00E-05	20	0.1	Cartoon	45.36	97.18	1.86	0.11
5	32	SGD	1,00E-04	20	0.9	5,00E-05	20	0.1	Sketch	24.26	99.52	4.51	0.02
6	32	SGD	1,00E-04	35	0.9	5,00E-05	20	0.1	Sketch	26.61	99.76	4.64	0.01
7	32	SGD	1,00E-05	20	0.9	5,00E-05	20	0.1	Sketch	21.14	94.49	2.81	0.18
8	32	SGD	1,00E-05	35	0.9	5,00E-05	20	0.1	Sketch	22.41	97.00	3.02	0.11
9	32	SGD	1,00E-04	35	0.9	5,00E-05	20	0.1	Art	49.02	99.88	2.43	0.008

Table 3: Results without domain adaptation

The combination of hyperparameters without domain adaption whose average accuracy is best is the one used in Number 2 and in Number 6. The accuracy reached on the art dataset with this combination is 49.02%.

Number	Batch Size	Optimizer	LR	Epochs	Momentum	Weight decay	Stepsize	Gamma	Gammap	Target dataset	Target Acc	Source Acc	Target Loss	Source Loss	Loss discriminator
1	32	SGD	1,00E-05	20	0.9	5,00E-05	20	0.1	0.1	Cartoon	32.95	84.37	2.43	0.51	0.00004
2	32	SGD	1,00E-05	20	0.9	5,00E-05	20	0.1	0.25	Cartoon	60.79	85.50	2.13	0.48	0.0002
3	32	SGD	1,00E-05	20	0.9	5,00E-05	20	0.1	0.4	Cartoon	64.06	91.73	1.80	0.25	0.001
4	32	SGD	1,00E-05	35	0.9	5,00E-05	20	0.1	0.1	Cartoon	39.94	90.53	2.02	0.26	0.0003
5	32	SGD	1,00E-05	35	0.9	5,00E-05	20	0.1	0.25	Cartoon	36.13	94.67	2.63	0.15	0.0005
6	32	SGD	1,00E-05	35	0.9	5,00E-05	20	0.1	0.4	Cartoon	64.20	0.94	1.59	0.16	0.006
7	32	SGD	1,00E-05	20	0.9	5,00E-05	20	0.1	0.1	Sketch	30.51	94.79	3.05	0.15	0.009
8	32	SGD	1,00E-05	20	0.9	5,00E-05	20	0.1	0.25	Sketch	34.52	94.37	2.68	0.17	0.01
9	32	SGD	1,00E-05	20	0.9	5,00E-05	20	0.1	0.4	Sketch	35.54	94.67	2.62	0.16	0.01
10	32	SGD	1,00E-05	35	0.9	5,00E-05	20	0.1	0.1	Sketch	36.66	96.86	2.89	0.10	0.006
11	32	SGD	1,00E-05	35	0.9	5,00E-05	20	0.1	0.25	Sketch	30.41	94.25	2.98	0.16	0.005
12	32	SGD	1,00E-05	35	0.9	5,00E-05	20	0.1	0.4	Sketch	30.37	95.74	3.95	0.11	0.007
13	32	SGD	1,00E-05	20	0.9	5,00E-05	20	0.1	0.4	Art	43.75	94.97	1.72	0.17	0.018

Table 4: Results with domain adaptation

The combination of hyperparameters with domain adaption whose average accuracy is best is the one used in Number 3 and in Number 9. The accuracy reached on the art dataset with this combination is 43.75%.

Even if the best resulting model is found without domain adaptation, it can be noticed that domain adaptation with alpha scheduling (Table 4) leads on average to higher accuracies and lower losses on the validation datasets with respect to the standard Alexnet (Table 3).