Machine Learning and Deep Learning Project Report: RGB-D Domain Adaptation

Gabriele Goletto
s276047@studenti.polito.it

Emanuele Gusso
s277882@studenti.polito.it

Vittorio Zampinetti
s276966@studenti.polito.it

Abstract

Domain adaptation techniques allow convolutional neural networks to learn the distribution of unlabeled target data through the exploitation of labeled data belonging to shifted domains. This field of research is still wide open and various methods are being tested in order to find optimal solutions to both sector specific and more general issues in DA. Indeed, with this work, we aim at presenting and analyzing a state-of-the-art domain adaptation method tailored for multi-modal RGB-D data [4] while trying to enhance it adopting one simple yet effective DA strategy. A systematic study on this topic will eventually shed light on some curious aspects of the domain adaptation framework. Code is available at `https://github.com/toyo97/rgbd-domain-adaptation`.

1. Introduction

The presented academic work focuses on a novel Unsupervised Domain Adaptation (DA) method for the domain transfer of multi-modal RGB-D data, from a synthetic dataset to a real one. Synthetic and real domains happen to have significantly different marginal distributions and this impacts on the performance of the network. Our aim is to analyse such method in the light of traditional DA settings, replicate the results shown in the original paper by Loghmani, M.R. *et al*. [4] and eventually enrich the method with an additional domain adaptation strategy. More specifically, here is a brief summary of our contribution:

i We implement the network architecture described in the original reference paper along with the dataset handling and transformation procedures;

ii run all the experiments necessary to replicate the baseline results as well as those of the main RR method;

iii perform some light hyperparameter tuning in order to try to improve the scores shown in the benchmark tables of the paper;

iv implement the two variants of the Adaptive Feature Norm approach by Xu, R. *et al*. [5], Hard AFN and Stepwise AFN;

v run the baseline experiments with AFN, again, trying to replicate the results shown in the reference paper;

vi test the performance of the two DA strategies combined, relative rotation and AFN, with another hyperparameter tuning procedure with the purpose of reaching some improvement in the overall performance.

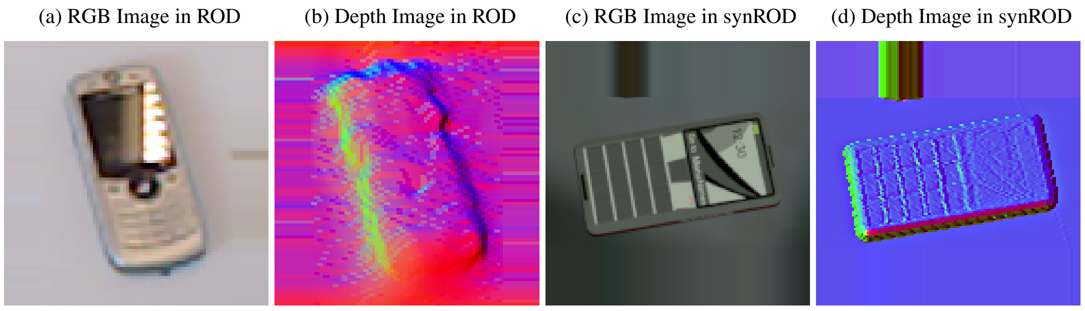
2. Datasets

Since the focus of the project was on synthetic-to-real domain shift two datasets have been used in the experiments: a source (sythetic) dataset and a taget (real) dataset. The datasets used are the ones of the original paper on Domain Adaptation in RGB-D context [4] synROD (source and syhtetic dataset) and ROD (target and real dataset).

- ROD (RGB-D Object Dataset) is a large dataset (41,877 RGB-D images) of 300 common household objects and it is composed of 51 categories arranged using WordNet hypernym-hyponym relationships. The objects into the categories are also organized into instances (e.g. the category ”apple” has five sub-folders which indicates the five instances of apple used to generate all the crops). To obtain all the images a Kinect camera has been used and the recorded objects have been placed on a turntable; in this way it has been possible to capture a whole rotation of synchronized and aligned RGB and depth images. Each rotation has been made with different angles with respect to the table in order to capture different views and take crops of the images from a wide range of perspectives. As stated in [4] it is *”arguably the most used RGB-D dataset in robotics for object categorization”*.

- synROD (synthetic ROD) is a dataset generated for the specific purpose of the synthetic-to-real domain shift with respect to the ROD dataset so it contains the same

Figure 1: Examples of cell phone images in the datasets



51 categories which have been generated from models from the free catalogs of public 3D model repositories, such as 3D Warehouse and Sketchfab, keeping just the models that present texture information in order to be able to render the RGB modality in addition to the depth (so using only the models which could be useful to exploit the multi-modality of the images for the recognition). From the selected 303 textured 3D models the images for all the 51 categories have been generated by sampling, during the rendering, the poses of the camera and the light source from an upper hemisphere of the plane with varying radius to obtain different perspectives of the objects. The dataset has been generated using on average six models per category, furthermore the split file used for the training has been generated by a random selection and extraction of approximately 40,000 objects crops in order to match the dimensions of ROD, obtaining in the end two datasets of comparable size.

The depth images have been colorized with surface normal encoding for a more practical usage, one example of RGB and depth images from each dataset can be found in Figure 1. [1] Even if the two datasets contain 51 categories, in four out of the 51 of synROD there are not enough models and images ("lime", "onion", "peach", "tomato") so just 47 categories have been used according to the given split files (which is provided together with the dataset). The datasets have been divided into three splits:

- **synROD train** which is used for training purposes in a supervised manner (37528 pairs of images);
- **synROD test** which is used for the validation of the model on the source domain (7302 pairs of images);
- **ROD** which is composed by the selected 47 classes of ROD and which will be used for validation of the model on the target domain and, in an unsupervised manner, for the training (32476 pairs of images);

[1]It can be noticed that the depth images from ROD dataset appear more jagged in the colorization with respect to the ones from synROD dataset because the latter are not real images thus they do not suffer from noise and occlusion.

2.1. Transformations

The pipeline of transformations which have been applied is slightly different among training and test samples. In fact, after the resizing to 256×256 pixels, the training images have been *randomly* cropped to 224×224 pixels (to fit the dimensions of ImageNet pictures) while the test images have all been *centrally* cropped (so there is no randomness in the process). It is worth to be noticed that both the RGB and the depth image of the training set must be randomly cropped in the same part and in order to achieve this result a custom coupled random crop transformation has been implemented and applied. Then the tensors have been normalised according to ImageNet mean and standard deviation.

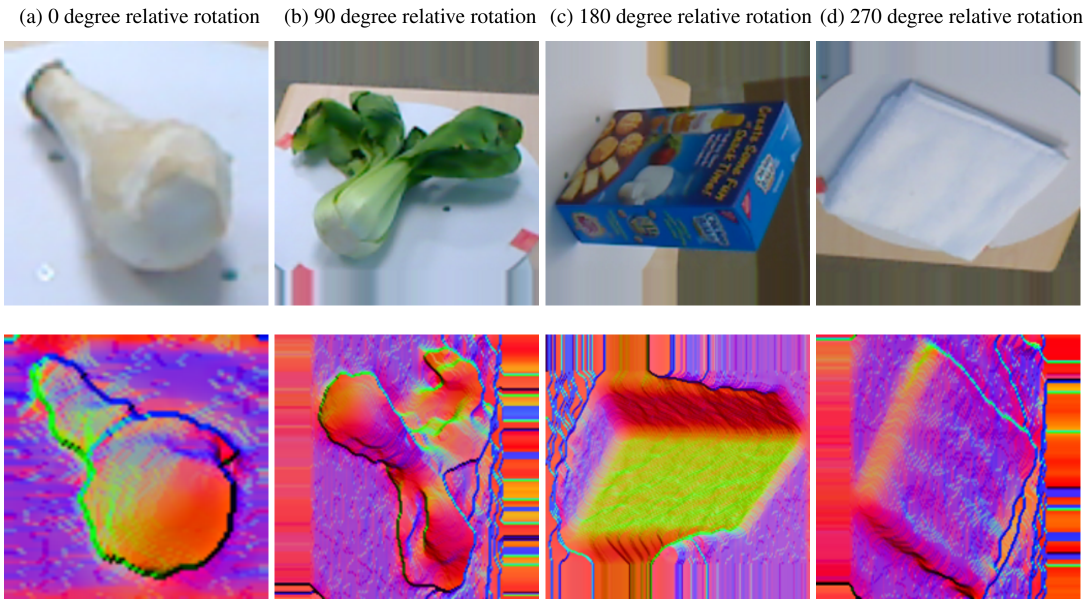
3. Method

In this section we present the method for RGB-D DA originally illustrated in [4], while, where considered necessary, adding some details more related to our actual implementation. Starting from a general description of the method (3.1), we then explain the network specification (3.2) and the composition of the loss function on which the training procedure is based (3.3).

3.1. Overview

The strategy proposed in Loghmani, M.R. *et al.* [4] consists in training a CNN to solve two tasks: the main classification task and a pretext (auxiliary) task which involves both RGB and depth images. This novel approach, applied to synROD$\rightarrow$ROD domain adaptation problem, leverages the inter-modal relations of RGB-D data and outperforms existing DA approaches originally designed on

Figure 2: Examples of relative rotations drawn from ROD dataset


(a) 0 degree relative rotation
(b) 90 degree relative rotation
(c) 180 degree relative rotation
(d) 270 degree relative rotation

single-modality data.
The pretext task consists in predicting the relative rotation between the RGB and depth images of a single sample. The pair is randomly and independently rotated by a multiple of 90 degrees and then fed to the network (see Figure 2). The output of the prediction is therefore an integer in $[0, 3]$ which represents the number of times the RGB image must be rotated clockwise to match the orientation of the depth image. Being a self-supervised task, samples drawn from both target and source datasets are transformed in the same way (the one just described) so to achieve the domain-invariance of the extracted features.

3.2. Network architecture

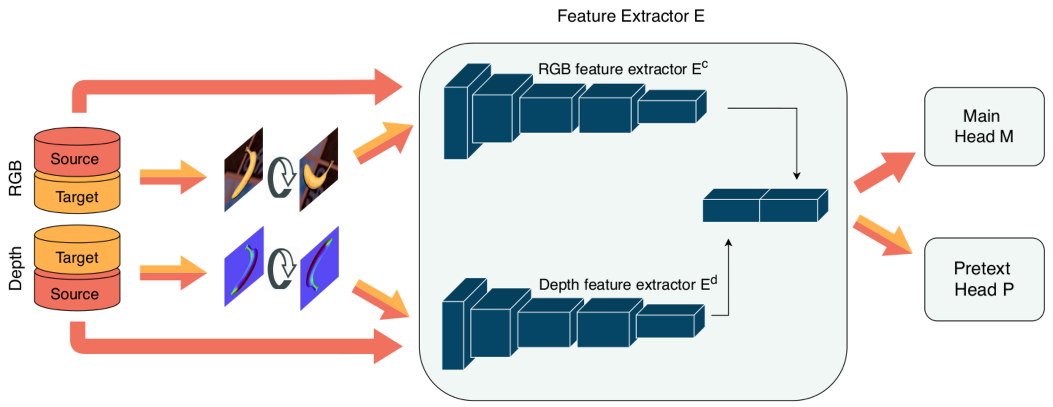
The network is built upon a single PyTorch module based on ResNet18 architecture and in particular it is composed of the following components:

- Two identical feature extractors consisting of a module derived from ResNet18 in its implementation from the PyTorch library: the main difference is that our architecture does not include the final fully connected and global average pooling layers. These two CNNs process the RGB and depth images respectively in order to provide us with the features extracted from the two modalities. The weights of both features extractors are initialized with the values of ResNet18 pretrained on ImageNet dataset.

- The main head which solves the classification problem. This network is made of an adaptive average pooling layer, a fully connected layer with 1000 outputs followed by a batch normalization layer and a last fully connected layer which computes the scores associated to each class label. A ReLU function is applied to the features coming out from the batch normalization layer and during the training phase a dropout with p 0.5 is performed on the input of the last fully connected layer. This network has been designed so that it can be used when dealing with both modalities or just one at a time (RGB or depth) as it fits both cases, single-modality and RGB-D samples, changing automatically the dimension of its first fully connected layer.

- The pretext head, which solves the relative rotation task consisting of a four-way classification problem since the possible rotations used are of multiples of 90 degrees. In the network in question convolutional layers are used differently from the main head in order to preserve the spatial information to better predict the relative rotation. In particular the network is composed of *conv(1×1, 100)*, *conv(3×3, 100)*, *fc(100)*, *fc(4)* where *conv(k×k, n)* indicates a 2D convolutional layer with kernel size $k \times k$ and n neurons and *fc(k)* indicates a fully connected layer with an output of dimension k. All convolutional and fully connected layers use batch normalization and all of them, except for the last fc layer, use a ReLU as activation function. As

Figure 3: Visualization of the architecture of the model adopted in RGB-D DA with relative rotation pretext task. Credits of the image to [4].



in the main head, a dropout with p 0.5 is performed on the input of the last fully connected layer and we will compute cross-entropy loss on the class scores given by the pretext head. Differently from the first convolutional layer which has stride equal to 1, the second convolutional layer has stride equal to 2 so that the 100 channels in output from it have smaller dimensions in order to speed up our work.

The main head and the pretext head are initialized with Xavier initialization [2] with biases initialized to zero. The network has been implemented in order to be able to distinguish if the passed batch of images should go to the main or to the pretext head and to identify the modality used.[2]

3.3. Optimization

Denoting with M and P respectively the main and the pretext head of the model, and with E the feature extractor (to be intended as the combination of both E^c and E^d, for RGB and depth images), we define $\hat{y}^* = M(E(x^*))$ and $\hat{z}^* = P(E(\tilde{x}^*))$ where input data x^* and transformed input data $\tilde{x}^*$ belong either to the source or the target domain. Then let y^s, z^s and z^t be simply the ground-truth labels of the main and pretext tasks for source and target samples. The objective function to be minimized is $L(\theta_e, \theta_m, \theta_p) = \mathcal{L}_m(y^s, \hat{y}^s) + \lambda_p \mathcal{L}_p(z^s, \hat{z}^s, z^t, \hat{z}^t) + \lambda_e \mathcal{L}_h(\hat{y}^t)$ where $\mathcal{L}_m$ and $\mathcal{L}_p$ are respectively the cross-entropy loss of the main and pretext tasks, $\mathcal{L}_h$ the entropy loss function and λ_p, λ_h the weights regulating the contributions of the losses. Specifically,

$$\mathcal{L}_m = -\frac{1}{N_s} \sum_{i=1}^{N_s} y_i^s \cdot \log\left(\hat{y}_i^s\right), \tag{1}$$

$$\mathcal{L}_p = -\frac{1}{\tilde{N}_s} \sum_{i=1}^{\tilde{N}_s} z_i^s \cdot \log\left(\hat{z}_i^s\right) - \frac{1}{\tilde{N}_t} \sum_{j=1}^{\tilde{N}_t} z_j^t \cdot \log\left(\hat{z}_j^t\right), \tag{2}$$

$$\mathcal{L}_h = -\frac{1}{N_t} \sum_{i=1}^{N_t} \hat{y}_i^t \cdot \log\left(\hat{y}_i^t\right), \tag{3}$$

where N_s, N_t, $\tilde{N}_s$, $\tilde{N}_t$ denote the size of source and target data before and after the transformation described in Subsection 2.1.

The role of the first loss is, of course, to encourage useful features for the classification (main) task; the second loss favours the domain shift between source and target data; the entropy loss instead acts as a regularization term, leveraging more confident predictions when feeding the target data to the main head. For what concerns the latter function, penalizing the entropy of the output is not always guaranteed to help improving the accuracy, but it has been shown effective in many domain adaptation experiments, like in [5] [4] [3]. Another helpful side-effect of entropy regularization is that it lets the running mean and standard deviation of the batch normalization layer in the main head to update according not only to the source data, but also to the target data which, belonging to different domains, likely have different statistics. This helps during testing phase since the main head will eventually perform better normalization.

4. Variation: AFN

Differently from the aforementioned domain adaptation method, which tries to make the model learn domain-invariant features through the minimization of the error on a pretext task, the Adaptive Feature Norm (AFN) approach presented in [5] suggests that the difference in the norm of

[2]If both RGB and depth representations are being processed, then the features related to the two modalities are extracted and concatenated before going through the chosen head.

the feature representations of source and target data could partly explain the common issues arise in domain adaptation. It has been shown that while the features of the source data have a quite large norm, the target data instead lie inside a sphere with very small radius. This could mean that the features do not contain enough information for the purpose of classification of target data. In particular, quoting the paper to which this section refers, there remain two hypothetical interpretations:

> 1) *Misaligned-Feature-Norm Hypothesis*: The domain shift between the source and target domains relies on their misaligned feature-norm expectations. Matching the mean feature norms of the two domains to an arbitrary shared scalar is supposed to yield similar transfer gains.
> 2) *Smaller-Feature-Norm Hypothesis*: The domain shift substantially relies on the excessively less-informative features with smaller norms for the target task. Despite non-rigorous alignment, adapting the target features far away from the small-norm regions can lead to safe transfer.

The idea is therefore to minimize at the same time the source error and the discrepancy between source and target in the norm of the extracted features or, following the second hypothesis, encouraging the model to learn features with higher norm.

4.1. Overview

After these observations, our work on this approach consists in the implementation and test both Hard and Step-wise AFN on (synROD $\rightarrow$ ROD), trying to replicate the baseline results shown in the reference paper [4].
The implementation of the Hard Adaptive Feature Norm (HAFN) algorithm is quite straightforward. Each forward pass over the main head (of both source and target data) will output not only the logits of the classification, but also the features produced right before the last fc-layer of the main head (which, in our case, is a vector of size 1000). Then we let the optimizer minimize the discrepancy between the mean feature norms[3] of the two domains by forcing the simultaneous convergence to a shared fixed R value.
Similarly, Stepwise Adaptive Feature Norm (SAFN) algorithm takes the embedded features and encourages their norm to enlarge by a step size Δr. SAFN architecture is sketched in Figure 4. More details on the loss function used are presented in subsection 4.3.

[3]The mean of the norms is, in practice, taken over the mini-batches.

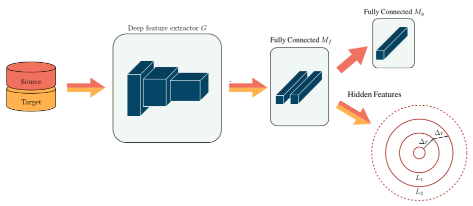


Figure 4: A simple visualization of the architecture adopted in the Stepwise variant of AFN approach. Both source and target data are sent through the feature extractor and, right before the last layer of the classification head, the output of the feature embeddings is produced. The optimization protocol encourages the norm of the obtained vector to increase of a step size Δr through backpropagation of the gradient of an objective.

4.2. Network architecture

In this section we describe the network architecture of the AFN variation (also represented in Figure 4).[4] Our implementation of the AFN network has basically the same architecture of the aforementioned model (see Section 3.2) because this domain adaptation technique does not require a different structure. On the other hand, since AFN algorithm is computed based on the L_2-norm of the hidden features and the standard dropout is L_1-preserving, we have to introduce a slightly different dropout operation which will be L_2-preserving. The standard dropout layer scales its output by a factor of $\frac{1}{(1-p)}$ but it can be easily proved[5] that in order to preserve the L_2-norm, the new dropout layer should scale its output by a factor of $\frac{1}{\sqrt{(1-p)}}$. This is achieved by multiplying the hidden features by a factor of $\sqrt{(1-p)}$ inside the forward method of the model, where p (the dropout frequency) is passed as a parameter to the network. Similarly to what is described above, the forward pass can be performed when dealing with both RGB and depth images or just one of the two and since the AFN technique is applied on the 'Hidden Features', in this case the forward method will return the class scores computed by the classifier along with the input of the last fully connected layer.

4.3. Optimization

We denote the first $l-1$ layers of the classifier as M_f, called bottleneck feature embeddings. The output of $\boldsymbol{f} = M_f(E(\cdot))$ consists in the domain-transferable features. Then, depending on the variation, we force their norm to get closer to a shared value R (Hard AFN) or to get larger each

[4]We would point out that in this paragraph with the term 'Hidden Features' we will denote the features which constitute the input of the last fully connected layer in the main head network.

[5]A detailed proof is present in the original paper.

iteration (Stepwise AFN).

In the Hard Adaptive Feature Norm setting, the objective function results in the one shown in Equation (5) after having defined the loss function in Equation (4)

$$\mathcal{L}_a = L_d(\frac{1}{N_s}\sum_{i=1}^{N_s}||\boldsymbol{f}(x_i^s)||_2, R) + L_d(\frac{1}{N_t}\sum_{j=1}^{N_t}||\boldsymbol{f}(x_j^t)||_2, R) \tag{4}$$

$$\mathcal{L}(\theta_e, \theta_m) = \mathcal{L}_m(y^s, \hat{y}^s) + \lambda_e \mathcal{L}_h(\hat{y}^t) + \mu \mathcal{L}_a(x^s, x^t) \tag{5}$$

with $L_d(\cdot,\cdot)$ taken as squared difference[6] and μ as the weight regulating the contribution of $\mathcal{L}_a$ to the total loss function. The parameter R is an hyperparameter of the method and there is no general rule for deciding which value assign to it. In particular, following the second hypothesis quoted at the beginning of the section, instead of choosing a fixed value for R, we can set it to the current norm value incremented by a step size of Δr, encouraging feature-norm enlargement. To this aim we define a new objective which replaces the previous one in the total loss function

$$\mathcal{L}_a = \frac{1}{N_s + N_t}\sum_{x_i \in S \cup T} L_d(||\boldsymbol{f}(x_i)||_2 + \Delta r, ||\boldsymbol{f}(x_i)||_2) \tag{6}$$

where S and T are respectively the source and the target datasets. It is clear however that such an implementation does not fix limits on the growth of the feature norm and with high values of Δr the objective function might explode. One could think about constraining the norm to a common fixed value R, resulting in a sort of mix of the two aforementioned variations. However in the original work it is shown that, with a reasonably small step size, an additional constrain on the norm does not lead to any improvement of the overall performances.
It is worth mentioning that HAFN algorithm minimizes the gap between the mean feature norms of the two domains in terms of Maximum Mean Feature Norm Discrepancy (MMFND) (defined in Equation (7)), while SAFN encourages a feature-norm enlargement at the step size of Δr with respect to individual examples (the mean operation is outside the L_2-distance). According to Xu, R. *et al.* [5] *"SAFN enables the optimization process more stable and fairly easy to trade off between the two objectives"*, this leads to more informative features and, as a consequence, higher accuracy on the target domain.

[6]It is worth mentioning that in the original paper L_d is taken as the L_2-distance (or, equivalently, Euclidean distance). This, however, would result in an ambiguous formulation since the distance is taken between two scalars, while L_2-distance is typically defined only on vectors. Moreover, the actual implementation of such formula in the code linked by the paper, does not include the square root operation, resulting, in fact, in a simple squared difference between scalars.

$$\text{MMFND}[\mathcal{H}, S, T] := \sup_{h \in \mathcal{H}} (\frac{1}{N_s}\sum_{x_i \in S} h(x_i) - \frac{1}{N_t}\sum_{x_j \in T} h(x_j)) \tag{7}$$

5. AFN and RR

The two methods described in the previous sections, RGB-D DA with relative rotation (RR) pretext task and AFN DA, can be combined together in a new domain adaptation procedure. In fact, the two methods have different purposes and therefore may have distinct impacts on the parameters of the network. The first one acts on the ability of the network to extract domain invariant features through the pretext task which helps both in merging the information of the two modalities and in learning to recognise source and target sample indistinctly. The second, instead, works on another characteristic of the images which is their final representation feature vector, in particular its norm. The implementation of this procedure does not require any modification to the model architecture with respect to the one described in Subsection 4.2 and can be applied just by combining the losses related to the two task in a single objective function, shown, for the sake of completeness in Equation 8. For a schematic view of the complete structure, see Figure 5. Finally, the steps performed in this method are listed in Algorithm 1 for further clarification.

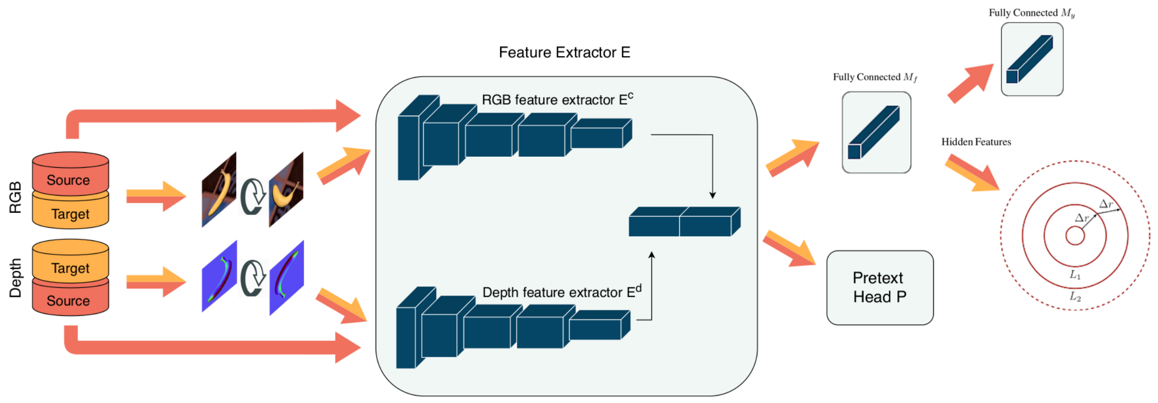
$$\begin{aligned}\mathcal{L}(\theta_e, \theta_m, \theta_p) = \mathcal{L}_m(y^s, \hat{y}^s) + \lambda_p \mathcal{L}_p(z^s, \hat{z}^s, z^t, \hat{z}^t) \\ + \lambda_e \mathcal{L}_h(\hat{y}^t) + \mu \mathcal{L}_a(x^s, x^t)\end{aligned} \tag{8}$$

Hopefully these two strategies can sum their contributions to the network in a non-destructive manner, leading to an overall improvement of the domain adaptation performance. This idea has been subject to various experiments which can be examined in Section 6.

6. Experiments

In this section we will report the results obtained with all the different aforementioned methods. In particular in Section 6.1 we report the results achieved without any kind of domain adaptation (source-only methods). In Section 6.2 there are the results obtained with the training method described in [4] which exploits the multi-modality of the RGB-D images through the usage of the relative rotation pretext task. In Section 6.3 there are the results obtained with the Stepwise AFN method applied on the different modalities of the images. In Section 6.4 we illustrate further trials using HAFN algorithm depicted in [5]. Then in Section 6.5 are reported the results obtained from an additional variation which combines the AFN methods (HAFN

Figure 5: Visualization of the architecture related to our variation consisting in both AFN and RR. In particular, this figure shows the Stepwise version of AFN.



Algorithm 1 SAFN + RR

```
Input
    Labeled source dataset $S = \{(x_i^s, y_i^s)\}_{i=1}^{N_s}$
    Unlabeled target dataset $T = \{x_i^t\}_{i=1}^{N_t}$
Output
    Object class prediction for the target data $\{\hat{y}_i^t\}_{i=1}^{N_t}$
 1: procedure TRAINING($S, T$)
 2:     Get transformed set $\tilde{S} = \{(\tilde{x}_i^s, z_i^s)\}_{i=1}^{N_s}$
 3:     Get transformed set $\tilde{T} = \{(\tilde{x}_i^t, z_i^t)\}_{i=1}^{N_t}$
 4:     for each iteration do
 5:         Load mini-batch from $S$
 6:         Compute main loss $\mathcal{L}_m$
 7:         Load mini-batch from $T$
 8:         Compute entropy loss $\mathcal{L}_h$
 9:         Compute SAFN optimization obj. $\mathcal{L}_a$
10:         Load mini-batches from $\tilde{S}, \tilde{T}$
11:         Compute pretext loss $\mathcal{L}_p$
12:         Update weights of $M$ from $\nabla\mathcal{L}_m$,$\nabla\mathcal{L}_h$,$\nabla\mathcal{L}_a$ [7]
13:         Upd. weights of $P$ from $\nabla\mathcal{L}_p$
14:         Upd. weights of $E$ from $\nabla\mathcal{L}_m$,$\nabla\mathcal{L}_p$,$\nabla\mathcal{L}_h$,$\nabla\mathcal{L}_a$
15:     end for
16: end procedure
17: procedure TEST($T$)
18:     for each $x_i^t$ in $T$ do
19:         Compute $\hat{y}_i^t = M(E(x_i^t))$
20:     end for
21: end procedure
```

and Stepwise AFN) with the relative rotation pretext task. Finally the last two sections (6.6 and 6.7) concern further ablation studies. The default set of hyperparameters provided (which have been used as a starting point for all the experiments) are reported here for the sake of completeness:

- Learning rate: 0.0003
- Learning rate fixed during all the training process
- Weight for the loss of the Pretext Head (λ_p): 1.0
- Entropy Loss Weight (λ_h): 0.1
- Batch size: 64
- Stochastic Gradient Descent Optimizer
- Momentum: 0.9
- Weight decay: 0.05
- Dropout: 0.5

We would point out that, due to the hardware limitations, we have conducted all the experiments with 10 or 20 epochs rather than 40 epochs as suggested also because we noticed that, after some tests, the best results in terms of accuracy were always obtained in the first epochs.

6.1. Source-only baseline

The source-only baselines can be performed by using just one modality (RGB or depth) or both the modalities in an end-to-end fashion (which means concatenating the features extracted from two distinct feature extractors). Before any tuning phase, we decided to run the source-only baseline experiments with the aforementioned default set of hyperparameters and we obtained the results reported in the relative column in Table 1. Apart from the depth only experiment, which showed better results with respect to the

[7]To be precise, $\nabla\mathcal{L}_a$ does not backpropagate on the last fully connected layer but just on the remaining layers denoted as M_f.

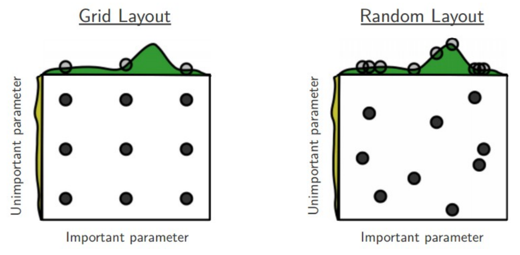


Figure 6: Differences among the two search algorithms

Table 1: Accuracy (%) of source-only methods for RGB-D images

Source-only methods				
Modality	Default	Tuning	Average	Stand Dev
RGB	50.12	51.96	50.59	1.67
depth	10.21	15.45	18.65	0.78
e2e	45.17	47.34	43.24	1.25

original paper, the other ones performed slightly worse and the accuracies had a decreasing trend. So, since in the original paper no kind of step decay policy was adopted on the learning rate, we thought that using a step scheduler would have helped in reaching better performances also because, as stated before, after some first experiments we noticed that the best results were already achieved during the first epochs. Besides, when conducting a training process of a neural network, learning rate is the most influential hyperparameter and this is the reason why we decided to perform a tuning on learning rate too, apart from step size and γ which are the hyperparameters related to the chosen scheduler. Furthermore as far as the search method is concerned, we preferred using a random search approach instead of a classical grid search because it often happens that some hyperparameters (e.g. learning rate) matter more than others and, as a consequence, a random search would allow us to discover better values for the important hyperparameters as explained in [1] and shown in Figure 6. A grid search would be better to be used when performing a finer search in the space of the hyperparameters after having identified a good region using firstly a random search. If we would have used a grid search as first approach it would have probably spent time evaluating unpromising regions of the hyperparameters space.

In order to do this search we randomly selected ten sets of configurations from the following ranges:

- fifty logspaced values from 10^{-5} to 10^{-2} for the learning rate;
- γ in (0.3, 0.1, 0.05, 0.02);
- step size in (2, 3, 4, 5, 6, 7);

We conducted the same random search when dealing with the three aforementioned settings and for each selected configuration we ran a 10 epochs training. The best results in terms of accuracy reached considering all the epochs are reported in Table 1.

After these experiments, for each modality, we analyzed the accuracy and loss plots obtained from the tuning runs and the default runs. We chose the configuration of hyperparameters which led to the best trends and we performed five runs, each for 10 epochs, with that configuration in order to compute mean and standard deviation statistics on the maximum accuracies obtained in each trial. This has been done to check if the method was robust and stable. Before doing the final five runs we also looked at the plots to avoid choosing a set of hyperparameters leading to maximum accuracy among all the configurations, although having a worse trend. This would have meant choosing a configuration that reaches an high accuracy in one epoch just by chance. In this section, in all the three cases, the maximum accuracies were reached with the three sets of hyerparameters for which the corresponding trends had also reasonable shapes. In particular:

- for the RGB modality we chose learning rate 6.3×10^{-5}, γ 0.05, step size 7;
- for the depth modality we chose learning rate 8.7×10^{-3}, γ 0.05, step size 5;
- for the e2e modality we chose learning rate 2.9×10^{-4}, γ 0.1, step size 2;

The five runs conducted with these three configurations have brought to the results reported in the third and fourth columns of Table 1. It can be noticed that the computed means are in line with the maximum values previously obtained (the average accuracy for depth mode is even greater than the maximum one reached in the experiments) and besides the values of standard deviation are quite low which means that the configuration of hyperparameters we chose as best ones are stable.

6.2. Relative rotation baseline

In this section we have tried to replicate the results obtained in [4] using the relative rotation as pretext task and considering both modalities. Firstly we trained the model for 10 epochs using the aforementioned default set of hyperparameters and the best accuracy reached was only 50.12%. That was much lower than the one in [4] so even in this case we performed a first tuning using a random search on the

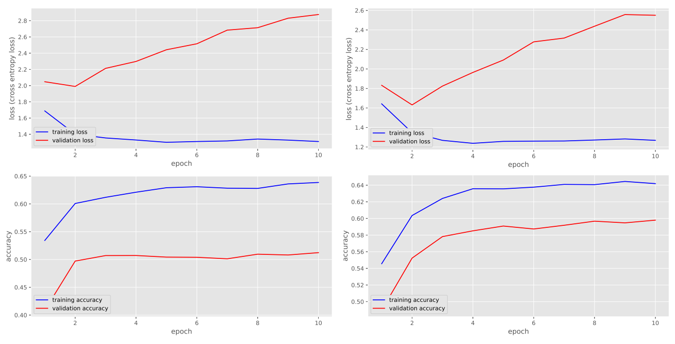
learning rate and introducing a step scheduler which makes use of step size and γ as hyperparameters. The reasons why we chose to adopt a random search and a scheduler are the same explained in Section 6.1. This tuning, which consists of ten training sessions of 10 epochs each, brought indeed to an increase in the best accuracy reached of $+8\%$. After this first tuning we performed a further search on the weight decay because it seemed to be quite high in the default set. So we ran three different trainings, again of 10 epochs each, testing three different weight decays (0.05, 0.005, 0.0005) and leaving the remaining hyperparameters as default: the best accuracy was found with weight decay 0.0005 and it was 59.80%.
At this point, being the results still lower than the reference ones, we thought that combining the best hyperparameters resulting from the first tuning with a weight decay of 0.0005 would have led to better performances because the weight decay deals with the regularization term. So we ran a training experiment with this setting for 10 epochs but the result was disappointing as shown in Table 2 in the fourth line. Moreover the trends of the accuracy and of the loss, depicted in Figure 8b, reach a plateau so we concluded that it was not even worth it to try training for more epochs. This could be explained by the fact that the value of γ used in this case was quite low (0.02) and step size was only 3: in fact the plateau is reached after 3 epochs. We could have softened the effect of the scheduler but we decided not to do this because, as can be seen in Figure 8b, we already noticed a big gap between the performances reached on the training set and on the validation one, and this is a clear symptom of overfitting. On the contrary, the training executed with the same learning rate, step size and γ but with the default weight decay showed, in addition to a better accuracy, a better trend as illustrated in Figure 8a.
Concerning the experiments with the default hyperparameters but with different weight decays (third line in Table 2), even if they brought to the maximum accuracy reached (59.80%) we decided to analyze the trends in Figures 7a and 7b: the validation loss has an important increasing evolution and this is due to overfitting as can be understood by the large gap between the two accuracies. In this case the fact that the validation accuracy still does not decrease is not significant because the increase of the loss means that the model is not able to predict the right label with confidence.
Since we were still not satisfied, using a weight decay of 0.0005, we tried a last tuning on learning rate, step size and γ (10 different trainings, each for 10 epochs, using the same random search technique previously explained) but even in this case the maximum accuracy reached and the trends were worse than what already obtained.
So in the end we decided to choose as best configuration of hyperparameters the one achieving an accuracy score of 58.13% (learning rate 1.46×10^{-4}, γ 0.02, step size 3) and we executed two[8] trainings, each for 10 epochs: we obtained an average accuracy of 57.65%, which is in line with the performance in the tuning phase with the same choice of hyperparameters, and a standard deviation of 0.33% which is a very small value but still we can not conclude with confidence that these hyperparameters bring to a stable model due to the low number of runs performed in this case. Please note that, although it is suggested to perform the same number of trials in all methods (source-only and RR), we carried out a heavier hyperparameters tuning on the latter one because we found complications in reaching the same accuracy score reported in [5].

Table 2: Accuracy (%) of RGB-D Relative Rotation

RGB-D Relative Rotation		
Hyperparameters	Best accuracy(%)	Weight decay
Default	50.12	0.05
1st tuning	58.13	0.05
Weight decay tuning	59.80	0.05, 0.005 0.0005
Best hyperparameters of 1st tuning	42.35	0.0005
2nd tuning	51.42	0.0005

Figure 7: Experiments with different weight decay values

(a) Weight decay 5×10^{-3} (b) Weight decay 5×10^{-4}



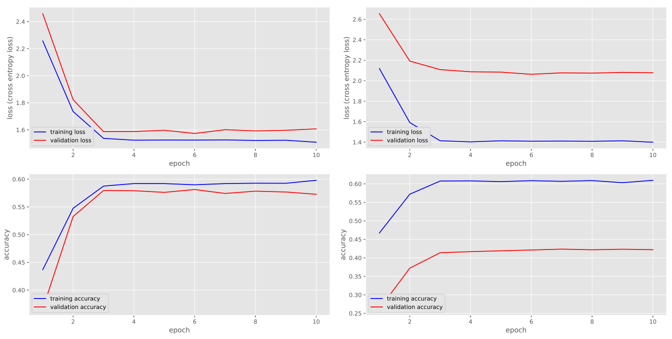
6.3. Stepwise AFN baseline

In this section we wanted to replicate the results reported in [4] as baselines with AFN which has been stated *"as the current state of the art"* for domain adaptation. The baselines refer to the same three modalities used in 6.1 (RGB/depth/e2e). With this new training technique two new hyperparameters have been added:

[8]In the final experiments involving the relative rotation task we have performed two trainings instead of five due to time constraints.

Figure 8: Experiments with different weight decay values and fixed γ 0.02, step size 3 and learning rate 1.46×10^{-4}

(a) Weight decay 5×10^{-2} (b) Weight decay 5×10^{-4}



- Δr which, as default, has been set to 1;
- μ which, as default, has been set to 0.05;
- Batch size, new default fixed to 32 instead of 64;

First of all we trained the model for 20 epochs using the default set of hyperparameters for the three different modalities and we reported the maximum values of accuracy reached in the first column of Table 3. The performances achieved after these first three runs are coherent with reference ones except for the e2e modality. To be consistent with all the previous experiments we decided to conduct the same tuning, which means that we carried out a random search on learning rate, step size and γ leaving the remaining hyperparameters as default. To summarize, this tuning consists of ten trainings, each for 10 epochs, and the best accuracies obtained for each modality, considering all the epochs, are written in Table 3.

After this tuning, similarly to what was done in the previous sections we chose to pick the best configuration of hyperparameters for each modality and to run five experiments in order to check the stabilty of the chosen set. Even in this case we did not simply choose the hyperparameters which brought to the maximum accuracy during its training but we firstly analyzed the plots obtained by all the trials in the tuning phase and we concluded that:

- for RGB modality the maximum accuracy was reached with the default set of hyperparameters and in that training the accuracy and the loss also showed reasonable and good trends, so for this modality we chose this set as the best one;
- for depth modality the maximum accuracy was reached with learning rate 3.7×10^{-3}, γ 0.1, step size 7 and the related plots are showed in Figure 9a. It can be noticed that the maximum accuracy is reached at the first epoch but after that it decreases substantially throughout all the epochs. So we chose as best configuration learning rate 5.7×10^{-3}, γ 0.05, step size 5 which brings to a slightly lower maximum value for the accuracy but features a much more stable trend during the training with an initial increasing of accuracy performances as shown in Figure 9b;
- for e2e modality the maximum accuracy was reached with learning rate 2.4×10^{-3}, γ 0.3, step size 7 and the related plots are shown in Figure 9c. Even in this case the maximum accuracy is reached at the first epoch and then it decreases. So we chose as best configuration learning rate 5.2×10^{-4}, γ 0.3, step size 3 which achieves a slightly lower maximum value for the accuracy but it has a much more stable trend as represented in Figure 9d.

The five runs with the three configurations chosen as just explained have brought to the results reported in the third and fourth columns of Table 3: the accuracy means are consistent, except for the RGB case, with the maximum values previously obtained with the chosen configurations[9] and the values of standard deviation are quite low, except for the e2e case[10], which means that the configuration of hyperparameters we chose as best ones are stable.

6.4. Hard AFN

Since in [5] two different variations of AFN are proposed, we decided to implement also the second one which is Hard Adaptive Feature Norm DA which has been explained in Section 4. The baselines reported in [4] refer to Stepwise AFN algorithm so in this case we do not have any benchmark to make a comparison with. So we decided to adopt as default setting the ones used in the experiments conducted in Section 6.3 except for the radius which, in this variation of AFN algorithm, replaces Δr and we fixed it to 25 which is the best value for this hyperparameter suggested in [5].[11] Starting with this setting, to be coherent with the previous sections, we performed the same tuning: a random search on learning rate, stepsize and γ testing ten different configurations each for 10 epochs. This work has been done for each modality (RGB/depth/e2e) and the best accuracies reached are reported in Table 4.

For all the three modalities the configuration of hyperparameters which brought to the maximum accuracy values

[9] In Table 3 the column 'chosen setting' reports this value.

[10] This is probably due to the low number of final runs.

[11] This value was obtained from a tuning on a different dataset, that of the AFN paper, we thought it was a reasonable value since the task of the analysis conducted in [5] was the same. Moreover, the authors also state that *"the preference setting of R still remains unsettled"* therefore, tuning also that hyperparameter, would have not been an easy task and required a big amount of time.

Table 3: Accuracy (%) of SAFN methods for RGB-D images

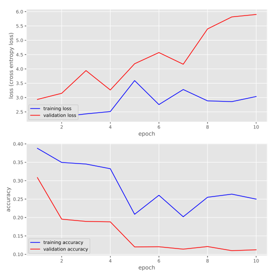
Baselines SAFN						
Modality	Default	Tuning	Chosen Setting	Average	Stand Dev	Standard Dropout
RGB	62.23	58.30	62.23	57.94	1.08	57.51
depth	26.21	30.85	27.10	26.79	1.34	12.73
e2e	52.14	57.63	57.18	55.65	2.13	58.46

Table 4: Accuracy (%) of HAFN methods for RGB-D domain

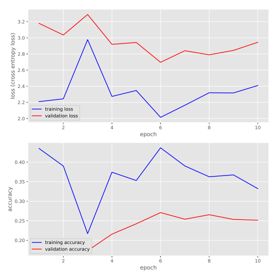
HAFN method on RGB/depth/e2e modality			
Modality	Tuning	Average	Stand Dev
RGB	60.20	56.98	0.62
depth	29.72	28.49	1.04
e2e	58.66	55.40	2.36

Figure 9: Experiments with SAFN algorithm

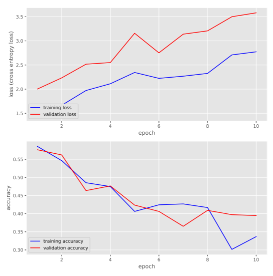
(a) Best hyperparameters SAFN (depth)



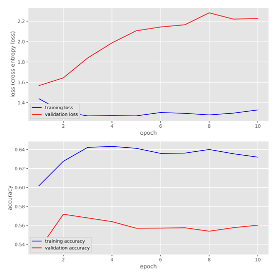
(b) Chosen hyperparameters SAFN (depth)



(c) Best hyperparameters SAFN (e2e)



(d) Chosen hyperparameters SAFN (e2e)



also showed reasonable shapes of loss and accuracy hence our decision to perform five runs, each one for 10 epochs, with those settings. The configurations used are here reported:

- for the RGB modality we chose learning rate 3.9×10^{-4}, γ 0.02, step size 3
- for the depth modality we chose learning rate 7.5×10^{-3}, γ 0.05, step size 6
- for the e2e modality we chose learning rate 5.2×10^{-4}, γ 0.05, step size 2

The results achieved are reported in Table 4.

6.5. Variation: AFN + RR

In addition to the motivation we gave in Section 5 we decided to adopt this variation since from the experiments it turned out that RR and AFN outperformed the source only methods and in particular we combined RR with both AFN variations (SAFN, HAFN). In both cases we decided to begin with a default configuration of hyperparameters which is constituted by the ones listed in Section 6 and:

- μ 0.05 in SAFN + RR and in HAFN + RR;
- Batch size 32 in SAFN + RR and in HAFN + RR;
- Δr 1 in SAFN + RR;
- radius 25 in HAFN + RR;

For both SAFN + RR and HAFN + RR we thought that instead of performing the usual tuning on the learning rate, step size and γ it would have been more reasonable tuning the learning rate (the most important hyperparameter) along with the weight decay, the weight for the loss of the pretext head (λ_p), the weight of the loss defined in AFN (μ) and the entropy loss weight (λ_h). That is because in this variation there are more contributions to the final loss, so it is important to find the right balance between them. For both variations we tested 10 configurations of these hyperparameters (with each training lasting 10 epochs) chosen, again, with a random search approach[12] using the following ranges:

- weight decay in (0.05, 0.005, 0.0005);
- fifty logspaced values from 10^{-5} to 10^{-2} for the learn-

[12]The reasons why the random search was preferred to the grid search are previously explained.

ing rate;

- λ_p in (0.5, 0.8, 1);
- μ in (0.01, 0.05, 0.1);
- λ_h in (0.05, 0.1, 0.2);

We realize that in these two algorithms there are more hyperparameters playing an important role with respect to the previous experiments but still we chose to test 10 configurations from the random search and we did not involve Δr and the radius in our search because we could not afford a deeper tuning due to our limitations.
Anyway the best accuracies in the tuning have been reached with the following configurations:

- in SAFN + RR: λ_h 0.05, λ_p 0.8, learning rate 5.4×10^{-5}, μ 0.05 and weight decay 0.0005 (the accuracy reached was 56.24%);
- in HAFN + RR: λ_h 0.05, λ_p 0.5, learning rate 7.2×10^{-5}, μ 0.05 and weight decay 0.05 (the accuracy reached was 57.23%);

In both cases these configurations also led to nice accuracies shapes as shown in Figures 10a and 10b; the trends are in fact increasing and the gap between the training and the validation accuracy is narrow: this means that our model is able to generalize. However, after 10 epochs the trends reach a plateau therefore we chose not to run any training for an higher number of epochs. For each of the two variations, with the aforementioned sets of hyperparameters, we performed two[13] runs of 10 epochs each and the results are reported in Table 5.
As far as we can conclude by doing only two runs, the average accuracies are in line with the performances reached during the tuning and the resulting standard deviations are very low.

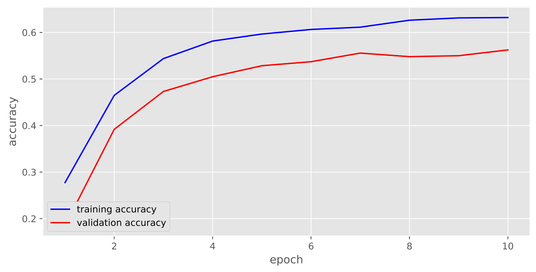
Table 5: Accuracy (%) of Relative Rotation methods

Accuracy of Relative Rotation methods			
Modality	Tuning	Average	Stand Dev
RR [4]	58.13	57.65	0.33
SAFN + RR	56.24	55.19	2.01
HAFN + RR	58.66	55.14	0.48

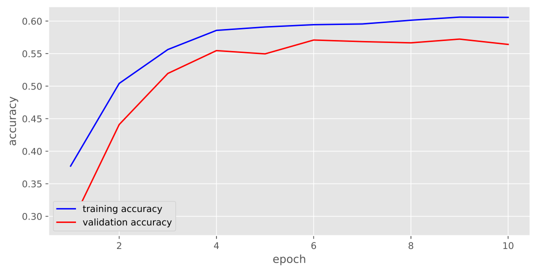
[13] As previously stated, in the final experiments, when the relative rotation task is present, we have performed two trainings instead of five due to our limitations.

Figure 10: Accuracies in the experiments of the variations

(a) SAFN with relative rotation



(b) HAFN with relative rotation



6.6. Further trials

Since in [4] the experiments conducted on SAFN baseline are performed using the standard dropout rather than the L_2-preserving dropout explained in [5], we also tried to implement the SAFN baseline without the rescaling factor ($\sqrt{(1-p)}$): in particular we ran a training of 10 epochs for RGB, depth and e2e modalities and the results are reported in the last column of Table 3. The maximum accuracy reached by the e2e modality has improved so we tried this variation also in the final ablation studies in 6.7.

Table 6: Ablation Studies

Ablation Study on SAFN + RR	
Method	Accuracy(%)
Target rotation	54.44
FC classifier	55.71
No entropy loss	52.78
Standard Dropout	53.39
Ours	56.24

6.7. Ablation Studies on SAFN + RR

In this section we illustrate the ablation studies we performed in order to better understand the influence and the impact that the different components of our variation SAFN

Table 7: Comparison of the baselines

Comparison of the results			
Method		Benchmarks [4]	Our results (avg)
Source only	RGB	51.17	50.59
	depth	15.50	18.65
	RGB-D e2e	47.70	43.24
Stepwise AFN [5]	RGB	64.63	57.94
	depth	30.72	26.79
	RGB-D e2e	62.40	55.65
Relative Rotation [4]		66.68	57.65
SAFN + RR		–	55.19
HAFN + RR		–	55.14

+ RR have on the overall performances. In particular we tried:

- to build the pretext head with the same architecture of the main head (i.e. using fully connected layers instead of convolutional ones);
- not to include the entropy loss, calculated on the target dataset while training, in the overall loss to minimize;
- to use only the target domain to solve the pretext task instead of using both domains;
- not to rescale by a factor of $\sqrt{(1-p)}$ the features coming out from the last dropout layer in the main head;

We executed only a single run of comparison (i.e. a training of 10 epochs) for each of the ablation studies and the results are reported in Table 6. In all the cases described we obtained slightly worse performances with respect to the best result reached with the original variation of SAFN + RR and this can be explained in the following ways:

- using convolutional layers instead of an adaptive average pooling layer followed by a fully connected one is a good idea because it helps to retain the spatial information needed to predict the relative rotation (which is in fact a task related to the position of the objects in the space); this is also stated in [4];
- the usage of the entropy loss has turned out to be useful and effective in this context[14]: it plays the role of a regularization term encouraging more confident predictions on samples from the target domain passing through the main head and it allows the running mean and standard deviation of the batch normalization layer in the main head to update according also to the target data;
- the network is capable of learning more informative features when solving the pretext task with both domains thanks to the higher diversity in the data provided; this is also stated in [4] and our experiment confirms this idea;
- as explained in [5] it is better to have a L_2-preserving dropout when dealing with AFN because this algorithm considers the L_2-norm of the hidden features. Probably the improvement observed in Section 6.6 related to the e2e modality, which made us think we could have had an improvement also in this case, was just limited to the e2e specific case and this is reasonable because when working in these contexts there is not a golden rule but it is often better to perform empirical trials and assess the improvement;

7. Conclusion

We have presented two different strategies in the domain adaptation framework, analysing their impact on the performance over a synthetic-to-real adaptation setting. After extensive experiments, we acknowledge that, unfortunately, our implementation do not manage to reach the same results of the benchmarks in [4]. Up to the writing of the present report, the reasons of this shift are still to be discovered; one possible explanation could be found in some small implementation details (unknown to us) not explicitly reported in the original paper. However we can notice reasonable differences between the source-only setting and the two DA methods scores: there is an evident increase in the performance with both AFN (Stepwise and Hard) and RR with respect to the source-only baselines. We also believe that, although it actually seems to worsen the accuracy, the combination of the two methods, RR and AFN, could eventually provide a more accurate and robust domain adaptation, more likely with wider and in-depth experiments that we could not perform due to our limited computational resources.

[14]As stated in Section 3.3 this is not always true but it has to be empirically proven.

In Table 7 we report an overview of our score results compared to those in the reference paper.

References

[1] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *JMLR*, page 305, 2012.

[2] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[3] Y. Grandvalet and Y. Bengio. Semi-supervised learning by entropy minimization. In *Advances in neural information processing systems*, pages 529–536, 2005.

[4] M. R. Loghmani, L. Robbiano, M. Planamente, K. Park, B. Caputo, and M. Vincze. Unsupervised domain adaptation through inter-modal rotation for rgb-d object recognition, 2020.

[5] R. Xu, G. Li, J. Yang, and L. Lin. Larger norm more transferable: An adaptive feature norm approach for unsupervised domain adaptation, 2018.