Machine Learning and Deep Learning project: RGB-D Domain Adaptation with Cross-Modality Self-Supervision

Akafak Sokeng Jordan Derlich
Politecnico di Torino
s270746@studenti.polito.it

Calvi Edoardo
Politecnico di Torino
s259044@studenti.polito.it

Garbarino Matteo
Politecnico di Torino
s265386@studenti.polito.it

Abstract

Multi-modal Domain Adaptation is a powerful, but complex task. Its aim is to combine useful features, obtained by aligning the underlying data distributions of the source and target domains, with the rich information given by multiple data modalities. What we propose is to replicate the experiments on this topic that are presented in the reference paper[6] and to make a variation that explores alternative methodologies to face this problem.

1. Introduction

Over the last few years we have been assisting to the impressive progress in the field of computer vision, where Convolutional Neural Networks (CNNs) have become the dominant tool due to their performances and despite the huge quantity of required data for meaningful trainings. The cost of generating labelled images is a limiting factor for the continuous progress of Deep Neural Network architectures. One possible workaround is to automatically generate synthetic annotated RGB-D images to enlarge the training pool. However, currently there exists a gap in terms of data distribution between artificial and real data, and this negatively impacts performances.
In this context, domain adaptation (DA) techniques can help reducing the domain shift, but most of these strategies have been developed for single modality data, which typically consists of RGB images. Nevertheless, the depth modality is more robust to color variations and contains additional geometric information that can improve performances, as shown in the work of *Loghmani et al.*[6].
Therefore, in this work we explore some DA techniques that leverage both modalities (RGB-D) for each domain in order to reduce the differences. The tried approaches perform two tasks in parallel, where the main task is the required image classification problem, while the other one is auxiliary and aims to bridge the gap between the distributions.
In the present work we mainly focus on two different self-supervised tasks:

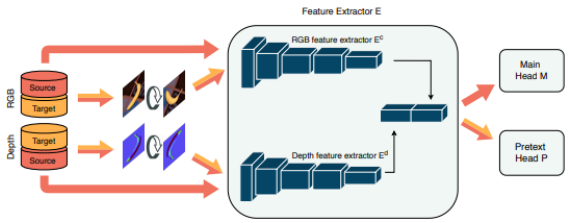


Figure 1. The model from the original paper to be replicated.

The first version is a replica of the experiment in the reference paper and is based on the relative rotation between an RGB image and the corresponding one in the Depth modality. In particular, the images from the two modalities are independently rotated, then the network tries to predict the relative rotation that separates them (figure 1). This pretext task encourages the model to learn geometric relationships between the two modalities.

The second one exploits the same model structure, but the transformation is different; it is a colour permutation, in which the order of the RGB channels in the images of both modalities is randomly and independently swapped, and the network is required to predict the transformation (more details on this in section 3.2).
Thanks to their self-supervised nature, both domains can be used for training for the pretext task. On the other hand, the main task can only use annotated source samples, with the unlabeled target samples available transductively.

The rest of the paper is organized as follows: in *section 2* all the details of the dataset and the preprocessing are presented, *section 3* introduces the model of the used CNN, *section 4* describes the experiments together with their results and *section 5* draws the conclusions.

The code of the experience is available on the GitHub repository of the project.

2. Dataset and data preprocessing

In this section we are going to talk about both the synROD and ROD datasets used to perform our experiments and the pre-processing applied on them. More specifically, *section 2.1* describes how we access and process the data available in the code, and *section 2.2* mainly focuses on how we loaded the dataset and applied data augmentation to the RGB-D pairs.

2.1. Dataset storage and initial pre-processing

Concerning the datasets, they are already available in two zip files that are stored in a Google Drive repository and imported into Google Colab. ROD is used as unlabelled *target dataset* and contains RGB-D real images from 51 different classes all represented in both modalities (RGB and Depth). synROD is the synthetic counterpart, with the same number of classes, and its images are used as labelled *source dataset*.
In order to be able to use these data, we first unzip both the files in Colab and then reorganise the images in folders as shown in figure 2; during the process four classes are discarded as suggested since they are underrepresented. The outcome is a subdivision in four folders that are later used to properly load the dataset with an ad-hoc class. After this initial process, some images which are not present in both modalities are discarded, since they are decoupled and cannot be used for multi-modal purposes. Finally, we adopt the provided split for training and test sets in synROD, so that the learning process has the same amount of training data as in the original experiments and therefore allow comparisons between results.

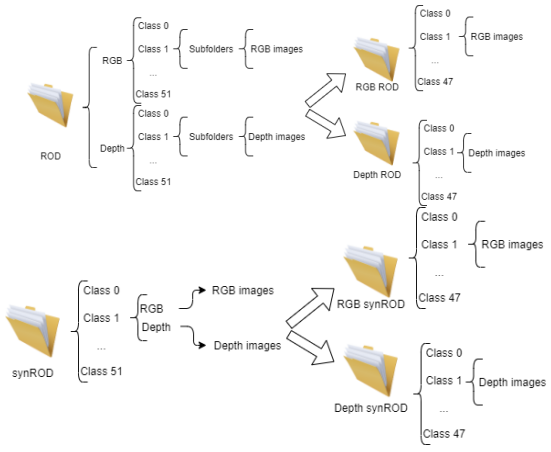


Figure 2. How synROD and ROD are preprocessed: they have different initial shapes, but are reshaped to a standard configuration suitable for the Pytorch[7] ImageFolder class.

2.2. Dual dataset and data agumentation

Having multiple modalities, their images must be preprocessed and appositely transformed before being used in the model. The *DualDataset* class can represent the dataset as *pairs* of RGB and Depth images. During the dataset allocation we apply the data augmentation technique (only on the source) used to automatically generate slightly modified images from the original ones at each epoch, but keeping the same label in order to increase the variety of the training dataset and avoid overfitting. In details, we perform two random transformations (random crop and random horizontal flip) paying attention to apply the same one to the RGB and Depth images. This is done through a re-definition of the used transformations in order to decide whether to apply the same transformation or not on both the elements of the RGB-D pairs. The dataset objects defined in this way are ready to be fed to the dataloaders that can require a regular version of the data as well as a rotated or permuted version of the images (more on this in later sections).

3. Approach

In this section we are going to talk about the approach used to build a suitable model for the multi-modal self-supervised DA problem and its purposes. More specifically, *section 3.1* presents the model architecture, while *section 3.2* shows the details of the implemented self-supervised tasks and *section 3.3* focuses on the loss functions and some optimization details.

3.1. Model Architecture

As it can be seen from figure 1, the model is made of three principal parts which are the *Feature Extractor (E)*, the *Main Task (M)* and the *Pretext Task (P)*.

The feature extractor is made of two identical branches E^c and E^d that respectively extract the features from RGB and Depth images; those features are then concatenated along the channel dimension and given as input to the main head M and/or the pretext head P depending on the situation. Both branches are defined as modified versions of the *ResNet18* architecture[4], without the final fully connected and global average pooling layers. The weights and biases of the extractors are obtained from pre-trained models on *ImageNet* dataset[2], therefore the model has prior knowledge that gets transferred.

The M head is the one which performs the supervised classification task, it only uses the source dataset as labelled data during training and performs class prediction on the unlabelled dataset at test time. Its structure is the following: first we have the average pooling layer with kernel size *(1,1)*, then a fully connected layer *fc(1000)* which uses batch normalization and the ReLU as activation function. We finally use the dropout layer with probability 0.5, fol-

lowed by a *fc(47)* layer as classifier.

The P head is the part of the model which performs a self-supervised task and can thus use both the transformed source and target datasets thanks to the different nature of the addressed problem. It is composed of a convolutional layer with 100 *1x1* filters, followed by another convolutional with 100 *3x3* filters, a *fc(100)*, and the final *fc(n)*, where n is the number of output classes and depends on the chosen task for any given experiment. A value *n=4* is used for relative rotation. All the convolutional and fully connected layers use batch normalization and ReLU as activation function except for the last one, which uses softmax and is subject to dropout.

3.2. Self-supervised tasks

Let us consider a sample of the source images (x_s^c, x_s^d), where x_s^c is the RGB image and x_s^d is the Depth image. In order to reduce the domain shift between the two data modalities we will perform different self-supervised tasks.

The first one performs a *relative rotation* between the two modalities. We apply a clockwise rotation of a multiple of 90° on each of the single images: $(\hat{x} = rot90(90 \cdot k, x)$ where k in $\{0, 1, 2, 3\}$, x is the original image and $\hat{x}$ is the rotated version. Therefore, by knowing the independent values of k for the two modalities, namely k^c and k^d, we can construct the label for the pretext task with a simple computation: $y_s^{rot} = (k^c - k^d) mod(4)$, where *mod* is the integer modulo operation.

The second self-supervised task is the *colour permutation* by means of channel swapping. The idea came taking inspiration from the *Lee et al.*[5] work that used color permutation as a data augmentation technique. The intuition behind this variation is therefore to use a similar approach in self-supervised DA. This is possible since in the datasets the Depth images have already been colorized with surface normal encoding [1], and therefore are represented through the three RGB channels. Hence, the color permutation can be applied on those even if each pixel is still representing the distance from the camera to the object. The initial idea is to implement a task where the two images of an RGB-D couple undergo two permutations out of the six possible (RGB, RBG, BGR, BRG, GBR, GRB) while enforcing a 50% probability of undergoing the same one $(x_{perm}^c, x_{perm}^d = make\_permutation(x^c, x^d))$. The new label $(y_{perm} = 0/1)$ associated to the couple is a binary indicator of whether the transformation was the same or not. In this manner, the task is a binary classifier and the classes are not unbalanced as it would be in the case of applying twice a random permutation out of six to the images, resulting with 83.3% times with a label and 16.7% of the times the other. Avoiding this case discourages the model to easily achieve high performances simply classifying always the same label.

However, the model showed a strange behavior and seemed unable to learn from this pretext task, as will be explained in section 4.3. In order to investigate and better explore this solution, a second variation of the auxiliary task is proposed for comparison. In this case the same permutation is applied at random to both the images and the network classifies which of the six possible transformations was applied.

3.3. Optimization

The loss functions adopted for the experiments are obtained by combining all the contributions, depending on the domain of the used data and the addressed task.

Cross Entropy has been used for all the cases where using labels for training is allowed, such as training the main task with source data, or training the pretext task with transformed data of either domain.

Binary Cross Entropy has been adopted in the pretext task of one of the variations, due to the binary nature of the presented classification problem.

Finally, since target data contains important information for the main task but cannot be used as labeled for this part, *Entropy* has been used as a loss component to measure how clearly the classifier can make a prediction on it.
The expressions of the losses are described in detail in section 4.

Each presented model is trained using a *SGD optimizer*, with the hyper-parameters shown in *table 1*.
The training process may be forced to an early stopping, in cases when the main loss on target does not show improvements for several epochs. This has the added benefits of reducing execution time and the potential overfitting on target data. In order to thin the latter even more, each model is saved every 5 epochs of its training process. This way, the best performing version of the model, which is typically picked near the epoch when accuracy peaks, can be later extracted. It should also be noted that the ideal approach for model selection involves a dataset that is disjoint from both the training and the testing parts, in order to not overfit the model on those sets. However, this is an open problem in the context of domain adaptation and is not meant to be part of this work.

4. Experiments

In this section we are going to talk about both the performed end-to-end experiments and the achieved results. If not differently specified, the parameters used to carry the experiments are the ones shown in *table 1*.

More specifically, *section 4.1* presents the source-only baseline implementation, *section 4.2* describes the domain adaptation experiments, *section 4.3* explores the proposed variation and *section 4.4* displays the results and makes some considerations.

Hyper-parameter	Value
Batch size	64
Learning rate	3e-4
Weight decay	0.05
Momentum	0.9
Number of epochs*	40
λ_1**	1
λ_2**	0.1

Table 1. Initial settings as in the reference paper.
(*) The number of epochs is fixed, but an early stopping criterion is set in order to reduce the execution time when possible.
(**) The specified values are used only in the experiments that require them (I.E. Loss weights in DA and Variation).

4.1. Source only baseline

This experiment is useful to establish a baseline score that allows to understand if the domain adaptation (later applied) brings positive results by accomplishing its duty.
For this experiment the auxiliary task is therefore not needed, only the *Feature Extractor (E)* and the *Main Task (M)* are allocated.
The source data ($S = \{((x_i^{sc}, x_i^{sd}), y_i^s)\}_{i=1}^{Ns}$), limited to the training split, are forwarded through the model ($\tilde{y}_i^s = M(E(x_i^{sc}, x_i^{sd}))$) in order to train the network. At training time, by the end of each epoch, the validation is performed on the entire target set and the validation losses obtained are used to handle the early stopping criterion. Moreover, the data augmentation technique is applied on the source images, but no further transformations as rotations or colour permutations are used.
In order to understand how the model behaves during the training, in figure 3 a visual representation of the trends of both the training loss and the validation accuracy is shown.
In fact, for the first few epochs, we observe a rapid decrease of the training loss while the validation accuracy increases and reaches the peak (47.8%) at epoch 4. Going on with the training, the training loss is still decreasing but the validation accuracy starts decreasing too (the model starts overfitting) and the training is stopped 10 epochs later.

Subsequently a hyper-parameter tuning phase is performed, testing different combinations of batch size, learning rate and weight decay, as shown in figure 4. Without having the possibility to play with any weight, the tuned hyper-parameters are quite standard and it is interesting to notice a different behavior of the models using a smaller weight decay and learning rate. Every model makes use of the early stopping criteria, and it is possible to see that some level of overfitting is happening for the hyper-parameter sets from 7 to 10 by judging the decrease in accuracy they have after their peaks.

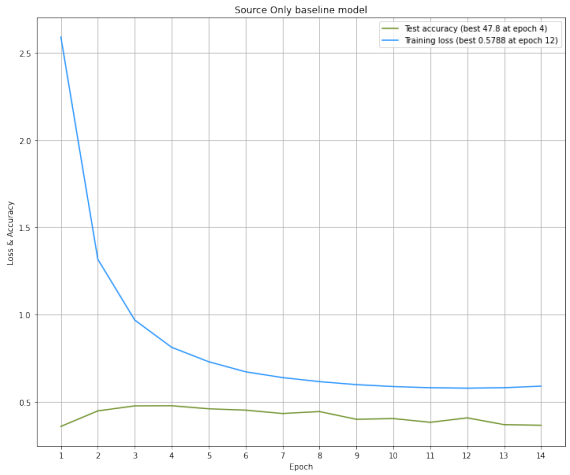
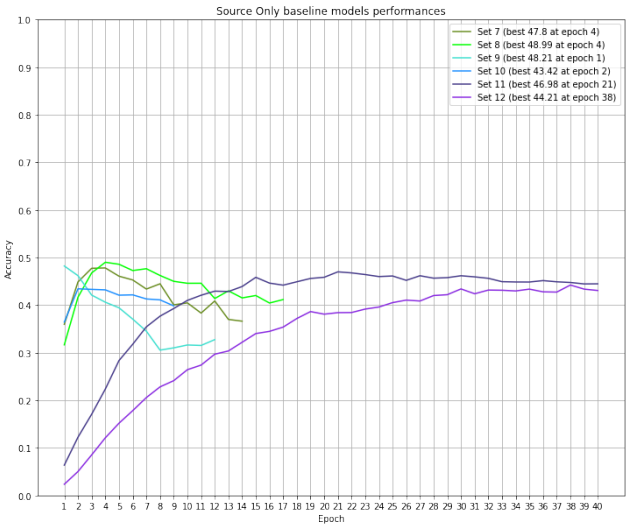


Figure 3. Scores of the source only base model.



Set \Params	Batch size	LR	Weight decay
S7*	64	3e-4	0.05
S8	128	3e-4	0.05
S9	64	6e-4	0.05
S10	128	6e-4	0.005
S11	64	3e-5	0.005
S12	128	3e-5	0.005

Source only hyperparameters sets
(*) Values corresponding to Table 1

Figure 4. Source only hyper-parameters tuning scores: validation accuracy trend (above) for each set of hyper parameter shown in the table (below).

4.2. Domain adaptation - relative rotation

In this experiment the cross-modality domain adaptation comes into play, so not only the *Feature Extractor (E)* and the *Main Task (M)* are allocated, but also the *Pretext Task (P)*. The self-supervised task addressed in this section is the relative rotation and at each training epoch an iteration on four data loaders in parallel is done: source, target, source transformed, target transformed. During the training step,

the data are forwarded through the network according to the pseudo code described in algorithm 1 and the single losses are computed. The *total loss* is built as a weighted linear combination of the single losses:

$$\Lambda_{tot} = \Lambda_s + \lambda_1 * (\hat{\Lambda}_s + \hat{\Lambda}_t) + \lambda_2 * \Lambda_t \qquad (1)$$

where λ_1 and λ_2 are hyper parameters representing the weight for the pretext task and the weight for using target data transductively, respectively. Their initial values are presented in *table 1*. The gradient is then computed and the weights, and all the network parameters, are updated during the back-propagation. Note that, during the training, a validation test is made, similarly to the one in sub-section 4.1 and the scores are collected to be later analysed (a preview is available in figure 5).

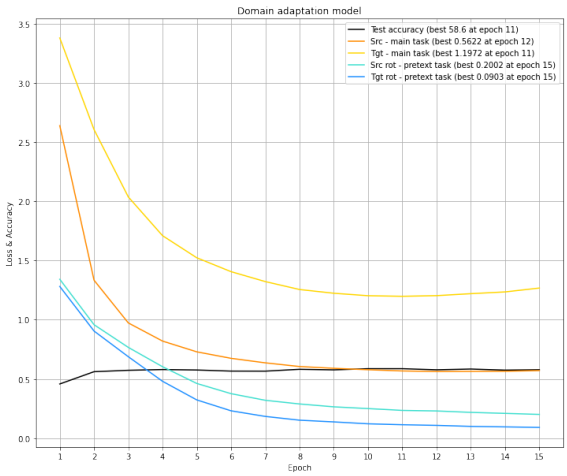
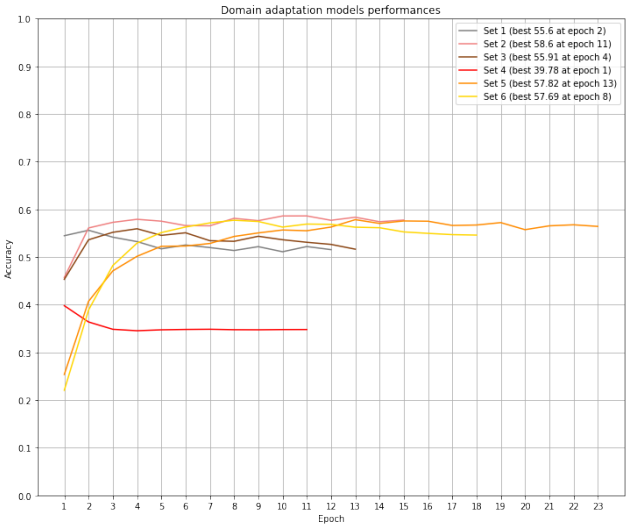


Figure 5. Scores of the DA model with relative rotation. It shows in black the validation accuracy which slightly increases during the training, and the other curves which represent the trend of the single losses calculated in the Main and the Pretext tasks.

Subsequently a hyper-parameter tuning phase is performed to fit our implementation. We tried different values for the weights λ_1 and λ_2, as well as for the learning rate. The main idea is to test how the model behaves assigning a different relevance to the auxiliary task; that is actually an open problem of this field. The outcome is that in this case the pretext task should not weight more than the main task. The results are shown in figure 6 where it is visible that the hyper-parameter set presented in the paper proved to be the most suitable. The plot also shows that models can oscillate after reaching convergence, and therefore further encourages the usage of early stopping techniques.

4.3. Domain adaptation - variation

The network goes through a similar process as presented in subsection 4.2, but the kind of transformation applied



Set \Params	LR	λ_1	λ_2
S1	6e-4	1	0.1
S2*	3e-4	1	0.1
S3	3e-4	2	0.1
S4	6e-4	1	0.3
S5	1e-4	1	0.05
S6	1e-4	0.75	0.1

Domain adaptation hyperparameters sets
(*) Values corresponding to Table 1

Figure 6. DA relative rotation hyper-parameters tuning scores (above) for each combination of hyper-parameter in the table (below).

on data for the auxiliary task is different. In both versions of this variation, color permutation is applied on both modalities, either independently or jointly.

In the first variation, the pretext task is a binary classification. The number of output neurons *n=1* in this case, and it represents the number of binary output features to be classified. The obtained score goes through an extra Sigmoid layer for normalization, and it is then fed to the Binary Cross Entropy.

In the second version of the variation, the pretext task has a number of output classes *n=6*. Here softmax and Cross Entropy are used as in previous experiments. The scores of the two experiments, performed with the starting values for hyper-parameters shown in *table 1*, are available in figure 7.

Algorithm 1 RGB-D Domain Adaptation

Input:
Labeled source data set $S = \{((x_i^{sc}, x_i^{sd}), y_i^s)\}_{i=1}^{Ns}$
Unlabeled target data set $T = \{((x_i^{tc}, x_i^{td})\}_{i=1}^{Nt}$
Output:
Object class prediction for the target data $\{\tilde{y}_i^t\}_{i=1}^{Nt}$

procedure TRAINING(S,T)
  Get transformed set $\hat{S} = \{((\hat{x}_i^{sc}, \hat{x}_i^{sd}), z_i^s)\}_{i=1}^{\hat{N}s}$
  Get transformed set $\hat{T} = \{((\hat{x}_i^{tc}, \hat{x}_i^{td}), z_i^s)\}_{i=1}^{\hat{N}t}$
  for each epoch **do**
    for each iteration **do**
      Load a batch B_s from S
      Compute loss Λ_s of B_s on M
      Load a batch B_t from T
      Compute loss Λ_t of B_t on M
      Load a batch $\hat{B}_s$ from $\hat{S}$
      Compute loss $\hat{\Lambda}_s$ of $\hat{B}_s$ on P
      Load a batch $\hat{B}_t$ from $\hat{T}$
      Compute loss $\hat{\Lambda}_t$ of $\hat{B}_t$ on P
      Update M weights with $\nabla\Lambda_s\ \nabla\Lambda_t$
      Update P weights with $\nabla\hat{\Lambda}_s\ \nabla\hat{\Lambda}_t$
      Update E weights with $\nabla\Lambda_s\nabla\Lambda_t\nabla\hat{\Lambda}_s\nabla\hat{\Lambda}_t$
    TEST(T)
procedure TEST(T)
  for each x_i^{tc}, x_i^{td} in T **do**
    Compute $\tilde{y}_i^t = M(E(x_i^{tc}, x_i^{td}))$

M = Main task, P = Pretext task, E = Feature Extractor

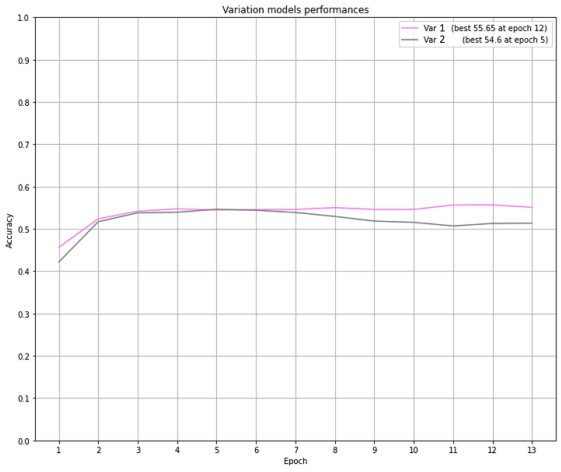
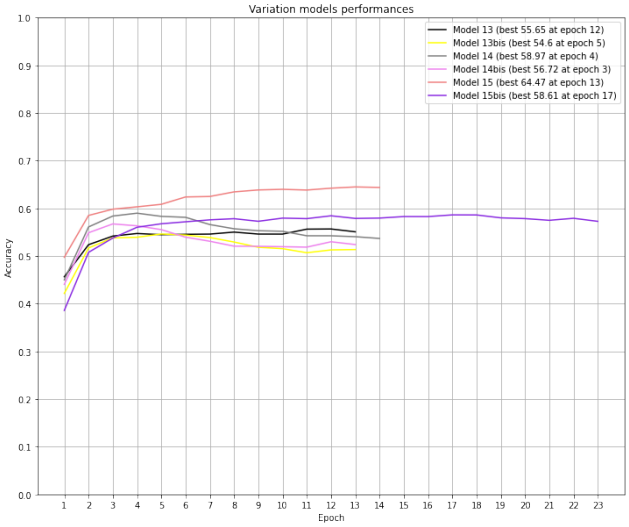


Figure 7. Scores of the variations models: trends of the accuracies of the two variations.

Subsequently a hyper-parameter tuning phase is performed. The criteria for the hyper-parameters choice is analogous to the one for the previous experiment. As shown in figure 8, each model stops when it satisfies the early stopping condition in order to save time. In this case the first variation is simply named with the hyper-parameters set number, while the second variation has the "bis" suffix in the legend. It is important to analyse another aspect of



Set \Params	LR	lambda1	lambda2
S13*	3e-4	1	0.1
S14	3e-4	0.5	0.1
S15	3e-4	1.5	0.1

DA variation hyperparameters sets
(*) Values corresponding to Table 1

Figure 8. DA variation hyper-parameters tuning scores

this experiment, that is the trend of the losses of the two versions. The reason why a second variation is proposed comes from the losses of the first one; as it can be seen in figure 9, the loss of both the source and target transformed data on the pretext task remain constant for the entire training time. This led us to think that the task might be too hard to be learned for the network, therefore it was interesting to explore a second version of this solution. Differently from the first one, that is a relative task, this second one is absolute and its losses decrease as expected, forming a learning curve. Moreover, it appears that higher values for the weight of the pretext task yield better accuracies even if the loss trend of the pretext task would suggest the opposite.

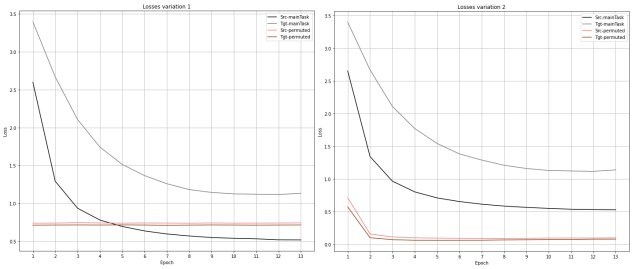


Figure 9. Variations losses comparison.

4.4. Results

Concerning the qualitative results of the realized experiments, table 2 reports the comparison between our scores and the ones obtained in the reference paper separately for each experiment.
The source only experiment is a complete success, having similar or higher accuracy for many hyper-parameter sets, one of which with the same configuration of the reference paper.
The domain adaptation with relative rotation instead didn't reach the desired accuracy, but still presents a huge improvement of around ten percentage points with respect to the base model. This clearly shows that the used pretext task achieves its purpose of reducing the domain shift by aligning the distributions of features of the two domains.

Finally, the two variations of the DA task using color permutation affect the results in two substantially different ways. The accuracy scores achieved using the first variation as self supervised task are considerably higher than the ones obtained using the other DA techniques. Moreover, it appears that the higher the weight of this pretext task the better the accuracies even if the loss trend of the pretext task would suggest the opposite.

In order to further investigate, we analysed the behavior of loss and accuracy for the pretext task alone in the first few epochs. In fact, given the big size of the adopted datasets, it could be possible for a task to reach convergence within the first epoch and remain stable from that point on. If this was the case, we would not have been able to detect it by checking the loss at the end of the epoch, as done previously. Therefore, we collect loss values for every mini-batch, as shown in figure 10. Interestingly, the trend of the loss is constant around its initial value before any training, if we don't consider the expected oscillations that come from sampling with such a high frequency within epochs. This is also reflected in the accuracy of the pretext task, that never improves from a random guess (figure 11).
This means that the model never learns how to solve the task, despite having some clear impact on the learned features and performances.

On the other hand, the second variation achieves results that are perfectly in line with the obtained relative rotation scores.

It must be observed that due to the huge requirement in terms of training times, it has not been possible to do multiple runs for all the experiments and thus report the values as mean and standard deviation. Being the reported values deterministic, it must be considered that a certain variability exists and could affect the scores of some percentage points.

In order to effectively visualize the results of the adopted DA techniques on the domain shift, t-SNE[8] has been applied. The two datasets are sampled with 250 points, while preserving class distribution. Then, this selection of data points goes through the best performing model of each approach, so that the results of t-SNE can be compared. As we can see in figure 12, all these models partially diminish the differences in terms of data distributions between source and target with respect to the baseline, with the relative rotation and the first variation being slightly better than the last. Nonetheless, variation 1 seems to have a different

Experiment	Our results	Ref. results*
Source only	48.99% (set8)	47.33%
DA - Relative rot.	58.60% (set 2)	66.68%
DA - Variation 1	64.47% (set 15)	N/A
DA - Variation 2	58.61% (set 15bis)	N/A

Table 2. Results comparison.
(*) Results achieved in the reference paper[6]

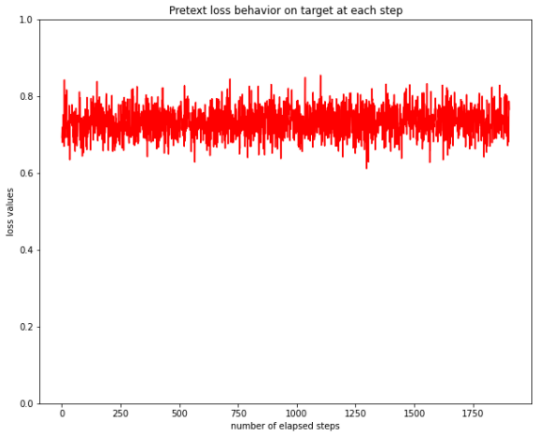


Figure 10. Loss values for the pretext task for each mini-batch over the first 3 epochs.

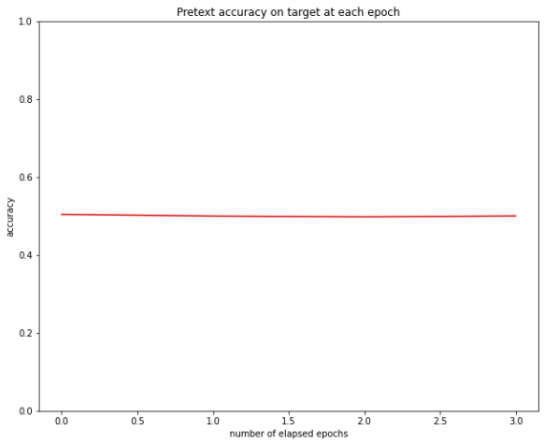


Figure 11. Accuracy for the pretext task over the first 3 epochs.

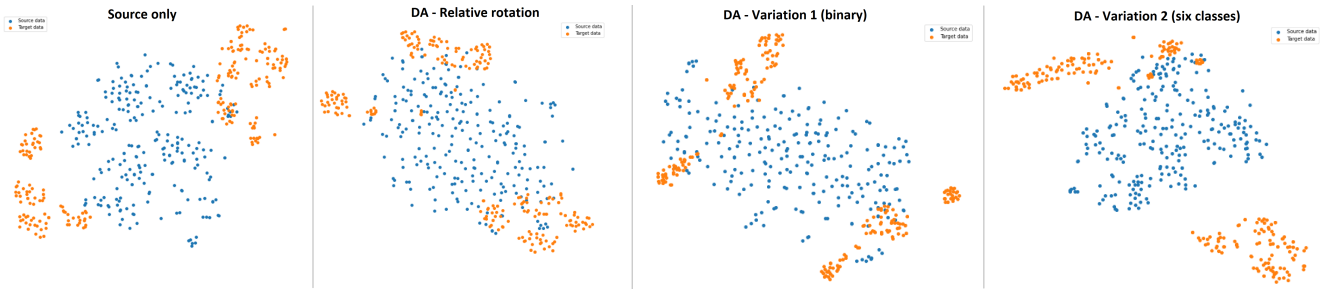


Figure 12. t-SNE[8] visualization of the features extracted from the last hidden layer of the main head M. Blue dots: source samples; Orange dots: target samples. The features displayed are extracted from the same set of images and shown for all the four experiments.

alignment in term of features for the two domains, despite having a better overlap than the baseline.

It should also be noted that the obtained representation varies significantly depending on the chosen random samples. Ideally, this problem could be reduced by using a bigger portion of the datasets, but it could not be performed due to hardware limitations.

5. Conclusions

The report proposes a new approach to tackle the problem of domain shift between the source and the target domains. It has therefore explored different DA techniques, which leverage both modalities of RGB-D images to solve auxiliary tasks. We have shown experimentally that the relative binary variation is the self-supervised task wherewith our implementation performs the best, reaching the maximum test accuracy equal to 64.47% which is close enough to the obtained score for relative rotation in the reference paper. Further research would be needed to understand the strange behavior of its learning process, and to ensure that the model doesn't learn shortcuts [3].

Despite not reaching the desired capability of solving said task, we have verified once more that relative tasks are powerful tools and can perform better than their absolute counterparts.

To conclude, all the self-supervised tasks actually improved the model performances with respect to the source-only baseline, and it can be safely stated that the proposed RGB-D DA techniques are at least partially reducing the domain gap.

References

[1] A. Aakerberg, K. Nasrollahi, and T. Heder. Improving a deep learning based rgb-d object recognition model by ensemble learning, 2017.

[2] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database, 2009.

[3] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction, 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[5] H. Lee, S. J. Hwang, and J. Shin. Rethinking data augmentation: Self-supervision and self-distillation, 2019.

[6] 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.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*, pages 8024–8035, 2019. editor: H. Wallach and H. Larochelle and A. Beygelzimer and F. d'Alché-Buc and E. Fox and R. Garnett.

[8] L. van der Maaten and G. Hinton. Visualizing data using t-sne, 2008.