Unsupervised Domain Adaptation through rotation as regression for RGB-D Object Recognition

Marco Gullotto
marco.gullotto@studenti.polito.it

Silvia Giammarinaro
silvia.giammarinaro@studenti.polito.it

Abstract

The work of Loghmani et al. [1] aims to teach the network the relationship between RGB and depth images belonging to two different domains, to weaken the effects due to the domain shift. For this reason, this self-supervised task is cross-modality because a random rotation is applied to both modalities and we want to determine its relative transformation. Starting from the work done by Loghmani et al. [1], we replicate some of their experiments on unsupervised domain adaptation for the recognition of RGB-D objects using the synROD and ROD datasets. After analyzing the results obtained, we implement a new architecture with different pretext tasks. The variations proposed by us consist of an absolute and a relative rotation. In the relative implementation we choose two random angles between 0 and 360 degrees to rotate both RGB and depth images independently: the goal is to predict the relative rotation between the images as in [1] in order to find geometrical relationships between them. For better performance we have also implemented the absolute task. In this one we choose a random angle and rotate the RGB and depth images of that angle: in this case CNN must predict the rotation angle. The project's code is uploaded in this GitHub repository.

1. Introduction

The algorithm that we have to replicate belongs to the class of domain adaptation algorithms. Domain adaptation is a technique used in deep learning to be able to handle different data distributions, between train and test set, and tasks. In this scenario, we define the synROD dataset as source domain $\mathcal{D}_S$, and ROD $\mathcal{D}_T$ as target domain. Before we begin to analyze the network architecture, it is worth explaining why we are using RGB-D images. We benefited from the work done by Eitel et al. [2] to manage the two data distributions. The RGB-D Object Dataset (ROD) incorporates the challenges faced by a robot into a real-life application and provides a useful tool for validating object recognition algorithms. For this reason, this dataset not only contains standard RGB images provided by a digital camera, but we also have information about the spatial properties of the object. Depth knowledge is very useful, especially in a robotic setting, and it is obtained by using a Kinect camera (like that of an Xbox console).

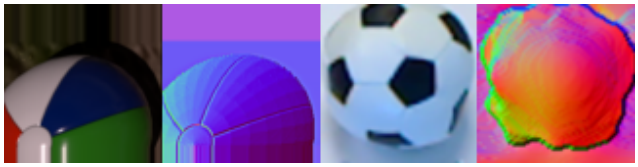


Figure 1: Samples representing the class Ball taken from the dataset synROD (left) and ROD (right) before applying the transformations needed for the pretext task.

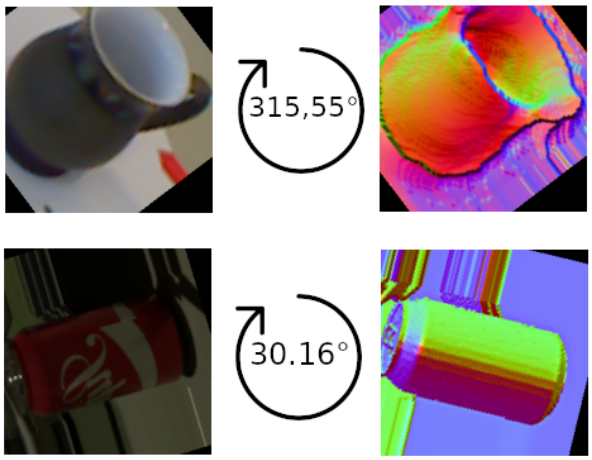


Figure 2: Examples of the absolute and relative rotation task. In the first line we report a sample taken from the ROD dataset ready for the absolute rotation task. In the second case we can see a sample from the synROD dataset used for the relative rotation task.

For this reason, in recent years, the ROD has become the state-of-the-art standard in the robotics community for the task of classifying objects. Further explanation of the

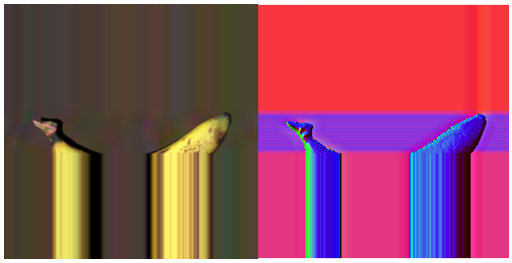


Figure 3: Image stretch maintaining the object ratio of a sample taken from the synROD dataset

dataset will be provided in the dataset section. We therefore define two different tasks to be achieved: the main classification task $\mathcal{T}_M$ to recognize the object (i.e. lemon, ball, banana), and the pretext task $\mathcal{T}_P$ to learn the absolute and relative rotation between the RGB image and the depth image. The task proposed in paper [1] is unsupervised because we did not consider the target labels during the training phase. Considering the synthetic-to-real domain shift the goal is to reduce the gap between these two distributions using domain adaptation algorithms. Introducing an artifact, we are able to train the CNN using not only the synROD dataset, but also the ROD one. In the original paper, the possible rotation angles were four: 90, 180, 270 and 360 degrees. Our variation consists in implementing a rotation as a regression algorithm to manage all the possible rotations of the images, from 0 to 360 degrees. An example of this operation is shown in figure 2. To implement this new model, we modify the network architecture, adding two streams for the pretext task in order to teach the network to recognize sine and cosine of its absolute and relative rotation angle. The results obtained using this pretext task show us how an artifact helps the objects recognition, as reported in the results section. Moreover, with this project we want to demonstrate that choosing the best pretext task is essential to help CNN extract domain-invariant features. To do this, we present the implementation of two new pretext heads: one for predicting the absolute rotation and one for the relative one. Our relative rotation method is an extension of the one reported in [1], while the absolute rotation task is not designed to deal with multi-modalities problems. Moreover, we report, method section, some experiments taken from [1]. In the first series of trials, the pretext head is not considered. Instead, we consider only the source domain and a single modality at a time to define our baselines (only RGB and depth experiments). Then we train the CNN with the concatenation of the features obtained by both modalities in an end-to-end fashion (source only RGB e2e). We therefore introduce the pretext head, useful for enforcing the main classification task, as reported in [1], [2].

2. Related work

In this section we report all the useful resources used for solving our domain adaptation problem. Starting from the network used, the preprocessing phase and a loss to handle domain-shifts. Eitel et al. [2] developed a CNN with two branches to manage the modalities RGB and depth. At each iteration the network is fed with a batch of twin images, as in fact we have two photos for the same object: one with the RGB format and one with the depth. Each stream takes as input one modality, the network extracts the features of the pictures and then the predictors are combined together as output. In addition, this work introduces a way to stretch the image in the desired input format of the network while maintaining the ratio between the objects. This stretching process is illustrated in figure 3. In fact, if we have a rectangular-shaped image and we try to resize it to a square shape ignoring the ratio of the original object, this could adversely affect the performance of the network. To overcome this problem, the approach chosen is the following: scale the longest side of the original image to 256 pixels, then tile the borders of the longest side along the axis of the shortest side. The method used for color encoding the depth images is introduced in [2]. Each pixel in the image is mapped to a range of colors between red and blue. Red symbolizes that the object is close to the sensor, while blue indicates that it is far away. This technique is used to distribute depth information on all three RGB channels. After defining the architecture and the output of the first part of the network, we gather from [3] the split into two task, main head and pretext head. The main stream predict the objects class. In the pretext task we apply a transformation (rotation) to the images and we see that the CNN predicts these transformations. This process, linked to the main task, is able to make the features extraction independent of the domain. By recognizing the different orientation of the images, CNN is able to recognize the salient features also in the target images. Moreover, we use a loss entropy on the target domain's labels as reported in [4]. This loss is introduced to reinforce the predictions made by the main task head. Furthermore, some layers of the network are initialized using the Xavier initialization method as mentioned in [5].

3. Dataset

Since a huge amount of data is needed to train the network, we used the SynROD and ROD dataset. These two datasets contain photos of objects commonly used in homes, offices, etc ... The object models for synROD have been selected so that each of them belongs to one of the fifty-one categories defined by ROD. All the photos were taken using synthetic objects available on websites such as 3D Warehouse and Sketchfab and a ray-tracing engine such as

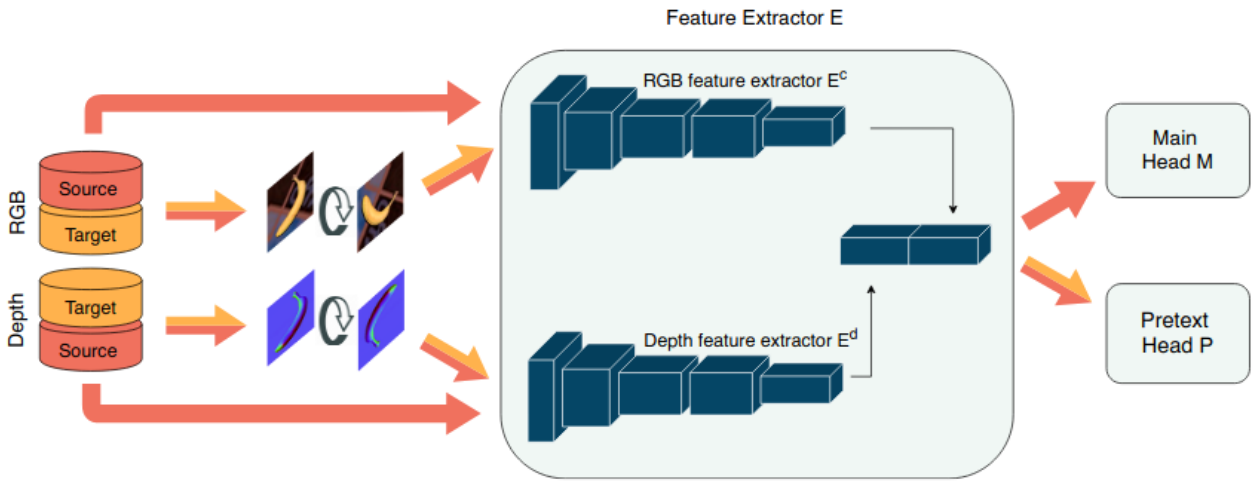


Figure 4: Net's architecture used in the Loghmani et al. paper. Our implementations modify the pretext task's architecture. Our architecture of the new pretext task is reported in figure 5

Blender. In this way they were able to simulate different types of lights, shadows, angles and distances from objects. It was thus possible to create a huge dataset with over fifty thousand photos. Having said that, not all categories have enough images to be properly learned by CNN so, to avoid this problem, we have reduced the number of categories to forty-seven, so we have discarded all the photos of the "lime, onion, peach, tomato" categories. Looking at some of these photos, we immediately noticed that some images have very anomalous lines around the edges. Furthermore, to obtain natural scenes, the background of all the photos is randomized using the images of the MS-COCO dataset: this will help the CNN to become more robust. The photos available from the SynROD datasets to train the main network stream are approximately 37500. As regards ROD, the useful images for both the adaptation domain and for the test part are approximately 32500. Altogether we have more than 70000 photos occupying about 15 GB of disk space. The first idea put in place was to try to load all the data from the disk into RAM using an ad-hoc dataset class. Using this class, all images were also preprocessed using the technique described in implementation details section. The main problem was Colab's limitations. In fact, having only 12 GB of available RAM, it was not possible to upload all the photos. Initially, we therefore thought of loading only one of the two datasets into the main memory however, in doing so, the complexity of the code increases considerably, while vice versa the performance does not improve much. So, due to Colab's limitations, every single image is loaded from the disk every time. Fortunately, this is not a big problem either in the training phase or in the test phase. The tesla k80 GPU, the one made available by Colab, is really fast and allows us to train the network in a relatively short period of time.

4. Method

4.1. Network architecture

Initially, the network architecture is the same as that used in the Loghmani's paper, in particular the one shown in figure 4. As shown in the picture, RGB and depth photos are sent through two features extractors E^c and E^d. These networks are made up of the first eighteen convolutional layers of a ResNet18. In other words, E^c and E^d are used, respectively, to process the RGB and depth images. The result of these two networks is then chained along the channel dimension to compose the final RGB-D feature. In the Loghmani's research [1], the global average pooling layers are also removed to try to exploit all the spatial features generated by these two feature extractors. At this point, these predictors are provided as input for both the main head M and the pretext head P. Now, both branches are used to solve a classification task, but the network structure of the two tasks is different.

- For the main task, the choice is to replicate, more or less, the layout of a Resnet. First a global average pooling layer, then a fully connected layer with thousand neurons and finally a last one fully connected layer with "Number of different classes" neurons. The first fully connected layer uses batch normalization and the ReLU activation function, while the last one uses a softmax activation function. The dropout is also used between these two layers, which will be discussed in

more detail in the next section.

- For the pretext task, to understand the relative rotation between the two photos, all spatial characteristics must be exploited. So, instead of introducing a global average pooling layer, we use two 2D convolutional layers the first kernel of size 1 x 1 and one hundred neurons the second still with one hundred neurons but with a kernel size of 3 x 3 and a stride of 2. Finally as in the main task, an FC (100) and an FC (4) are introduced. The last layer has four outputs because it represents the four possible 90 degree rotations. In addition, this time the first fully connected layer uses batch normalization and the ReLU activation function, while the last one uses a softmax activation function.

Once the structure of the network has been explained, from the point of view of optimization, the problem becomes minimize:

$$\mathcal{L} = \lambda_p \cdot \mathcal{L}_p + \mathcal{L}_M + \alpha \cdot \mathcal{L}_{en}$$

where:

- $\mathcal{L}_M$ is the cross-entropy loss of the main task. The cross-entropy loss is very similar to the negative log-likelihood loss, and it is commonly used to train a classification problem with C classes. This can be described by the expression:

$$loss(x, class) = -log(\frac{e^{x[class]}}{\sum_j e^{x[j]}})$$

- $\mathcal{L}_p$ is the loss of the pretext's task and is still a cross-entropy one. This time the cost function is composed of two terms: the first is represented by that produced starting from the source data, the second is instead that produced starting from the target photos. λ_p is a weight to regulate the contribution of this term. In [1] is set equal to 1.

- $\mathcal{L}_{en}$ this cost is the one introduced by the Pietro Morerio et al. [4]. Since we can predict the label of the target but we don't have the label to compute, for instance, a cross-entropy loss, we replace it with the entropy one:

$$\mathcal{L}_{en} = -\sum_{x \in batch} < f(x), log(f(x)) >$$

where $f(x)$ represents the rotation predictions of the network for the x image. This term is also weighted with a term α that regulates the contribution of this loss. In this case in according with Loghmani α is set to 0.1.

4.2. Rotation as regression

Before starting with the new task we introduce two different definitions of rotation used in our variation of the algorithm.

- Absolute rotation: we choose a random angle and we rotate both the RGB images and the depth with respect to it. We perform the rotation of the images every time we have to retrieve them from the dataloader, therefore the rotation for a given input is different with each iteration;

- Relative rotation: we generate two random angles to rotate the two images differently and independently, then compute the difference between the RGB angle and the depth angle as in [1].

That said, the CNN implementation doesn't change because we only change the way we calculate labels with regards to the rotation angles of the images. From now on we consider the case of relative rotation because the algorithm is similar to the absolute one.
Predicting the relative rotation between RGB and depth images allows the network to learn significant features of both the source and target data. Indeed, as shown in Loghmani et al. [1] result, this type of domain adaptation significantly improves CNN performance. But rotating the image 90 degrees multiple can be a little reductive because in this way only three rotations are applied: 90, 180 or 270 degrees. So, after a few epochs, all the possible combinations of relative for an RGB and a depth photo rotations end. In order to have more variations and further improve the result obtained, we can try a different approach based on the same idea. Instead of rotating the image only in multiples of 90 degrees, we tried to rotate the image by an arbitrary angle between 0 and 360 degrees. In this way, we can achieve infinite relative rotation between the RGB image and the depth image. This intuition introduces two new problems: first of all, we don't have a "perfect" rotation of the image. For example, if we rotate of 22 degrees an image, we will have black space on the corners. However, this problem can be easily solved. So, all photos are resized to 256 pixels and then center cropped. In this way, part of the black area is eliminated. The new task is no longer a classification task but a regression one. So, we can no longer use a cross-entropy loss but we must use a different one (this topic will be further explored). To solve this problem we have also changed the structure of the network (figure 5). It is very similar to the old one, in fact, we still use the "Conv (1 x 1,100), Conv (3 x 3, 100), FC (100)" layers (with ReLU activation function) but we changed the last one to a " FC (1) " needed for regression task. The output of this branch will be directly the relative rotation angle in degrees. A very common loss when dealing with a regression problem is the

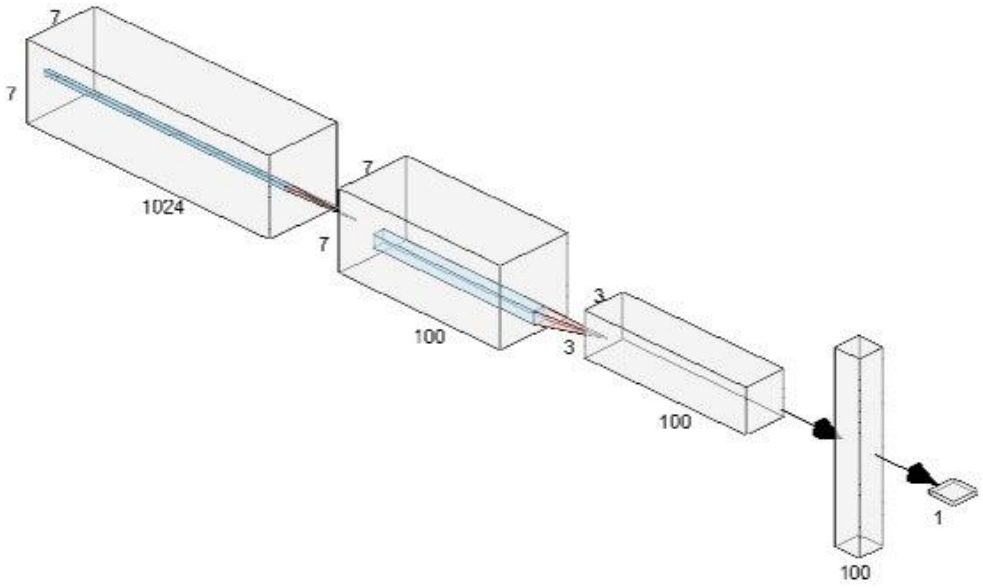


Figure 5: The pretext task is composed of two identical streams to predict the sine and the cosine of the angles. The figure show the architecture of one of the two.

so-called mean sum of squared error (MSE).

$$loss(y_n, x_n) = (y_n - o_n)^2$$

So, the goal we want to achieve is to find some weights w that minimize the distance between our prediction and the real label y. Given this loss function, to update the weights of the last fully connected layer, it is necessary to compute a gradient descent task:

$$\frac{1}{2}\frac{\partial \sum_n (y_n - o_n)^2}{\partial w_i} = \sum_n (y_n - o_n) \cdot (-\frac{\partial o_n}{\partial w_i}) = \sum_n (y_n - o_n)(1 - o_n)x_n^i$$

and finally update the weight:

$$w \leftarrow w + \eta(y_n - o_n)(1 - o_n)x_n$$

This was our basic idea which, however, works quite well. The problem is that using this loss function if the net, for example, predicts an angle of 360 degrees and the real label is 0 degrees the error should be null but: $(0 - 360)^2 >> 0$. The problem is that there is no convex loss functions suitable for this task. Our proposal to solve this problem is as follows: if we denote by ϑ the angle of the relative rotation between the RGB image and the depth one, we can compute for each angle:

$$\vartheta \rightarrow (sin(\vartheta), cos(\vartheta))$$

The network this time does not predict the degrees of the angle but the corresponding sine and the cosine value. Given these two values we can calculate the relative rotation angle as:

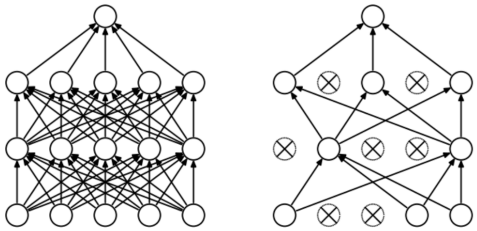
$$output \leftarrow atan2(sin_{predicted}, cos_{predicted})$$

with a loss function:

$$\mathcal{L}_{regr} = \frac{1}{2}((sin_{pred} - sin(y))^2 + (cos_{pred} - cos(y))^2)$$

The last step is to add a new branch to the network. The new branch will be the same as the old "Conv (1 x 1,100), Conv (3 x 3, 100), FC (100), FC (1)" but, this time, the first branch will predict the sine value and the second the cosine value. The representation of the branch can be seen in figure 5.

To improve the final result we also used batch normalization and dropouts. Even if we use normalized input data of the network after going through a convolutional layer,



(a) Standard Neural Net (b) After applying dropout.

Figure 6: Dropout

the result will no longer be normalized due to non-linearity. Ideally, normalization should be conducted over the entire training set but, since we deal with SGD, we will use a normalization step that fixes the means and variances of the inputs at each layer. So at each step for each input batch, we can compute the empirical mean:

$$\mu = \frac{1}{batchsize} \sum_{i}^{batchsize} x_i$$

and the empirical variance:

$$\phi^2 = \frac{1}{batchsize} \sum_{i}^{batchsize} (x_i - \mu)^2$$

finally:

$$\widehat{x} = \frac{x - \mu}{\phi}$$

Dropout, on the other hand, is another important technique that helps the network to intelligently update the weight of the fully connected layer. A very common phenomenon that occurs in CNN is that some weights of the fully connected layers are too much greater than others. To avoid this problem there is the regularization term, but often it is not enough to guarantee a good weight distribution. This means that the decision made by the network will be based only on a few neurons. Hence, the risk is to lose part of the discriminative power of the network. So, the dropout (figure 6) works as follows: at each step, some neurons are "turned off" with a certain probability p. The network has yet to predict the new angle, so in turn it is forced to use all the neurons in the network. Furthermore, using this technique we create a sort of ensemble effect that also helps prevent overfitting problems. When dealing with a huge network, it is often suggested to use p = 0.1 to prevent the network from slowing down too much, but since we have enough computational power, we keep it at 0.5. From an optimization point of view, the new problem becomes:

$$\mathcal{L} = \lambda_p(\mathcal{L}^S_{sin} + \mathcal{L}^S_{cos} + \mathcal{L}^T_{sin} + \mathcal{L}^T_{cos}) + \mathcal{L}_M + \alpha \cdot \mathcal{L}_{en}$$

Where the α remains equal to 0.1, because nothing has changed from that point of view. Instead, the lambda value decreases from 1 to 0.5: this is because the regression loss is greater than that of classification, so, to allow the network to learn significant features from the pretext task we reduce the value of λ_p. 0.5 is the best value found after different trials with different value of λ_p.

5. Experiments

5.1. Implementation details

For the development of our experiments, we consider the algorithm made by Loghamni et al. [1]. For each variation, we define four sets, $S, T, \widetilde{S}, \widetilde{T}$. The set $S = \{((x_i^{sc}, x_i^{sd}), y_i^s)\}_{i=i}^{N_s}$ contains the samples and the labels of the synROD dataset, which are used to compute the cross-entropy loss $\mathcal{L}_m$. The set $T = \{(x_i^{tc}, x_i^{td}))\}_{i=i}^{N_t}$ is unlabeled and it contains the images taken from ROD. The pair (x_i^{*c}, x_i^{*d}) simbolizes the RGB and depth images. The set T is used to compute the entropy loss and test our models. Futhermore, the sets $\widetilde{S}, \widetilde{T}$ are the sets in which we apply the transformation to the photos, absolute or relative rotation, and their labels are composed by the angle and the class. The only difference reported in the algorithm is the introduction of the two MSE losses $\mathcal{L}_{cos}$ and $\mathcal{L}_{sin}$, which are computed with $\widetilde{S}, \widetilde{T}$ respectively.

Most of the hyperparameters are taken from the Loghmani's paper [1] but now they will be reported in a short summary in which the parameters we used will be added in full. CNN is trained using the SGD optimizer with momentum 0.9, learning rate 3e-4 and batch size 32. Due to Colab limitations, we cannot use batch size equal to 64, but to maintain the same hyperparameters we use gradient accumulation. Not quite like having 64 photos in batch, but it's still a good compromise. The weight decay term is set to 5e-2 and the dropout is 0.5. As mentioned above, we include entropy-minimization with α equals to 0.1 as specific regularization for DA in both regression and classification tasks. While the cross-entropy loss in the pretext task is multiplied by 1, the regression loss in the second part is multiplied by 0.5 (for both the absolute and relative case), as explained above. The weights of the two "Resnet 18" are initialized using the values of the pre-training network trained on ImageNet. To boost the performance of our baseline (RGB only or depth only) we also initialized the *FC(1000)* using pre-trained weights. Unfortunately, this is not possible for the domain adpatation case because the structure of the net is different from the basic one. Then the rest of the network is initialized with Xavier initialization. This means that the weights are chosen from a uniform distribution in [-1.1] and then scaled by $\frac{1}{\sqrt{n}}$. Where n is the number of the network input connections at a layer. This should initially help the stability network, especially if we use the Relu or

Algorithm: RGB-D Domain adaptation with rotation as regression

Data: Labeled source domain
$S = \{((x_i^{sc}, x_i^{sd}), y_i^s)\}_{i=i}^{N_s}$
Unlabeled target domain
$T = \{(x_i^{tc}, x_i^{td}))\}_{i=i}^{N_t}$

Result: Object class prediction $\{\widehat{y}_i^t\}_{i=1}^{N_t}$

1: **procedure** TRAINING(S, T)
foreach *iteration* **do**
Get transformed set
$\widetilde{S} = \{((\widetilde{x}_i^{sc}, \widetilde{x}_i^{sd}), z_i^s)\}_{i=i}^{\widehat{N}_s}$
Get transformed set
$\widetilde{T} = \{((\widetilde{x}_i^{tc}, \widetilde{x}_i^{td}), z_i^s)\}_{i=i}^{\widetilde{N}_t}$
Load batches from S, T, $\widetilde{S}$ and $\widetilde{T}$
Compute $\mathcal{L}_M$ from S
Compute $\mathcal{L}_{en}$ from T
Compute $\mathcal{L}_{cos}^S, \mathcal{L}_{sin}^S$ from $\widetilde{S}$
Compute $\mathcal{L}_{cos}^T, \mathcal{L}_{sin}^T$ from $\widetilde{T}$
Update weights of M from $\nabla\mathcal{L}_M, \nabla\mathcal{L}_{en}$
Update weights of P_{cos} from $\nabla\mathcal{L}_{cos}^S$, $\nabla\mathcal{L}_{cos}^T$
Update weights of P_{sin} from $\nabla\mathcal{L}_{sin}^S$, $\nabla\mathcal{L}_{sin}^T$
Update weights of E from $\nabla\mathcal{L}_M, \nabla\mathcal{L}_{en}, \nabla\mathcal{L}_{cos}^S, \nabla\mathcal{L}_{sin}^S, \nabla\mathcal{L}_{cos}^T, \nabla\mathcal{L}_{sin}^T$
end
2: **end procedure**
3: **procedure** TESTING(T)
foreach (x_i^{tc}, x_i^{td}) *in* T **do**
Compute $\widehat{y}_i^t = M(E(x_i^{ct}, x_i^{dt}))$
end
4: **end procedure**

TanH functions according to [5]. The input to the network is RGB synchronized and depth images preprocessed as follows:

- The images (RGB and depth) are resized from the initial dimension into 256 x 256 pixels. At this point, the two images are flipped horizontally with a probability of 0.5. This is a kind of data-augmentation technique.
- The photos are rotated by random angles, the difference between these two angles represents our relative rotation. While in absolute rotation, the two images are rotated by the same angle random angle.
- Images are randomly cropped when rotated a multiple of 90 degrees, and center cropped when rotated by an arbitrary angle. In the first case, we use a random cropping to add diversification between each epoch, therefore, again, a sort of data-augmentation.

5.2. Results

Table I shows the result obtained trying to replicate the Loghmani et al. [1] experiments and results obtained using absolute and relative rotation as regression. Let's begin with some baselines to try to understand if the methods we used work properly. Initially, we provide the network with only one image at a time to work on, using only RGB photos. The result obtained is quite good with an accuracy score of around 50%. On the other hand, if only depth images are used, the result obtained is really poor: accuracy of 10%. By combining RGB and depth photos, and training the network in end-to-end way, the result obtained is surprisingly worse than that resulting from the experiment carried out using only RGB. For this first set of experiments we only used the source domain $\mathcal{D}_S$. The results obtained until now are very similar to the ones obtained in [1] but the same behaviour does not occur when we try to use their domain adaptation technique. After three different trials, the score we got is in fact equal to $57.6\% \pm 1.2\%$, while that obtained by Loghmani and his team is around 66%. Nothing explicitly justifies this gap, but the result is still a confirmation that this domain-adaptation technique works properly. In fact, we have a clear improvement if we consider RGB only and the RGB-D e2e experiments. At this point we can report the result obtained by the rotation as a regression variant. The accuracy obtained with this method is also greater than that obtained by Loghmani's. After three trials conducted using relative rotation as pretext task, the score obtained is $59.0\% \pm 0.8\%$, which means that accuracy also fluctuates less than before. It's not a big difference, but this confirms that rotating the image representation of an arbitrary angle introduces more variability into the dataset and helps feature extractors to get more meaningful features. The result obtained using absolute rotation is less than we expected. The accuracy score, after three trials, is $55.0\% \pm 0.9\%$, therefore worse than both techniques with relative rotation. As already mentioned, this may depend on the fact that absolute rotation is not designed for multi-modal problems like this. Having said that this method also exceeds the result obtained from the source only and the e2e networks as well, also this domain adaptation technique can help CNNs to learn more significantly features so that the domain shift effect is less obvious. The values listed in Table I show the standard deviation only for domain adaptation techniques because they are the ones that from our point of view are really interesting. As for the baseline methods, it is not particularly important to have a precise representation, because they represent only a starting point to be overcome.

6. Conclusions

The result we have obtained is in accordance with to the one reported by the Loghmani et al [1]. Of course there

Method	Modality	SynROD → ROD
Source Only	RGB	51.4%
	Depth	13.1%
	RGB-D e2e	47.9%
Loghmani et al. impl.	RGB-D	57.6% ± 1.2%
Abs. Rotation as regresion	RDB-D	**55%** ± 0.9 %
Rel. Rotation as regresion	RDB-D	**59%** ± 0.8%

Table 1: *Accuracy % of several methods for rgb-d domain adaptation on synthetic-to-real shift SynROD to ROD*

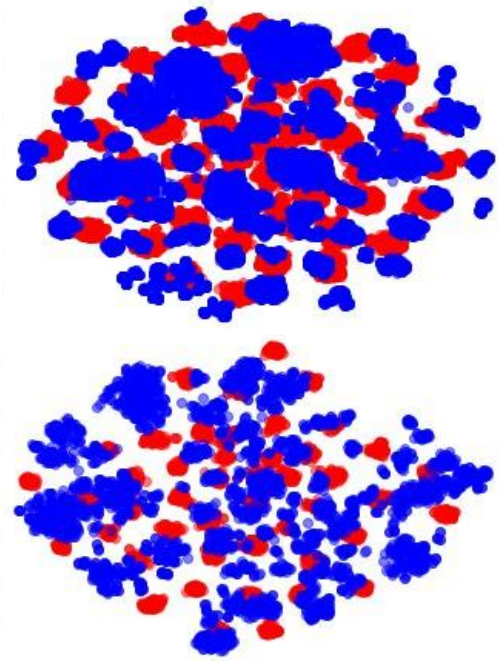


Figure 7: t-SNE visualization of the SynROD and ROD features extracted from the last hidden layer of the main head M. Red dots: SynRod samples; blue dots: Rod samples. When adapting the two domains with our method (below), the two distributions align much better compared to the one without domain adaptation (above)

is a certain gap between their performance and ours, but it is still evident that the domain-adaptation method from the synthetic to the real world that they introduced works correctly. To underline this fact, we also used the t-SNE algorithm in a part of the SynROD and ROD datasets to show that the two distributions are more aligned after applying the Loghmani's method. Having said that, it is relevant that our variation, that is rotation as regression, exceeds the result obtained with rotation as classification. The result is not particularly better, but since with the regression the images can be rotated by many different angles than before, this constitutes a sort of data-augmentation. Having said that, however, it should be reported that some of the tests we have done do not increase the overall performance.

- The first experiment was to change the color of the depth images into grayscale images. This idea came to our mind because the depth images of both SynRod and ROD don't have the same color distribution: synROD has colder tones and ROD more warm tones. This is due to the fact that the images are taken at a different distance from the object, as seen in [2]. The performance achieved using this technique does not improve at all performance, but we must emphasize that there has not been a big drop in the accuracy score. So, perhaps, further analyzes should be carried out in the future to go into this aspect.

- As mentioned by Eitel et al. [2] we tried to implement a data augmentation technique for the depth data in order to make the network more robust. The depth sensors are particularly affected by a non-negligible amount of noise which can be translated into black points near the edges of the objects. So to simulate this effect in the SynROD dataset which is synthetic, therefore it isn't influenced by the aforementioned problem, let's add a particular kind of noise called "pepper". This type of noise introduces random black points into the image, so in this way we can simulate what happens in the real world. Unluckily it is not so good perhaps because the black points are whole located in depth images and not only in the edges of the objects. Perhaps, using a more complex noise generation algorithm, higher performance can be achieved. Unfortunately, the one implemented by Eitel et al. [2] cannot be applied because the SynROD dataset has no noise pattern.

- The last attempt was to try adding another dataset called Autonomous-Robot-Indoor-Dataset (ARID) to the SynRod train set, but due to the limitation of Google Colab's memory, it was not possible to load it entirely, so we stopped this interesting process. As suggested in the last two points, we hope that further research will be conducted on the problem of the DA for RGB-D and that this will lead to better performance.

References

[1] Mohammad Reza Loghmani, Luca Robbiano, Mirco Planamente, Kiru Park, Barbara Caputo, and Markus Vincze. Unsupervised domain adaptation through inter-modal rotation for rgb-d object recognition. 2020.

[2] Andreas Eitel, Jost Tobias Springenberg, Luciano Spinello, Martin Riedmiller, and Wolfram Burgard. Multimodal deep learning for robust rgb-d object recognition. 2015.

[3] Jiaolong Xu, Liang Xiao, and Antonio M. López. Self-supervised domain adaptation for computer vision tasks. 2019.

[4] Pietro Morerio, Jacopo Cavazza, and Vittorio Murino. Minimal-entropy correlation alignment for unsupervised deep domain adaptation. *CoRR*, abs/1711.10288, 2017.

[5] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010.