Unsupervised Domain Adaptation for RGB-D Object Recognition

Matteo Bunino
Politecnico di Torino
matteo.bunino@studenti.polito.it

Christian Cancedda
Politecnico di Torino
christian.cancedda@studenti.polito.it

Giacomo Garrone
Politecnico di Torino
giacomo.garrone@studenti.polito.it

Abstract

Convolutional neural networks require enormous amounts of data in order to achieve state of the art performances, however the data labeling process is a considerably expensive operation, well known to be one of the main bottlenecks in supervised learning. This issue can be partially addressed by relying on synthetic generated data, whose labeling is obtained without supervision, having the drawback of being slightly different from real data.
Domain adaptation techniques allow to learn robust domain invariant features from synthetic data during training by reducing the domain shift between synthetic and real data distributions.
This solution can be as well applied in the robotic vision field in which images come in both RGB and depth modalities, requiring the design of ad-hoc inter-modal self supervised tasks.
In this paper we propose a novel self-supervised task for a domain adaptation algorithm aimed to reduce the domain gap between RGB-D synthetic and real images, tailored to leverage their intermodal relations. We compare it to the model presented in [1], which is our main reference, and to the source-only baselines to prove its effectiveness versus simple training without domain adaptation.
We also propose a different initialization schema from [1] that improves its accuracy by 1.32% and subsequently we try different approaches to enhance the performances of models, such as a data augmentation pipeline tailored to RGB-D data and a multi-task learning approach. Eventually, we propose an improvement of the model of [1] by trying different features extractors, increasing its accuracy by 3.42%.

1. Introduction

Convolutional neural networks rely on large amounts of image data in order to be trained properly. However, gathering and labeling real world images is a costly operation which is partially solved by synthetic generated data. This approach has the benefit that the examples can be labeled automatically without human supervision, but on the other hand it has the drawback that synthetic images are slightly different from their real counterpart.
Synthetic and real domains are said to be shifted among each other. Under the assumption that the two distributions from which data are generated only differ for a peculiar domain shift, domain adaptation techniques allow for the reduction of the effect of this factor. This was proposed for the first time in [2].

Robotics systems strongly rely on recognition tasks while interacting with the real world, therefore an effective solution that has been worked out is the use of RGB-D cameras that provide an additional information with respect to common cameras: the depth image. Depth images have a different nature from RGB ones: each pixel represents the distance from the camera, having a value which is not bounded in [0, 255] such as RGB pixels. The addition of depth images to the analyses allows us to take into consideration more information than with RGB images alone. The latter have a rich content of texture details, whereas depth images convey more information about geometry and shape of the object. It is important to deploy a suited approach to depth images, due to their difference with RGBs, in fact in [3] it was proved that a CNN trained from scratch on depth images can learn very different features from the ones learned on RGB images.

Our work is focused on Domain Adaptation techniques applied to RGB-D images, aimed to reduce the domain

shift between source (synthetic) and target (real) domains.
The current state of art is rich of proposed models that already employ domain adaptation, but most of them address this topic using RGB images only. In this case we want to tailor a domain adaptation model to the multimodal nature of RGB-D images. This requires to focus on the meaning of multimodal representation of an object.
A multimodal representation is the set of representations in different domains of the same object, that in this case it is composed by

- its visual appearance, that mapped on a 2D may loose much of its 3D information, namely the RGB modality.
- its geometric shape where only the 3D structure is considered, discarding all the texture information, which is the depth modality.

Based on this multimodal nature of data examples, we want to design a self-supervised task that leveraging the intermodal relations learns to abstract from the domain.
The idea behind self-supervised domain adaptation is that, along with a main classification task, is solved a self-supervised pretext task that implicitly regards intrinsic properties of the image and for which the label is not needed, hence it can be done on both source and target domain images.
As a result, the features learned after solving both tasks will be domain invariant and more suited to properly classify images from different visual domains. In the multimodal setting, exploiting the different modalities, we are able to design more robust pretext tasks that enhance intermodal relations as well.

Before delving into the implementation of a novel domain adaptation model, we repeated the results obtained in [1], trying to improve the presented results using some well known techniques in deep learning.
Our self-supervised proposed task is a task aimed at predicting the relative difference between the zoom that has been applied independently both to RGB and depth modalities.
In our work, we show that the choice of the pretext tasks is relevant in achieving better performances of the trained model.
Another important factor is the weight initialization of the model which we found to be a key point to obtain good results. At the beginning of this project this topic was not much considered, but reading and observing the effects of Xavier initialization on our model we understood how much influence in the training it can have.
Another aspect took into account in the optimization process is the design of a proper data augmentation pipeline with the aim of reducing overfitting, improving robustness and replicating the data augmentation step involving depth images, proposed in [4].
To furtherly optimize our models we tried different networks as features extractors and a multi-task learning approach inspired by [5].

2. Related work

One of the key references is the paper of Eitel et al.[4] which explores the possibility to use multimodal data for RGB-D object recognition. Furthermore, it proposed an intuitive data augmentation schema suited for depth images.
The leitmotiv of our work is the comparison with the model and the results proposed in [1]. In that paper the self-supervised task is implemented as the prediction of the relative rotation between RGB and depth modalities, previously independently rotated. This has inspired our work and its outstanding performances with respect to the current state of the art has encouraged us to propose an alternative that stems from it.

2.1. Usupervised Domain Adaptation

Methods for unsupervised domain adaptation in computer vision can be divided into three main classes. The first, aims to align source and target domains in some feature space. This is done by optimizing for some measurement of distributional discrepancy. One popular measurement is the maximum mean discrepancy (MMD) [6] but there exists other discrepancy-based methods as well [7, 8].
The second class contains methods based on adversarial learning. One of these, is the domain adversarial network proposed in Ganin et al. [9], in which the pretext task consists of a discriminator branch that has to classify the domain of the given image samples. However, in this case RGB-D data has not been used and the domain adaptation is instead achieved by means of a gradient reversal layer. The proposed layer backpropagates positive gradients, thus maximizing the discriminator loss that arrives to the feature extractors; this allows domain independent features to emerge. Similar works are [10], [11], [12].
The third class leverages the solution of a self-supervised tasks in parallel with the main recognition task. A self-supervised task has the benefit of being carried out without the need of labels, therefore it can be executed on both source and target examples. This allow to learn robust domain invariant features.
Although there exist examples of self-supervised domain adaptation applied to RGB-D data [13][14][15][16], they all lack something in what is our focus, namely adapting a source (synthetic) domain to a target (real) domain of RGB-D images, leveraging intermodal relations.
As far as we know, the only work addressing this issue is [1]. In this case, the pretext task consist in predicting the relative zoom of RGB and depth modalities. This enhances the multimodal relations because the pretext task learns that

when there is a rotation the pixels in both modalities behave in the same way, pointing out the features that enable to recognize better across domains. In their work, they showed that these features are the shapes of the objects, rather than their texture.
Eventually, in [5] is presented a novel technique of multi-task learning. In that paper it is argued that solving K pretext tasks, rather than one, can further improve the performances of the main task, entailing a better domain alignment. The proposed self-supervised task are, for instance, the prediction of the absolute rotation and the location of patch, that can be implemented as classification or as regression problems.

3. Domain adaptation and multi modal data

The standard Domain Adaptation problem considers paired samples $x, y \in X \otimes Y$ sampled from different distributions, namely the source and the target distributions $S(x,y)$ and $T(x,y)$, with $S \neq T$. The paired samples consist of items x and their labels $y \in Y$ from the finite set $Y = \{0, 1, ..., L\}$. Each item x_i is composed of multimodal data, represented with the components $x_i^{(rgb)}$ and $x_i^{(depth)}$. In our setting we consider the marginal distributions $S(x)$ and $T(x)$ from which the training sets are constructed. In the case of the samples $x_i \sim S(x)$, labels are known at training time, while these are unknown for the samples $x_i \sim T(x)$. Therefore, the training phase consists of a supervised classification task that allows the model to extract RGB features and depth features from the source domain. In order to tackle the absence of target domain labels, a self-supervised task is employed. The self-supervision is performed jointly on source and target domain images to allow the model to learn features that are meaningful for both domains. More specifically, for each domain we consider the RGB and depth representation of each item. Then for each $x_i^{(rgb)}$ and $x_i^{(depth)}$ a self-supervised label is generated based on the selected type of pretext task. By training jointly on the described main and pretext tasks the aim is to use the learned domain invariant and RGB-depth features to predict labels on unknown target domain data.

4. Datasets

4.1. Description

For our tests we considered the SynROD and the ROD datasets. The first one was collected and created as described in [1] and consists in both RGB and depth version of artificially generated images, while ROD consists in real RGB and depth images. These sets contain 51 classes of common objects.

4.2. Improvements

Out of all the categories, 47 are considered, leaving out *'lime', 'onion', 'peach', 'tomato'*. Furthermore, subsets of images for each class are sampled from SynROD and ROD. Overall, following the proposed synROD split we obtain 37528 synROD train samples 7301 synROD validation images, totaling $N_{synrod} = 44829$ samples. Regarding ROD, the considered ROD subset that is used for testing the configuration consists of $N_{rod} = 32476$ images out of the 41877 present in the original ROD.

4.3. Usage

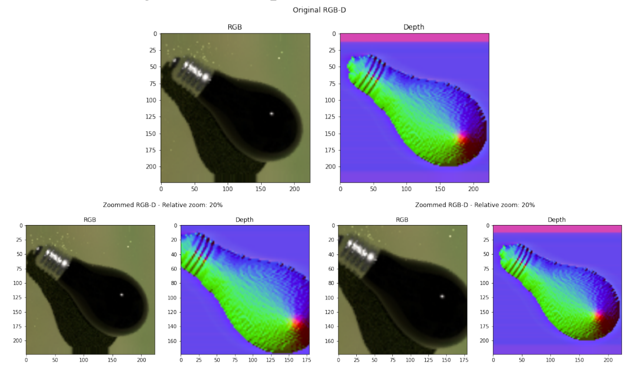
We generate the training and test sets by pairing for a given sample its RGB and depth representation along with its label. Therefore, we have a source main task training set $S_{train} = \left\{ \left(x_i^{(rgb)} x_i^{(depth)} \right), y_i \right\}_{i=1}^{N_{synrod}}$, a source pretext task training set $S_{pretext} = \left\{ \left(x_i^{(rgb)} x_i^{(depth)} \right), z_i \right\}_{i=1}^{N_{synrod}}$, a target pretext task training set $T_{pretext} = \left\{ \left(x_i^{(rgb)} x_i^{(depth)} \right), z_i \right\}_{i=1}^{N_{rod}}$ and a target test set $T_{test} = \left\{ \left(x_i^{(rgb)} x_i^{(depth)} \right), y_i \right\}_{i=1}^{N_{rod}}$. Here, z_i is generated pretext task label.

5. Method

5.1. Pretext task

In this section we present the types of pretext task that have been tested on the dataset configuration presented in the previous paragraph. Starting from the previous work on relative rotation classification pretext task[1], we extend the experiments by proposing two different pretext task variations based on the same principles. That is, the application of relative transformations to the RGB and depth representation of the samples. We propose a relative zoom regression and an alternative classification pretext as means of learning RGB-D invariant features with the purpose of generalizing also to different domains the learned characteristics. In order to avoid loss of information caused by an high zoom applied to both images, we randomly select only one which is then zoomed in of a percentage in the range $[0, 40]$. An example can be seen in Figure 1. Therefore, the defined label generation procedure for the pretext task consists of a first step in which one between the RGB and depth image is selected with equal probability. Then, a random number between 0 and 40 included is sampled from a uniform distribution and the selected image is zoomed-in of that amount, while the other is left as is. By following this procedure it is possible to generate a "relative zoom difference" random variable that is uniformly distributed. Since the zoom

Figure 1. Example of 20% relative zoom


Original RGB-D
RGB
Depth
Zoommed RGB-D - Relative zoom: 20%
RGB
Depth
Zoommed RGB-D - Relative zoom: 20%
RGB
Depth

difference values follow a total ordering, the relative zoom percentage prediction can be seen as a regression problem. An alternative to this, is to discretize the zoom intervals and predict the relative zoom difference percentage as a specific category, thus considering this setting as that of a classification problem. The discretization we implement is every 10% of zoom, resulting in 5 classes: 0%, 10%, 20%, 30% and 40%. This can be justified noticing that a zoom transformation is always discrete, because it is done at pixel level.
In our work, we consider both variations of this pretext problem. The proposal of the two types of zoom tasks stems from the work done by [17], which states that cross-entropy helps in finding better local optima compared to those obtained with mean squared error in comparable environments.

5.2. CNN architecture

The CNN architecture employed in this study is composed of two feature extractor networks, whose outputs are concatenated along the filter dimension and then fed into a main and a pretext branch. These are dedicated respectively to the main classification task and the given pretext task. In this analysis we propose different variations of this architecture, which consist in different feature extractor architectures combined with modifications to the pretext branch. The starting CNN architecture is the one proposed in [1] that consists of two convolutional feature extractors of which one is dedicated to RGB images and the other to the respective depth representation. For this purpose, two ResNet18 without the last global average pooling and fully connected layers have been used. The two networks are initialized with pretrained imagenet weights. Their output features are concatenated and the result is forwarded either to the main task branch or to the pretext task branch. The former will be referred to as M, while the latter as P. The concatenation of the feature extractor filter banks will be referred to as F. Regarding the structure of the M branch, it consists of a global average pooling layer, followed by a fully connected that uses batch normalization, a ReLU activation function and dropout 0.5. This branch terminates with a softmax-fully connected layer with $num\_classes$ output neurons. The P branch instead consists of a sequence of two 2D convolutional layers that use batch normalization and ReLU activation function. The first of the two utilizes 1x1 filters with stride = 1, while the one that follows uses 3x3 filters with stride = 2. Padding is set to 0 for both. At the end of this sequence, a global average pooling operation is performed before feeding the result into another fully connected layer with batch normalization, ReLU and dropout 0.5. Its output ends up in the final fully connected output layer which consists of: 4 neurons with softmax output for the relative rotation task; 1 neuron for the relative zoom regression; 5 output neurons followed by a softmax output for the relative zoom classification task. While the two ResNet18 are pretrained, the M and P branches have been initialized with Kaiming normal initialization [18] in our best performing configurations. In this case, the two ReLU activations of the pretext branch convolutional layers have been substituted with the PReLU activation proposed in [18], each setup with 1 parameter in order to limit overfitting. An overview of the architecture can be seen in Figure 2.

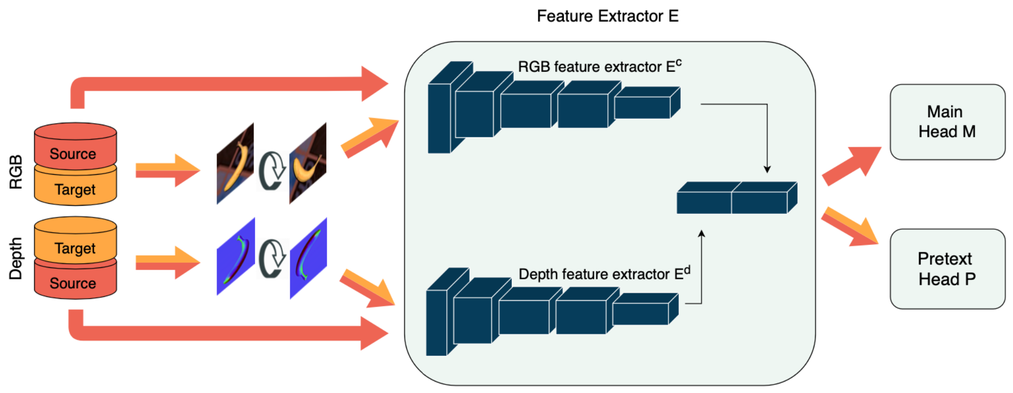
The substitution of the ResNets with two inceptionV3 models yields a variation of the architecture (Figure 3) that allows us to evaluate how different feature extractors affect domain adaptation performances. In order to make use of the auxiliary modules that are part of the inception models, we propose a modification that adapts them to the context of this problem. Removing the global average pooling and fully connected output layer from the auxiliary modules allows for the concatenation of their output filter banks. These convolutional outputs are then fed into a fc-softmax output layer M_{aux}dedicated to the main task and into a fc-output layer reserved for the pretext task P_{aux}, thus mimicking the structure of the M and P branches. The M and P branch, attached to the concatenation of the output of the RGB-inception and of the Depth-inception, maintain the same structure proposed in the previous multi ResNet18 architecture.

5.3. Initialization

5.3.1 Default Initialization

Reading [1] we learned about Xavier initialization [19] and implemented it in all layers, convolutional and fully connected with biases initialized to 0. This was our first initialization scheme which we refer to as *default*. The reason to use this kind of initialization is because in deep neural networks the mean and variance change from layer to layer and this type of initialization prevents this behaviour. Dur-

Figure 2. CNN architecture as seen on Robbiano et al. [1]



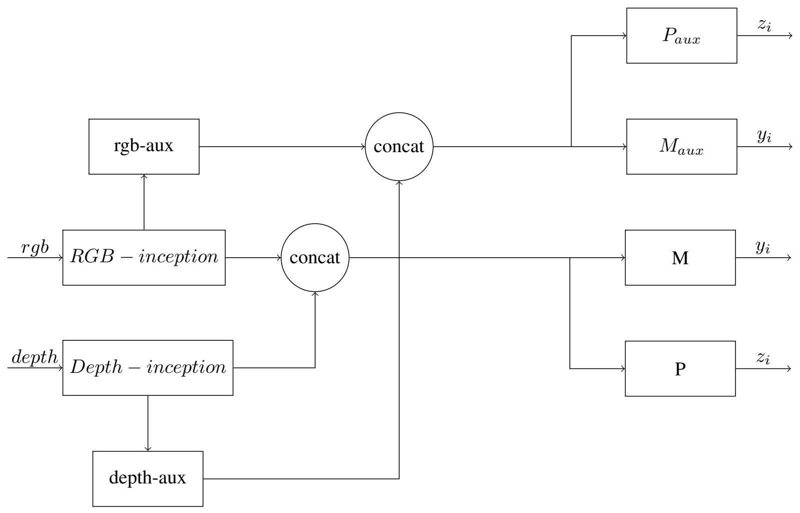


Figure 3. Proposed RGB-Dinceptionv3 architecture

ing our research we found that this kind of initialization was intended to be used on activation functions such as sigmoid and hyperbolic tangent, while in our model we used ReLU. This led us to search for other kind of initializations such as Kaiming, described below.
Xavier uniform initialization consists in setting the weights of the network sampling from a uniform distribution defined as follows:

$$W \sim U[-\frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in}+n_{out}}}]$$

where n_{in} and n_{out} are rispectively the number of input units and the number of ouput units in the weight tensor. At first, initialization of the model was underestimated also due to the batch normalization layers which were implemented in order to reduce vanishing or exploding mean and variance between layers. After obtaining poor results, we further investigated the reasons of these outcomes and noticed that initialization played a key role in training. Since it is so important, we researched various initialization methods and tried them in our models.

5.3.2 Xavier Initialization

For this type of initialization, that we refer to as *Xavier*, we used Xavier uniform initialization only on convolution layers, while used a normal distribution with 0 mean and 0.01 variance on the fully connected with bias set to 0. We found this initialization to be much more performing than 5.3.1. This choice for linear layers was proposed on [20]. After reading many works, we noticed that this is a common practice and it is considered as state of the art for fully connected layers initialization.

5.3.3 Kaiming Initialization

After the remarkable initialization method proposed by Glorot et al. [19], that was not specifically designed for rec-

tified activation functions (ReLU), a further step was made by Xe et al. [18] in which was introduced a generalization of the rectified activation function, called Parametric Rectified Linear Unit (PReLU) and a novel initialization method originally thought for convolutional layers, tailored for rectified units.
They proved that replacing the parameter-free ReLU activation by a learned parametric activation unit improves classification accuracy. The definition of PReLU is

$$f(y_i) = max(0, y_i) + a_i \cdot min(0, y_i)$$

where a_i is a learnable parameter and the subscript i in a_i indicates that such parameter can be separately learned for each channel or as an alternative, the PReLU can have just one parameter for all channels. This configuration introduces only one extra parameter for each layer. We will try both configurations. If a_i is a small fixed value, ($a_i = 0.01$) the PReLU becomes the Leaky ReLU in [21] which was introduced to avoid zero gradients.
The introduction of new learnable parameters increases the capacity of the model raising the risk of overfitting, but this number is negligible compared to the total number of parameters for each layer. As stated by Xe et al., this can increase the accuracy while having a small overfitting risk.
Suited to this activation units, Xe et al. introduce an initialization schema inspired by [19]. They propose to initialize the weights of the layers as normally distributed with zero mean and variance

$$Var = \frac{2}{n}$$

where n is the number of connections of a response, namely $n = c \cdot k \cdot k$, and the bias initialized to zero, $b = 0$.
This kind of initialization proved to be effective in very deep models, outperforming the one of Glorot et al.

Fully connected layers were initialized as in the previous paragraph, using a normal distribution with 0 mean and 0.01 variance. We implemented this method in the pretext task for the convolutional layers, as a variation to the one proposed by [1].

5.4. Data Augmentation

A common practice in deep learning to increase the capability of a model to generalize and to increase its robustness is indeed data augmentation.
Our proposed augmentation technique is tailored separately for each modality, trying to leverage their peculiar properties.
The augmentation carried out on RGB images is inspired by [20] and is based on the assumption that in a couple RGB-D the depth image is immune to changes in lighting conditions, such as the change of sun illumination during the day. For this reason, exclusively on the RGB image it has been applied color jittering, remembering that the depth is just the result of a colorization preprocessing, and its colors does not have the same semantic meaning.
As stated in [4], depth cameras are not perfect and suffer from light reflections, thus real depth images are subject to noise and occlusions phenomena. These are a missing information comparable to random erasure of some areas. Therefore, in the cited paper is proposed an advanced augmentation procedure applied to synthetic depth images, which are too ”clean”, consisting in erasing some pixels to simulate real disturbs.
We decided to adopt a similar augmentation strategy, implementing the Cutout (or random erasing) proposed in [22], encouraged by its outstanding results. The cutout is performed with a 50% probability, a variable scale between (0.02, 0.08) and a variable aspect ratio between (0.03, 0.33) Eventually, for each original RGB-D image another is created by flipping it horizontally. The color jittering augmentation is performed only on one of the two.
This data augmentation strategy has been devised with the focus to produce augmented examples that are not too dissimilar with the ones in the test set. Therefore we applied only augmentation consistent with it. Since all the images in our dataset are square and the objects are centered, it does not make sense to generate square crops of them, while it is consistent to flip or rotate them because in the test set there will be flipped or rotated objects belonging to the same class. This assumption was subsequently confirmed by extensive experiments, which helped us to tailor an *ad-hoc* augmentation method.

5.5. Multiple pretexts

As presented by Sun et al. [5], using more than one self-supervised task may induce a stronger alignment among source and target domains. This is because the solution of each pretext tasks involves learning some specific domain invariant features. Having more pretext tasks may produce even more diverse and equally robust features. Following their idea and inspired by their results, we tried to implement a model having two pretext tasks:

- prediction of the relative rotation between RGB and depth, as presented in [1].
- prediction of the relative zoom, our proposed variation.

The zoom prediction task is implemented both as a classification and as a regression problem, in different trials.
The loss function becomes:

$$L = L_{main} + L_{rotation} + L_{zoom} \tag{1}$$

Note that the pretext losses are added directly, without being multiplied by any factor. This follows the implementation of Sun et al. paper.

Our expectations are that using this configuration will leverage the cooperation between both auxiliary tasks, leading to better results than the ones obtained implementing just one or another.

5.6. Training

The proposed training procedure requires the computation of the predictions for the main classification task in the context of the source domain, and of the pretext task in the source and target domains. Therefore, the losses that are employed in order to train the model are the mean squared error and the cross-entropy loss. Due to the fact that no target domain label are available at training time, an entropy minimization regularization procedure is also used for the purpose of learning partial information also on the target domain main classification task.

Let θ_M, θ_P and θ_F be the parameters of the main branch, the pretext branch and of the feature extractors respectively. Let

$$\hat{y}_i^s = M(F(\hat{S}); \theta_M, \theta_F, \hat{S})$$

be the output of the main branch, received after the RGB and depth images of the source batch $\hat{S}$ have been fed into the respective feature extractors and then concatenated. Let

$$\hat{z}_i^s = P(F(\hat{S_{mod}}); \theta_P, \theta_F, \hat{S_{mod}})$$

be the output of the pretext branch, received after the rgb and depth images of the modified source batch $\hat{S_{mod}}$ have been fed into the respective feature extractors and then concatenated. After defining the outputs related to the target domain with only a change in the notation of the previous from s to t, we denote the losses computed by means of the defined values as follows:

$$L_{main} = -\frac{1}{Ns}\sum_{i=1}^{Ns} y_i^s \log(\hat{y}_i^s)$$

$$L_{reg\_pretext} = \frac{1}{Ns}\sum_{i=1}^{Ns}(z_i^s - \hat{z}_i^s)^2 + \frac{1}{Nt}\sum_{i=1}^{Nt}(z_i^t - \hat{z}_i^t)^2$$

$$L_{clf\_pretext} = -\frac{1}{Ns}\sum_{i=1}^{Ns} z_i^s \log(\hat{z}_i^s) - \frac{1}{Nt}\sum_{i=1}^{Nt} z_i^t \log(\hat{z}_i^t)$$

The complete loss consists of the sum of the main task loss and the two pretext tasks losses weighted by a factor λ: $L_{tot} = L_{main} + \lambda L_{pretext}$. In the case of the inception architecture variation, the loss considers also the contribution given by the auxiliary modules. In particular, we have a L_{main_aux} loss obtained by forwarding the concatenated auxiliary outputs to the M_{aux} layer and a $L_{pretext\_aux}$ contribution obtained by feeding the same output to the P_{aux} layer instead. These terms are weighted with a coefficient set to 0.3 and are added to the L_{main} and to the $L_{pretext}$ losses, respectively.
Therefore, in this case the total loss formulation becomes $L_{tot} = L_{main} + 0.3L_{main\_aux} + \lambda(L_{pretext} + 0.3L_{pretext\_aux})$ This result can be summarized by considering a generic main loss $L_M = L_{main} + 0.3L_{main\_aux}$ and a generic pretext loss $L_P = L_{pretext} + 0.3L_{pretext\_aux}$.
As mentioned before, regularization techniques such as entropy minimization and weight decay are also employed in order to reduce overfitting. Both are implemented and weighted in the parameter update formula. The entropy minimization is weighted with a coefficient τ set to 0.1, while the weight decay wd value is set to 5e-2. While the these two hyperparameters are kept constant for each tested configuration, the learning rate and λ have been fine tuned by means of grid search.

Algorithm 1 RGB-D neural network training

```
Input
    source set  $S = \left\{(x_i^{(rgb)}, x_i^{(depth)}), y_i\right\}_{i=1}^{N_{synrod}}$
    target set  $T = \left\{\left(x_i^{(rgb)} x_i^{(depth)}\right)\right\}_{i=1}^{N_{rod}}$
    network
Output
    test set predictions $\hat{y}$
procedure TRAINING_EPOCH($network, S, T$)
    for $i = 1$ to $N_{batches}$ do
        load main task source batch $\hat{S}$ from S
        load target batch $\hat{T}$ from T
        load pretext batch $\hat{S_{mod}}$ from S
        load pretext batch $\hat{S_{mod}}$ from T

        do network forward pass

        compute main source loss $L_{main}$
        compute source pretext loss $L_{sp}$
        compute target pretext loss $L_{tp}$
        compute entropy loss $L_{ent}$
        $Loss \leftarrow L_{main} + \lambda(L_{sp} + L_{tp})$

        update all parameters $\theta$ of the network
        $\theta \leftarrow \theta - \eta\nabla_\theta(Loss(\theta) + \tau L_{ent}(\theta)) - wd \cdot \theta$
    end for
end procedure
procedure TEST(network, T)
    for $i = 1$ to $N_{test\_batches}$ do
        load test batch from T
        predict target main task labels $\hat{y}^{(t)}$
    end for
end procedure
```

The empirical loss computed in the presented training procedure is composed of three contributions. The main loss L_{main}, calculated by forwarding a source batch through the two feature extractors and the M branch of the network. Therefore, this loss does not depend on the P branch parameters. The auxiliary contribution $0.3L_{main\_aux}$ has to be added to this term, in the case of the inception variation. The L_{sp} and L_{tp} losses are computed by forwarding the source or target pretext batch into the two feature extractors and then sending their concatenated output to the P branch. Thus, these two contributions do not depend on M branch parameters. The auxiliary contribution $L_{pretext\_aux}$ has to be added to this term, in the case of the inception variation. The vector θ contains all parameters of the network.

6. Experiments

In this section we present our experiments. It is divided in three main parts. In the first we discuss about the source-only baselines, in the next one we report the results obtained by implementing the model described in [1] and our variation to it and in the last part we expose the results of diverse optimization strategies that we applied to the models exposed in the previous subsection.
In the first two parts (6.1, 6.2) the presented results are the ones obtained on the target dataset after an hyperparameters optimization procedure, carried out using a grid search method. During this phase, many hyperparameters sets have been tried for each configuration and the best performing are reported in Appendix A - Tuning results. These models have been tuned and tested following precisely the instructions reported in our reference paper [1] without any modification

- using all the classes and all the images present in ROD and synROD datasets.
- initializing the convolutional and linear layers of the pretext task with the *default* initialization, as reported in paragraph 5.3.1.

It is worth of notice that we tried two ways of pre-processing the images: the first one normalizing them using ImageNet mean and standard deviation for both RGB and depth modalities, while the second using ImageNet mean and standard deviation for RGB images and a custom mean and standard deviation computed on the depth dataset for the depth modality. We decided to try this because RGB and depth images belong to different domains and their shift is not slight. We calculated mean and standard deviation of the depth dataset and found that they are quite different from ImageNet. In these parts we used both ImageNet only normalization and depth custom normalization.
In the last section (6.3) we optimize our models by using a sub sample of both datasets, as described in paragraph 4.2, trying a different initialization method, different feature extractors and other practices taken from literature.

6.1. Source-only baselines

The performance improvement obtained by means of domain adaptation and multi modal data on the target domain main classification task has been evaluated by considering various baseline models that do not make use of a pretext task, hence not implementing domain adaptation. We used five types of baselines: RGB only, depth only, RGB-D, RGB-D e2e and RGB-D e2e (main head). All these baseline experiments were performed using the complete datasets with all the 51 classes. Another consideration about these baselines is that the layers added were all initialized using Xavier uniform initialization [19] which during our work we found to be not the most efficient on fully connected layers. All these baselines were implemented following Robbiano et al. [1] paper without using any dataset splits or less than 51 classes. The ones related to the inception architecture variation are implemented by using the same principles adopted for the others, but by substituting the resnet18 architecture with inceptionv3 wherever they are present. The architecture used for the RGB-D e2e inception baseline is equal to the inception variation proposed in the 2 section but without the P branch.

Best results are available in table 1.

6.1.1 RGB only

The network architecture used to evaluate this baseline is a single pretrained ResNet18 with the last fully connected with 51 output neurons. During training only RGB images from the source dataset (synROD) were feed to the network and at the end the model was tested on the target RGB dataset (ROD). This is a standard non Domain Adaptation algorithm which however didn't cut a poor figure because it was the best baseline model. For hyperparameters tuning we fine tuned learning rate in [0.0001, 0.001, 0.01], batch size [32, 64] and weight decay [0.05, 0.005] using 20 epochs and step size 15. After tuning we also re-trained the best model found 5 times to see the mean and the consistency of the training.

6.1.2 Depth only

For this part we used the same architecture as for RGB only but this time we also tried training the model normalizing the depth images using ImageNet means or our custom depth means. The results are very similar in accuracy, but ImageNet gave better results. As expected, training a model on depth only features didn't lead to good results. That is

Source-only baselines		
RGB	imagenet	50.93
Depth	imagenet	12.50
	depth custom	11.71
RGB-D	imagenet	45.90
	depth custom	50.00
RGB-De2e	imagenet	44.10
	depth custom	43.16
RGB-De2e (main head)	imagenet	46.00
	depth custom	42.21

Table 1. Best baseline results

because depth images are not much detailed and the feature extractors find it very difficult to obtain domain invariant features using only depth representation of the objects. As before we performed a grid search algorithm using the same hyperparameters as for the RGB only section and than re-trained the best models found 5 times each.

6.1.3 RGB-D

The network used for the RGB-D baseline is the combination of the RGB only and depth only networks where the last fully connected layers are removed and the concatenation of the two extracted features is used as input for a new fully connected layer. For this method the two feature extractors were already trained and frozen and only the last fully connected was trained. We used our best models found in the previous RGB only and depth only as the two feature extractor. We optimized hyperparameters also in this method but with batch size fixed at 64.

6.1.4 RGB-D e2e and RGB-D e2e (main head)

In RGB-D e2e the network is the same as RGB-D but the feature extractors are not frozen while in RGB-D e2e (main head) baseline the network is the same as RGB-D, but instead of the last fully connected, the extracted and concatenated RGB and depth features are fed to the main branch of the network. The feature extractors are not frozen and start the training pretrained on ImageNet. In this way the model is trained in an end to end fashion. Also for this method we ran a grid search for best hyperparameters.

6.2. Base models

6.2.1 Relative rotation pretext

For this model we tested various configurations changing the values of learning rate and lambda. Learning rate was chosen among [0.0003, 0.001] and lambda was tuned on the values [0.01, 0.1, 0.5]. SGD optimizer with momentum 0.9 was used. Epochs were always 20, weight decay fixed at 0.05, batch size 64 and step size 7.

Base models		
Relative rotation	imagenet	53.82
	depth custom	51.02
Relative zoom (centered)	imagenet	53.84
	depth custom	53.88
Relative zoom (decentered)	imagenet	51.98
	depth custom	52.46

Table 2. Best base models results

6.2.2 Variation - Relative zoom pretext

For our variation we implemented the regression problem of predicting the relative zoom between RGB and depth, as a pretext task. We explored two possible kind of zoom: centered an decentered.

Centered zoom
In this case the zoom is performed towards the center of the image.
In our experiments we tested various configurations and tuned the learning rate and lambda hyperparameters in order to achieve the best performing configuration. The model has been trained on 20 epochs, with batch size of 64. Regarding the optimizer, SGD with momentum 0.9 has been employed. The hyperparameters considered in the analysis are learning rates in the range between $[1e-4, 1e-2]$ and values of $\lambda \in [0.001, 1.0]$.
As showed in Table 6.2, this is the best performing zoom task, probably due to the fact that this task is easier to solve. For this reason we only considered this one in further steps.

Decentered zoom
Since zooming always to the center of the image may lead to give less information to the pixels that are near the borders, we also tried to perform a zoom in a random location of the images, the same for both RGB and depth.
The model has been trained on 20 epochs, with batch size in the interval [32, 64]. Regarding the optimizer, SGD with momentum 0.9 has been employed. The hyperparameters considered in the analysis are learning rates in the range between $[7e-5, 5e-4]$ and values of $\lambda \in [0.05, 0.2]$.
This model has proved to perform worse than the centered zoom. This may be due to the fact that the objects in the images are centered and there is no useful information close to the borders.

6.3. Optimized models

To further optimize the results obtained from tuning and get results closer to the ones presented by [1], we perform a sampling of both ROD and synROD datasets as described in paragraph 4.2 and try different techniques presented in literature. Each of these is commented below and the results are reported in table 3.

Xavier initialization
In this setting, we initialize the model using the initialization method described in the paragraph 5.3.2. This has proved to be effective on all models, compared to our first initialization schema that we refer to as ”Default initialization” in paragraph 5.3.1.

Kaiming initialization
This initialization schema entails the introduction of parametric activation units, the PReLUs. We experimented both ways to configure them: with one parameter for each channel or just one for each layer. Even if with a small difference, the best configuration appeared to be the second one, that uses just one parameter for all the channels of the previous layer. The PReLUs were introduced only after the convolutional layers in the pretext task, following the idea in [18].
We believe that in a domain adaptation setting the overfitting caused by introducing the additional parameters may have a stronger impact than in other applications.

Different feature extractors
We tried two features extractor other than the original resnet18: a resnet34 and Google Inception v3. The resnet34 as a feature extractor did not produced better results than the resnet18, showing a slightly stronger tendency to overfitting which is sensible, due to its higher capacity.
On the other hand, the inceptionv3 variant showed promising results even with its RGB, Depth, RGB-D e2e baselines. Overfitting occurs easily also in this case if the learning rate is too high. However, a proper learning rate scheduling paired with the early stopping strategy criterion allowed to achieve in most runs an accuracy of 67%. After multiple tests with fixed $\%\ lr$ reduction every k ($k \in [3, 4, 5]$) epochs for 15 epochs and starting from $lr = 3e-4$, we noticed that an exponential lr decay reduces noticeably the overfitting over the epochs. Coupling this with a non improvement early stopping criterion, it allowed the model to achieve the highest accuracy reported in the ”optimized models” table. A relatively high starting lr of 3e-4 is required to achieve the mentioned performances in at most 5 epochs. Training over a longer 15 epochs, with a moderate lr such as $1e-4$ or $6e-5$, $\gamma \in [0.1, 0.2, 0.5]$ and reduction $epoch\_num\_decay$ at the end of epochs $1, 4, 9$ or at $1 - 8$, allows to achieve more robust models. All these results are obtained by using Kaiming initialization and a Pbranch configured with PReLU activations at the end the of the convolutional layers.

Data augmentation
The data augmentation process was always tested jointly with a Kaiming initialization and proved to be capable of entailing slightly better results on average, besides providing a faster convergence.
The augmentation preprocessing had a remarkable positive effect on the performances of the pretext tasks.

Optimized models		
Relative rotation	Xavier init	65.11 (64.15)
	Kaiming init	68.31 (65.32)
	Inception v3	70.1 (66.49)
	Data augmentation	65.47 (64.30)
Relative zoom (regression)	Xavier init	62.46 (59.99)
	Kaiming init	65.37 (62.41)
	Inception v3	67.21 (64.30)
	Data augmentation	61.06 (59.23)
Relative zoom (classification)	Xavier init	65.11 (62.89)
	Kaiming init	65.78 (64.07)
	Inception v3	69.55 (65.91)
	Data augmentation	65.69 (64.30)
Double pretext	Zoom as regression	62.51 (60.73)
	Zoom as classification	65.82 (64.19)

Table 3. Best results, the average result is in brackets.

6.3.1 Variation - Predict relative zoom using classification

As expected, using a cross-entropy loss led to better performances rather than using the MSE loss. This phenomenon is visible after few epochs of training: at the beginning the MSE loss is very large and good overall improvements are made, but from a certain point on the MSE loss start to decrease slowly, compared to its value. On the other hand the cross-entropy is never as large as MSE, but it keeps decreasing over the epochs.

6.3.2 Double-pretext model

This model proved to be effective when in the right conditions. In fact, when the rotation pretext was implemented as a regression task, its performances were negatively influenced by it, being pretty similar to the ones obtained by the regression problem alone. On the other hand, when that pretext task and all the losses took into account into the total loss equation (1), the model was able to leverage the multiple pretexts providing a good domain alignment.
It was implemented using the Kaiming initialization method.

7. Conclusions

Our work is basically divided into two main sections.
In the first one we precisely follow the instructions described in [1] training their presented model, our variation to it and a set of source-only baselines. Here we prove

that domain adaptation has a remarkable impact on the performances of the main classification (recognition) task and it is able to reduce the domain shift between source and target domains. While being able to prove the importance of domain adaptation showing the entity of improvement that can be obtained by our variation, we were not able yet to reach the same results of [1]. This asked us to investigate deeper to find different strategies to improve our results, which led to the second part of our work.
In this following part we reach and overtake the performances presented in [1] with different approaches at the same time and even when we do not surpass its performances, we still are able to experiment other promising optimizations, that together prove our results are consistent and not obtained by chance, underlying the robustness of the implemented domain adaptation algorithms.
To obtain the aforementioned results, a pivotal step has been to deepen our knowledge on initialization methods. After many trials with different configurations, we found how much of an impact it has on performance. This shows that the initial state of a model is fundamental in order to obtain good results. The fact that the input signal can propagate without vanishing or exploding in deep networks is essential for a proper training. The best suited to our model was the Kaiming initialization method [18] and the usage of PReLU after convolutional layers.
We tried several techniques to improve our results such as a data augmentation pipeline tailored for this specific RGB-D domain that proved to be capable of increase the generalization ability of the models.
Giving more importance to the feature extraction phase, we implemented some variations using Inception v3 that reached outstanding results, far better than all the other models.
We eventually further investigate in the filed of self-supervised domain adaptation, trying a multitask learning approach that, in our case, utilizes two pretext tasks. This method proved to be more stable than the ones implementing just one pretext task, but at the same time it proved to suffer from the negative influence of a bad pretext task.

We have proposed a variation to the self-supervised task in [1] which at the beginning has been though as a regression problem, but after extensive experimenting and reasoning on the results, we have guessed the superiority of cross-entropy loss with respect to MSE. Since our hypothesis has been confirmed by literature [17], the pretext task has been converted to a classification problem.

8. Source code

Our source code can be found here `github.com/matbun/RGB-D-Domain-Adaptation`.

References

[1] M. Reza 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.

[2] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting visual category models to new domains," 2010.

[3] F. M. Carlucci, P. Russo, and B. Caputo, "(de)2co: Deep depth colorization," 2018.

[4] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, "Multimodal deep learning for robust rgb-d object recognition,"

[5] Y. Sun, E. Tzeng, T. Darrell, and A. A. Efros, "Unsupervised domain adaptation through self-supervision," 2019.

[6] M. Long, Y. Cao, J. Wang, and M. I. Jordan, "Learning transferable features with deep adaptation networks," 2015.

[7] B. Sun, J. Feng, and K. Saenko, "Return of frustratingly easy domain adaptation," 2016.

[8] R. Xu, G. Li, and J. Yang, "Larger norm more transferable: An adaptive feature norm approach for unsupervised domain adaptation," 2019.

[9] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," 2016.

[10] P. Russo, M. Carlucci, T. Tommasi, and B. Caputo, "From source to target and back: symmetric bi-directional adaptive gan," 2018.

[11] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," 2017.

[12] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," 2014.

[13] L. Spinello and K. O. Arras, "Leveraging rgb-d data: Adaptive fusion and domain adaptation for object detection," *in ICRA. IEEE*, p. pp. 4469–4474, 2012.

[14] J. Hoffman, S. Gupta, J. Leong, S. Guadarrama, and T. Darrell, "Crossmodal adaptation for rgb-d detection," *in ICRA. IEEE*, p. pp. 5032–5039, 2016.

[15] X. Li, J. Fang, J. Zhang, J. Wu, and T. Darrell, "Domain adaptation from rgb-d to rgb images," *Signal Processing*, vol. 131, p. pp.27–35, 2017.

[16] W. Jing and Z. Kuangen, "Unsupervised domain adaptation learning algorithm for rgb-d staircase recognition," *URL http://arxiv. org/abs/1903.01212*, 2019.

[17] H. N. Pavel Golik, Patrick Doetsch, "Cross-entropy vs. squared error training: a theoretical and experimental comparison," 2016.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," 2015.

[19] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," 2010.

[20] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," 2012.

[21] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," 2013.

[22] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," *URL http://arxiv. org/abs/1708.04896*, 2017.

Appendix A - Tuning results

Below are reported the results obtained from tuning hyperparameters, to select the best model. For each model were tried numerous hyperparameters sets, but for the sake of simplicity we report only the first 8, sorted by test accuracy and test loss.
Note that the this tuning has been carried out following our first setting, namely the one that follows strictly the instructions depicted in [1] without any sort of model optimization introduced by us or dataset sampling. Furthermore, in this setting our *default* initialization (see paragraph 5.3.1).

Baselines

Below are reported the results obtained for source-only baselines.

RGB only

lr	batch size	epochs	wd	step size	test acc	test loss
0.0001	64	20	0.05	15	0.5093	1.8653
0.001	64	20	0.005	15	0.4805	2.1300
0.0001	32	20	0.005	15	0.4747	2.1812
0.0001	64	20	0.005	15	0.4650	2.0451
0.001	32	20	0.005	15	0.4446	2.2374
0.0001	32	20	0.05	15	0.4432	2.2913
0.01	64	20	0.005	15	0.3575	2.5825
0.01	32	20	0.005	15	0.3183	2.7631

Table 4. RGB only imagenet

Depth only

lr	batch size	epochs	wd	step size	test acc	test loss
0.01	64	20	0.05	15	0.1250	3.5694
0.01	32	20	0.005	15	0.1028	4.1456
0.0001	32	20	0.005	15	0.0999	4.1085
0.0001	64	20	0.005	15	0.0912	4.1118
0.0001	64	20	0.05	15	0.0906	3.7069
0.01	32	20	0.05	15	0.0889	3.5768
0.01	64	20	0.005	15	0.0877	4.0453
0.0001	32	20	0.05	15	0.0718	3.7070

Table 5. Depth only imagenet

lr	batch size	epochs	wd	step size	test acc	test loss
0.0001	64	20	0.05	15	0.1171	3.5974
0.0001	64	20	0.005	15	0.1054	3.9521
0.0001	32	20	0.005	15	0.1041	4.3763
0.01	64	20	0.05	15	0.0971	3.8458
0.001	64	20	0.005	15	0.0893	4.4546
0.0001	32	20	0.05	15	0.0743	3.6579
0.001	32	20	0.005	15	0.0682	4.1154
0.01	32	20	0.05	15	0.0652	3.5313

Table 6. Depth only custom

RGB-D

lr	batch size	epochs	wd	step size	test acc	test loss
0.0001	64	20	0.005	15	0.4590	1.9803
0.001	64	20	0.05	15	0.4526	2.0435
0.01	64	20	0.05	15	0.4515	2.0503
0.0001	64	20	0.05	15	0.4413	2.0848
0.01	64	20	0.005	15	0.4353	2.2449
0.001	64	20	0.005	15	0.4343	2.2061

Table 7. RGB-D imagenet

lr	batch size	epochs	wd	step size	test acc	test loss
0.001	64	20	0.05	15	0.5000	1.8656
0.0001	64	20	0.05	15	0.4937	1.8905
0.0001	64	20	0.005	15	0.4888	1.8487
0.01	64	20	0.05	15	0.4887	1.8980
0.001	64	20	0.005	15	0.4756	1.9885
0.01	64	20	0.005	15	0.4682	2.0784

Table 8. RGB-D custom depth

RGB-D e2e

The epochs are always 20 and the step size is always set to 15 epochs.

lr	batch size	weight decay	test accuracy	test loss
5E-05	32	0.005	0.441	2.1147
0.0001	32	0.005	0.4173	2.3858
0.0001	64	0.005	0.412	2.2883
1E-05	32	0.05	0.4023	2.277
0.0001	64	0.05	0.3815	2.2543
5E-05	32	0.0005	0.3709	2.3778
1E-05	32	0.005	0.361	2.3872
0.001	64	0.005	0.3468	2.5656

Table 9. RGB-D e2e imagenet normalization

Lr	Batch size	Weight decay	test accuracy	test loss
0.0001	64	0.05	0.4096	2.2903
5E-05	32	0.005	0.4080	2.1565
0.0001	64	0.005	0.4058	2.2229
5E-05	32	0.0005	0.4033	2.2537
5E-05	64	0.0005	0.3989	2.3599
5E-05	64	0.005	0.3888	2.2226
0.0001	32	0.005	0.3801	2.3288
0.0005	64	0.005	0.3598	2.5516

Table 10. RGB-D e2e depth custom normalization

RGB-D e2e (main head)

lr	batch size	epochs	wd	step size	test acc	test loss
0.0001	64	10	0.05	7	0.4601	2.1649
0.0001	64	10	0.05	15	0.4204	2.2758
5e-05	64	10	0.05	7	0.3850	2.3770
0.0005	64	10	0.05	15	0.3347	3.0336
0.001	64	10	0.05	15	0.2379	3.2400
1e-05	64	10	0.05	7	0.1911	3.1952

Table 11. RGB-De2e imagenet

lr	batch size	epochs	wd	step size	test acc	test loss
0.0001	64	10	0.05	15	0.4221	2.2759
0.0001	64	10	0.05	7	0.4187	2.2948
5e-05	64	10	0.05	7	0.3919	2.3525
0.0001	64	20	0.05	15	0.3871	2.4571
0.0005	64	20	0.05	15	0.3370	3.1122
0.0005	64	10	0.05	15	0.3175	3.0982
0.001	64	20	0.05	15	0.2626	3.2579
0.001	64	10	0.05	15	0.2126	3.3198

Table 12. RGB-De2e custom depth

Relative rotation pretext

lr	batch size	epochs	wd	step size	lambda	test acc	test loss
0.0003	64	20	0.05	7	0.1	0.5382	2.1224
0.001	64	20	0.05	7	0.5	0.4883	2.5974
0.001	64	20	0.05	7	0.1	0.4779	2.6576
0.0003	64	20	0.05	7	0.01	0.4714	2.6461
0.0003	64	20	0.05	7	0.5	0.4669	2.4292
0.001	64	20	0.05	7	0.01	0.4564	2.7144

Table 13. Relative rotation imagenet

lr	batch size	epochs	wd	step size	lambda	test acc	test loss
0.0003	64	20	0.05	7	0.1	0.5102	2.4668
0.0003	64	20	0.05	7	0.5	0.5063	2.6151
0.0003	64	20	0.05	7	0.01	0.4752	2.6535
0.0003	64	20	0.05	7	0.5	0.4510	2.7076
0.001	64	20	0.05	7	0.01	0.4477	2.8090
0.001	64	20	0.05	7	0.1	0.4473	2.7325

Table 14. Relative rotation custom depth mod

Relative zoom pretext

Centered zoom

lr	batch size	epochs	wd	step size	lambda	test acc	test loss
0.0003	64	20	0.05	7	0.01	0.5384	2.0868
0.0003	64	20	0.05	7	0.1	0.4967	2.1883
0.001	64	20	0.05	7	0.001	0.4895	2.6280
0.001	64	20	0.05	7	0.01	0.4844	2.5791
0.0003	64	20	0.05	7	0.001	0.4657	2.5166
0.001	64	20	0.05	7	0.1	0.3253	3.0232

Table 15. Centered zoom imagenet

lr	batch size	epochs	wd	step size	lambda	test acc	test loss
0.0003	64	20	0.05	7	0.01	0.5388	2.2637
0.0003	64	20	0.05	7	0.001	0.5007	2.3078
0.001	64	20	0.05	7	0.001	0.4899	2.6244
0.0003	64	20	0.05	7	0.1	0.4869	2.2126
0.001	64	20	0.05	7	0.01	0.4790	2.6020
0.001	64	20	0.05	7	0.1	0.3487	2.8336

Table 16. Centered zoom depth custom

Decentered zoom

The epochs are always 20 and the step size is always set to 15 epochs.

lr	batch size	weight decay	lambda	test accuracy	test loss
0.0001	64	0.05	0.05	0.5198	2.1809
0.0001	64	0.05	0.2	0.5093	1.8903
0.0005	64	0.05	0.05	0.4836	2.2282
0.0001	32	0.05	0.2	0.4784	1.8289
0.0001	32	0.05	0.05	0.4703	2.3972
0.0005	64	0.05	0.2	0.4392	2.3717
7E-05	64	0.05	0.3	0.4195	2.0459
0.0001	64	0.05	0.3	0.417	1.9687

Table 17. Variation - relative zoom, imagenet normalization

lr	batch size	weight decay	lambda	test accuracy	test loss
0.0001	64	0.05	0.2	0.5246	1.8156
0.0001	64	0.05	0.05	0.5244	2.0689
0.0005	64	0.05	0.05	0.499	2.288
0.0001	32	0.05	0.2	0.4947	1.9531
0.0001	32	0.05	0.05	0.481	2.1769
0.0005	64	0.05	0.2	0.4434	2.3361
0.0001	64	0.05	0.3	0.4338	2.0987
7E-05	64	0.05	0.3	0.4334	2.0628

Table 18. Variation - relative zoom, custom depth normalization