RGB-D Domain Adaptation with Cross-Modality Self-Supervision

Lidia Fantauzzo
Politecnico di Torino
s273117@studenti.polito.it

Valentin Nelu Ifrim
Politecnico di Torino
s271413@studenti.polito.it

Matteo Latino
Politecnico di Torino
s269317@studenti.polito.it

Abstract

One of the attractions of the last few years concerns the automatically generated synthetic data that require no manual annotation, for example in robotics. They could be used to make predictions on real data, however the difference in domain negatively affects the performance of a neural network. Unsupervised Domain Adaptation (DA) is used to learn a model from a source distribution which performs well on a different target data distribution. So the aim is to implement a DA method dealing with the multi-modal nature of RGB-D data, which is capable of reduce the shift between the two domains by inserting a secondary or pretext *task in a convolution neural network that exploits the relation between RGB and Depth images. This pretext task consists in predicting the relative rotation between the two modal images. Moreover, we apply a technique that tries to give a visual explanation of how the network makes its statements regarding the classification of images. Such method is called Grad-Cam and it results in a graphical representation of an activation map that highlights the the important regions in the image for predicting the concept.*

1. Introduction

In this paper we want to expose our work related to the implementation of a convolutional neural network with unsupervised domain adaptation. The purpose is to optimize a main classification task with the aid of a pretext task. This implementation has already been carried out by a research group led by professor Barbara Caputo [2], so our project aims at re-implementing some of their experiments and try to explore a possible variation. The network is trained by multi-modal RGB-D data. The standard RGB images, obtained with a digital camera, provide information about color, texture and appearance by a three-dimensional array that explicitly stores a color value for each pixel. Depth images have values according to how far are the objects in the scene, so pixels represent distance from camera. This modality brings additional geometric information and is more robust to lighting and color variations. The main classification task consists in predicting the class from where images belongs: in particular 47 different categories. This task intents to learn the network using synthetic images and adapt the model to perform well on real data. So the experiments will be mainly divided in two steps: the first will consist in training the CNN, without domain adaptation, only with the synthetic dataset and validate and test it on the real dataset. The second part will introduce the domain adaptation and so the pretext task, with the aim of showing the benefits brought by this new added branch.

Synthetic data, created with computer graphics software, does not requires manual annotation. This is an enormous advantage due tue the high costs of obtaining great amounts of annotated data to train a CNN.

The shift of domain distribution between synthetic and real data needs a solution in order to adapt the network to recognize images from an unknown domain. This problem is addressed using the aforementioned pretext task and its underlying double modality: a couple of

matching RGB and Depth images, both from real end synthetic data, are randomly and independently rotated and sent to the network. The support task assigned to the domain adaptation is to predict the relative rotation between the two modalities. For this task both source and target data can be used due to the self-supervised nature of the task: the labels to be predicted are not the class but the relative rotation that it is stored in memory before training. The experiments carried out have shown that the introduction of this new task improves the performance of the CNN on objects categorization by reducing the shift between the real and synthetic domains. Moreover, the previous DA approaches do not fully explore the relation between RGB and Depth. The connection between these two allows the network to extract additional information about image features useful for the main task.

The problem of the interpretability in convolutional neural network is a high barrier that prevents the use of this powerful tool in many circumstances where the transparency of the decision process is fundamental. In order to try to break down these barriers, many methods are being developed to allow us to inspect the work and the decision roles inside the network. In this regard, we implemented the Gradient-weighted Class Activation Mapping (Grad-CAM), which uses the gradient that flows through the CNN and the feature maps produced by the convolutional layers to produce an heat map. This heat map, overlapped with the original image that was being analysed by the network, is able to show the areas of the image that mostly contributed to the final decision according to the neurons.

The paper initially describes and explores the dataset used for this project. Then, the successive sections will describe the methods, such as the network and the optimization, used during the experiment. In the next section, there are the discussion of the trials done on the network, and the correspondent results. At the end, our variation is presented, the implementation of the Grad-Cam methods and the relative outcomes.

2. Dataset

In this section, we describe the dataset used for our project (2.2) and the classes defined to retrieve and elaborate the images. (2.3).

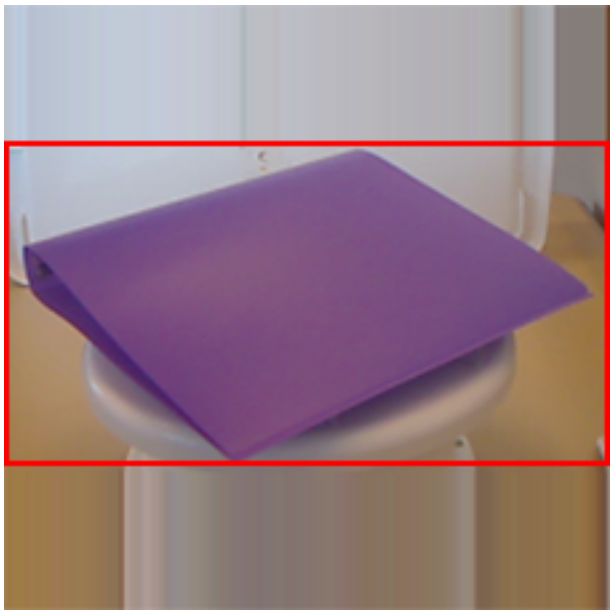

Figure 1: The red rectangle highlights the original image, while the areas above and below are replicas of the first and last row of pixels respectively

2.1. RGB-D images

In this project a corresponding colorized depth-encoding picture of the same subject is provided for every RGB image. Moreover, as explained in [1], preprocessing the RGB image is not particularly problematic, while some encoding decision has to be made for the depth data. In particular in this experiments the depth values are also encoded as three channels images: the depth data are normalized to lie between 0 and 255. Then, a jet colormap on the given image is applied in order to convert the one channel into three (colorizing the depth). It is also important to highlight that, as explained in [1], all the images present in the dataset are square crops. In case of an original image in form of a rectangle, the last line of pixels was repeated on the shorter. The same technique was also applied by the authors of [2] while building the SynROD dataset. Figure 1 shows an example of the transformation.

2.2. Data Exploration

The project needed two dataset containing the same classes but with images collected in different conditions. The first one is the popular RGB-D Object

Dataset, in short ROD, a dataset used in robotics for object categorization that contains real RGB-D data. The images come from 51 categories such as notebook, tomato, toothpaste and coffee mug. The second dataset is the SynROD, a synthetic dataset that has been made by the authors of the article [2], taking the same object models that form the classes in the ROD dataset. Due to the pivotal role of both the shape and the color of the objects, the dataset contains the same image in two modality: RGB and Depth. Depth image is an image channel that contains information related to the distance of the surfaces of scene objects from a viewpoint encoded as described in Section 2.1.
The total classes in the ROD dataset were 51 but in order to remain coherent and keep the same labels for the ROD and SynROD, only the 47 common classes were considered. While the SynROD dataset is quite balanced, with about 700 images in each class, ROD exhibits greater variance i the number of samples, ranging from 200 to 1200 images per category. Figure 2 shows the distribution of synROD and ROD in each of the 47 classes for a sample of 30000 images.

2.3. Dataset Usage

The use of the two dataset has been done with the implementation of two different classes, ROD and SynROD. In both classes for each object the two images has to be loaded, one per modality (RGB and Depth) since they are both required by the implemented Network.
The synROD class loads the path of the images from a textual file, provided by [2], in which the images to use are indicated and saves them in two different dictionary respectively for the RGB and Depth pictures, and when it is required the synROD class load the associated PIL images from both dictionary, then it invokes the needed transformations over them and it provides the correct labels for every step of the process.
A flag called *pretext* is used to distinguish the cases in which the net requires the images with the class label for the main task and the case in which the net require the rotated image with the relative rotation between the two modalities for the pretext task. Differently from the synROD class, the ROD class saves directly the PIL images (and not the path) on the main memory and then it applies the required transformations over them. In this case, in addition to the *pretext* flag which operates as in the synROD class, there is also a *validation* flag with priority over the previous one and applies a different set of transformations, required for the validation and the test process. Both classes provide also

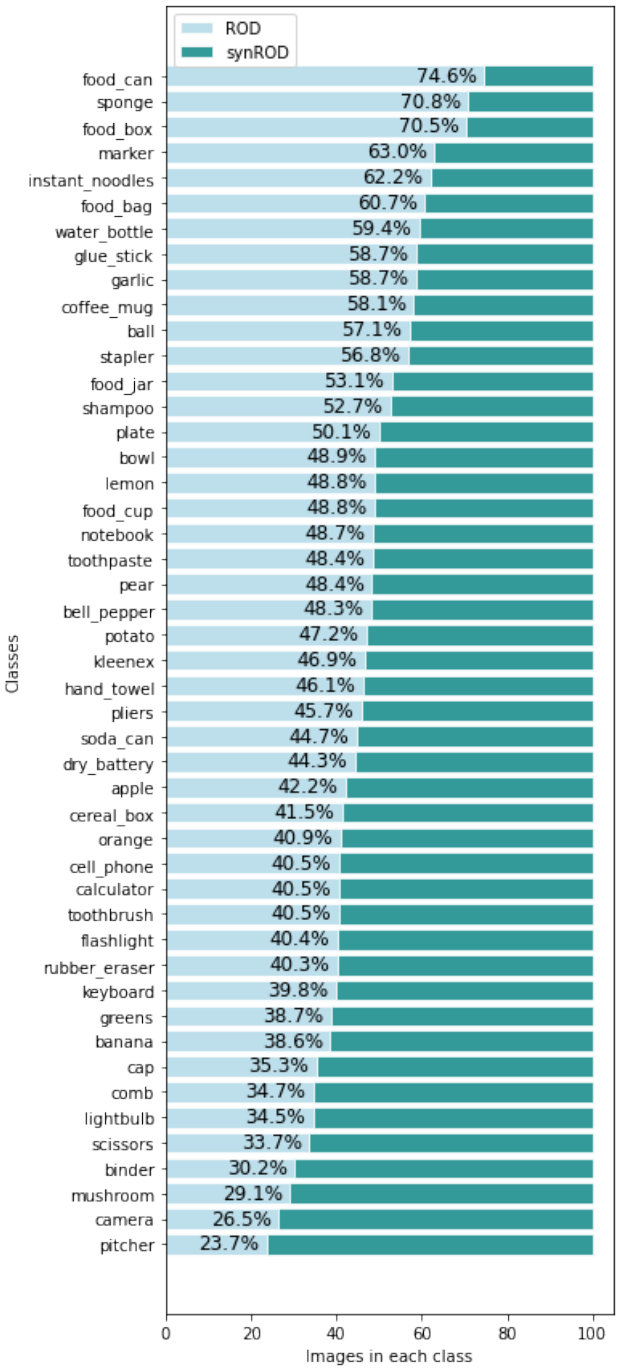

Figure 2: For each class, the percentage of ROD and SynROD images.

a set of functions that give access the different dictionaries and allow setting the flags to the wanted value.
The data-loaders using those classes, at each iteration will return a batch size of images per modality and one label: this last value will depend on the aforementioned *pretext* flag and *validation* flag, which if they are left as default the data-loaders will yield the class labels. To shuffle as much as possible the data, different data-loaders are used for the main and the pretext task.
The SynROD dataset is used only for training, and independently from the assigned task the following transformations will always be applied:

- **Resize**: all images are scaled to be 256 by 256 pixels (usually the applied transformation scales the picture such that the shorter edge will match the input size, but as previously mentioned, all used data are squares of different dimensions);
- **Random Crop**: from the 256 by 256 square 3 channels pictures a randomly positioned crop of 224 by 224 pixel is made on both RGB and Depth images in order to match the input of the two *ResNet-18* used at the beginning of the Network architecture;
- **Random Horizontal Flip**: it mirrors horizontally the images with a probability of 50%; as the previous transformation, this one is applied simultaneously over RGB and Depth image (if the color image is rotated also the depth one will be);
- **ToTensor** & **Normalized**: they are eventually used to transform the images into tensors and to normalize them according to the values used for ImageNet training (since two pretrained versions of the *ResNet-18* are used).

While the first and the last transformations were standard ones, it was necessary to override the two random transformations in order to apply the same random transformations to both modality. To be consistent with real applications where artificial intelligence machines process the same image in both modalities simultaneously the same crops in both modalities are needed. It will also be fundamental in the pretext task, where the crops have to be made in the same way for the two modalities to be analyzed and to predict the relative rotation. The ROD dataset was harder to manage since it was both used for training and for validation, for this reason an additional flag is used (*validation* flag) that when set to *True* the random transformations are substituted with a centre crop of 224 by 224 pixels. For the pretext task, in the training phase there are two major additions to the previous task. Firstly, the same transformation that were applied on the SynROD in the previous task, are now applied to the ROD as well, since the ROD dataset is used during the classification task in an unsupervised way as regularization. Moreover, the network has to be fed with the randomly rotated images from both dataset in both modalities, beside having applied the other transformations previously described for the training. The rotation of the images is done independently for RGB and Depth images but they are always returned coupled, for the relative rotation prediction. This transformation consists in four possible random rotations, by $0, 90, 180, 270$ degrees, each of which corresponds to a label from 0 to 3. In particular, the rotation is the clockwise angle from RGB to Depth and is encoded accordingly. This can be interpreted as the number of times the first image must be rotated clockwise of 90 degrees to get the in the same position as the second one.
The initial dimension of the SynROD dataset was of 37528 pictures, but since the ROD datset has only 32476 images, the size of SynROD has been reduced in order to have the same number of batch for both of them and always have the iterators synchronized, if that were not be the case, an overhead during training had to be add to re-initiated the data-loaders accordingly. To further optimize execution times, since was possible, despite the limited resources, it was chosen to load the ROD dataset in RAM during the initialization phase while the synROD is loaded while trained.

3. The Methods

In this section are exposed the network that was implemented to fulfill the task and the optimization methods used.

3.1. Introduction

The aim of the experiments is to re-implement the convolution neural network with domain adaptation which was able to overcome the existing DA methods for the classification of RGB-D data ([2]). This network is built with the purpose of solving a main supervised classification task using labeled source data and unlabelled target data, and a pretext self-supervised task. This last one tries to predict the relative rotation between the same image in two modalities, RGB and depth, that have been rotated independently. The possible rotations are four multiples of 90 degrees whose label are the digits from 0 to 3. For this task, the labels are generated automatically so it is possible to use both the source and the target data to train the network. This additional task makes the CNN learn more features that improves the performance of the main task without the need of a huge amount of annotated data, since it also allows to align the source domain features with the target domain features.

3.2. The network

The network that is re-implemented it is shown in Figure 3. The section *E* contains the features extractors which receive the source and target images for the main task and the source and target rotated images for the pretext task. This section is divided in two streams of convolutional neural networks, E^c and E^d, which respectively extract features from RGB and Depth data. The couple of tensor images are send to the network and each of them passes through the corresponding branch. This branches are implemented as two *ResNet-18* architectures without the final fully connected and global average pooling layers. This is due to the fact that the data will be further processed before the classification. At the end of the *E* section, batches of resulting feature maps will be concatenated along the channel dimension in order to obtain the RGB-D features: it will result in a tensor of size $[batch\_size \times 1024 \times 7 \times 7]$. If the *main_task* flag is set as *True* the concatenated tensor is sent to the Main Head branch (M), otherwise is sent to the Pretext Head branch (P). The Main Head is implemented using an Adaptive Average Pooling, a Fully Connected layer with an output of 1000 neurons, a Batch Normalization layer, a ReLU activation layer, the Dropout and lastly a Fully Connected layer having as output a neuron for each class. The final Softmax activation function, used by the last fully connected layer is already integrated into the loss Cross Entropy Function that used during training phase. On the other hand, the Pretext head solves the classification regarding the relative rotation of the images. It is made by two 2D convolutional layers, one with 100 output channels and a 1×1 kernel and another one with 100 neurons and a 3×3 kernel; the second convolutional layer uses a stride of two, yielding as as output a tensor of shape $[batch\_size \times 100 \times 3 \times 3]$ that will be fed to the two Fully Connected Layers with 100 and 4 neurons respectively. Between every two layers there are a Batch Normalization Layer and ReLU as activation function. Before the last Fully Connected Layer there is again a Dropout Layer. The two convolutional layers, added in this head, allow to extract further features for the relative rotation classification; it has been proved by the authors of [2] that this leads to superior performance compared to adopting the architecture of M for both heads.

3.3. Initialization

The two branches of the feature extractor are initialized with the values of a pre-trained *ResNet-18*, manually coping the weights, the biases for the Convolutional Layers and also the *running_mean* and *running_var* for the BatchNormalization Layer. The remaining layers of the network architecture were initialized with a xavier uniform initialization. Those weights were used as a starting point for all the training procedures.

3.4. Optimization

As optimizer for the losses the stochastic gradient descent with momentum is used. The objective function to minimize during the training phase is a linear combination of some loss functions. A Cross Entropy Loss (L_m) is exploited to penalize wrong results over the main classification task with the Source data (SynROD). Then two additional Cross Entropy Loss objects are used for the pretext task, one for the Source and one for the Target data

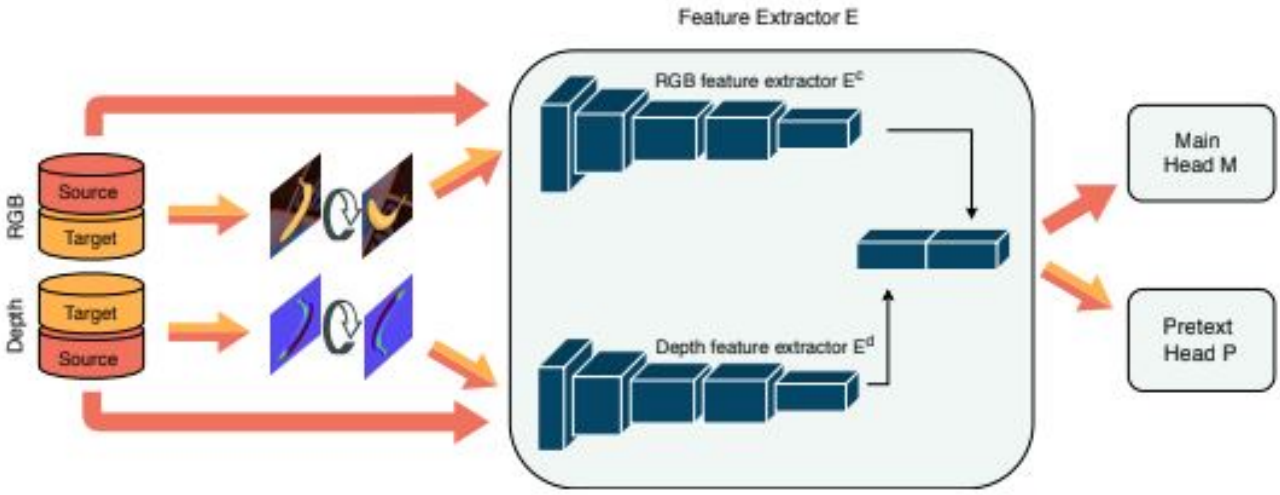

Figure 3: The network [2]

which sum gives L_p. This two losses are combined with λ_p, a weight to regulate the contribution of the corresponding pretext loss term. Moreover an Entropy Loss (L_e) is computed over the results of the unsupervised classification of the Target data as regularization with a weight of 0.1. The Algorithm 1 briefly summarize one step of the training procedure. Since the limited resources available on *Google Colab* wasn't possible use a batch size of 64 as specified by [2], to overcome this problem, it is used a gradient accumulation approach. The batch size was reduced to 32 and the the weights of the network are updated only after the back propagation of the losses of 2 consecutive batches.

Algorithm 1: Network train

Data:
S = source batch processed for main task
Sp = source batch processed for pretext task
T = target batch processed for main task
Tp = target batch processed for pretext task
if *main_task* **then**
  Compute main loss L_m
  Update weights of M from ∇L_m
  Update weights of E from ∇L_m
else
  Compute main loss L_m
  Compute entropy loss L_e
  Compute pretext loss L_p
  Update weights of M from ∇L_m and ∇L_e
  Update weights of P from ∇L_p
  Update weights of E from ∇L_m, ∇L_e and ∇L_p
end

4. Results

4.1. BaseLine

At first the network was trained without domain adaptation, using only the main branch. Due to hardware limitations an exhaustive hyperparameter was not possible, for this reason the suggested default values given by [2] were used and shown in the Table 1. In particular is important to mention that the best results were obtained after few epochs and then the validation accuracies tended to oscillate around a certain value or drop drastically. As validation score it was computed the accuracy over a random subset of the *ROD* images, *VALIDATION_SIZE* images to be precise, a hundred batches. Due to this oscillating behaviour of the accuracy validation score a stopping policy to stop training was implemented. If the accuracy scores in the last four epochs were always worsening at each epoch the train would stop. This trick allowed more testes. The obtained results are summarized in Table 3. As it is possible to notice the mean result were not as high as the ones presented in [2] (47.70%), but some

HYPER PARAMETER	VALUE
BATCH_SIZE	64
LR	3.0×10^{-4}
MOMENTUM	0.9
WEIGHT_DECAY	0.05
NUM_EPOCHS	30
VALIDATION_SIZE	BATCH_SIZE$\times$100

Table 1: Baseline Hyperparameters

runs got even higher results of 48.11%. The scores that this experiments tries to reach are the one regarding the *Source only, RGB-D e2e synROD→ROD*. As aforementioned the needed epochs to reach a maximum were in average five, while the training did not last for more than 12 epochs due to progressive worsening.

4.2. Pretext

The next step was to repeat the experiments using also the pretext branch. This time two additional hyperparameters were used: $ALPHA$ and $ENTROPY\_LOSS\_WEIGHT$, the first one is a regularization parameter for the back propagation of the pretext loss L_p of both ROD and synROD (the aforementioned λ_p). The second one instead is a regularization parameter for the back propagation of the entropy loss L_e. The Table 2 summarise the chosen hyperparameters. Despite the recommendations on the needed hyperparameters the maximum test accuracy reached was of 53.01% while there was an average of 51.76%. From the Table 3 it is possible to notice that averagely speaking the training lasted longer: after some epochs there was no improvement at all and the same stopping policy as before was applied.

HYPER PARAMETER	VALUE
BATCH_SIZE	64
LR	3.0×10^{-4}
MOMENTUM	0.9
WEIGHT_DECAY	0.05
NUM_EPOCHS	30
VALIDATION_SIZE	BATCH_SIZE$\times$100
ALPHA	1
ENTROPY_LOSS_WEIGHT	0.1

Table 2: Pretext Hyperparameters

5. Variation: Grad-CAM

In this section it is exposed the implementation of the Gradient-weighted Class Activation Mapping (Grad-CAM), and the outcomes obtained by the application of this methods on the presented network.

5.1. Introduction

The field of neural network is treated as a *black box* approach: these deep learning methods are as powerful as they are not interpretable. This gives rise to a big obstacle for the full exploitation of those algorithms, because in some social context, it is really necessary to be aware of the reasons that led the network to make a certain decision; for example, in the field of job recruitment, an automatic machine could act in a discriminatory way due to a bias, in the data or embedded in its implementation code. For this reason the knowledge of how an artificial intelligence powered machine operates and takes its decisions is very important. In this regard, having a tool that allows to inspect the way in which the network takes its decisions reveals to be really useful. The *CAM*, or Class Activation Map, identifies discriminative regions to understand where the network is looking to classify an image. The only-CAM methods works only with a net that does not contain any fully connected layers, so it requires that the convolutional layers directly precede the Softmax layer. So this is a drawback because CAM can be applied to networks with a specific architecture, that perform global average pooling over convolutional maps immediately before prediction. For this reason, generalization of CAM is used, the *Grad-CAM* [4], which can be used on every kind of architecture and does not requires any modification or re-training of the original network structure. It is chosen to implement the Grad-CAM, a methods that uses the gradient flowing toward the last convolutional layer, to produce a map which highlights the area of the image that positively contributed to the resulting classification. In the next

	TESTED ACC	VALIDATION ACC	BEST EPOCH	TOTAL EPOCHS RUN
BASELINE	$46.82 \pm 0.70\%$	$46.84 \pm 0.86\%$	4.50 ± 1.43	8.60 ± 2.27
PRETEXT	$51.76 \pm 1.3\%$	$49.92 \pm 1.70\%$	8.5 ± 2.21	13.11 ± 3.43

Table 3: Experiment Results

sections all the details will be discussed.

5.2. Base Theory

As stated in [4], the last convolutional layers is the best compromise between high-level semantics and detailed spatial information. In fact deeper networks capture more information, but this information is lost when passed to the fully connected layers. For this reason extracting the feature maps from the last convolutional layer before the fully connected layers can be the best practice, although this methods can be applied in any layer of a CNN. Convolutional layers look for semantic class-specific information in the image which are able to discriminate the different classes. Grad-CAM exploits the gradient information flowing through the network toward the last convolutional layer of the CNN to create an heat map that highlights the regions of the image that have had some relevance in the decision made by the network. Thanks to this outcome, besides being a useful approach to demystify the work of CNN as said before, this can be also useful to verify that the network is indeed looking at the correct patterns in the image and activating around those patterns.
The algorithm is used to obtain the class-discriminative localization map Grad-CAM $L^c_{Grad-CAM} \in \mathbb{R}^{u \times v}$, of width u and height v for a class c, as discussed in [4]. The gradient of y^c, the score for the class c before the softmax, is computed with respect to feature map activation A^k of the last convolutional layers, $\frac{\partial y^c}{\partial A^k}$. These gradients flowing back are global-average-pooled over the width and height dimensions (indexed by i and j respectively) to obtain the neuron importance weights α^c_k

$$\alpha^c_k = \frac{1}{Z} \sum_i \sum_j \frac{\partial y^c}{\partial A^k},$$

where Z is the product between u and v. The α^c_k captures the importance of feature map k for the target class c. The Grad-CAM heat map is a weighted combination of feature maps with a ReLU activation function

$$L^c_{Grad-CAM} = ReLU \sum_k \alpha^c_k A^k. \quad (1)$$

The size of the result is the same of the kernel of the last convolutional layers (or of the convolutional layers used), in our case, applying this approach to our network we will obtain a 7×7 heat map. The application of the ReLU activation function is performed due to the fact that only the features that contribute positively to the decision regarding the selected class are evaluated. These features that have a positive influence on the class of interest are the pixels whose intensity should be increased in order to increase y^c. Negative pixels are likely to belong to other classes so if we did not apply the ReLU, the heat map would show information also about other classes, instead of focusing on the target one.

5.3. Implementation

The code is organised in a main class called *GradCAM* which inherits from an other class called *_BaseWrapper*. The object of GradCAM class is instantiated with the model on which the method is applied. Then the forward function is called with the images in the two modalities. Here the images are fed to all the net to obtaining the scores. Eventually the softmax is applied and the results are sorted. Then the back propagation takes place calculating the gradient of the score of the class predicted for the image, with respect to the feature maps only for the class of interest. The generation of the heat map is done by a function called *generate* that receives the name of the convolutional layer of which reveals the areas of the image that have activated it most. During the forward and the backward propagation the feature maps and the gradients of each layer were saved respectively using the two functions *forward_hook* and *backward_hook*. In order to do this, some handlers were instantiated

and automatically called during the two events. So the *generate* function finds the feature maps and the gradients corresponding to the target layer and then a global average pooling is applied on the gradients to obtain the α_k^c's. The equation in (1) is computed and result is up-sampled to obtain a tensor with the same dimension of the initial image tensor. Then it is normalised between 0 and 1 to generate an heat map.
In this implementation the code of Kazuto Nakashima ([3]) was used as a starting point.

5.4. Application

After having trained the network with the pretext task, the model was saved in order to use it for the *Grad-CAM* variation. Two lists of 20 images each (10 per modality) are created, respectively containing images correctly and wrongly classified by the network from both modalities. The images are transformed as required by the network and than are passed to the Grad-CAM method. The heat maps obtained are overlapped with the original RGB images and displayed, as shown in Figure 4 and Figure 5.
Looking at the correct classified images is possible see that in all the cases the network is focused on the object in the centre of the image and not in the background. In particular, in some cases, the network looks at the shape of the overall object like in the Figures 4.a, 4.f, 4.b, 4.i and 4.e. It is possible to see that the heat-map covers almost all the object in the image, this means that the entire object is analyzed by the network to allow the correct classification. Instead in other cases the net focuses the analysis on a specific sub-part of the object that allows to distinguish it from all other classes, like in Figure 4.c in which the *Grad-CAM* is activated by the handle and the blades of the scissors; in Figures 4.d, 4.g and 4.h the top of the bottle is identified by the Model as the feature that most characterizes the water bottle and the soda can, that despite having a similar shape overall, they have a totally different lids.
Regarding instead the images miss-classified, it is possible that even if also in this cases the network looked for the correct features, understanding the shape of the object ignoring the background, but it was not enough to correctly label the object. In some cases is the shape of the object itself that is not helpful at all since can be similar across different classes. In Figures 5.b, 5.c and 5.d the spherical shape led to miss-classification: the apple as a lemon, the pear as a lemon and the ball as a light bulb. In Figure 5.h the keyboard is classified as a calculator, having both a rectangular shape and a set of keys. In this case however the Network yielded really close scores for both classes.
In other cases instead, the Network activated on only a sub portion of the object like on the top of it or at its edges (Figures 5.f, 5.g, 5.i, 5.l and 5.e). Even if those are the region of the image in which is possible see some characteristics that allow to identify the correct class, the Network wrongly classify them since is not able to identify very well the different classes. For example in the Figure 5.f the heat-map is centered on the lid of the food jar and as it was possible to observe in other images it is a good way to analyze this type of objects.The food jar and soda can have a similar tops which let to these miss-classifications. The bowls (Figures 5.i and 5.l) in which the network is focuses on the edges and confuses it with with coffee mugs: both objects have similar shapes, and without environment references it would be possible also for humans to mistake.

6. Conclusion

In this work, a re-implementation of a method used to overcome the problem of the classification of images coming from different domain with different modalities, has been carried out. The methodology consisted in implementing a *ResNet-18* with a domain adaptation branch and a double modality setting. Then the network was trained with synthetic images in order to be able to classify real images. Along whit this task, the network learned how to recognize the relative rotation between the same image in different modalities. The Baseline phase obtained results close with the ones reached by the authors of [2].Regarding the Pretext phase, although the results have demonstrated that the introduction of a domain adaptation task to reduce the shift between the two domains was not enough. Even if the results did not get nearly sd high as the ones in [2].
The implementation of the Gradient-weighted Class Activation Mapping (Grad-CAM), to make CNN-

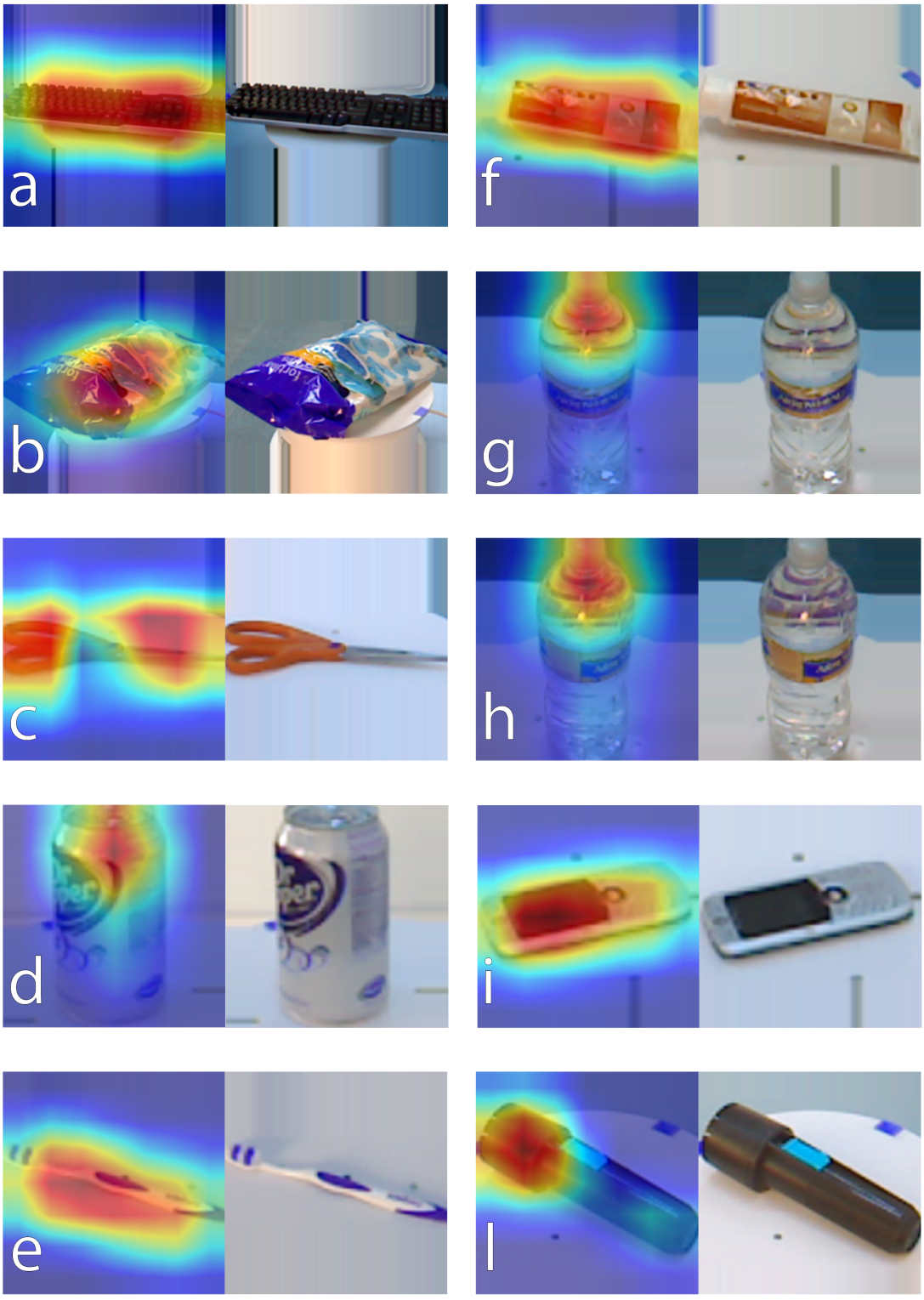

Figure 4: Grad-CAM correctly classified images

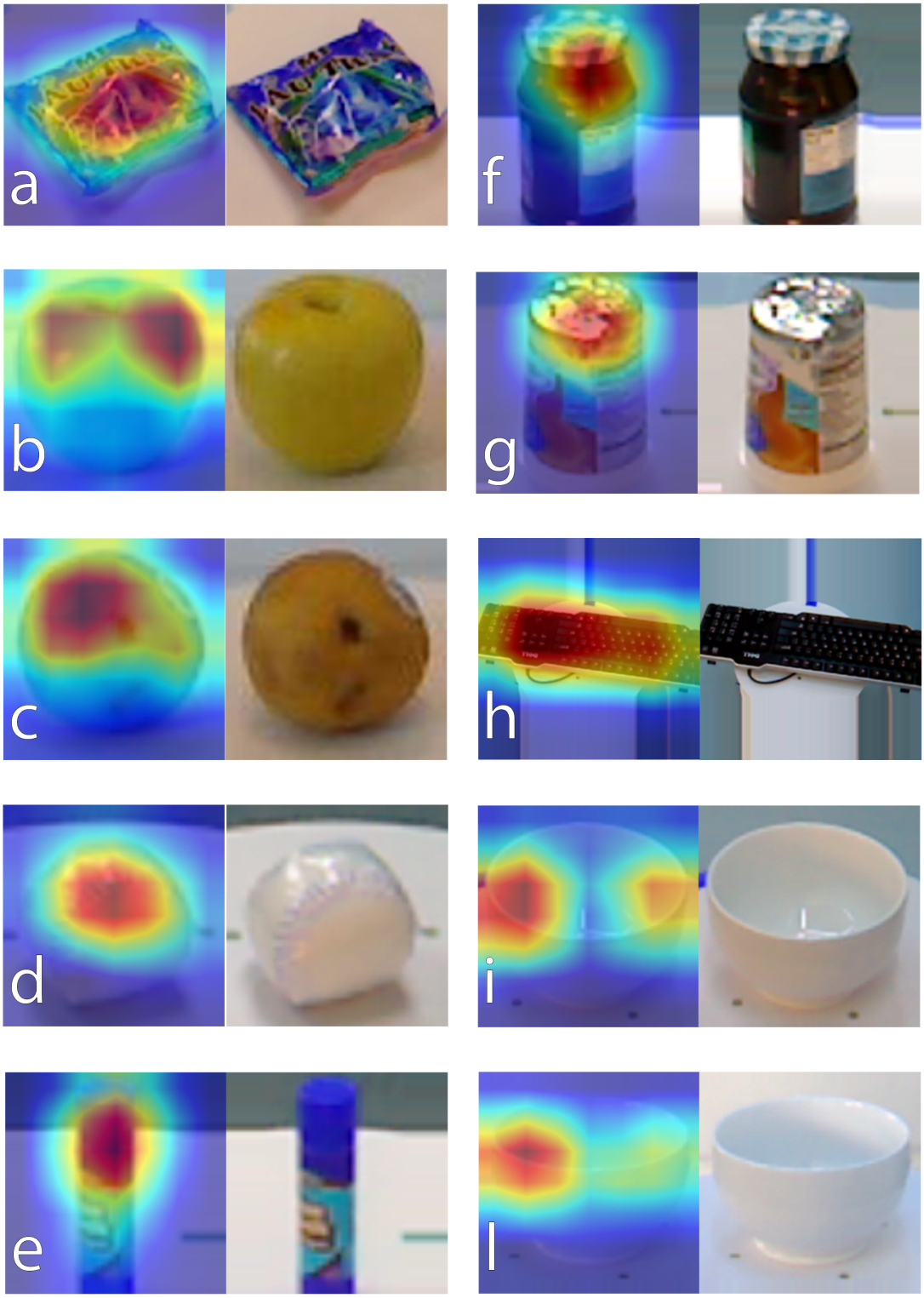

Figure 5: Grad-CAM wrong classified images

based models more transparent by producing visual explanation, gave some remarkable results. When the Network was classifying correctly the class of an image, the features that made it activate were reasonable understandable. On the other hand, some of the incorrectly classified images were difficult to distinguish even for the human eye due to some misleading characteristics, for example the case of the coffee mug mistaken for a bowl and the yellow round pear classified as a lemon.

At the flowing link it is possible to see the code used during this project: Git Hub Repo- `https://github.com/IAmNelu/MLDL`

References

[1] A. Eitel, J. T. Springenberg, L. Spinello, M. A. Riedmiller, and W. Burgard. Multimodal deep learning for robust RGB-D object recognition. *CoRR*, abs/1507.06821, 2015.

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

[3] K. Nakashima. Grad-cam with pytorch. `https://github.com/kazuto1011/grad-cam-pytorch`, 2020.

[4] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization, 2019.