POLITECNICO DI TORINO

Master Degree in Computer Engineering

Master Thesis

Deep Domain Adaptation through Inter-modal Self-supervision

Supervisor
prof. Barbara Caputo

Co-supervisors:
doct. Mirco Planamente
doct. Mohammad Reza Loghmani

Candidate
Luca Robbiano

March 2020

Deep Domain Adaptation through Inter-modal Self-supervision
Master thesis. Politecnico di Torino, Turin.

March 2020.

The work in this thesis has been submitted to the *International Conference on Intelligent Robots and Systems (IROS) 2020* and for publication on *IEEE Robotics and Automation Letters (RA-L)*.

Acknowledgements

In this section I want to thank all the people who have been supporting me in these years and during the development of this work. Since not all of them are comfortable with English, this part will be in Italian.

Vorrei ringraziare prima di tutto la mia famiglia per essermi stati vicini e avermi sostenuto in questi anni di studio.

Ringrazio poi la professoressa Barbara Caputo, per avermi offerto questa opportunità di tesi e per avermi dato la possibilità di lavorare presso il laboratorio VANDAL, e i miei correlatori, Mirco Planamente e Mohammad R. Loghmani, con cui ho collaborato nello svolgimento di questa ricerca. Grazie a loro e alla loro pazienza ho imparato molto in questi mesi. Ringrazio anche tutti i ragazzi del laboratorio, con cui ho imparato cosa significhi letteralmente vivere la ricerca.

Un enorme grazie va infine ai miei amici, che mi sono stati vicini e mi hanno sostenuto in questi anni, anche nei momenti più difficili. Un ringraziamento particolare a Ilaria, Elian, Andrea e Silvia e a tutto il resto della Cluster. Questi anni non sarebbero stati gli stessi senza i momenti folli che abbiamo vissuto assieme. Un grazie speciale anche a Teresa, per aver sempre saputo trovare le parole giuste nei momenti giusti, a Michela per essere sempre stata presente e di supporto negli ultimi anni e a Giovanni, per le nostre passeggiate con annesse chiacchierate filosofiche, sempre spunto di riflessione.

Un grazie enorme anche a tutti coloro che non ho esplicitamente nominato, non per importanza ma per vincoli di spazio...e tempo, visto che i ringraziamenti sono l'ultima cosa e come al solito sto consegnando all'ultimo. Lo sapete, lavoro meglio sotto pressione.

Abstract

Computer vision in robotics makes heavy use of RGB-D data. However, collecting large manually annotated datasets is extremely time-consuming and therefore costly. A potential solution is to automatically generate synthetic datasets and to use them in order to make predictions on the real data. Nevertheless, the domain shift between the synthetic dataset (source domain) and the real data (target domain) partially invalidates the effectiveness of this solution, yielding an accuracy significantly lower than the one that would be obtained using labelled real data. In order to overcome this issue, multiple domain adaptation methods have been developed. These methods can also be employed on multimodal data, such as RGB-D data, but they do not explicitly exploit the natural relationship between modalities. We propose a novel domain adaptation method which allows reducing the domain shift by forcing the convolutional neural network to learn the connection between RGB and Depth images through a secondary self-supervised task. Extensive experiments on object categorization and instance recognition show that the exploitation of inter-modal relation can significantly enhance the performance of the main classification task.

Contents

List of Figures

List of Acronyms

AFN Adaptive Feature Normalisation

CNN Convolutional Neural Network

DA Domain Adaptation

DAN Deep Adaptation Network

DANN Domain Adversarial Neural Network

e2e End to End

FC Fully Connected

GD Gradient Descent

GRL Gradient Reversal Layer

HAFN Hard Adaptive Feature Norm

HB HomeBrewed db

MK-MMD Multiple Kernel Maximum Mean Discrepancy

MMD Maximum Mean Discrepancy

MMFND Maximum Mean Feature Norm Discrepancy

RKHS Reproducing Kernel Hilbert Space

ROD RGB-D Object Dataset

RR Relative Rotation

SAFN Stepwise Adaptive Feature Norm

SGD Stochastic Gradient Descent

SVM Support Vector Machine

Chapter 1

Introduction

Since the advent of the first computers, researchers tried to push the boundaries of their capabilities in order to solve complex tasks. Machine-learning algorithms play a key role in this endeavour, by enabling computers to learn from experience. In the last few years, the breakthrough of deep neural networks profoundly transformed machine learning, allowing to accomplish tasks previously thought to be out of reach for computers.

Computer vision is one of the fields that have been affected the most by the deep learning revolution: the introduction of AlexNet [38] and its victory in the ILSVRC [64] competition in 2012 showed how deep Convolutional Neural Networks (CNNs) could be effectively trained on hundreds of thousands of images achieving significantly better performances than the previous state-of-the-art methods. Other fields had indirect benefits from these results: for instance, robotics makes extensive use of computer vision algorithms to make robots more autonomous.

Figure 1.1: Centauro is a robot designed at the Italian Institute of Technology (IIT) that makes use of RGB-D sensors. Image from [52].

The use of robots has been standard practice in the industry for decades and is still becoming more and more relevant, but other areas are increasingly benefitting

from robotic workforce. Robots have been successfully used for medical applications, space exploration and educational purposes [4], and are rapidly becoming part of our daily lives by assisting humans in their everyday tasks.

In order to correctly perform their tasks, robots usually need to collect information about the surrounding environment. Emulating what humans and other animals do, many robots make use of images to obtain such information and take consequent actions thanks to computer vision. However, because robots physically interact with the environment, they have a particular need for robustness: bad decisions taken by robots can often be very costly and even life-endangering.

For this reason, robots are often equipped with additional sensors, which provide supplementary data to improve the understanding of the context. Among the most frequently used ones there are depth sensors, which along with a standard RGB camera provide visual information enriched by an extra channel encoding distance. This multimodal type of data is usually referred to as RGB-D.

The employment of RGB-D data has been empirically proven to be an effective technique to improve the robustness of computer vision algorithms: RGB-D cameras provide a precise pixel-wise measure of the distance between the sensor and portrayed objects. The additional spatial and geometric information helps the robot to understand the surrounding space better, being less sensitive to light conditions and different textures.

The availability of low-cost RGB-D sensors encouraged their spreading in robotics. However, the lack of large scale multimodal datasets to train the computer vision models is currently a significant bottleneck for the wider adoption of this kind of sensors. CNNs need a massive amount of training samples to be effectively trained for practical purposes, and the production of the required manually annotated datasets can be very expensive.

A possible solution to this problem is to generate large training datasets automatically. Three-dimensional models can be used to render a virtually unlimited amount of samples for each class with different light conditions and backgrounds. However, despite the rendering photorealism achievable thanks to modern raytracing algorithms, the difference between the synthetically generated images and the real world heavily affects the effectiveness of this approach.

The ability of a model to perform well despite the domain shift (i.e. the difference between the training domain, called source, and test domain, called target) is addressed by a specific research field, known as Domain Adaptation (DA). Unsupervised DA aims to minimise the effect of the domain shift by exploiting an unlabelled collection of samples from the target distribution. Since unlabelled samples can be created without needing per-image human intervention, they are much cheaper to collect. Therefore, a domain-adapted training algorithm takes as input a large dataset of labelled data from the source domain and a dataset of unlabelled samples from the target domain. The resulting trained network should be able to perform well on both datasets. In our RGB-D synthetic-to-real setting, we are particularly

interested in the performances on the target domain, since the source dataset is synthetic and produced specifically for training purposes.

Multiple strategies have been proposed to perform DA and can be used for this purpose, but none of them was specifically designed to handle multimodal data. We believe that the additional information provided by multimodality can effectively be exploited to better adapt to the target domain. Our approach consists of encouraging the neural network to learn more transferable features by learning a relation between the two modalities, thus becoming able to balance the lack of information of each channel by using the information provided by the other one.

In chapter 2 we present a summary of the approaches already studied in the literature, pointing out the main differences with our method. In chapter 3 we provide the basics about neural networks required to understand the details of our method, which is described in chapter 5. In chapter 6 and 7 we describe in detail our implementation and the experiments we conducted to compare it with four baseline methods described in the literature, which we describe in chapter 4.

Chapter 2

Related Works

Several approaches have been proposed in the literature for RGB-D object classification. While traditional single-modality RGB object recognition has been based on unsupervised feature extraction for a long time, many of the successful RGB-D approaches kept employing hand-crafted feature extractors [10, 41]. In the last few years, however, some studies suggested extending the use of unsupervised feature learning for RGB-D data as well. Some of the proposed strategies involve clustering algorithms [9], random forest classifiers [3] or convolutional filters combined with recurrent neural networks [70].

More recently, multi-stream CNNs have been proposed [19, 67]. This approach employs colourisation methods for the depth channel in order to better leverage transfer learning. CNNs have been proven to be superior to other methods due to their ability to automatically learn a hierarchical set of filters with multiple abstraction levels [38]. Therefore we follow the same approach and implement a dual-stream CNN for our method.

With respect to unsupervised DA, no method has been proposed in literature specifically for RGB-D data. Nevertheless, general-purpose DA methods can be adapted for use in a multimodal setting [11]. A first distinction within DA techniques can be drawn between shallow and deep methods, the latter designed to work with deep neural networks. Because shallow DA is not of interest to our work and our method is based on deep neural networks, we focus on deep DA. More specifically, since in our setting the source and target domain share the same feature space and no labelled target samples are available during training, we are interested in homogeneous unsupervised DA [75].

Some of the methods, like [49, 77], encourage the network to reduce the domain shift by directly penalising the discrepancy between source and target distributions.

Another approach is to use an adversarial objective to boost domain confusion; for example [20] makes use of a domain discriminator and a gradient reversal layer to prevent the feature extractor from learning domain-specific features.

A third approach is to use a secondary task trained to reconstruct the input data.

Xavier and Bengio [24] suggest extracting a hidden feature representation from a stacked denoising autoencoder (SDA) trained on both source and target. Less computationally expensive methods include applying some kind of random transformation on the input and using an auxiliary classifier to reverse it. Xu *et. al.* [76] propose to randomly rotate source and target images by multiples of 90°. By using a four-way classifier to predict the rotation, the feature extractor is encouraged to learn domain-invariant features. With the same purpose, Carlucci *et. al.* [12] suggest solving a jigsaw puzzle by predicting the correct permutation of the input crops.

Our method falls in this last category, but an inter-modal secondary task is used: both modalities are randomly and independently transformed, and the auxiliary classifier is trained to predict the relation between the transformed modalities.

Chapter 3

Background

This chapter introduces the technical foundation on which our work is based. Section 3.1 provides a brief overview of Neural Networks. In section 3.2 we focus on CNNs, which is the specialised type of network used for our work. Lastly, in section 3.3, we describe the recently introduced Residual Networks, which we adopted to implement our method.

3.1 Neural Networks

Neural networks are machine learning models loosely inspired by the way the biological brain processes information. Like the animal brain, they work through simple interconnected units. The name network is due to their common directed-graph representation. When the graph is acyclic, the model is said to be a feedforward network, in contrast to recurrent networks.

Feedforward neural networks can be formalised mathematically as compositions of functions. For example

$$f(\mathbf{x}) = (f_3 \circ f_2 \circ f_1)(\mathbf{x}) = f_3(f_2(f_1(\mathbf{x}))) \tag{3.1}$$

is a network composed of three functions, called layers. f_1 first processes input data, then its output is processed by f_2 and f_3. Therefore f_1, f_2 and f_3 are referred to as the first, second and third layer. Each layer is composed of multiple units defined as

$$h(\mathbf{x}) = g(\mathbf{w} \cdot \mathbf{x} + b), \tag{3.2}$$

where $\mathbf{w}$ and b are model parameters to be optimised during the training of the network, and $g(\cdot)$ is an *activation function* (note that, depending on the context, the activation function is sometimes considered a separated layer itself).

Activation functions are crucial for neural networks since they prevent the model from being a composition of linear functions, which would be a linear function

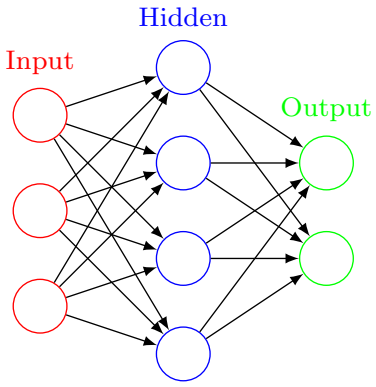


Figure 3.1: Three-layer neural network.

itself, thus being unable to solve non-linear problems. A high number of activation functions have been proposed in the literature [56]. Some of the most used are:

- Hyperbolic Tangent

$$g(x) = \tanh(x) \triangleq \frac{1 - e^{-2x}}{1 + e^{-2x}} \tag{3.3}$$

- Sigmoid

$$g(x) = \sigma(x) \triangleq \frac{1}{1 + e^{-x}} \tag{3.4}$$

- ReLU [34]

$$g(x) = \text{relu}(x) \triangleq \max\{x, 0\} \tag{3.5}$$

However, modern neural networks mostly use ReLU because it can be defined piecewise by linear functions, allowing to preserve their useful properties and making models more easily optimisable by using gradient-based algorithms. Another important activation function often used with multiclass classification is the Softmax function, which is a multidimensional generalisation of the logistic function:

$$\text{softmax}(\mathbf{x}) \triangleq \left[\frac{e^{x_i}}{\sum_{n=1}^{|\mathbf{x}|} e^{x_n}} \right]_{i=1}^{|\mathbf{x}|}. \tag{3.6}$$

While ReLU is the generally preferred activation function for input and hidden layers, Softmax is the most used one for output layers in C-classifiers because its output can be interpreted as a probability distribution over C classes.

The simplest type of neural network layer is the Fully Connected (FC) layer, in which each output unit is connected to all the neurons of the following layer:

$$f_i(\mathbf{x}) = g(\mathbf{W} \cdot \mathbf{x} + \mathbf{b}), \tag{3.7}$$

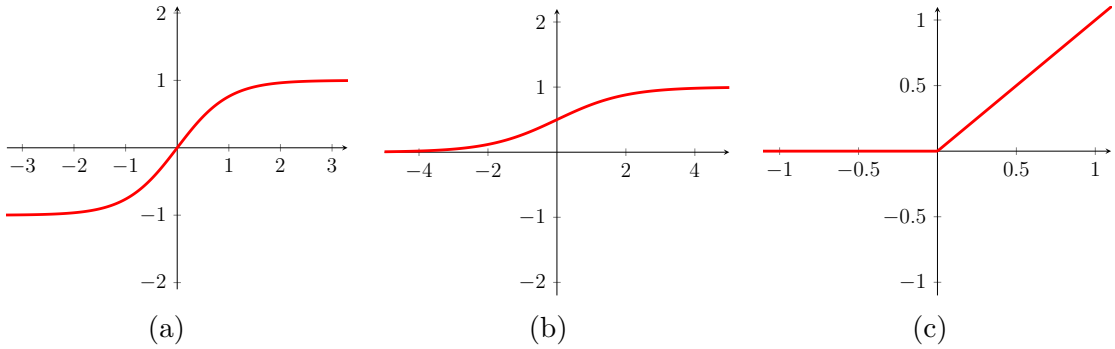


Figure 3.2: Plots of different activation functions. (a) tanh, (b) sigmoid, (c) relu.

where the activation function $g(\cdot)$ is computed element-wise when defined as a scalar function.

FC layers do not have any hyperparameters, except for the number of neurons, which for input and output layers is bound to the dimensionality of input and output data. At the cost of a high number of parameters and high computational requirements due to dense matrix multiplication, FC layers offer a high expressive power (capacity): provided it has enough neurons, a network having at least one hidden layer can approximate any function required by practical applications [15, 31, 32, 45].

3.1.1 Optimisation

Training a neural network means solving an optimisation problem in order to minimise the cost of the errors made by the network. In machine learning terminology, the cost is called loss. In a supervised setting, given a sample $(\mathbf{x}, \mathbf{y})$ and a prediction $\hat{\mathbf{y}} = f_\theta(\mathbf{x})$, the loss function $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ measures the quality of the prediction $\hat{\mathbf{y}}$ with respect to the ground truth $\mathbf{y}$. The choice of the loss function depends on the task. For classification problems, a popular one is the cross-entropy loss

$$\mathcal{L}_c(\mathbf{y}, \hat{\mathbf{y}}) = \sum_i y_i \log(\hat{y}_i). \tag{3.8}$$

Training a network consists of finding the optimal parameters for the model

$$\theta_{\text{opt}} = \arg\min_\theta \mathcal{L}(\mathbf{y}, f_\theta(\mathbf{x})). \tag{3.9}$$

Unfortunately, finding the global minimum θ would require strong assumptions on the form of f. Typically such assumptions are not satisfied, therefore the only viable way to proceed is searching a local minimum.

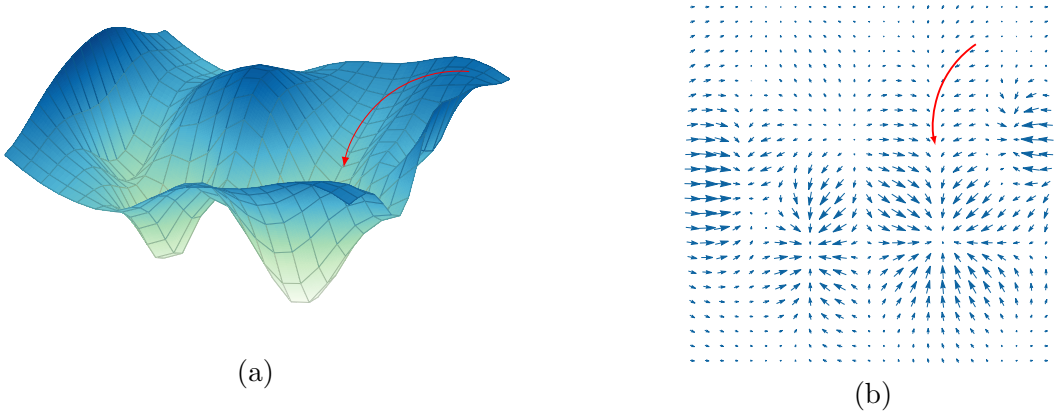
Several iterative optimisation algorithms allow to solve this problem by exploiting the gradient of the loss function with respect to the network parameters, $\nabla\mathcal{L}$. The training procedure consists of two repeated steps:

- During **forward propagation**, the network computes predictions for the training input data, by using the current weights.
- The gradient is computed through the **backpropagation** algorithm, and it is used to update the weights.

The simplest gradient-based optimisation algorithm is the vanilla Gradient Descent (GD) [14], which computes the gradient of the cumulative loss for the whole dataset at each step and then updates the weights as

$$\Delta\,\theta = -\eta \nabla \mathcal{L}, \tag{3.10}$$

where η is a value called learning rate. Like all gradient-based optimisation methods, GD works by exploiting the fact that a multivariate function $f(\mathbf{x})$ decreases fastest in the opposite direction of its gradient ∇f (figure 3.3).



(a) (b)

Figure 3.3: Gradient Descent. (a) Plot of a loss function of two parameters, (b) plot of the negative gradient of the loss function. The parameters are optimised by following the negative gradient.

Vanilla GD, though effective with small amounts of data, cannot be easily scaled to large datasets, which would require to compute the loss function over all the samples before each weigh update. In order to solve this issue, GD is usually approximated by using Stochastic Gradient Descent (SGD) [62], which computes the loss and its gradient only on a small subset ("batch") of the whole dataset at each step.

A number of variations of SGD have been proposed. For example, a "momentum" [61, 63, 73] is added to the update term in order to remember the previous update at each step:

$$\Delta\,\theta_n = \mu\,\Delta\,\theta_{n-1} - \eta \nabla \mathcal{L}. \tag{3.11}$$

A variant of this method is the Nesterov momentum [53, 54, 73], which is similar to standard momentum except that the gradient $\nabla\mathcal{L}$ is computed at $\theta + \mu\,\Delta\,\theta_{n-1}$

instead of θ. Some other popular optimisation algorithms, like AdaGrad [18] and Adam [37], use different learning rates for each parameter and automatically update them during training.

3.1.2 Batch Normalisation

When training a neural network, we update the weight of every layer at the same time. However, the weight update is computed under the assumption that the other functions of the function-composition chain do not change. In practice this assumption is not satisfied, thus preventing to train very deep networks effectively due to a phenomenon known as *internal covariate shift* [68]. The distribution of the input of each layer changes after each training step due to the update of the weights of the previous layer.

A common method to mitigate the effects of internal covariate shift is to use a lower learning rate. Nevertheless, this solution has the drawback of significantly slowing down the training process. Observing that a neural network usually benefits from the normalisation of its input, Ioffe and Szegedy [33] suggest normalising the input of hidden or output layers. For this purpose, a batch normalisation layer is proposed. Given an input $\mathbf{x}$, it can be normalised as

$$f_{\mathrm{BN}}(\mathbf{x}) \triangleq (\mathbf{x} - \boldsymbol{\mu}) \oslash \boldsymbol{\sigma}, \tag{3.12}$$

where $\oslash$ denotes the element-wise division. $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are the mean and standard deviation of each feature, respectively:

$$\begin{aligned} \mu_i &= \mathrm{E}[x_i] \\ \sigma_i &= \sqrt{\delta + \mathrm{Var}[x_i]}, \end{aligned}$$

with δ being a small positive constant to avoid division by 0 and the undefined gradient $\sqrt{a}$ at $a = 0$. During training the statistics are computed for the current batch, and at test time it is possible to use the statistics previously collected, thus allowing inference for single samples.

Because batch normalisation can reduce the expressive power of the model, batch normalisation layers are often defined as

$$f_{\mathrm{BN}}(\mathbf{x}) \triangleq \boldsymbol{\gamma}[(\mathbf{x} - \boldsymbol{\mu}) \oslash \boldsymbol{\sigma}] + \boldsymbol{\beta}, \tag{3.13}$$

where $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are learnable parameters. This formulation allows the output features to have any mean and variance, but both these value are directly learned as single parameters to optimise, thus not affecting the benefit gained with batch normalisation.

3.1.3 Regularisation

Optimisation algorithms are designed to minimise the training loss. However, when developing a machine learning model, we aim to get a low test loss, which does not always happen due to overfitting. Overfitting happens when the model learns the training data noise, thus closely fitting the training samples but failing to generalise when fed with new input. Overfitting is strictly linked with the model capacity: higher representational power leads to higher exposition to overfitting. Since deep models often have a high number of parameters, they have a large capacity, therefore being particularly prone to overfit the training data.

The act of modifying an algorithm in order to minimise specifically the test error without taking into account the training error is known as regularisation. Several regularisation methods have been described in the literature, many of which act by restricting the capacity of the model. One of the most popular approaches consists of adding a parameter norm penalty. L_2 Parameter regularisation, also known as weight decay [39], defines a regularised loss

$$\begin{aligned} \tilde{\mathcal{L}} &= \mathcal{L} + \alpha\, \mathcal{L}_r \\ \mathcal{L}_r &\triangleq \frac{1}{2} \left\| \mathbf{w} \right\|_2^2, \end{aligned}$$

where $\mathcal{L}$ is the main loss and $\mathbf{w}$ denotes the weights of the network, with $\boldsymbol{\theta} = (\mathbf{w}\,;\, \mathbf{b})$. The biases $\mathbf{b}$ are usually left unregolarised in order not to reduce too much the capacity of the model.

3.1.4 Dropout

A common regularisation practice in machine learning is to use ensembles of separately-trained models in order to improve the performances. This method, called bagging, is not very suitable for use with big neural networks due to the high amount of time and memory required to fit this kind of models.

Dropout (Srivastava *et. al.* [72]) provides an inexpensive way to approximate an ensemble of a large number of models while training a single network. During training, random units from the hidden layers of the network are temporarily removed at each step. From a mathematical perspective, this can be formalised by defining a "dropout layer" as

$$f_d\left(\mathbf{x}\right) \triangleq \mathbf{m} \odot \mathbf{x}, \tag{3.14}$$

where $\odot$ denotes the Hadamard product. At test time $\mathbf{m}$ is a mask with constant value $\mathbf{1}$, while during training it is defined as

$$\mathbf{m} = (m_1, m_2, \ldots, m_k) \tag{3.15}$$

$$m_i \sim \mathcal{B}\left(p\right), \tag{3.16}$$

where $\mathcal{B}$ is the Bernoulli distribution. The only hyperparameter of this type of layer is $p \in (0\,,\,1)$. The most common of choice is $p = 0.5$.

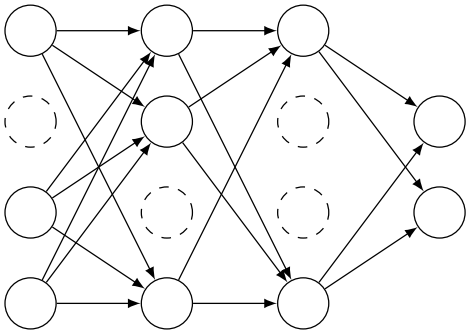


Figure 3.4: Neural Network after applying dropout. At each iteration, random units are temporarily disabled. At evaluation time all the units are active.

3.2 Convolutional Neural Networks

FC networks can model virtually any kind of function, but they tend to be unusable when the complexity of data and task grows beyond a certain point. The number of parameters composing the model is often huge, therefore making the training of such networks infeasible due to memory requirements and computational time. This problem arises because in FC networks, each unit of a layer interacts with every unit of the following one. This interaction is regulated through a weight which is unique for every possible pair of neurons. A network taking a 64×64 px image as input and with 4,096 units in the first hidden layer would need $64^2 \cdot 4{,}096 = 16{,}777{,}216$ weights (64 MiB assuming 32-bit single-precision arithmetic) just for the first two layers.

However, this large amount of parameters is often redundant: weights that are useful to process some parts of the input are likely useful for other parts of the input data. CNNs (LeCun *et. al.* [42, 43]) solve the problem by exploiting three principles:

- **Sparse interactions**: each unit is connected only to a subset of the units of the following layer.
- **Parameter sharing**: the same weights are applied to multiple portions of the input.
- **Equivariant representations**: CNNs are equivariant to translation (i.e. if translating the input yields the same output, translated).

Besides FC layers, the two essential blocks of a CNN are convolutional layers and pooling layers.

3.2.1 Convolutional layer

From a mathematical point of view, the convolution is an operation on two functions $x, w : \mathbb{R} \to \mathbb{R}$ defined as

$$h(t) = (x * w)(t) \triangleq \int_{-\infty}^{+\infty} x(a)\, w(t-a)\, \mathrm{d}a. \tag{3.17}$$

Actual implementations are bound to work with finite-precision arithmetic, therefore a discretized convolution has to be defined. Given two functions $x, w : \mathbb{Z} \to \mathbb{R}$, their discrete convolution is:

$$h(t) = (x * w)(t) \triangleq \sum_{a=-\infty}^{+\infty} x(a)\, w(t-a). \tag{3.18}$$

Moreover, neural networks often need to process multidimensional data (e.g. images or videos), hence a multidimensional definition of convolution is required. Given two functions $x, w : \mathbb{Z}^2 \to \mathbb{R}$, their bidimensional discrete convolution is

$$h(i\,,\,j) = (x * w)(i\,,\,j) \triangleq \sum_{m} \sum_{n} x(i\,,\,j)\, w(i-m\,,\,j-n). \tag{3.19}$$

This definition can be easily extended to n-dimensional functions. Because it is possible to process only finite data, we can assume both x and w to have a limited support. This allows referring to them as n-degree tensors $\mathbf{x}$ and $\mathbf{w}$.

Referring to CNNs, the two operands $\mathbf{x}$ and $\mathbf{w}$ are called respectively *input* and *kernel* (or *filter*), while the result $\mathbf{h} = \mathbf{x} * \mathbf{w}$ is called *feature map*. A single convolutional layer is defined by one or more kernels of size $d \times k_1 \times \cdots \times k_{n-1}$ and takes as input a tensor of size $d \times s_1 \times \cdots \times s_{n-1}$. Since this work focuses on static image recognition, from this point we will assume three-dimensional input tensors of size $c \times h \times w$, where c, h and w are the number of channels (typically 3), the height and the width of the input respectively. The size of the kernel ($c \times k_h \times k_w$) is an important hyperparameter which defines the receptive field of each neuron. Kernels are usually square-shaped ($k_h = k_w = k$) and much smaller than the input tensor, with sizes ranging between 3 and 7.

The output of the layer is computed by convoluting the input with each filter and by stacking the resulting feature maps. Therefore the output volume is defined by the input volume and the number of kernels. Nevertheless, in practice, some other hyperparameters affect the output size. The stride S defines the step size while sliding the input tensor during the convolution. Sometimes it can be useful to zero-pad the input, and the size of the padding is an additional hyperparameter P. The output volume can then be computed as

$$F \times \left(\frac{H - K + 2P}{S} + 1\right) \times \left(\frac{W - K + 2P}{S} + 1\right), \tag{3.20}$$

where F is the number of kernels of the layer.

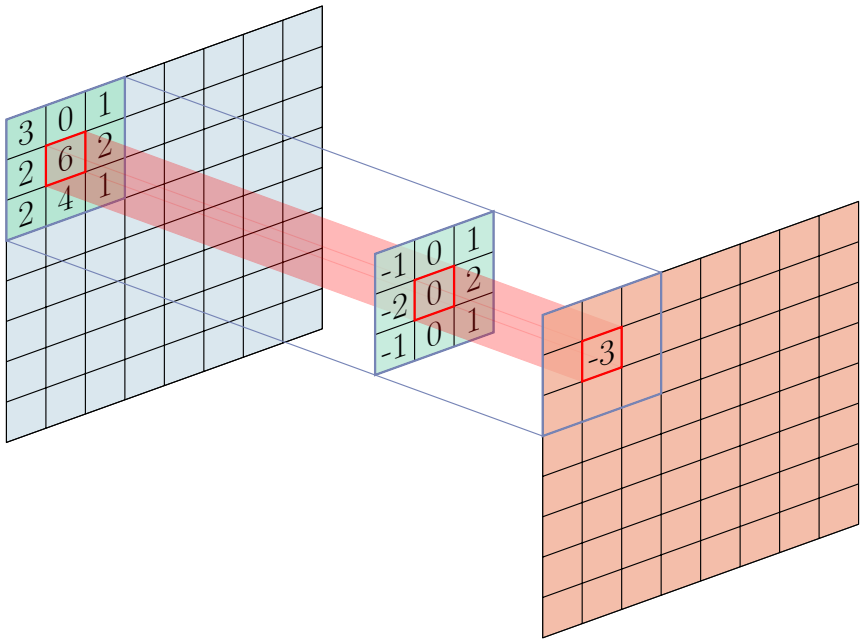


Figure 3.5: Convolution operation with a 3×3 kernel, stride 1 and padding 1.

3.2.2 Pooling layer

The output of a convolutional layer is usually fed to an activation function, which operates element-wise without changing the size of the tensor. After the activation layer, a pooling layer is often present. It does not have any learnable parameters, and its role is to reduce the dimensionality of the input by replacing each value with a summary of its neighbourhood. Pooling is useful because while reducing the dimensionality, it is approximately invariant to small translations. When appended to a convolutional layer with multiple filters, the block can learn to be invariant to any transformation, which is useful for generalisation.

Multiple types of pooling layers have been proposed in the literature. The most common functions are $\max(\cdot)$ [79] and $\text{avg}(\cdot)$, but others are sometimes used (e.g. L_2-norm or weighted average). The pooling function is computed over a rectangular (typically squared) neighbourhood on each channel, so the tensor depth is unaffected. The size of the neighbourhood is a hyperparameter, corresponding to the kernel size of convolutional layers. Similarly to convolutional layers, pooling layers take stride and padding as additional hyperparameters, and the output volume can be computed in the same way.

Some studies [71] suggest deprecating the use of pooling layers, advocating that dimensionality reduction can be effectively achieved by using convolutional layers alone with higher stride values, but at the moment most architecture still make use of them.

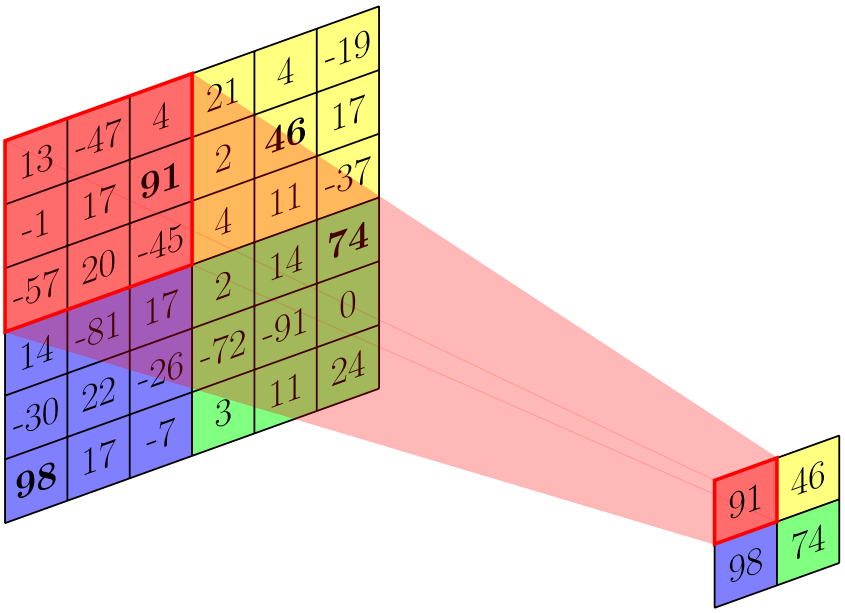


Figure 3.6: Max pooling with a 3×3 filter and stride 3.

3.3 Residual Networks

Due to their ability to learn features on multiple abstraction levels, Deep Neural Networks quickly became the dominant approach to computer vision and many other machine learning fields. Several studies [69, 74] show how the depth of the network (i.e. the number of layers) plays a central role in determining the model performances. All the most successful architectures participating in the ILSVRC [64] competition feature a high number of layers.

Though evidence suggests employing deeper and deeper networks, a major issue arises when stacking a high number of layers [7, 23]. During back-propagation, the gradient gets very small, thus preventing the optimisation algorithm from effectively updating the weights of the first layers. This problem, known as *vanishing gradient*, has been addressed by carefully initialising the network weights [23, 29, 44, 65] and normalisation layers [33].

A second problem when training very deep networks is the *degradation* of the performances: the accuracy gets saturated during training and then starts decreasing rapidly. Since the training loss diverges, we can conclude that degradation is not caused by overfitting. A network with more layers should be able to perform at least as well as a shallower version of the same network because the additional layers can be set to be the identity mapping $f(\mathbf{x}) = \mathbf{x}$, thus making the two models perform identically.

He *et. al.* [30] propose a solution to both degradation and vanishing gradient by introducing Residual Networks. The primary component of the proposed architecture

is the residual block (figure 3.7). Instead of trying to learn the desired mapping

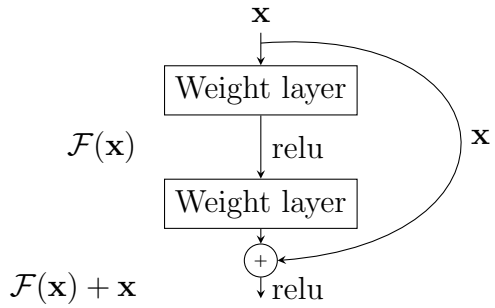


Figure 3.7: Illustration of a residual block.

$\mathcal{H}(\mathbf{x})$, the model tries to fit the residual

$$\mathcal{F}(\mathbf{x}) = \mathcal{H}(\mathbf{x}) - \mathbf{x}. \tag{3.21}$$

The original mapping is then rebuilt as $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$. The hypothesis is that it is easier to optimise $\mathcal{F}$ than to fit $\mathcal{H}$. More specifically, if the optimal mapping were the identity function $\mathcal{H}(\mathbf{x}) = \mathrm{id}(\mathbf{x}) = \mathbf{x}$, it would be easier to fit it by putting the residual $\mathcal{F}$ to 0 than by combining multiple non-linear layers. In real-world problems, the optimal mapping is not expected to be the identity function, but it could be closer to the identity than it is to a zero mapping, thus making it easier to optimise the residual mapping.

In order to compute $\mathcal{F}(\mathbf{x}) + \mathbf{x}$, $\mathbf{x}$ and $\mathcal{F}(\mathbf{x})$ must share the same dimensionality. Depending on the specific form of $\mathcal{F}$, this requirement is not always satisfied. For example, pooling layers (see 3.2.2) apply a dimensionality reduction. Therefore, a linear projection is used to reduce or increase the dimensionality when required:

$$\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + W_s \mathbf{x} \tag{3.22}$$

The authors propose several architectures, with a number of layers varying from 18 (figure 3.8) to 152. Multiple stacked residual blocks compose each of the suggested networks.

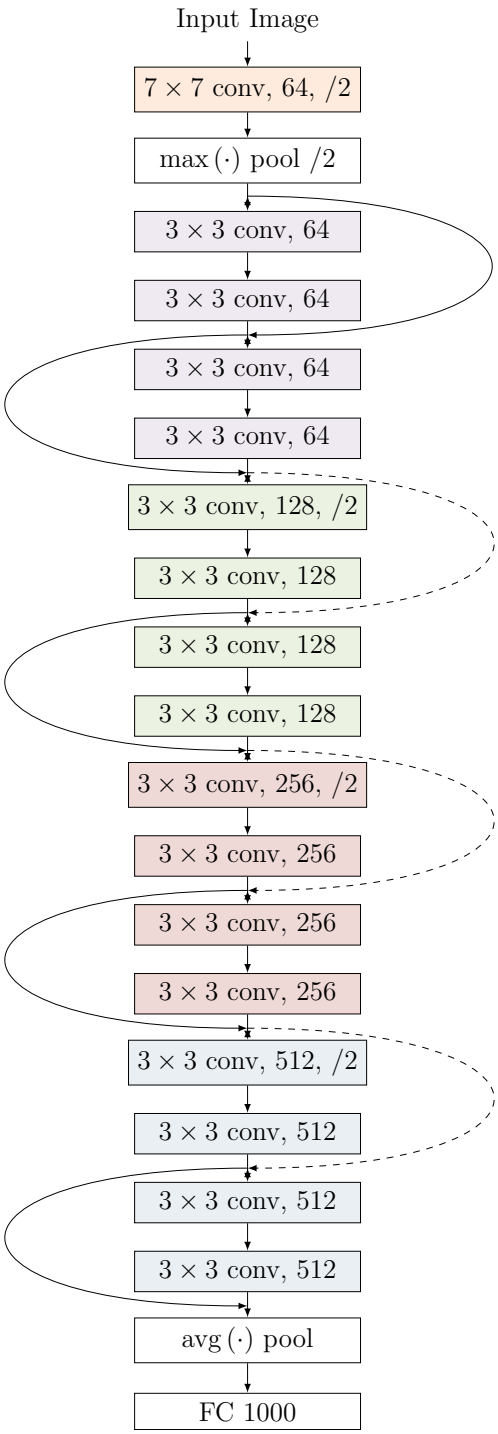


Figure 3.8: ResNet-18 Architecture. The dashed arrows denote a projection due to dimensionality reduction.

Chapter 4

Domain Adaptation

One of the main challenges in training deep neural networks for classification is the need for massive amounts of labelled data in order to make the model generalise well. However, large datasets are often expensive to be collected and annotated, and sometimes an adequate amount of data is unavailable.

Fortunately, even when it is difficult to obtain enough data from the domain of interest, it is possible to exploit data from other domains, which might be much cheaper to collect and annotate and in some cases could be even produced automatically. Nevertheless, the shift between the domain of interest (target domain) and the domain of the training data (source domain) heavily affects the performances of the model on target data.

Deep DA solves this problem by using unlabelled target data, which is usually less costly to collect due to the absence of the manual annotation process. Several DA methods have been explored in literature [75]. In this section we describe four of them.

4.1 Domain Adversarial Neural Network

Ganin *et. al.* [20] suggest promoting the generation of domain-invariant features through an adversarial learning approach. The proposed architecture, called Domain Adversarial Neural Network (DANN) (figure 4.1), is composed of a deep feature extractor G_f, a classifier G_y and a domain discriminator G_d. The domain discriminator is a binary classifier trained to distinguish source and target samples, while the classifier is trained to solve the main label prediction task.

Both the classifiers are fed with features extracted by G_f, but the discriminator G_d gets the features through a Gradient Reversal Layer (GRL). The GRL behaves like the identity function at forward time but multiplies the gradient by a negative constant during backpropagation.

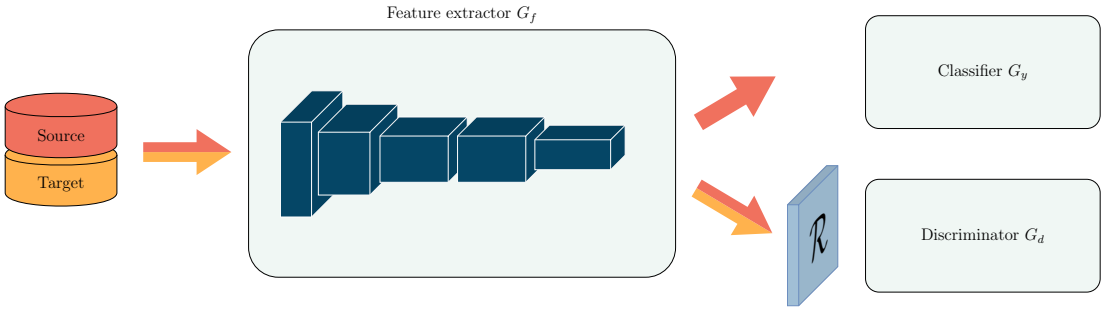


Figure 4.1: Domain Adversarial Neural Network. The features extracted by G_f are used by G_y to predict the sample classes and by G_d to discern between source and target samples. The gradient reversal layer $\mathcal{R}$ prevents G_f from learning features useful to make such distinction. The classifier G_y is trained using only source samples, while the discriminator G_d is trained using both source and target data.

We can define GRL formally as a "pseudo-function" $\mathcal{R}(\mathbf{x})$:

$$\begin{aligned} \mathcal{R}(\mathbf{x}) &= \mathbf{x} \\ \frac{\partial \mathcal{R}}{\partial \mathbf{x}} &= -\lambda \, \mathbf{I}, \end{aligned} \tag{4.1}$$

where $\mathbf{I}$ is the identity tensor. Reversing the gradient forces the feature extractor to make source and target as indistinguishable as possible for the domain discriminator, therefore generating similar feature distributions for both domains and reducing the domain shift.

The value of λ can either be constant or change during training. The authors propose the increasing scheme

$$\lambda = \frac{2}{1 + \mathrm{e}^{-\gamma \cdot p}} - 1, \tag{4.2}$$

where p linearly grows from 0 to 1 during training and γ is a hyperparameter.

4.1.1 Motivation

From a theoretical point of view, the use of a GRL for domain adaptation optimises the $\mathcal{H}$-divergence between source and target distribution.

Definition 4.1.1. $\mathcal{H}$-divergence (Ben-David *et. al.* [5, 6], Kifer *et. al.* [36]) *Given two domain distributions $\mathcal{D}_S^X$ and $\mathcal{D}_T^X$ over X, and a hypothesis class $\mathcal{H}$, the $\mathcal{H}$-divergence between $\mathcal{D}_S^X$ and $\mathcal{D}_T^X$ is*

$$\mathrm{d}_{\mathcal{H}}\left(\mathcal{D}_S^X, \mathcal{D}_T^X\right) \triangleq 2 \sup_{\eta \in \mathcal{H}} \left| \Pr_{\mathbf{x} \in \mathcal{D}_S^X} [\eta(\mathbf{x}) = 1] - \Pr_{\mathbf{x} \in \mathcal{D}_T^X} [\eta(\mathbf{x}) = 1] \right|.$$

The $\mathcal{H}$-divergence quantifies the capability of the hypothesis class $\mathcal{H}$ to distinguish from examples from the two distributions. Ben-David *et. al.* [6] proved that if $\mathcal{H}$ is symmetric, the empirical $\mathcal{H}$-divergence between two samples $S \sim \left(\mathcal{D}_S^X\right)^n$ and $T \sim \left(\mathcal{D}_T^X\right)^{n'}$ can be computed as

$$\hat{\mathrm{d}}_{\mathcal{H}}\left(S\,,\,T\right) \triangleq 2\left(1 - \min_{\eta \in \mathcal{H}}\left[\frac{1}{n}\sum_{i=1}^{n} I\left[\eta\left(\mathbf{x}_i\right) = 0\right] + \frac{1}{n'}\sum_{i=n+1}^{N} I\left[\eta\left(\mathbf{x}_i\right) = 1\right]\right]\right), \quad (4.3)$$

where $I\left(\cdot\right)$ is the indicator function defined as 1 if the argument is true, 0 otherwise.

In order to compute the exact empirical $\mathcal{H}$-divergence using (4.3), one should take into account all the possible classifiers of interest (i.e. belonging to $\mathcal{H}$), which is generally infeasible. However, Ben-David *et. al.* [6] suggest that an estimation of $\hat{\mathrm{d}}$ can be computed by training a model G_d to predict if a given sample belongs to the source or target distribution. Denoting with ϵ the generalisation error of G_d, $\hat{\mathrm{d}}_{\mathcal{H}}$ can be approximated as

$$\hat{\mathrm{d}}_{\mathcal{H}} \sim \hat{\mathrm{d}}_{\mathcal{A}} \triangleq 2\left(1 - 2\epsilon\right), \quad (4.4)$$

where $\hat{\mathrm{d}}_{\mathcal{A}}$ is called *Proxy* $\mathcal{A}$*-distance* and $\mathcal{A} \subseteq X$.

Let us consider a shallow neural network composed by a single hidden layer

$$G_f\left(\mathbf{x}\,;\,\mathbf{W},\mathbf{b}\right) = \sigma\left(\mathbf{W}\cdot\mathbf{x} + \mathbf{b}\right), \quad (4.5)$$

that maps the input $\mathbf{x}$ into a D-dimensional space, and an output layer $G_y : \mathbb{R}^D \rightarrow \left[0\,,\,1\right]^L$

$$G_y\left(G_f\left(\mathbf{x}\,;\,\mathbf{W},\mathbf{b}\right)\,;\,\mathbf{V},\mathbf{c}\right) = \mathrm{softmax}\left(\mathbf{V}\cdot G_f\left(\mathbf{x}\,;\,\mathbf{W},\mathbf{b}\right) + \mathbf{c}\right). \quad (4.6)$$

$\left(\mathbf{W},\mathbf{b},\mathbf{V},\mathbf{c}\right)$ are the parameters to optimise when training the model. Assuming to use the classification loss

$$\mathcal{L}_y\left(G_y\left(G_f\left(\mathbf{x}_i\right)\;\;y_i\right)\right) = \log\frac{1}{G_y\left(G_f\left(\mathbf{x}_i\right)\right)_{y_i}}, \quad (4.7)$$

where $\left(\mathbf{x}_i\,,\,y_i\right)$ is a training sample, fitting the model consists of solving the optimisation problem

$$\min_{\left(\mathbf{W},\mathbf{b},\mathbf{V},\mathbf{c}\right)}\left[\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}_y^i\left(\mathbf{W},\mathbf{b},\mathbf{V},\mathbf{c}\right) + \lambda\,\mathcal{L}_r\left(w,b\right)\right] \quad (4.8)$$

where $\lambda\,\mathcal{L}_r$ is a regularisation term.

Ganin *et. al.*suggest defining a domain discriminator $G_d : \mathbb{R}^D \rightarrow \left[0\,,\,1\right]$

$$G_d\left(G_f\left(\mathbf{x}\,;\,\mathbf{W},\mathbf{b}\right)\,;\,\mathbf{u},z\right) = \sigma\left(\mathbf{u}\cdot\left(G_f\left(\mathbf{x}\,;\,\mathbf{W},\mathbf{b}\right) + z\right)\right), \quad (4.9)$$

trained by using the loss

$$\mathcal{L}_d\left(G_d\left(G_f\left(\mathbf{x}_i\right)\ d_i\right)\right) = d_i \log \frac{1}{G_d\left(G_d\left(\mathbf{x}_i\right)\right)} + \left(1 - d_i\right) \log \frac{1}{1 - G_d\left(G_d\left(\mathbf{x}_i\right)\right)}, \quad (4.10)$$

with d_i the binary domain label for the sample $\mathbf{x}_i$. We can then define a DA regularisation term in equation (4.8)

$$\mathcal{L}_r\left(\mathbf{W}, \mathbf{b}\right) = \max_{\mathbf{u}, z} \left[-\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_d^i\left(\mathbf{W}, \mathbf{b}, \mathbf{u}, z\right) - \frac{1}{n'} \sum_{i=n+1}^{N} \mathcal{L}_d^i\left(\mathbf{W}, \mathbf{b}, \mathbf{u}, z\right) \right]. \quad (4.11)$$

From (4.4) we have that $2\left(1 - \mathcal{L}_r\left(\mathbf{W}, \mathbf{b}\right)\right)$ approximates $\hat{\mathrm{d}}_{\mathcal{H}}$. The complete loss can be rewritten as

$$\begin{aligned} \mathcal{L}\left(\mathbf{W}, \mathbf{b}, \mathbf{V}, \mathbf{c}, \mathbf{u}, z\right) = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_y^i\left(\mathbf{W}, \mathbf{b}, \mathbf{V}, \mathbf{c}\right) - \\ \lambda \left(\frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_d^i\left(\mathbf{W}, \mathbf{b}, \mathbf{u}, z\right) + \frac{1}{n'} \sum_{i=n+1}^{N} \mathcal{L}_d^i\left(\mathbf{W}, \mathbf{b}, \mathbf{u}, z\right) \right), \end{aligned} \quad (4.12)$$

from which we can conclude that training the classification model $G_y \circ G_f$ while regularising by $\mathcal{H}$-divergence minimisation, leads to minimise the loss with respect to a subset of the parameters to maximise it with respect to the others.

4.2 Deep Adaptation Network

In order to reduce the domain shift of the deep features, a direct approach could consist of adding a loss function to penalise the discrepancy of source and target features. Such discrepancy is more significant in the higher layers of the network, which tends to learn more specific features.

Long *et. al.* [49] encourage the network to align source and target by minimising the Maximum Mean Discrepancy (MMD) [27] between deep features in the higher FC layers. MMD is a measure of the discrepancy between two distributions. More specifically, it is a distance defined on the space of probability by measuring the distance between their *mean embeddings*. Given a kernel k and its associated Reproducing Kernel Hilbert Space (RKHS) $\mathcal{H}_k$, the distance between the mean embeddings of two distributions p and q is defined as

$$\mathrm{d}^2\left(p, q\right) \triangleq \left\| \mathbf{E}_p\left[\phi\left(\mathbf{x}^s\right)\right] - \mathbf{E}_q\left[\phi\left(\mathbf{x}^t\right)\right] \right\|_{\mathcal{H}_k}^2. \quad (4.13)$$

The authors propose a Deep Adaptation Network (DAN) (figure 4.2), composed of a convolutional feature extractor and a FC classifier.

Batches of data coming from both domains are fed into the network. Source labelled samples are used to train the classifier by backpropagating the classification loss. The output resulting from target data is discarded, but the hidden features are used to compute the DA loss

$$\mathcal{L}_{\text{MMD}} \triangleq \sum_{l \in L} \mathrm{d}^2 \left(\mathcal{D}_s^l \, , \, \mathcal{D}_t^l \right), \tag{4.14}$$

where L is the set of the adapted layers and $\mathcal{D}_s^l$, $\mathcal{D}_t^l$ are the distributions of the features at layer l for source and target respectively. The loss to minimise is then

$$\mathcal{L} = \mathcal{L}_c + \lambda \, \mathcal{L}_{\text{MMD}}, \tag{4.15}$$

with $\mathcal{L}_c$ being the main classification loss and λ a trade-off hyperparameter.

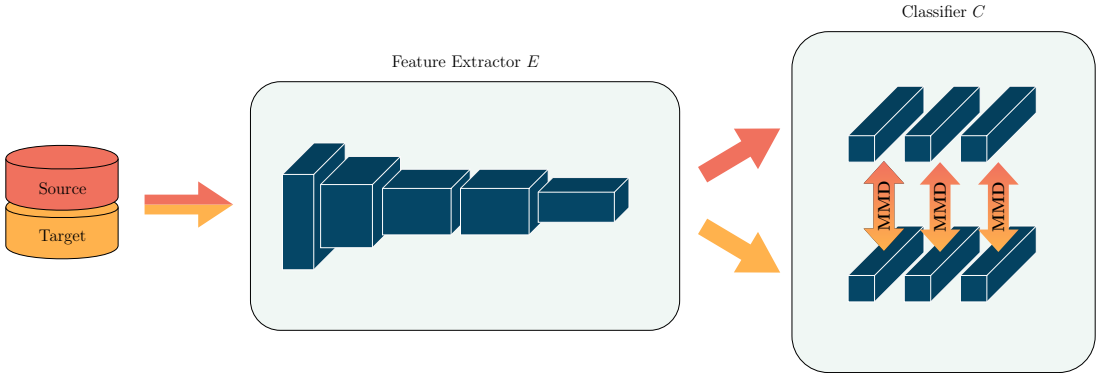


Figure 4.2: Architecture of a Deep Adaptation Network. The classifier C takes as input the features extracted by the deep feature extractor E. The Maximum Mean Discrepancy loss is computed between the features of the Fully Connected layers. Note that C is a siamese network (i.e. the two branches share the same weights).

Instead of using basic MMD with a specific kernel, the authors follow a multi-kernel approach implementing Multiple Kernel Maximum Mean Discrepancy (MK-MMD) (Gretton *et. al.* [26]), with a kernel defined as the convex combination of m positive-semidefinite kernels

$$\mathcal{K} \triangleq \left\{ k = \sum_{u=1}^{m} \beta_u k_u \; : \; \sum_{u=1}^{m} \beta_u = 1 \, , \; \beta_u \geq 0 \; \forall u \right\}, \tag{4.16}$$

where the coefficients $\{\beta_u\}$ are selected with the constraint that the resulting composite kernel is characteristic.

MK-MMD can be computed using (4.14) with complexity $O(n^2)$. Since CNNs are usually trained with large datasets, the exact discrepancy is therefore too

computationally expensive to compute. Instead, the linear-complexity unbiased estimate

$$\hat{\mathrm{d}}^2(p,q) \triangleq \frac{2}{n_s} \sum_{i=1}^{\frac{n_s}{2}} g_k\left(\mathbf{x}_{2i-1}^s, \mathbf{x}_{2i}^s, \mathbf{x}_{2i-1}^t, \mathbf{x}_{2i}^t\right) \tag{4.17}$$

is used, where

$$\begin{aligned} g_k\left(\mathbf{x}_{2i-1}^s, \mathbf{x}_{2i}^s, \mathbf{x}_{2i-1}^t, \mathbf{x}_{2i}^t\right) = & \, k\left(\mathbf{x}_{2i-1}^s, \mathbf{x}_{2i}^s\right) \\ & + k\left(\mathbf{x}_{2i-1}^t, \mathbf{x}_{2i}^t\right) \\ & - k\left(\mathbf{x}_{2i-1}^s, \mathbf{x}_{2i}^t\right) \\ & - k\left(\mathbf{x}_{2i-1}^t, \mathbf{x}_{2i}^s\right). \end{aligned} \tag{4.18}$$

4.3 Self-supervised Rotation

Self-supervision employs supervised learning algorithms in settings where external pre-labelled data is not available. The idea is to define an artificial problem, tailored to be trained by using supervision coming from the data itself. Training a model to solve this auxiliary (or "pretext") task will hopefully make it learn deep representations useful to solve the main task.

Several pretext tasks have been explored in the literature, like image inpainting [59], solving jigsaw puzzles [55], grey-scale image colourisation [78], predicting the location of a patch of the image [17] or predicting audio from video [57]. Gidaris *et. al.* [22] propose to rotate unlabelled training images by random multiples of 90° and to predict the rotation by using a four-way classifier.

This kind of approach has been mainly used to exploit transfer learning: the network is first trained to solve the pretext task, then it is fine-tuned to solve the main task. More recently, self-supervision has been used in other settings, like DA. Several studies [12, 21] show how it is possible to encourage the network to learn transferable features by adding a secondary self-supervised task, which can be solved without using target labels but at the same time helps to learn domain-invariant features.

Following the example of Gidaris *et. al.*, Xu *et. al.* [76] use absolute rotation: during training input images are randomly rotated by multiples of 90°. The secondary task aims to predict the rotation angle, thus promoting the generation of features useful to solve the main classification task on the target domain. More formally, given a training sample $\mathbf{x}$ (from either source or target domain) and a randomly selected $j \in [0, 3]$, the network takes as input the transformed example

$$\tilde{\mathbf{x}} = \mathrm{rot}\left(\mathbf{x}, j \cdot 90°\right) \tag{4.19}$$

and uses it to predict j.

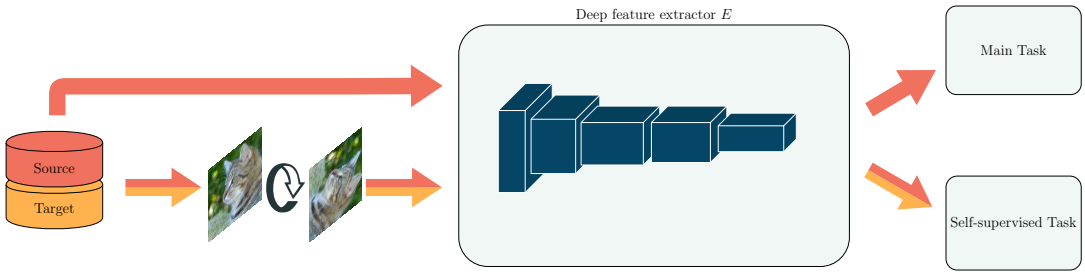


Figure 4.3: Domain Adaptation through image rotation prediction. Input images from both domains are randomly rotated and the self-supervised task is trained to predict the rotation angle.

4.4 Adaptive Feature Normalisation

The approach proposed by Xu *et. al.* [77]. is based on two hypotheses:

- **Misaligned-Feature-Norm Hypothesis**: the domain shift between source and target features causes the feature-norm expectations to be misaligned. Encouraging the norms to be similar is expected to cause the network to learn more transferable features, thus yielding better performances on target.
- **Smaller-Feature-Norm Hypothesis**: the domain shift causes the target features to be less informative. Empirically, non-adapted target features can be shown to have smaller norms, suggesting to encourage the network to generate higher-norm features.

Given the first hypothesis, the authors propose Maximum Mean Feature Norm Discrepancy (MMFND) to measure the distances between the distributions of source and target. With reference to the network shown in figure 4.4, composed by the deep feature extractor G and the FC classifier $F = F_f \circ F_y$, the MMFND can be defined as:

$$\text{MMFND}\left(\mathcal{H}, \mathcal{D}^s, \mathcal{D}^t\right) = \sup_{h \in \mathcal{H}} \left(\frac{1}{N_s} \sum_{\mathbf{x}_i \in \mathcal{D}^s} h\left(\mathbf{x}_i\right) - \frac{1}{N_t} \sum_{\mathbf{x}_i \in \mathcal{D}^t} h\left(\mathbf{x}_i\right) \right), \tag{4.20}$$

where $\mathcal{H}$ is the space of all the possible $h = \|\cdot\|_2 \circ F_f \circ G$.

Without any restrictions on h, the MMFND distance will probably diverge or increase to high values. This problem can be addressed by restricting the mean feature norm, which is bound to converge to a scalar R. Both source and target feature norms will then converge to the same value, thus making the MMFND vanish. The constraint is enforced through an additional loss term. The cost function to optimise is then

$$\mathcal{L} = \mathcal{L}_c + \lambda \, \mathcal{L}_{\text{AFN}}, \tag{4.21}$$

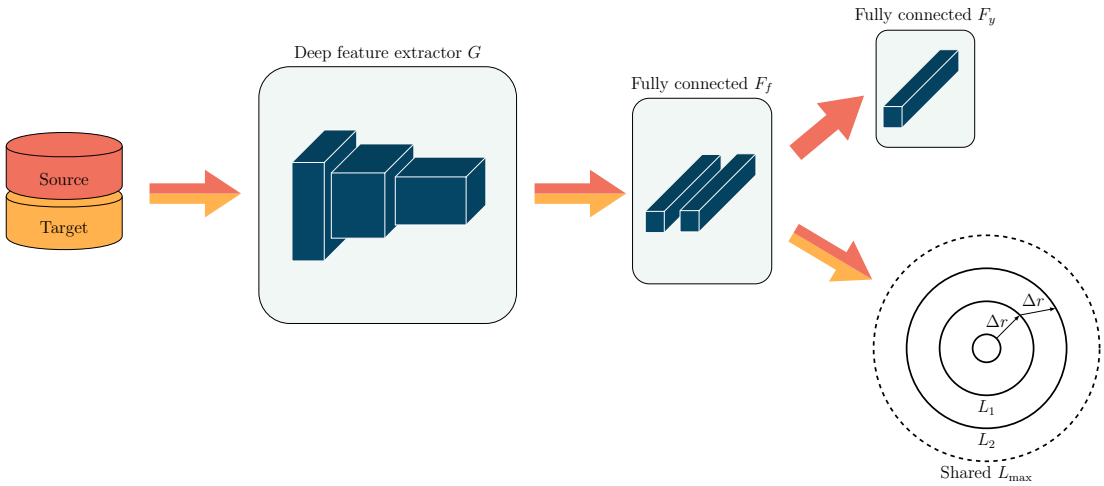


Figure 4.4: Example of a network adapted by using Adaptive Feature Normalisation. G is the backbone, a convolutional general feature extractor. The extracted features are fed into the Fully Connected classifier $F = F_f \circ F_y$. Feature-norm adaptation works by adding a constraint to the features extracted before the last Fully Connected layer of F, F_y.

where $\mathcal{L}_c$ is the main classification loss, λ is a trade-off hyperparameter and

$$\mathcal{L}_{\text{AFN}} = \mathcal{L}_d \left(\frac{1}{N_s} \sum_{\mathbf{x}_i \in \mathcal{D}^s} h(\mathbf{x}_i) \ , \ R \right) + \mathcal{L}_d \left(\frac{1}{N_t} \sum_{\mathbf{x}_i \in \mathcal{D}^t} h(\mathbf{x}_i) \ , \ R \right), \tag{4.22}$$

with $\mathcal{L}_d$ being the Euclidean distance.

Coherently with the Smaller-Feature-Norm Hypothesis, empirical evidence proves that higher values of R yield more transferable features. However, the value of R cannot be arbitrarily increased: large values would cause the gradient of the feature-norm loss to get extremely large, leading the model to explode. In order to overcome this limit the authors developed a variant of the method, called Stepwise Adaptive Feature Norm (SAFN), in contrast to the plain version, named Hard Adaptive Feature Norm (HAFN).

Instead of encouraging the feature norm to converge to a fixed value R, the new approach consists of pushing the norm to increase progressively. The proposed loss function is

$$\mathcal{L}_{\text{AFN}} = \frac{1}{N_s + N_t} \sum_{\mathbf{x}_i \in \mathcal{D}_s \cup \mathcal{D}_t} \mathcal{L}_d \left[h \left(\mathbf{x}_i \ , \ \theta_0 \right) + \Delta r \ , \ h \left(\mathbf{x}_i \ , \ \theta \right) \right], \tag{4.23}$$

were θ_0 and θ are the model parameters in the last iteration and current iteration respectively.

Chapter 5

Multimodal Domain Adaptation

Our goal is to perform DA by exploiting the existing relationship between the two modalities of an image, RGB and Depth. We work in a classification setting, so we are interested in predicting the labels associated with target images using only labelled source data and unlabelled target data.

Our approach consists of adding a secondary "self-supervised" inter-modal task to the main classification one. The purpose of the self-supervised task is to force the network to learn domain-invariant features that will be useful to classify target data correctly. Several secondary tasks could be used. We focus on predicting the relative rotation between the two modalities. Before being fed to the network, the RGB and the Depth images are randomly and independently rotated by multiples of 90°. The secondary classifier is trained to recognise the relative rotation between the two. Figure 5.1 illustrates the architecture of our network, which is composed of three parts:

- The two-stream deep convolutional **feature extractor** E, which is composed of two single-modality feature extractors (E_c and E_d)
- The **main classifier** M
- The **secondary classifier** P

5.1 Feature extraction and main task

The feature extractor E is composed of two independent convolutional networks with the same architecture, E_c and E_d, aimed to extract features from RGB ($\mathbf{x}^c$) and Depth ($\mathbf{x}^d$) images. Given the input RGB-D image $\mathbf{x} = \left(\mathbf{x}^c, \mathbf{x}^d\right)$, we compute the features

$$\mathbf{h} = \left(\mathbf{h}^c, \mathbf{h}^d\right) = \left(E_c\left(\mathbf{x}^c\right), E_d\left(\mathbf{x}^d\right)\right). \tag{5.1}$$

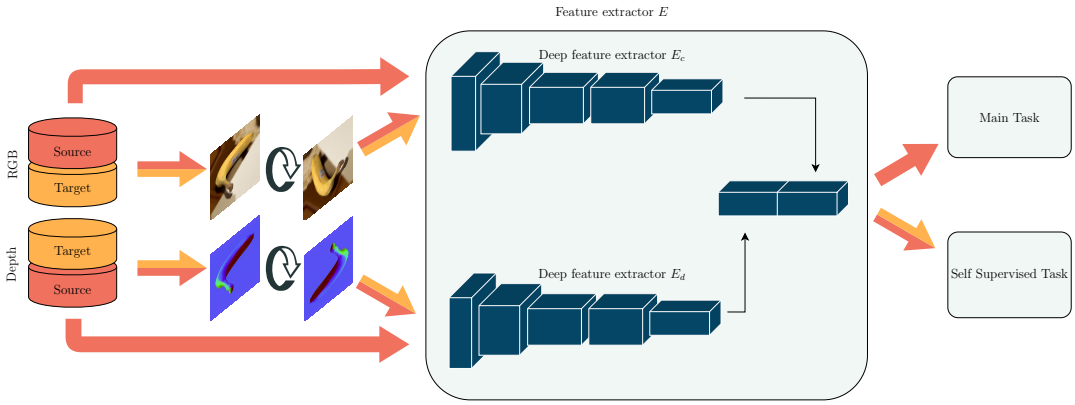


Figure 5.1: Source labelled and non-rotated images are used to train the feature extractors and the main classifier for object classification. Both source and target samples are randomly rotated and used to train the feature extractors and the secondary classifier, in order to predict the relative rotation between modalities.

The extracted features $\mathbf{h}$, resulting from the concatenation of the features extracted from RGB and Depth, are then fed to the main head M, a FC network used to predict the object label.

5.2 Secondary task

The relative rotation classifier P takes as input the features extracted by the rotated modalities and tries to guess the relative rotation between the two. This inter-modal task makes the network learn a relationship between the modalities, resulting in the extraction of domain-invariant features.

Another benefit of using relative rotation over simple rotation is that it does not require to make any assumptions on the dataset. In contrast, a self-supervised task based on absolute rotation can only work under the hypothesis that the pose of the objects portrayed in the dataset is coherent in the majority of the images. For some datasets this is not the case, and even when this requirement is satisfied it prevents from using random rotation as data augmentation technique.

Given an RGB-D sample $\mathbf{x} = \left(\mathbf{x}^c, \mathbf{x}^d\right)$, we randomly define $j, k \in [0, 3]$. The transformed sample which is fed into the network is then

$$\tilde{\mathbf{x}} = \left(\tilde{\mathbf{x}}^c, \tilde{\mathbf{x}}^d\right) = \left(\text{rot}\left(\mathbf{x}^c, j \cdot 90°\right), \text{rot}\left(\mathbf{x}^d, k \cdot 90°\right)\right), \tag{5.2}$$

where $\text{rot}(\cdot, \alpha)$ is a clockwise rotation by α degrees around the centre of the image. Let then z be the hot-encoded relative rotation between the two images

$$z = \text{one\_hot}\left((k - j) \mod 4\right). \tag{5.3}$$

The self-supervised task consists of predicting z given the transformed $\tilde{\mathbf{x}}$, which gives the number of times the RGB image has to be rotated by 90° in order to be aligned to the Depth one.

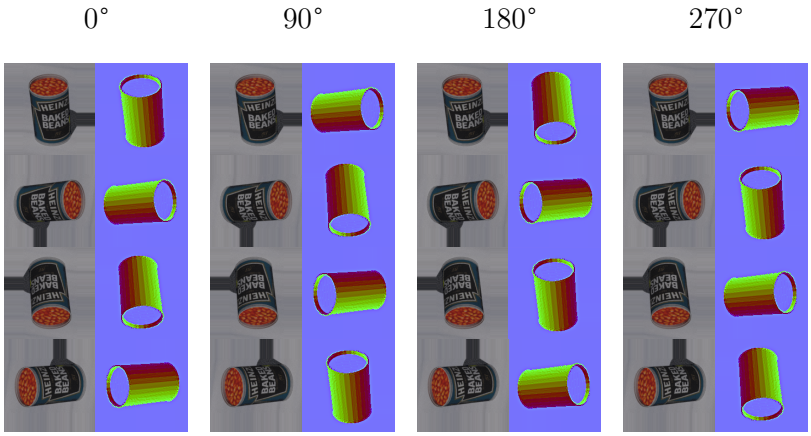


Figure 5.2: All the 16 possible combinations of relative rotations between modalities.

5.3 Loss

During the training, the optimisation algorithm updates the weights in order to minimise the value of the loss function:

$$\mathcal{L} = \mathcal{L}_m + \lambda\, \mathcal{L}_p + \epsilon\, \mathcal{L}_r. \tag{5.4}$$

$\mathcal{L}_m$ is the main classification loss, it is needed to train the model to classify the objects and it is defined as a cross-entropy loss:

$$\mathcal{L}_m = -\frac{1}{N_s} \sum_{i=1}^{N_s} y_i^s \cdot \log\left(\hat{y}_i^s\right). \tag{5.5}$$

$\mathcal{L}_p$ is the classification loss of the self-supervised task and it is weighted by the coefficient λ. Like the main loss, it is a cross-entropy function, defined as

$$\mathcal{L}_p = -\frac{1}{\tilde{N}_s} \sum_{i=1}^{\tilde{N}_s} z_i^s \cdot \log\left(\hat{z}_i^s\right) - \frac{1}{\tilde{N}_t} \sum_{i=1}^{\tilde{N}_t} z_i^t \cdot \log\left(\hat{z}_i^t\right). \tag{5.6}$$

$\mathcal{L}_r$ is a generic regularisation term, weighted by the coefficient ϵ.

Chapter 6

Implementation Details

In this section we describe the details of our implementation of the network and the training algorithm. Specifying implementation choices that may affect experiment results is needed for reproducibility, but it is important to remember that the DA method exposed in chapter 5 is independent of these details.

For our experiments, we defined the convolutional feature extractors E_c and E_d as the ResNet-18 (section 3.3) without the last FC and average pooling layers.

6.1 Main classifier

The main classifier M takes as input the features extracted by E and solves a C-way classification problem, with C being the number of classes in the dataset. The classifier is a two-layer FC network, following an average pooling layer. The first FC layer is composed of 1000 units, uses ReLU activation function and implements both batch normalisation and dropout, while the second FC layer (C units) uses Softmax activation. The detailed structure of the network is shown in figure 6.1.

6.2 Secondary classifier

Similarly to M, the secondary classifier P takes as input the features extracted by E and solves a 4-way classification problem. For this classifier we used a different architecture, implementing it as a CNN composed of two stacked convolutional layers and two FC layers, without global pooling.

The convolutional layers are defined to have 100 kernels (1×1 the first, 3×3 the second). The FC layers have 100 and 4 units respectively. Both the convolutional layers and the first FC one implement ReLU activation function, batch normalisation and dropout. The last layer uses Softmax activation function. The classifier architecture is described in figure 6.2.

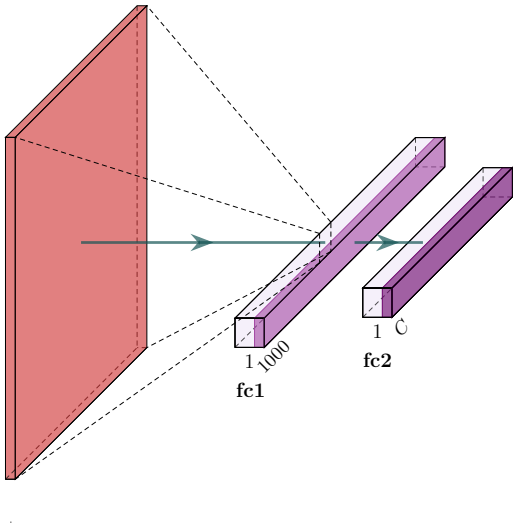


Figure 6.1: Architecture of the main classifier. Batch normalisation and dropout layers are not represented.

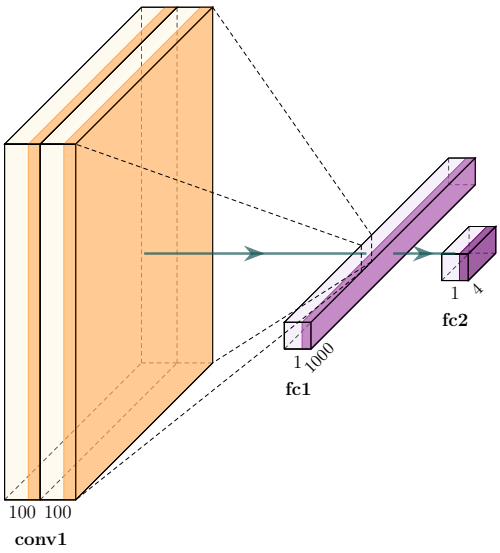


Figure 6.2: Architecture of the relative rotation classifier.

6.3 Training

The network is trained using SGD with momentum $\mu = 0.9$, learning rate $\eta = 3{\times}10^{-4}$ and weight decay 0.05 (see 3.1.3). We add entropy-minimisation [51] as a DA specific regularisation loss, as described in section 5.3. Both M and P have dropout layers with $p = 0.5$.

All the parameters of the network are optimised during training. The weights of E_c and E_d are initialised from models pre-trained on ImageNet [16], while M and P are randomly set using Xavier initialisation [23].

In each iteration, the network is fed with synchronised pairs of RGB and Depth images, independently for object classification and DA. The detailed process is described by algorithm 1.

Algorithm 1 Training algorithm

Require:
Labelled source dataset $S = \{(\mathbf{x}_i^s, y_i^s)\}_{i=1}^{N_s}$
Unlabelled target dataset $T = \{\mathbf{x}_i^t\}_{i=1}^{N_t}$
Ensure:
Object class prediction for target data $\left\{\hat{y}_i^t\right\}_{i=1}^{N_t}$
procedure TRAIN(S, T)
Compute transformed source dataset $\widetilde{S} = \{(\tilde{\mathbf{x}}_i^s, z_i^s)\}_{i=1}^{\widetilde{N}_s}$
Compute transformed target dataset $\widetilde{T} = \left\{\left(\tilde{\mathbf{x}}_i^t, z_i^t\right)\right\}_{i=1}^{\widetilde{N}_t}$
for each iteration **do**
Load batch from S
Compute main classification loss $\mathcal{L}_m$
Load batches from S and T
Compute DA loss $\mathcal{L}_p$ and regularisation loss $\mathcal{L}_e$
Update weights of M from $\nabla\mathcal{L}_m$
Update weights of P from $\lambda\nabla\mathcal{L}_p$ and $\epsilon\nabla\mathcal{L}_e$
Update weights of E from $\nabla\mathcal{L}_m$, $\lambda\nabla\mathcal{L}_p$ and $\epsilon\nabla\mathcal{L}_e$
end for
end procedure
procedure TEST(T)
for each $\mathbf{x}_i^t$ in T **do**
Compute $\hat{y}_i^t = M(E(\mathbf{x}_i^t))$
end for
end procedure

The images in the datasets we use for our experiments are not square-shaped. We scale to 256px the longer side of the pictures and get 256×256 images by tiling the pixels of the longer sides. Before being fed to the network during training, the samples are randomly cropped to 224×224. Random horizontal flip (with $p = 0.5$) is used as an additional data augmentation technique. At test time, images are centre-cropped to 224×224.

6.4 Preprocessing

Before the steps described in section 6.3, additional preprocessing is required for Depth data in order to exploit effectively transfer learning.
Depth images are thus preprocessed using *surface normal++* encoding [1, 2], a colourisation technique which maps a 1-channel depth image to a 3-channels RGB-like space.

Because depth images are often incomplete and have missing values, a median filter is first applied considering only non-missing values in a small neighbourhood of each missing pixel (Lai *et. al.* [41]), therefore filling all the missing values without significantly blurring the image. In order to solve the border problems that could manifest when applying such filter, padding with border replication is used. A second, bilateral, filter is applied to reduce noise, then the actual surface normal vectors are computed pixel-wise. For each pixel two vectors

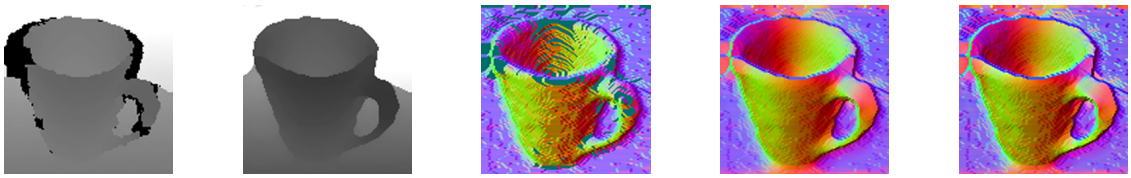
$$\mathbf{v_1} = \left(1\ ;\ 0\ ;\ \frac{\partial z}{\partial x}\right),$$
$$\mathbf{v_2} = \left(0\ ;\ 1\ ;\ \frac{\partial z}{\partial y}\right)$$

are defined. The "surface normal" vector is then computed as the cross product

$$\mathbf{n} = \mathbf{v_1} \times \mathbf{v_2} = \left(-\frac{\partial z}{\partial x}\ ;\ \frac{\partial z}{\partial y}\ ;\ 1\right). \tag{6.1}$$

The vector $\mathbf{n}$ is then normalised as $\hat{\mathbf{n}} = \frac{\mathbf{n}}{\|\mathbf{n}\|_2}$ and remapped from the $[-1\,,\,1]^3$ space to $[0\,,\,255]^3$ to be interpreted as an RGB triplet. Finally, the resulting image is sharpened by using a filter which increases contrast in high-frequency zones, like the edges.

Figure 6.3 shows the steps of the colourisation procedure.



(a) (b) (c) (d) (e)

Figure 6.3: Surface Normal++ steps. (a) Original depth image, (b) missing values are filled by using the median filter, (c) surface normal colourised image without bilateral filter, (d) surface normal colourised image with bilateral filter, (e) sharpened final result. Images from [2].

Chapter 7

Experiments

In this section, we describe different experiments to assess the capabilities of our Relative Rotation (RR) approach and the potential of inter-modal self-supervision in general. For this purpose, our DA method is compared with four existing methods described in chapter 4.

7.1 Datasets

The first challenge developing an RGB-D DA method is the lack of standard benchmark datasets for evaluation. In order to evaluate DA methods, two distinct datasets presenting the same object classes are needed. Source and target datasets must have different origins for the domain shift to be considerable. More specifically, we are interested in a synthetic-source and real-target scenario.

7.1.1 synROD - ROD

Due to the absence of synthetic data suitable for this purpose, we collect a new synthetic dataset called synROD and we model it on the existing dataset ROD [40]. ROD is the primary reference dataset for object recognition in robotics [13, 19, 48]. It is composed of 41,877 RGB-D pictures taken from 300 objects divided into 51 classes. The images are captured from different angles with a camera placed at about 1 m from the object.

In order to synthesize the new dataset, we select 303 textured models from public 3D repositories. The models are then rendered using Blender [8] with a ray-tracing engine, to get photorealistic images. Each rendered scene is generated by randomly selecting a subset of the available objects and by dropping them on a virtual plane using a physics simulator. The size of the subset varies from 5 to 20, and the model selection is conditioned in order to ensure balanced classes. The background is created by using random images from the Microsoft COCO dataset [47].

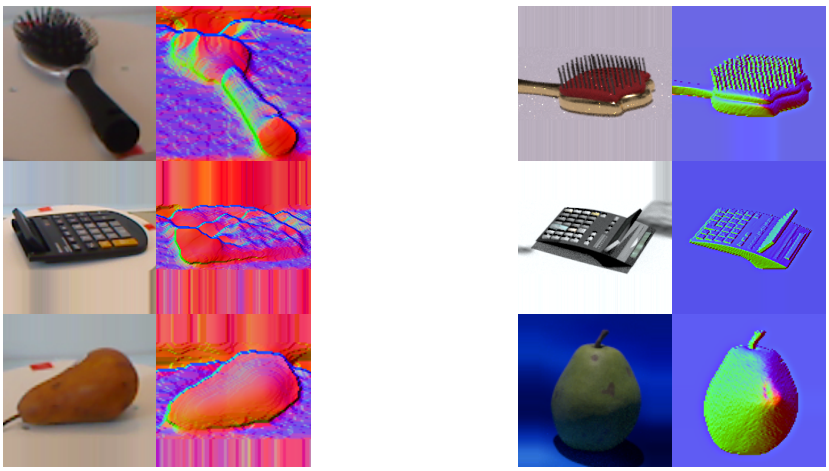


Figure 7.1: Examples from the two datasets ROD [40] (on the left) and synROD.

7.1.2 synHB - HB

HomebrewedDB [35] is a dataset featuring objects from 33 categories. The dataset contains 3D object models obtained using a scanner and validation sequences consisting of images captured with two RGB-D cameras. HB was conceived for 6D pose estimation, but we fix it for instance recognition and DA.

For this purpose, we crop the validation scenes to get images containing only the object instance. We refer to the resulting 22,935 samples as realHB. This part of the dataset is aimed to be the target domain in our DA setting.

The synthetic counterpart of realHB, synHB, is generated by rendering the scanned object models following the same procedure used for synROD.

7.2 Baselines

We compare our novel method with four existing DA techniques: GRL (section 4.1), MMD (4.2), Rotation (4.3) and AFN (4.4). GRL, MMD and AFN are general-purpose methods, since they are not limited to computer vision tasks. The first two are the most established DA algorithms, and the latter can arguably be considered state of the art. *Rotation* is specifically designed for use in computer vision problems, and it is the most relevant to our method.

Since RGB-D DA has not been explored in literature yet and none of the methods mentioned above was designed for use with multimodal data, we adopt two approaches to evaluate them on RGB-D data:

- **RGB-D** We first train until convergence two single-modality classifiers, adapting RGB and Depth separately. Then we freeze the feature extractors and

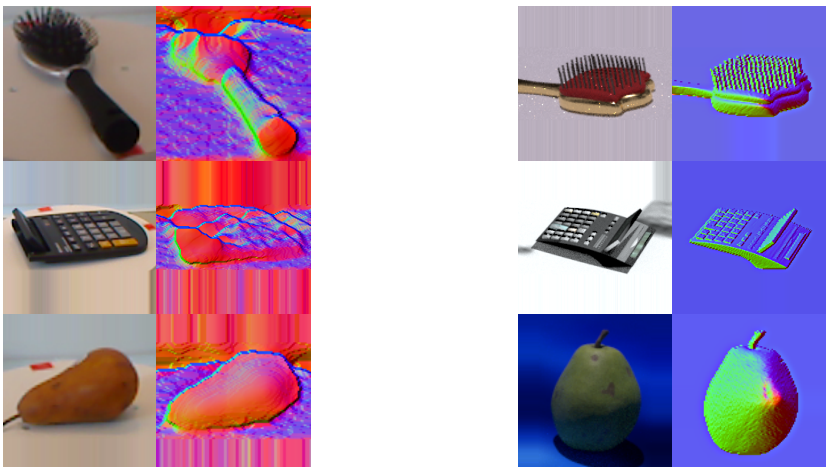


Figure 7.2: Examples from the two datasets HB [35] (on the left) and synHB.

train a single FC layer fed with the concatenation of the extracted features (figure 7.3).

- **RGB-D e2e** We perform DA while training a single classifier for both modalities, concatenating the extracted features (figure 7.4).

7.3 Results

This section reports the results obtained by confronting the selected baselines and variations of our method. The main challenge when evaluating DA methods is the need for target labels in order to assess the actual performances on target data. Labels are usually available for benchmark datasets like realHB and ROD, but they should not be used to select the best model because they are not available in a real-world scenario. Therefore we use target labels only after having chosen the best model to use for the test. Since there is not a standard protocol for model selection in a DA context, we proceed by manually choosing models by heuristically inspecting evaluation source losses. More specifically, we adopt the following criteria:

- For *GRL*, we identify an interval of epochs where the discriminator accuracy is stable around a reasonable value (~ 0.5). In such interval we select the model having the minimum classification loss on the source test set.

- For *MMD*, *Rotation* and *Relative Rotation*, we find a compromise between the minima of classification and DA specific loss, focusing on minimising the classification loss.

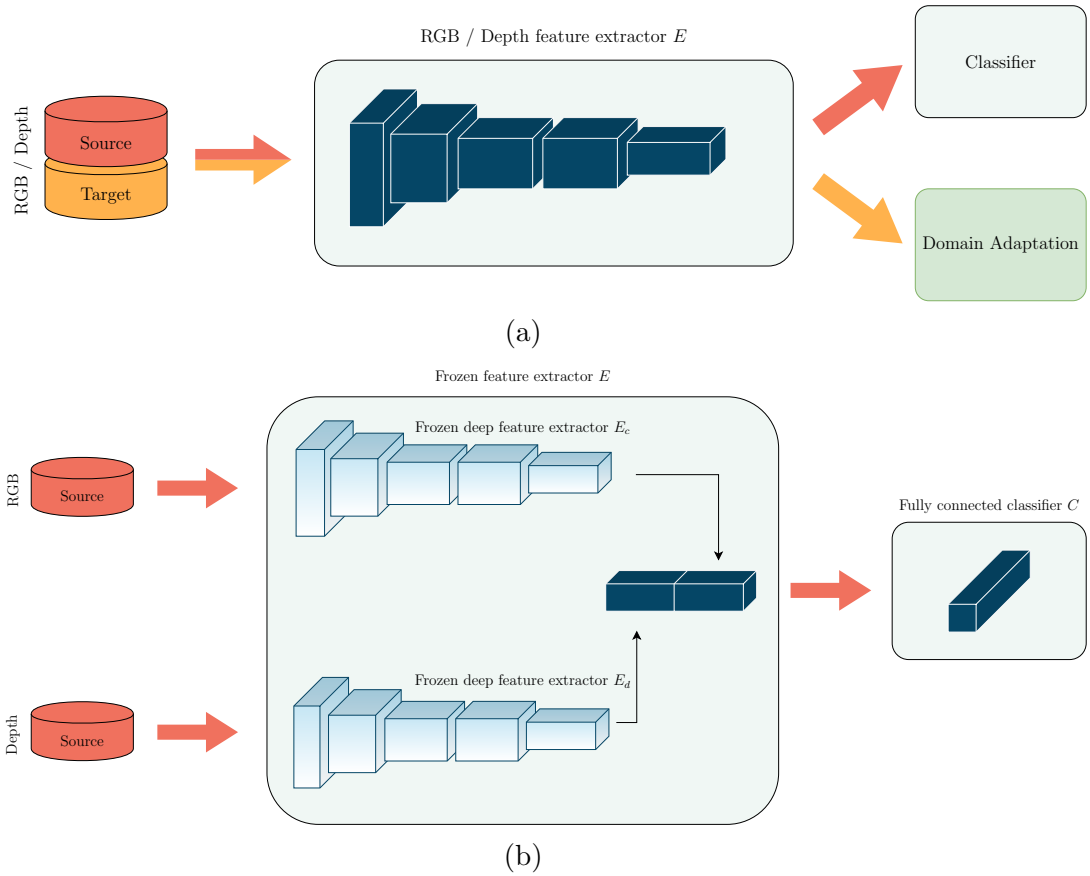


Figure 7.3: RGB-D setting. First, two single-modality feature extractors E_c and E_d are trained using both labelled source data and unlabelled target data to perform DA with the selected method. For this purpose, the output of E is fed into a classifier M (a). After convergence a new network is built and trained, using E_c and E_d as frozen feature extractors and a FC layer C as classifier (b).

- For *AFN* we just select the model exhibiting the minimum test classification loss.

7.3.1 Ablation study

In order to understand how different components of our method influence the overall performance, we perform an ablation study. Table 7.1 presents the accuracy obtained with three different versions of Relative Rotation:

- **Target rotation**: we use labelled source data only for classification and DA is performed solely with unlabelled target samples.
- **FC classifier**: the self-supervised task is solved using the same FC classifier

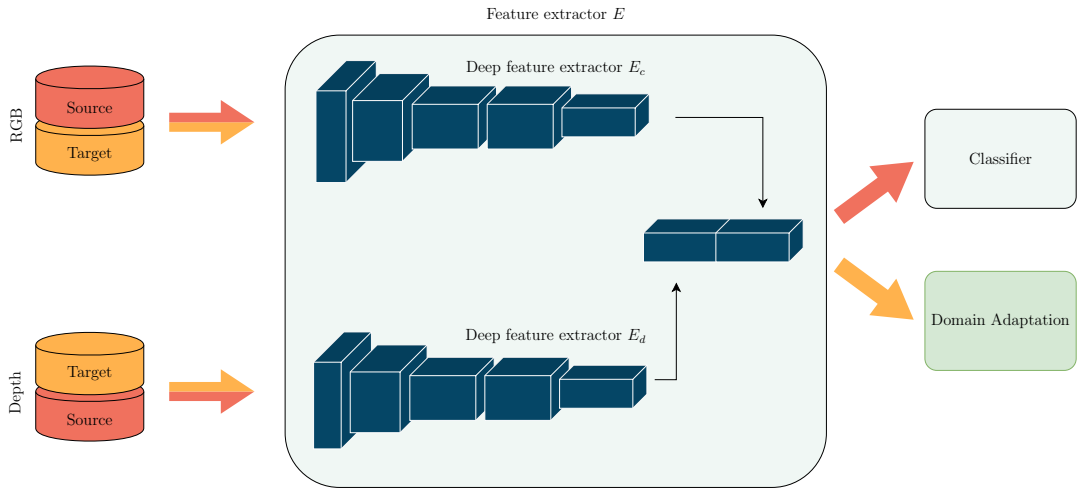


Figure 7.4: RGB-D e2e setting. The feature extractors E_c and E_d are trained simultaneously while performing Domain Adaptation on the concatenation of RGB and Depth features.

we use for the primary task. Both source and target are used for DA.

- **Ours**: the final version of our method. We use a convolutional classifier in order to solve the secondary task (see section 6.2).

Method	synROD → ROD	synHB → HB
Target rotation	64.60	86.32
FC classifier	64.20	86.49
Our	**66.68**	**87.28**

Table 7.1: Accuracy (%) of variations of our method. Using convolutional layers instead of a pooling layer in the rotation classifier helps to exploit spatial information, causing the network to better adapt to the target domain.

Performing DA only by predicting relative rotation of target samples is sufficient to improve the accuracy significantly over "source only" (table 7.2), but it is not as effective as using both source and target datasets. The network learns more useful features when the self-supervised task is trained with data coming from both domains due to the higher diversity in data.

Finally, we evaluate the performance of our method when using the same classifier architecture for the secondary task and the main task. The network performs better when convolutional layers are used instead of a pooling layer, because spatial information is more easily preserved, therefore helping to solve the relative rotation task.

7.3.2 Baseline comparison

Table 7.2 shows that general-purpose DA methods do not always perform well on multimodal data. For example, *AFN* and *MMD* yield higher accuracy on single-modality RGB data than on multimodal RGB-D images. These results are due to the less informative nature of Depth data: without an effective method to exploit the additional information provided by the Depth channel, training on RGB-D data can produce lower performance than using the RGB modality alone.

Figure 7.5: Comparison between the synthetic datasets used in our experiments and PACS [46]. The latter portrays subjects in their natural position, so it is easy to guess if the image has been rotated. Instead, the first two datasets contain pictures taken from random angles, which makes it almost impossible to guess the absolute rotation.

Rotation suffers from an additional issue: as already mentioned in section 5.2, it

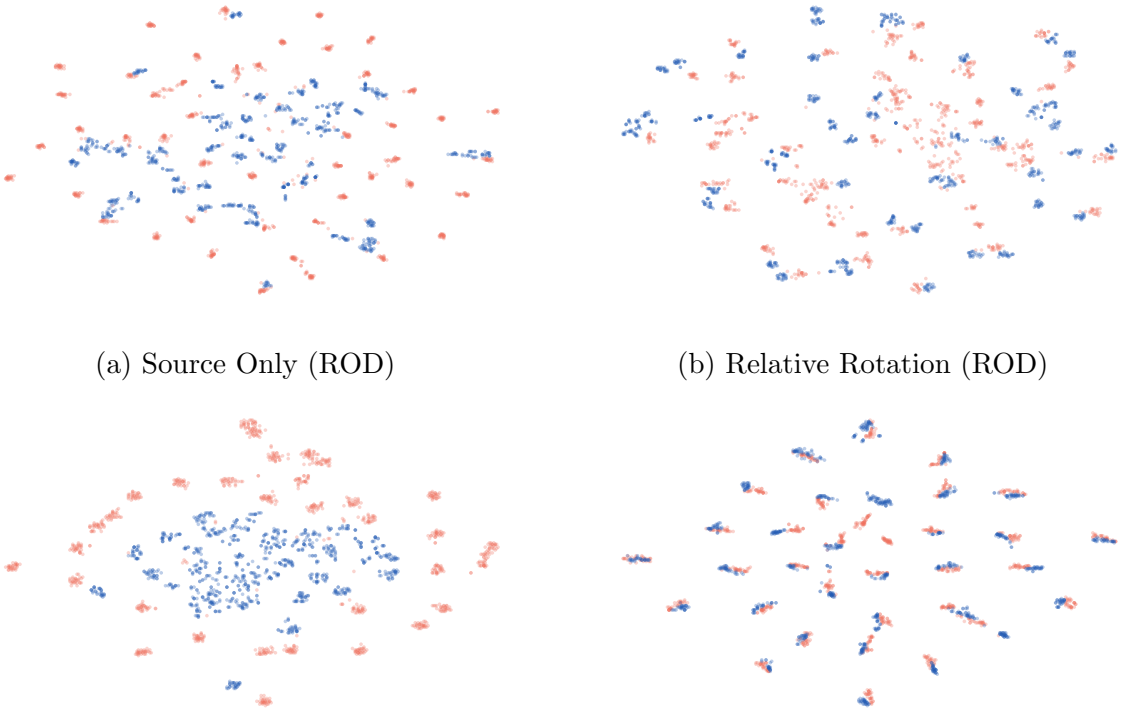
can only work under the hypothesis that the objects in the dataset are portrayed in a pose that is coherent through the majority of the samples. As figure 7.5 shows, this requirement is not satisfied by the synthetic datasets we used, making the self-supervised task an ill-posed problem. Since the secondary task cannot be solved by natural means, the network learns to solve it by overfitting or by using shortcuts [55], thus not helping the feature extractor to learn domain-invariant features. The attempt to answer an impossible problem is not just ineffective, but it can severely damage the ability of the network to generalize, yielding even lower performances than the ones obtained in a "source only" setting, as shown by the results of *Rotation* on Depth. Nevertheless, this method still produces an improvement over "source only" in some cases (RGB-D, RGB-D e2e and RGB). Such performance gain is not due to the rotation task, but it is caused by the update of batch normalisation layers while training on target. Depth images do not benefit as much of batch normalisation on target because of the higher domain shift and less informative channel.

Despite being related to *rotation*, our method does not share this weakness. In addition, it does learn inter-modal relationships, thus effectively exploiting multimodal data: table 7.2 proves that predicting the relative rotation between modalities is an efficient DA method, indeed outperforming all the selected baselines on both datasets.

From a qualitative point of view, the effectiveness of our method can be shown by plotting the t-SNE [50] visualisation of the features of the main classifier when the network is fed with data coming from the source and target domains. Figure 7.6 clearly shows that using our method source and target features are better aligned and form well-defined clusters.

Method		synROD → ROD	synHB → HB
Source only	RGB	52.13	51.17
	Depth	7.56	15.50
	RGB-D	50.57	49.71
	RGB-D e2e	47.70	49.45
GRL	RGB	57.12	74.74
	Depth	26.11	29.52
	RGB-D	59.09	75.23
	RGB-D e2e	59.51	74.95
MMD	RGB	63.68	74.95
	Depth	29.34	28.24
	RGB-D	62.10	77.96
	RGB-D e2e	62.57	77.26
Rotation	RGB	63.21	84.46
	Depth	6.70	5.62
	RGB-D	63.33	83.99
	RGB-D e2e	57.89	84.15
AFN	RGB	64.63	84.04
	Depth	30.72	31.67
	RGB-D	61.19	83.06
	RGB-D e2e	62.40	86.49
Ours (RR)		**66.68**	**87.28**

Table 7.2: Accuracy (%) of several RGB-D Domain Adaptation methods.



(a) Source Only (ROD)

(b) Relative Rotation (ROD)

(c) Source Only (HB)

(d) Relative Rotation (HB)

Figure 7.6: t-SNE [50] projection of the features extracted from the last hidden layer of the main classifier M. Red dots represent source samples, while blue dots denote target samples.

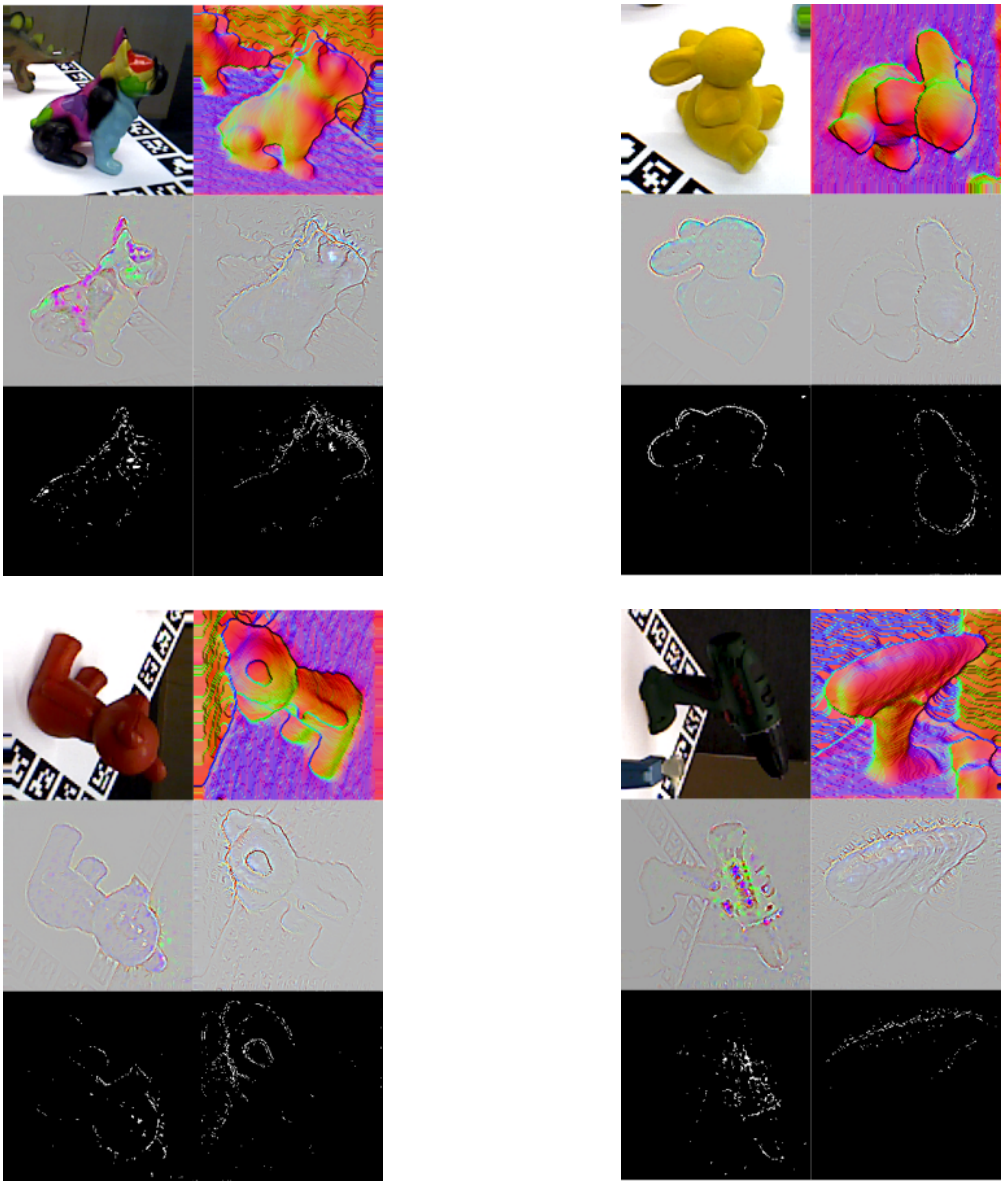


Figure 7.7: Visualisation of the most relevant pixels to predict the relative rotation between modalities. The first row displays the randomly rotated RGB-D image fed into the network. The second row shows the pixels that maximally activate the last layer of the feature extractors. The pixels are determined by guided backpropagation [71]. The third row highlights the peaks through binarisation to make visualisation easier.

Chapter 8

Conclusions and future work

This work shows the effectiveness of exploiting inter-modal relations in order to perform DA. We propose the first multimodal DA method specifically tailored for RGB-D data. Our approach consists of randomly rotating both the modalities and training a network to predict the relative rotation between the two images. This task works alongside the main classification head, which is trained on labelled source samples, and helps it to better generalise on target data.

In order to assess the performance of our novel method, we define two synthetic-to-real benchmarks for DA and object classification. For this purpose we adapt and use the existing dataset HB, and we collect a new dataset called synROD as a synthetic counterpart to the already existing ROD. Our experiments show that the inter-modal self-supervised task effectively reduces the domain shift and outperforms all the baselines we considered, thus proving that exploiting inter-modal relations is the key to perform DA on multimodal data.

Further research can be done in order to experiment with other inter-modal self-supervised tasks. For example, it could be possible to make the network solve a cross-modality jigsaw puzzle [12, 55] or determine the relative shift between modalities. Since it is unlikely that a specific self-supervised task is proven to be optimal in every possible scenario, it would be interesting to study the effectiveness of different tasks depending on the dataset. Ultimately, the self-supervised transformation could be learnt at training time through an encoder, yielding a general-purpose multimodal DA method that would go beyond computer vision and could be used in scenarios where there are no obvious hand-crafted self-supervised tasks.

Another issue that could be addressed by future research is the lack of a standard benchmark protocol for DA methods. As already discussed in section 7.3, the knowledge of target labels should not be employed until the evaluation of the chosen model. In a real-world scenario, target labels are not available at all, and requiring them in order to select the model would nullify the purpose of DA itself. A possible approach could involve training an anomaly detection algorithm, for example a one-class SVM [66] or autoencoder-based network [28], on source deep representation

features. The unsupervised model could then be tested on both source and target features, and the main classification model could be selected based on the inability of the anomaly detector to identify target samples.

We hope that this work encourages the community to further research on the topic of multimodal DA.

Bibliography

[1] A. Aakerberg, K. Nasrollahi, and T. Heder, «Improving a deep learning based RGB-D object recognition model by ensemble learning», in *2017 Seventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*, Nov. 2017, pp. 1–6.

[2] A. Aakerberg, K. Nasrollahi, C. B. Rasmussen, and T. B. Moeslund, «Depth Value Pre-Processing for Accurate Transfer Learning based RGB-D Object Recognition.», in *Ijcci*, 2017, pp. 121–128.

[3] U. Asif, M. Bennamoun, and F. Sohel, «Efficient RGB-D object categorization using cascaded ensembles of randomized decision trees», in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 1295–1302.

[4] M. Ben-Ari and F. Mondada, «Robots and Their Applications», in *Elements of Robotics*. Cham: Springer International Publishing, 2018, pp. 1–20.

[5] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. Vaughan, «A theory of learning from different domains», *Machine Learning*, vol. 79, no. 1-2, pp. 151–175, 2010.

[6] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, «Analysis of representations for domain adaptation», 2007, pp. 137–144.

[7] Y. Bengio, P. Simard, and P. Frasconi, «Learning long-term dependencies with gradient descent is difficult», *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, Mar. 1994, ISSN: 1941-0093.

[8] *Blender*, `https://www.blender.org/`, Accessed: 2020-03-06.

[9] M. Blum, Jost Tobias Springenberg, J. Wülfing, and M. Riedmiller, «A learned feature descriptor for object recognition in RGB-D data», in *2012 IEEE International Conference on Robotics and Automation*, May 2012, pp. 1298–1303.

[10] L. Bo, X. Ren, and D. Fox, «Depth kernel descriptors for object recognition», in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 821–826.

[11] S. Bucci, M. R. Loghmani, and B. Caputo, *Multimodal Deep Domain Adaptation*, 2018. arXiv: 1807.11697 [cs.LG].

[12] F. M. Carlucci, A. D'Innocente, S. Bucci, B. Caputo, and T. Tommasi, «Domain Generalization by Solving Jigsaw Puzzles», in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2019, pp. 2224–2233.

[13] F. M. Carlucci, P. Russo, and B. Caputo, «$(DE)^2CO$: Deep Depth Colorization», *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2386–2393, Jul. 2018, ISSN: 2377-3774.

[14] A. Cauchy, «Méthode générale pour la résolution des systemes d'équations simultanées», *Comp. Rend. Sci. Paris*, vol. 25, pp. 536–538, 1847.

[15] G. Cybenko, «Approximation by superpositions of a sigmoidal function», *Mathematics of control, signals and systems*, vol. 2, no. 4, pp. 303–314, 1989.

[16] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, «ImageNet: A large-scale hierarchical image database», in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2009, pp. 248–255.

[17] C. Doersch, A. Gupta, and A. A. Efros, «Unsupervised Visual Representation Learning by Context Prediction», in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 1422–1430.

[18] J. Duchi, E. Hazan, and Y. Singer, «Adaptive subgradient methods for online learning and stochastic optimization», *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, 2011.

[19] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, «Multimodal deep learning for robust RGB-D object recognition», in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2015, pp. 681–687.

[20] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. March, and V. Lempitsky, «Domain-Adversarial Training of Neural Networks», *Journal of Machine Learning Research*, vol. 17, no. 59, pp. 1–35, 2016.

[21] M. Ghifary, W. B. Kleijn, M. Zhang, D. Balduzzi, and W. Li, «Deep Reconstruction-Classification Networks for Unsupervised Domain Adaptation», in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 597–613, ISBN: 978-3-319-46493-0.

[22] S. Gidaris, P. Singh, and N. Komodakis, *Unsupervised Representation Learning by Predicting Image Rotations*, 2018. arXiv: 1803.07728 [cs.CV].

[23] X. Glorot and Y. Bengio, «Understanding the difficulty of training deep feedforward neural networks», in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Y. W. Teh and M. Titterington, Eds., ser. Proceedings of Machine Learning Research, vol. 9, Chia Laguna Resort, Sardinia, Italy: Pmlr, May 2010, pp. 249–256.

[24] X. Glorot, A. Bordes, and Y. Bengio, «Domain adaptation for large-scale sentiment classification: A deep learning approach», 2011.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. The MIT Press, Nov. 2016, ISBN: 978-0-262-03561-3.

[26] A. Gretton, B. Sriperumbudur, D. Sejdinovic, H. Strathmann, S. Balakrishnan, M. Pontil, and K. Fukumizu, «Optimal kernel choice for large-scale two-sample tests», vol. 2, 2012, pp. 1205–1213.

[27] A. Gretton, K. Borgwardt, M. Rasch, B. Schölkopf, and A. J. Smola, «A Kernel Method for the Two-Sample-Problem», in *Advances in Neural Information Processing Systems 19*, B. Schölkopf, J. C. Platt, and T. Hoffman, Eds., MIT Press, 2007, pp. 513–520.

[28] J. Guo, G. Liu, Y. Zuo, and J. Wu, «An Anomaly Detection Framework Based on Autoencoder and Nearest Neighbor», in *2018 15th International Conference on Service Systems and Service Management (ICSSSM)*, Jul. 2018, pp. 1–6.

[29] K. He, X. Zhang, S. Ren, and J. Sun, «Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification», in *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec. 2015, pp. 1026–1034.

[30] ——, «Deep Residual Learning for Image Recognition», in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 770–778.

[31] K. Hornik, M. Stinchcombe, and H. White, «Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks», *Neural networks*, vol. 3, no. 5, pp. 551–560, 1990.

[32] K. Hornik, M. Stinchcombe, H. White, *et al.*, «Multilayer feedforward networks are universal approximators.», *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[33] S. Ioffe and C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift», in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ser. Icml'15, Lille, France: JMLR.org, 2015, pp. 448–456.

[34] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, «What is the best multi-stage architecture for object recognition?», in *2009 IEEE 12th International Conference on Computer Vision*, Sep. 2009, pp. 2146–2153.

[35] R. Kaskman, S. Zakharov, I. Shugurov, and S. Ilic, «HomebrewedDB: RGB-D Dataset for 6D Pose Estimation of 3D Objects», in *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*, Oct. 2019, pp. 2767–2776.

[36] D. Kifer, S. Ben-David, and J. Gehrke, «Detecting change in data streams», in *Vldb*, Toronto, Canada, vol. 4, 2004, pp. 180–191.

[37] D. Kingma and J. Ba, «Adam: A method for stochastic optimization», 2015.

[38] A. Krizhevsky, I. Sutskever, and G. E. Hinton, «Imagenet classification with deep convolutional neural networks», in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[39] A. Krogh and J. A. Hertz, «A Simple Weight Decay Can Improve Generalization», in *Proceedings of the 4th International Conference on Neural Information Processing Systems*, ser. Nips'91, Denver, Colorado: Morgan Kaufmann Publishers Inc., 1991, pp. 950–957, ISBN: 1558602224.

[40] K. Lai, L. Bo, and D. Fox, «Unsupervised feature learning for 3D scene labeling», in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 3050–3057.

[41] K. Lai, L. Bo, X. Ren, and D. Fox, «A large-scale hierarchical multi-view RGB-D object dataset», in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 1817–1824.

[42] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, «Gradient-based learning applied to document recognition», *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, ISSN: 1558-2256.

[43] Y. LeCun *et al.*, «Generalization and network design strategies», *Connectionism in perspective*, vol. 19, pp. 143–155, 1989.

[44] Y. LeCun, L. Bottou, G. Orr, and K.-R. Muller, «Efficient backprop», *Neural Networks: Tricks of the Trade. New York: Springer*, 1998.

[45] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, «Multilayer feedforward networks with a nonpolynomial activation function can approximate any function», *Neural networks*, vol. 6, no. 6, pp. 861–867, 1993.

[46] D. Li, Y. Yang, Y. Song, and T. M. Hospedales, «Deeper, Broader and Artier Domain Generalization», in *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct. 2017, pp. 5543–5551.

[47] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, «Microsoft COCO: Common Objects in Context», in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 740–755, ISBN: 978-3-319-10602-1.

[48] M. R. Loghmani, M. Planamente, B. Caputo, and M. Vincze, «Recurrent Convolutional Fusion for RGB-D Object Recognition», *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2878–2885, Jul. 2019, ISSN: 2377-3774.

[49] M. Long, Y. Cao, J. Wang, and M. I. Jordan, «Learning Transferable Features with Deep Adaptation Networks.», in *Icml*, F. R. Bach and D. M. Blei, Eds., ser. JMLR Workshop and Conference Proceedings, vol. 37, JMLR.org, 2015, pp. 97–105.

[50] L. van der Maaten and G. Hinton, «Visualizing Data using t-SNE», *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.

[51] P. Morerio, J. Cavazza, and V. Murino, «Minimal-Entropy Correlation Alignment for Unsupervised Deep Domain Adaptation.», in *ICLR (Poster)*, OpenReview.net, 2018.

[52] (2018). Nasce il Centauro, un nuovo robot quadrupede per il supporto dell'uomo, [Online]. Available: `https://opentalk.iit.it/nasce-il-centauro-un-nuovo-robot-quadrupede-per-il-supporto-delluomo` (visited on 03/23/2020).

[53] Y. Nesterov, *Introductory lectures on convex optimization: A basic course*. Springer Science & Business Media, 2013, vol. 87.

[54] ——, «A method of solving a convex programming problem with convergence rate», *Soviet Mathematics Doklady*, vol. 27, pp. 372–376,

[55] M. Noroozi and P. Favaro, «Unsupervised Learning of Visual Representations by Solving Jigsaw Puzzles», in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 69–84, ISBN: 978-3-319-46466-4.

[56] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, *Activation Functions: Comparison of trends in Practice and Research for Deep Learning*, 2018. arXiv: `1811.03378 [cs.LG]`.

[57] A. Owens, J. Wu, J. McDermott, W. Freeman, and A. Torralba, «Ambient sound provides supervision for visual learning», *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9905 Lncs, pp. 801–816, 2016.

[58] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, «Automatic Differentiation in PyTorch», in *NIPS Autodiff Workshop*, 2017.

[59] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros, «Context Encoders: Feature Learning by Inpainting», in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 2536–2544.

[60] *PyTorch*, `https://pytorch.org/`, Accessed: 2020-03-19.

[61] N. Qian, «On the momentum term in gradient descent learning algorithms», *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[62] H. Robbins and S. Monro, «A stochastic approximation method», *The annals of mathematical statistics*, pp. 400–407, 1951.

[63] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, «Learning representations by back-propagating errors», *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[64] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, «ImageNet Large Scale Visual Recognition Challenge», *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[65] A. M. Saxe, J. L. McClelland, and S. Ganguli, *Exact solutions to the nonlinear dynamics of learning in deep linear neural networks*, 2013. arXiv: `1312.6120 [cs.NE]`.

[66] B. Schölkopf, J. C. Platt, J. C. Shawe-Taylor, A. J. Smola, and R. C. Williamson, «Estimating the Support of a High-Dimensional Distribution», *Neural Comput.*, vol. 13, no. 7, pp. 1443–1471, Jul. 2001, ISSN: 0899-7667.

[67] M. Schwarz, H. Schulz, and S. Behnke, «RGB-D object recognition and pose estimation based on pre-trained convolutional neural network features», in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 1329–1335.

[68] H. Shimodaira, «Improving predictive inference under covariate shift by weighting the log-likelihood function», *Journal of statistical planning and inference*, vol. 90, no. 2, pp. 227–244, 2000.

[69] K. Simonyan and A. Zisserman, «Very deep convolutional networks for large-scale image recognition», cited By 5548, 2015.

[70] R. Socher, B. Huval, B. Bath, C. D. Manning, and A. Y. Ng, «Convolutional-recursive deep learning for 3d object classification», in *Advances in neural information processing systems*, 2012, pp. 656–664.

[71] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, *Striving for Simplicity: The All Convolutional Net*, 2014. arXiv: `1412.6806 [cs.LG]`.

[72] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, «Dropout: A Simple Way to Prevent Neural Networks from Overfitting», *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958, 2014.

[73] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, «On the importance of initialization and momentum in deep learning», in *International conference on machine learning*, 2013, pp. 1139–1147.

[74] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, «Going deeper with convolutions», in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2015, pp. 1–9.

[75] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, «A Survey on Deep Transfer Learning», in *Artificial Neural Networks and Machine Learning – ICANN 2018*, V. Kůrková, Y. Manolopoulos, B. Hammer, L. Iliadis, and I. Maglogiannis, Eds., Cham: Springer International Publishing, 2018, pp. 270–279, ISBN: 978-3-030-01424-7.

[76] J. Xu, L. Xiao, and A. M. López, «Self-Supervised Domain Adaptation for Computer Vision Tasks», *IEEE Access*, vol. 7, pp. 156 694–156 706, 2019, ISSN: 2169-3536.

[77] R. Xu, G. Li, J. Yang, and L. Lin, «Larger Norm More Transferable: An Adaptive Feature Norm Approach for Unsupervised Domain Adaptation», in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct. 2019, pp. 1426–1435.

[78] R. Zhang, P. Isola, and A. A. Efros, «Colorful Image Colorization», in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 649–666, ISBN: 978-3-319-46487-9.

[79] Zhou and Chellappa, «Computation of optical flow using a neural network», in *IEEE 1988 International Conference on Neural Networks*, Jul. 1988, 71–78 vol.2.