POLITECNICO DI TORINO

Master degree in Computer Engineering

Master Thesis

Towards Visual Generalization with Graph Deep Learning

Supervisors
prof. Tatiana Tommasi

Co-Supervisors:
prof. Barbara Caputo
doct. Davide Boscaini

Piero Cavalcanti
252117

December 2019

Abstract

Starting from the great potential of graph convolutional neural networks, the core idea of this thesis project is to tackle visual tasks via graph-based deep learning methods. This of course requires a principled way to map images into graphs, followed by an extensive evaluation of the visual recognition performance that graph-based models can provide. Our intuition is that, passing through graphs, it could be possible to maintain the semantic information of a picture while loosing all the specific idiosyncrasies which limit generalization in traditional image-based models. Still the graph construction is not trivial and tailored adaptive strategies may be needed. The proposed research direction inevitably involves several fields: from image segmentation and graph networks to multiple variants of cross-domain learning and self-supervision. A preliminary study allowed us to search for the most effective components to design our method. Then, we shifted our attention on more challenging cases where training and test data belong to two different distributions. The obtained promising results pave the way to new approaches based on graph-based learning for image understanding.

Acknowledgements

This chapter has to be intended as excluded from the academic content of the Thesis. To allow reading to people who are unfamiliar with English language, the further text will continue in my mother tongue.

Al termine del mio percorso di laurea magistrale al Politecnico di Torino, ci tengo a ringraziare tutte le persone che mi hanno sostenuto e motivato in questi due anni, senza le quali non sarei arrivato a questo punto.
In primo luogo, ringrazio la professoressa Tatiana Tommasi e la professoressa Barbara Caputo per avermi dato la possibilità di far parte a tutti gli effetti del gruppo *VANDAL* in questi mesi e di avermi fatto provare cosa significa davvero fare ricerca ad alti livelli. Ringrazio Davide Boscaini per l'impegno, la passione e gli insegnamenti trasmessi lungo tutto il periodo di collaborazione. Spero che questa esperienza rappresenti solo l'inizio di una lunga carriera in questo campo, di cui rappresentate di diritto il trampolino di lancio.
Ringrazio la mia famiglia, in ogni suo membro, per non avermi mai fatto mancare il sostegno e la sicurezza di avere qualcuno su cui contare in ogni momento. Mio padre, mia madre e mia sorella per avermi ascoltato nelle mie noiose elucubrazioni sulla tecnologia e su ciò che di fantastico stavo studiando, e per avermi dato modo di seguire le mie passioni senza pormi mai alcun limite. Grazie, non sarei qui senza di voi. Mia nonna, i miei zii e i miei cugini per avermi fatto sentire fiero di ciò che stavo facendo e di ciò a cui appartengo.
Voglio ringraziare la tribù Skopelos, varie famiglie che ne formano una più larga, di cui mi sentirò per sempre parte. In particolare Carlo, Francesco, Raffaele e Toto che mi accompagnano da una vita ormai e che, nonostante la lunga lontananza, riescono a farmi sentire a casa ogni volta che torno. Il vostro sostegno è stato più prezioso di quanto possiate immaginare. Ringrazio gli amici del liceo e Francesco, Carlo e Vincenzo per non essersi ancora annoiati di me nonostante tutto il tempo passato insieme.
Un ringraziamento speciale va a chi mi ha sostenuto in questi anni di vita a Torino: Federico, Flavio, Francesco, Francesco, Giuseppe, Mattia, Nico, Toppa, Uccio. Grazie per tutto ciò che abbiamo condiviso qui a Torino e tutto ciò che continuiamo a condividere ora che siamo sparsi per il mondo. I miei coinquilini, vecchi e nuovi, con

cui ho passato la maggior parte del mia vita torinese e che mi hanno sostenuto ed aiutato anche nei momenti peggiori. Ringrazio i compagni di corso Luca, Giuseppe, Gaetano, Totò, Giorgio, Gianluca, Vincenzo, e tutti coloro che hanno dovuto sopportare la mia ansia da esami e chi mi ha sostenuto in questo impervio percorso di magistrale.
Infine, vorrei dedicare questo progetto di tesi a due persone che hanno un valore speciale nella mia vita, Sara e Ciro. Quando ero indeciso sulla direzione da intraprendere, ho ricevuto un consiglio, l'ultimo di una lunga serie di riflessioni: "Vai a Torino, diventa un ingegnere". Se dicessi che questa è stata l'unica ragione che mi ha spinto a scegliere questa strada, che ora più che mai sento fatta per me, sarei ingiusto. Ma dal primo momento in cui ho messo piede in questa città sapevo già chi avrei dovuto ringraziare.

Contents

Chapter 1

Introduction

Almost a century has passed since the concept of *Robot* was conceived, and since those days relevant progresses have been reached in terms of automatic physical tasks and, mainly, cognitive tasks. Although it is true that the boundary between human and machine continues to be sharp, the more the time passes, the shorter this range becomes. Artificial Intelligence is a discipline whose aim is to constitute, design and test the cognitive tasks of the automatic components: basically, it develops the intelligence of machines.
One of the most interesting fields of research in this context is Computer Vision, which is attracting the growing interest of the scientific community due to the incredible results of the state-of-the-art algorithms. Computer vision is a long-standing discipline which deals with recognizing and exploiting knowledge within images. Common challenges in Computer Vision are image classification, object recognition and detection, semantic segmentation and localization.
The history of Computer Vision took an unexpected turn thanks to the introduction of the back-propagation training method and, moreover, with the development of the first *deep neural network*, which, since its debut, has beaten all previous results. The presentation of *AlexNet* [1] at the Neural Information Processing System (NIPS) Conference, in 2012, can be considered to be the first milestone in Deep Learning, as the application of the CNN led to a rapid increment of accuracy. Since then, Deep Learning methods became the undisputed star of Computer Vision research, as they consistently increase the score on classical benchmarks and they are still undefeated when compared with alternative automatic learning models.
The current overview greatly differs from the one of almost ten years ago. Modern networks have overcome human recognition ability [2], while deep learning approach has widely spread in different areas and with different types of information: from videos [3], with the introduction of the new recurrent CNN architecture and Long-Short Term Memory, to sounds and speech, where Natural Language Processing is constantly based on deep learning models.

Among the new frontiers of Deep Learning, the analysis of 3D shapes and graphs is one of the most attractive scenarios for researchers, as its range of possible applications is wide and heterogeneous. Moreover, it is evident that the adaptation of 2D-based architecture to the world of three-dimensional and non-Euclidean geometry is not trivial. Since the introduction of the topic, there have been various formulations of possible solutions to perform neural networks analysis on non-Euclidean geometry, with different results and applications. All the adaptation approaches of standard Convolutional Neural Networks to the non-Euclidean geometry or 3D objects come together under the name of **Geometric Deep Learning**.
The present work mainly focuses on graph-based geometric deep learning, or **Graph Deep Learning**. The first attempt to apply neural networks to graphs was conducted by Scarselli *et al.* [4], in 2005, but it remained unnoticed until the problem was revived by Bruna *et al.* [5]. In their paper, the authors introduced for the first time a deep neural network on graphs based on spectral domain, leaning on an analogy between Fourier transforms and results deriving from the graph Laplacian operator.
Subsequently, this approach was intensely lightened by Kipf and Welling [6], using simple filters operating on 1-hop neighborhoods of the graph, avoiding the explicit Laplacian eigenvectors computation based on recurrent Chebyshev polynomials. The introduction of graph-based neural networks represents a long step forward compared to the spectral approach, as it conveys a better generalization and a faster computation. Since that, a number of new approaches have been presented to perform graph classification, node classification and other tasks in the field of graph geometry. The last attempts include the capture of a finer structure of the neighborhood, analyzing vertices of graphs similarly to the way neighboring pixels are treated in conventional CNNs. Those approaches [7], [8] represent the starting point of our research project. Although some results have been presented as secondary experiments in graph-based Geometric Deep Learning [7], [9], we are not aware of previous work focusing on the application of non-Euclidean Deep Learning methods for Visual Learning tasks as their main purpose. As a matter of fact, research has proven that geometric deep learning methods can work on simple image classification tasks, such as digits recognition, and analogies suggest that the use of graph-based neural networks on images is coherent and can be done with both the full input or just a set of attention points.

Contribution of the thesis

Starting from these assumptions, **we created a full framework in order to extract significant pixels from the image and structure a coherent set of vertices, that will then be interpolated to build a graph**. The construction of the graph was based on two main stages: first, the segmentation of the original sample into a set of correlated points; and second, the computation of a significant point as summary or mean of the whole set. After the down-sampling of the input, we used different criteria with the aim to create the connections of each vertex through edges, which must keep semantic information about the relations among the nodes. Those two stages were not trivial and requested many experimental trials: we evaluated different segmentation techniques and different edge definitions in order to perform the graph construction. Such research was conducted considering different domains in the field of *digits recognition*: we considered 3 datasets, *MNIST, MNIST-M* and *USPS*. Although they have been widely used to investigate the domain shift problem due to the large variety of covered styles, only MNIST had previously been involved in graph-learning experiments. In the first part we focused on segmentation, comparing different algorithms while maintaining a fixed basic Graph Convolutional Neural Network, known as ECC Network [7]. This lead us to recognition results lower than those those obtained with standard CNNs, but produced with much less input information. Then, **we focused on designing new deeper graph-based network to emphasize their learning power**. We obtained different architectures, with different strengths and weaknesses: some of those backbones originated from concepts borrowed from existing architectures, others were obtained by logical deductions. As a result, we were able to obtain competitive results with respect to standard convolutional networks still using a heavily reduced number of points for each sample of the set. This proves that knowledge from image can be extracted from minimal number of pixels by exploiting the relations among them and their arrangement. Following, **we shifted our attention on more challenging cases where training and test data belong to two different distributions**. We leveraged on a recent domain generalization method, which has shown how combining supervised and self-supervised knowledge can improve generalization [10]. More specifically, such method includes a jigsaw puzzle task which is simple and intuitive for 2D images, but needs a relevant redesign to be included in graph networks. The first strategy we proposed was a sequence of patch-based clustering, graph vertices grouping and group shuffling. The puzzle tasks consisted in learning and then predicting the right permutation index. We also considered a second adaptation method based on the alignment of the representative features of the different domains by reversing the contribution of a domain discriminator [11]. As a result, the features extracted would be the best ones to confound the different domains in input. Starting from this concept, we developed a suitable method for graph-based geometric deep learning.

This work has to be regarded as a mere starting point for a more in-depth analysis and evaluation of the described intuition that connects images and graphs to generalization. Future developments envisage a smoother integration of all the components analyzed thus far, towards a fully learnable end-to-end model able to perform image segmentation for graph extraction and cross-domain visual recognition.

Chapter 2

Preliminaries on Visual Learning

2.1 Machine and Deep Learning

In the big data era in which we are living, Machine Learning became a core task to exploit the huge amount of accessible information and in that way to sustain the technological, cultural and social growth. Since it has proven to be effective, machine learning technology has spread and now supports many aspects of modern society: it underpins a large part of object identification, natural language processing, automatic recommendation systems and so on. It is even a key power in social networks, e-commerce system and research browser, and in many other components of our daily routine. Increasingly it is achieved by using Convolutional Neural Networks as main technique.
For many times, convolutional neural networks meant significant amount of works and resources used to tune and test the convolutional parameters in a non-automatic fashion. Today, the modern deep learning techniques overcome the issue by automatic learning the parameters that compose the network architecture with a reiterate process called back-propagation. Deep Learning is a sub-part of the conventional **Representation Learning**, that are based on understanding the necessary representation of the data to perform results in classification or detection, etc. In deep learning methods, this is achieved by multiple levels of representation, obtained in a hierarchical way by applying subsequent module on the retrieved input (starting from the original sample), making possible to extract even slightly abstract representations. These kind of high representations takes the name of **features**. Composing these levels of representations makes possible to reach incredible prediction results, even with very complex raw input data. To deeper understand

this application, let us make a representative example in computer vision (image-related) tasks: a digital image is submitted to the network on its regular format, means as an array of pixels. Generally, the first module layer (**convolutional layer**) simply detects the presence of edges in a particular orientation and location over the image. Then, the second layer, where the first layer output is processed, recognizes simple patterns, exploiting the retrieved edges. The representation is higher on the third layer, when the patterns are assembled to identify presence of particular objects in the image. As layers processes the images among the hierarchical structure of the network, as deeper and higher becomes the image analysis. Deep Learning structures are today based on thousand and thousand of parameters tuned each time a batch of images is processed to the network: it is unimaginable to expect to design all the network by human engineers, while it is today made by an automatic learning procedure. This is a key potential in machine learning process: the deep learning methods achieve state-of-the-art in a very large set of different applications without asking discouraging network design phase. It just requires little manual engineering, and can easily take advantage of increases in the computational power and storage capability that characterizes our epoch.

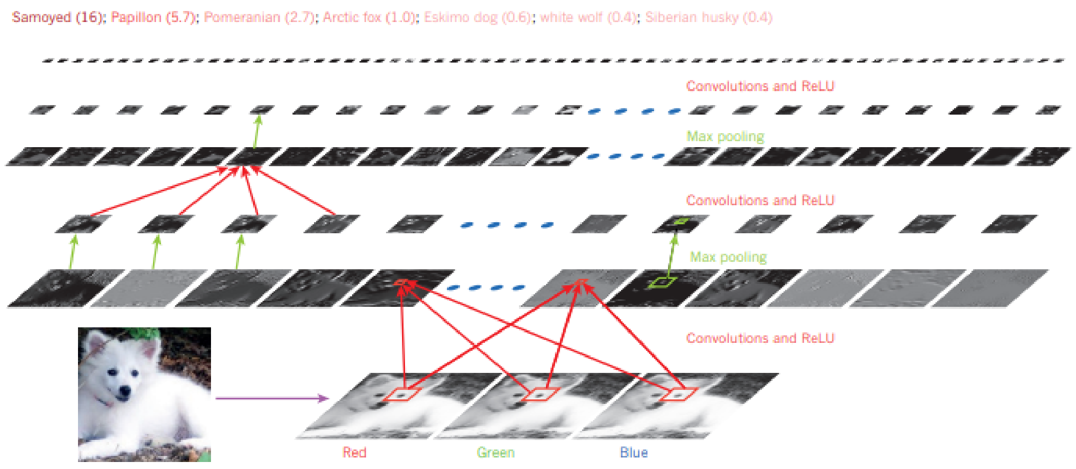
Samoyed (16); Papillon (5.7); Pomeranian (2.7); Arctic fox (1.0); Eskimo dog (0.6); white wolf (0.4); Siberian husky (0.4)
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Max pooling
Convolutions and ReLU
Red
Green
Blue

Figure 2.1: Example of a convolutional neural network applied in a computer vision task (from [12])

The most diffuse form of machine learning is the standard **supervised learning**. The supervision property means the capability to compare the predictor results with empirical evidence (commonly refers to as *ground truth*) that gives the possibility to understand whether or not the prediction is correct. In case we want to automatically classify images, we can model the task in a supervised manner by collecting a large set of images and assign to each of them a label representing the class of the image (*i.e. understanding the subject in the image*). The supervised setting makes the network able to perform automatic tuning of the network parameters (or *weights*) with the usage of a *loss function*, that measures the distance between the ground truth (representing the ideal behaviour of the net) and the network outputs. Exploiting the result of the loss function, network tunes every weights in such a way that the next output will reduce the gap from the ground truth. The concept is both simple and effective. The weights of the networks are nothing but real numbers, regulated according to the network expected behavior. As more layers are used, as more the number of weights grows; and in turn as more the number of weights grows, as more training dataset has to be large. Today, modern networks perform tuning on hundreds of millions of these adjustable weights, and thus requires hundreds of millions of labelled samples.

It is clear that the process of adjusting the weight vectors is a crucial part of deep learning: it is possible by the computation of a proper **gradient** vector representing for each weights the quantity of increment or decrement in the distance from the ground truth if the weights were increased by a value *small as you wish.* The weights vector is adjusted according to the negative gradient vector, since we want to get the *global minimum* of the loss function. Indeed, it can be proven that the loss function has a convex trend, and negative gradient vector indicates the way to obtain the steepest descent in the trend. Leaving the theory, practical application of the gradient descent principle is the **Stochastic Gradient Descent**. Stochastic optimization, briefly, is an optimization process where assumptions are taken by a probabilistic approach, in contrast with deterministic methods. In our case, the SGD consists on optimizing the weights vector by submitting to the network a small batch of the whole dataset and adjust it accordingly. So the small batch represent an exemplar of the entire population of the dataset, and then the estimation results slightly noisy. Despite that, SGD performs an optimization procedure that overcomes efficiently other complex methods for weights tuning.

As long as the modules represent relatively smooth functions of their inputs and internal wights, the gradients can be computed by a simple backwards propagation procedure. The **back-propagation method** is a key point in the CNN workflow, and it is based on the chain rule of the derivatives. Indeed, the module gradient can be computed by working backwards from the gradient with respect to the output of that module. Repeated application of the back-propagation allow to propagate gradients through all modules, starting from the output, through to initial layers.

The final step is to validate the results by testing on an unseen set of samples (called, therefore, *test set*, instead of the "seen" training set): it allow to simulate the behavior of the network when his application is required for new, unknown data. Standard practice is to split the original sample set in two subsets, training and test set, and then use just the first one to tune the net. As we will see, this separation has its limitations, due to the fact the both subsets belongs to the same distributions, so the test samples are quite similar in comparison to the training samples, and it do not stress enough the generalization ability of the machine.

2.2 Neural Networks

General neural networks are known as interconnected hierarchical structures of simple processing elements, units or nodes, whose trigger mechanisms are based on animal neurons. In the human brain, for example, there are 100 billion of neurons, which communicates with electrical impulses through the inter-neuron connections, the so-called *synapses*. The processing ability of the network is stored in the inter-unit connection strengths, or *weights*, obtained by a process of adaptation to a set of training sample. The artificial equivalents of synapses are modelled by a single number (the weight) so that each input is multiplied by it before being sent to the equivalent of the cell body. Here, the weighted signals are summed together by simple arithmetic addition to supply a node activation. When the sum, or activation, is computed, it is compared with a determined threshold; if the activation exceeds the threshold, the unit produces a high-valued output (conventionally "1"), otherwise it outputs zero. Thus, a simple neuron output can be represented in the following form:

$$y = f\Big(\sum_{i=1}^{n} w_i x_i + b\Big) \tag{2.1}$$

This is the canonical form to reproduce a single neuron function. x_i are the input signals from the m neurons of the previous layer, b is the *bias* term. f represents the activation function, that is typically a non-linear function such as *step function*, *sigmoid*, and, the most commonly used in modern application, *ReLU*. The adaptation procedure, that has effect during the training phase, adjust the weights and the bias term.

A neural network is the result of the combination of aggregation of multiple neurons, structured in layers. Layers are interconnected in a hierarchical way, means in a direction so that the output of a layer will be the input of the following layer. The input of the initial layer is the input data, while the output of the last layer represents the result, i.e. in classification the predicted class of belonging. In fully-connected layers, the neurons within a layer are inter-connected with all the neuron of the previous and the following layer. At each connection is assigned a specific tunable (or, better, *learnable*) weight. Neural networks are based on various layers, and architectures are often based on *hidden layers*, meaning the layers posed between the input and output layers.

A typical architecture of fully-connected layers can be seen in figure 2.3. The first layer processes the raw input(s), while the last one returns the desired output(s).

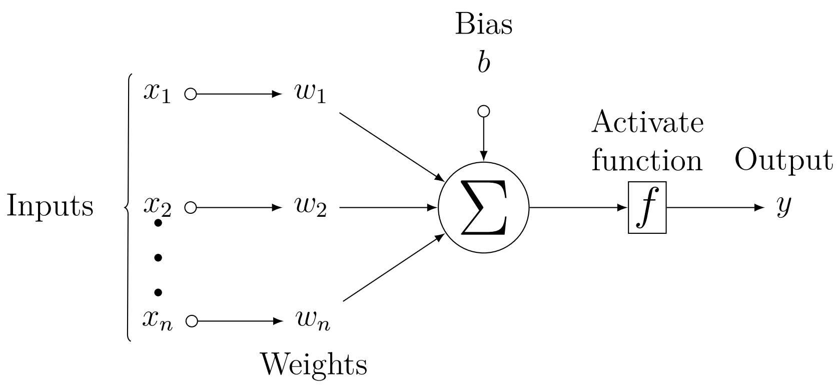

Figure 2.2: Representation of a neuron output function

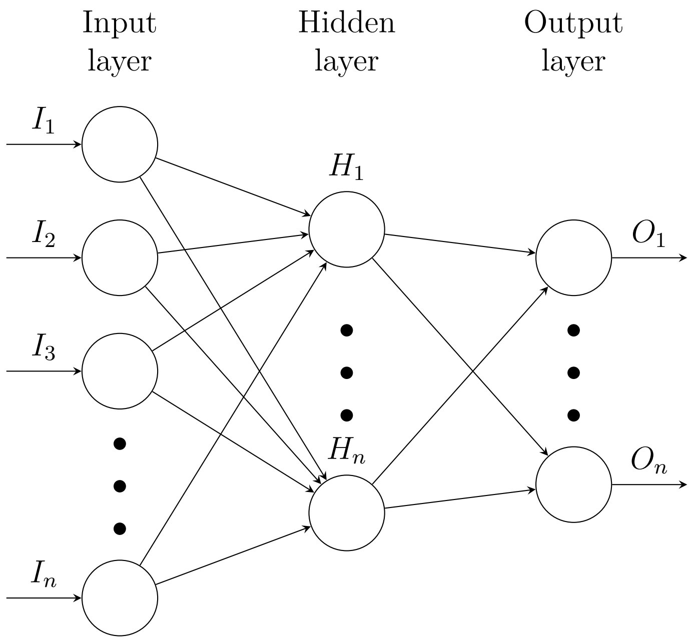

Figure 2.3: General diagram for a 3-layers NN architecture.

More hidden layers can be posed between the initial and last layer. For a general classification task, the output layer is composed of m neurons, where m represent the number of classes. Each neurons outputs a probability percentage, commonly from 0 to 1, and the predicted class will be k where $k - th$ neuron returns the highest value.

As standard machine learning frameworks that works with a supervised approach, Neural Networks has to be educated, given a labelled input, to outputs the relative label. This learning process is generally known as **training**, and the set of input sample with a manually-assigned label is so named training set. Training phase

is constituted of two sub-parts that sharing the task of giving to the network the possibility to learn: the **forward propagation** and **backward propagation**. The first part is based on the evaluation the performance of the network in the current state: each input sample is submitted to the neural network architecture and the outputs are processed according to the result of each neuron computation. The initial setting of a untrained neural networks is random, so the first evaluation score is generally far from optimal values. To make possible to reduce the error-rate between the desired outputs and the obtained ones, a loss function is computed. Typical example of loss function in deep learning methods is the *cross entropy loss*, or logarithmic loss ($-\sum_i y'_i \log_2 y_i$). The **backward propagation** allow to convey the loss information (the model literally learn from his mistakes): starting from the final layers, the loss signal is propagated in the opposite direction respect to the normal output computation (also said as *forward propagation*), through the hidden layers, that will perturb the weights of the whole network. The impact of the loss signal change among neurons according to the contribution of each of them to the final output. The perturbation of the network parameters is driven by the Gradient Descent, that allow to clarify the *direction* to follow to minimize the prediction errors, described by the loss. This technique, briefly described in the previous sections, use derivative calculation on the loss function, with respect of the model parameters, to orient the increment or decrement of each weight according to reaching the global minimum of the loss function. Since loss has not a close form solution, it is not possible to obtain unambiguous assignments for the weights. Otherwise, thanks to the fact that loss function is convex, we can approximate with the gradient the descending direction of the function, that will conduct to relative or global minima. This is done in batches of data in epochs, means consecutive iterations, that simulate the behavior of the entire dataset in a stochastic way.
Given a general loss function such as:

$$C(W, b, x^i, y^i) = \frac{1}{2}|f_{(W,b)}(x^i) - y^i| \tag{2.2}$$

Where W, b represents respectively the weights and the bias term of the neural networks, the update rule of the parameters will be:

$$W_{i,j} = W_{i,j} - \alpha \frac{\partial C(W, b)}{\partial W_{i,j}} \tag{2.3}$$

$$b_i = b_i - \alpha \frac{\partial C(W, b)}{\partial b_i} \tag{2.4}$$

α parameter, where generally $0 < \alpha < 1$, representing the impact of the gradient step, is called *learning rate*. The negative signs is justified by the fact that gradient always points in the direction in which the value of the loss function increases. Learning rate is one of the typical **hyperparameters** that constitute the learning process: one hyper-parameter can be considered the optimizer, that can be *SGD* or other alternatives such as *Adam*, or the **batch size** that indicates the number of elements considered in each iteration of the learning process. Another common hyper-parameter is the *learning rate decay* that allow to decrease in a specific moment the learning rate, since parameter optimization requires to minimize the rate as the model approaches to the solution.

2.2.1 Convolutional Neural Networks

Convolutional Neural Networks starts from the simple concept that in images (and also in other kind of euclidean data) the neighboring pixels are highly correlated, and it has sense to do not consider each pixel individually but exploit small windows for the feature extraction. This application allows to obtain strong results even with a great saving of parameters to tune and, thus, in efficiency of the model. Despite the similarities with the general neural network models, there are particular elements in the architecture that differs from non-convolutional models:

Convolutional Layers: the core of the CNN, allow to perform the convolutional operation on small window of the input volume. Convolutional layer are based on set of learnable filters, or *kernels*, that are basically three-dimensional matrices of weights with dimension equal to activation volume on which the convolution will be applied. "Convolution" name derived from the so called operator, generally represented by $\star$. In a general practical example, consider a 5×5 image, that is digitally represented as a three-dimensional square matrix of 5×5 elements. Let us define a generic kernel of size 3×3: the output of the convolutional operation will be a matrix 3×3 where each element is calculated by sliding the kernel matrix over the input matrix (the image) with a stride of 1 pixel and performing an element-wise multiplication. The values of the filters application represents the the feature detection. Changes in filters parameters made by back-propagation determine changes in the effect of the convolution and, thus, in the features detected. More than one filter can be applied on a single input volume, and the several outputs define the **depth** of the feature map produced. The **stride** define the number of pixels "jumped" between two filter application. By using a stride larger than one pixel, it could be necessary to define a **0-padding** along the frames of the image so that it is possible to divide exactly the input matrix by the kernel size.

Activation Layer: Generally, after a convolutional layer, a non-linear function is applied on the outputs. The application of such operation is used to make the features extracted more expressive. ReLU multiplication with the signal just "switch" the negative values into into 0, and it is formalized as:

$$Relu(x) = max(0, x)$$

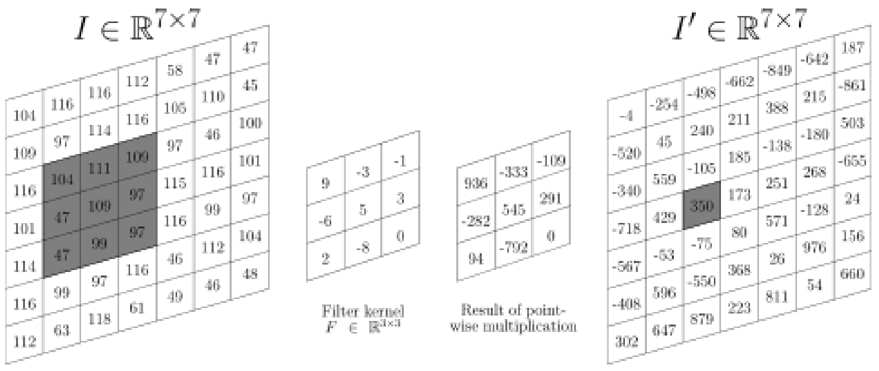
$I \in \mathbb{R}^{7\times7}$
Filter kernel $F \in \mathbb{R}^{3\times3}$
Result of point-wise multiplication
$I' \in \mathbb{R}^{7\times7}$

Figure 2.4: Convolutional operation: the input matrix (left) is convoluted with the 3×3 filter (center) on each 3×3 window to obtain the output volume (right).

. ReLU represents the most used operator in CNN application, but other alternatives are also used in Conv Architecture, such as:

$$\text{LeakyRelu} = \begin{cases} x & \text{if } x \geq 1 \\ 0.01x & \text{otherwise} \end{cases} \tag{2.5}$$

$$\text{ELU} = \begin{cases} x & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases} \tag{2.6}$$

Pooling Layer: sometimes it could be useful to reduce the dimensionality of the input information in a way that can preserve the semantic information within. Pooling, or spatial pooling, perform this kind of operation by down-sampling neighbors elements returning one resuming value. The most common Pooling Layer is the Max Pooling, that returns the max elements among all the elements in a $n \times n$ window of the input volume. Since even in the pooling layer there is a concept of local application and sliding, there is a similarity with the convolutional layer: we define the dimension of the filter, the number of local element considered in the max detection, and the stride define in the same way seen for convolution.

Another common pooling operator is the **Average Pooling** that simply extracts the mean of the values in the considered input window. The main goal of pooling layers is to perform a strong reduction of the input in a way that allow to manage features without loss of information, but there are other important properties obtained by using pooling, such as the robustness to transformations (local distortion, translation) or noisy samples, the reduction of overfitting risk and so on.

Fully Connected Layer: standard CNN architectures are structured with couple of sequential "Conv-Pooling" layers unit disposed in hierarchical order

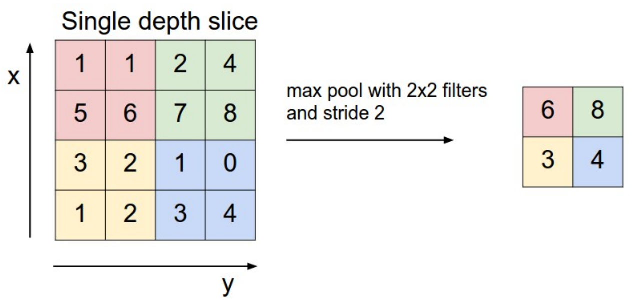

Figure 2.5: Example of max pooling application

to exploit features detection. Sometimes these layers are interspersed with pooling layers to perform dimensionality reduction of the processed units and to let the model learn in a more efficient way. To perform the actual prediction, it is good norm, particularly for regression and classification tasks, to exploit the entire output volume to formulate the results of deep model computation. In this cases is used the *Fully Connected Layer*, that, in a traditional neural networks fashion, connect each neurons with the all the activations of the previous layer. In the final part of the architecture, more fully-connected layers can be positioned in order to make more complex features analysis for computing the predictions: in this cases, often a **Dropout** layer is used between two fully connected layers. Dropout [13] is a form of model regularization based on randomly switch-off some of the neurons of the FC layers during the training. In this way, an ensemble method is simulated because of the different neurons structure that the net assume by dropping-out nodes. It allow to better generalize and to avoid the overfitting phenomenon.

Convolutional Neural Networks are mostly composed by these three layers, that can be assembled in various forms to improve the ability to perform predictions on the task submitted. There are several common architectures in the field of CNN that are relevant and marked by name. Here we resume briefly some of them with their characteristics, in chronological order:

- **LeNet**: first convolutional neural network developed by LeCun.
- **AlexNet**: a more robust, larger and deeper version of the basic LeNet was developed in 2012 and outperformed state-of-the-art. Firstly introduced subsequent convolutional layers without pooling.
- **GoogLeNet**: Google release in 2014 overcomes other competitors by introducing the *Inception Module* and other arrangements that make possible to dramatically reduce the number of networks weights.

- **ResNet**: ResNet, acronym of Residual Network, introduces a simple but terrifically effective novelty by using *skip-connections* to add residuals. The problem of going deeper with convolutional networks is that, when too much layers are connected, the back-propagation of the loss disrupt the information signal and gradually disperse it. This problem is commonly known as **Vanishing Gradient Descent**. The results is that, when the number of layers in the deep network reaches a threshold, the performance gets saturated or degrades. To skip connections, by simply adding a previous signal to the processed output, keep the gradient more stable and do not affect performances. ResNet is the most used architecture used in deep learning, a sort of state-of-the-art architecture from which starts to works.

2.3 Segmentation Algorithms

The core idea of the thesis project is to bridge images to graph representations by a function map that is able to incorporate pixels in similar regions (named as *clusters* or *superpixels* hereafter) and then computes a graph by a unique point from the cluster. This part focus the main algorithms proposed for the segmentation of the input images.

2.3.1 SLIC Superpixels

SLIC Superpixels, acronym of *Simple Linear Iterative Clustering* [14] is an efficient segmentation method that allow to retrieve a selected number (or a close approximation of it) of clusters given an input image. The algorithm uses a particular distance measure among five-dimensional vectors $labxy$ representing the pixels, where lab encode the pixel color in CIELAB space, and x and y the coordinates in the 2d space. Firstly, said given parameter k as the desired number of clusters, k cluster centers are located according to regular distance and than moved in a 3×3 range around the center, selecting the lowest gradient position to avoid non-optimal center positioning (*e.g.* noisy pixels). Then each pixel is assigned at the nearest cluster. This process is executed iteratively until convergence. A brief algorithm presentation in form of pseudo-code is presented is illustrated below:

Algorithm 1 Efficient Superpixel Segmentation

```
Result: Retrieve $K$ optimal segmentation clusters
Given the cluster centers $C_k$, initialize it as centers of a regular grid of step S  Move
  cluster centers in the range of $3 \times 3$ pixels by following the lowest gradient criteria
  to estimate best position  repeat
    for each $C_k$ do
        Assign the neareast pixel to the cluster center in the range of $2S \times 2S$ square
          neighborhood, using the similarity measure formulated for this specific task
    end
    Find new cluster centers positions and residual error $E$, as $L1$ distance from
      previous center position and the new one
until $E <= threshold$;
```

SLIC algorithm is commonly used to perform image or video-frame segmentation, thanks to the strong efficiency, parallelizable (gpu) computation [15], and the possibility to easily select the number of superpixel in output. Due to this reasons, SLIC has been chosen as the main segmentation algorithm for our superpixel-based experiments.

2.4 The Domain-Shift Problem

Basic assumption in theoretical models of machine learning is that training instances are drawn according to the same probability distribution as the test samples. It means that when a model fits on a training set with a certain probability distribution, it is assumed that will be tested on samples with a representation that is close to the one already seen. This hypothesis permits to provide basic guarantees on the correctness of future decisions. Nevertheless, this scenario seems to be hardly applicable in real world applications. Actually, it is more common that the training and test set are drawn from different probability distribution, even with great differences over the two samples set. Distance between domains [16] are due to several reasons that do not make possible to align the two probability distributions, such as time, space, samples source device and so on. Familiar examples of different domains in various machine learning scenarios are:

- For face recognition tasks, training samples are obtained under some set of lighting or occlusion conditions that will probably change during the test.

- In speech recognition, acoustic models trained on one speaker have to be capable to work on other voices.

- In natural language processing, part-of-speech taggers, parsers, and documents classifiers are trained on carefully annotated training sets, but then applied on text from different genders or styles.

- In autonomous driving semantic segmentation context, differences among cities and/or landscapes can represent a typical problem.

The general setting of the problem is that the **source** domain (the domain from witch the training set) mismatches the **target** domain distribution (the domain from witch will be drawn the test samples). While the data drawn from the one are labeled, the second ones are not.

The goal of the cross-domain learning research is to develop algorithms that are robust to **domain shifts**, means that are able to fit the target domain(s) by only leveraging the source domain(s). In this way, we get over the problem of collecting labels for new samples, even avoiding the models to overfit on the available training set.

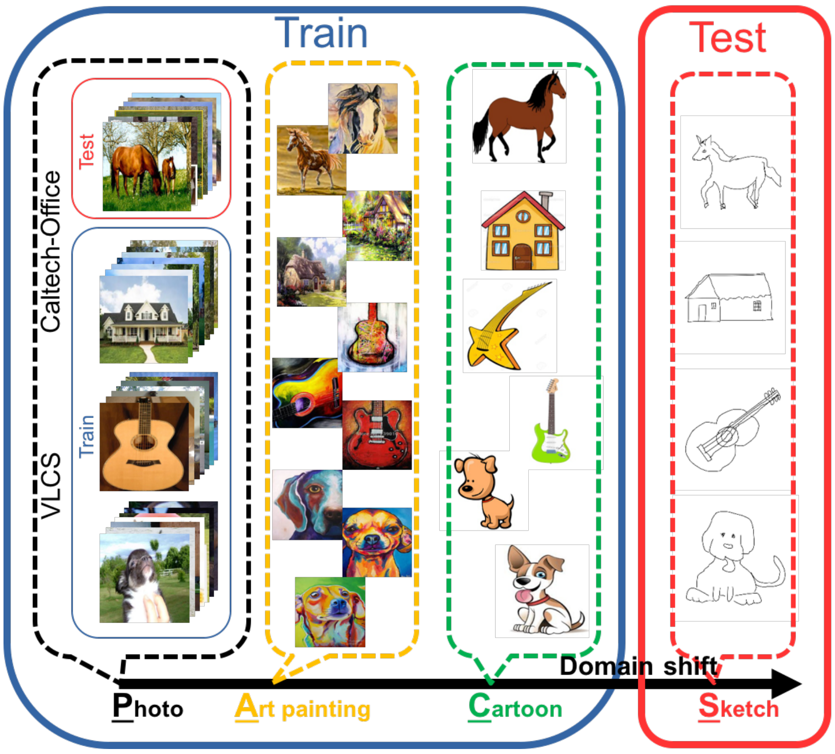

Figure 2.6: Typical examples of domain shift problem
: is it possible to generalize among the three source domains to perform good accuracy on the unseen target? Courtesy of [17].

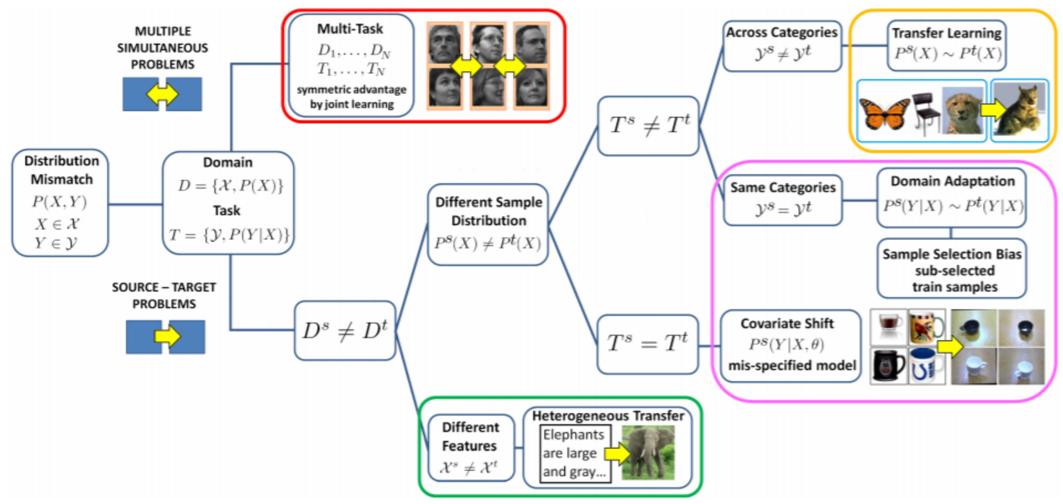

Figure 2.7: Explanatory diagrams representing common domain shift problems such as Domain Adaptation, Transfer Learning, Multi-task Learning. Courtesy of [18]

.

2.4.1 Domain Adaptation

Let's define $X \in \mathscr{X}$ as the input variable (with $\mathscr{X}$ means the input variable space) and with $Y \in \mathscr{Y}$ the output variable or *labels* (where $\mathscr{Y}$ indicate the label space). x and y will be specific values of X,Y. Furthermore we call domain $D = \{\mathscr{X}, P(X)\}$ the pair of feature space and marginal distribution on the data, while a task $T = \{\mathscr{Y}, f(X)\}$ is the pair of label space and prediction function, where the last one can be written in probabilistic terms as $P(Y|X)$.

Domain adaptation defines the problem where, exploiting information from a source domain D_s, we try to solve the learning problem on the target domain D_t, by respecting the following assertions:

- $D_s \neq D_t$, describes cases where samples of the target and the samples of the source correspond to different domain (and probability distribution).
- $T_s \neq T_t$, where T_s and T_t stands for, respectively, source task and target task.
- Label space $\mathscr{Y}$ is identical for both domains, so it is true $\mathscr{Y}_s = \mathscr{Y}_t = \mathscr{Y}$. It means that the discrete or continuous set of assignable labels is composed of the same values for both the target and source.
- $P_s(Y|X) \sim P_t(Y|X)$. Conditional probabilities describe the relation between the input variable and the label, so this hypothesis say that, in context of domain generalization, the probability distributions of the two different domains

are quite similar (but not totally).

Domain adaptation could be particularly critical for service companies, where all machine learning components deployed in a given service solution should be customized for a new customer either by annotating new data or, preferably, by calibrating the models in order to achieve a contractual performance in the new environment.

Domain Adaptation is exploited in two setting, different in the availability of information on target data:

- **Semi-Supervised** Domain Adaptation, when only few samples of the target are labeled
- **Unsupervised Domain Adaptation**, where there is not availability of labels on any samples of the target domain.

There are no differences on the source domain, that is in general fully labeled and with enough samples to perform the supervised training of the model.

All these definitions refers to case where a single source domain is used in supervised adaptation. However, a better option is to use a multiples source domain approaches, also known as **multi-source domain adaptation**, which, relying only on known domain labels, are able to exploit the specificity of each source domain.

Several works has attempted to reduce the problem of DA [19]: recent works focused on settings where on cases where the source data are drawn from multiple distributions [20], [21], and with source classes that are only partially covered by the target [22], [23]

2.4.2 Domain Generalization

As multi-source DA asset, **domain generalization** approach aims to extract general knowledge from several related source domains, in order to learn a model for a new target domain. But, in contrast to domain adaptation, where unlabeled target instances are available to adapt the model, in domain generalization the model do not have any access to samples of the target. So domain generalization experiments result more tricky than adaptation because the source has to be leverage in order to learn a general representation that allow to reach valid knowledge even for unseen domains. A typical example of domain generalization challenge is the one related to **PACS dataset** [17], which is composed of instances taken from four domains. The four domains (*art paining, sketches, photo, cartoon*) have to be treated in a sort of cross-validation setting to test the generalization power of the model: for each domain, the model has to be trained on the other three and then tested on the selected one. The test accuracies of the four experiments, that can be even very different from each other, will represent the relative generalization capability on a certain domain. The average of the scores will represent the benchmark of the overall ability of the model to learn in a domain-agnostic way.

For DG setting with no target data available at training time, a part of the previous research works showed model-based strategies to prevent domain specific signatures from multiple sources. They are both methods that build over multi-task learning [24], or domain specific aggregation layers [25]. Other research results exploit source model weighting [20], or validation measures on tests virtually defined from the sources available [26]. Other feature-level approaches search for a data representation able to capture information shared among multiple domains. This was based on the use of convolutional-based auto-encoders in [26]–[28], while [29] proposed to determine an embedding space where samples of same classes but from different source domains are projected nearby. [26] propose to adversarially exploit class-specific domain classification modules to cover the cases where the covariate shift assumption does not hold and the sources have different class conditional distributions.

2.4.3 Transfer Learning

Transfer learning focuses on the possibility to pass useful knowledge from a source task to a target task with different label sets $\mathscr{Y}_s$, that differs from $\mathscr{Y}_t$, when the corresponding domains are not the same but the marginal distributions of data are related [30].

In contrast to domain adaptation, now the classes contained in the source and target set are not the same. Thus it is always necessary to evaluate in practice how much the tasks are related and whether it is really worth to rely on the source knowledge when solving a new learning problem[18]. Transfer Learning is commonly used by the scientific community in different forms, according to the availability of labeled samples for source or target problems.

- When **labels are available for both source and target**, we can perform transfer learning in a supervised way. TL is often performed in this form when we want to avoid the overfitting of the models when the number of target samples is too low: in this case giving more samples, even from different domains, let the model learn in a more general and loose fashion. Noticeable kind of this transfer learning form is the *one-shot learning*, when training of the model is based on only one example of the target domain.

- When no label are available for the target domain, the task takes the name of **transductive transfer learning**. This category embrace also the *zero-shot learning*. Zero-shot learning refers to the cases in which you the model has to classify data based on very few or no labeled samples, meaning on the fly. Briefly, zero-shot learning is about exploiting deep learning networks, pre-trained by supervised learning, in new tasks, without the possibility to another supervised training phase.

- Cases in which labels are available for the target domain but not for the source are called *self-taught learning* [31]. In these situations, the application involves unsupervised methods to extract useful information from the source domains, that will be used to improves the supervised learning phase of the target domain. Thus, source knowledge is collected by an high-level representation.

Transfer Learning approach, when properly used, gives sensible advantages in comparison to a learn "from scratch":

- **Initial performances** generally achieve, for target problem, stronger results compared with the random-initialized networks. This is an natural consequence of starting from a reasonable initial state respect to an optimized but casual setting. [32]
- Starting with a prior extracted knowledge means **shorter time** to complete the learning process of the target task. [32]
- Often, TL allow to overcomes scores made by learning from scratch. This means that we can get **greater final scores** with application of transfer learning approach, when possible. [32]

2.4.4 Multi-Task Learning

Multi-Task Learning is based on cases where multiple sets of data can be used for training the network. With this setting, it is impossible to formulate the problem with a source and a target task, but we want to learn from the sets at the same time. The sets in which we are relying, supposed to be in number equal to N, will share the same feature space $\mathscr{X}$, with domains that presents different but close probability distributions $P^i(X) \sim P^j(X)$ for $i, j \in \{1, .., N\}$. Other main assumption is that the label space $\mathscr{Y}$ is even in common or is it possible to formalize a mapping functions that traduce in a deterministic way $\mathscr{Y}^i$ into $\mathscr{Y}^j$. Experiments have shown that the model performance can be improved by adding new tasks to the network, that has to be achieved simultaneously. This happens thanks to the fact that each various simultaneous task works as bias for the others, driving the inductive learner to prefer some settings than others: the multi-task bias causes the inductive learner to prefer hypotheses that explain more than one task [33]. Some solutions can be used to face the relatedness of these tasks: simplest approach includes the usage of closeness measures. The impact of each task in the learning process can be weighted according to the importance of the problem, so it is possible to implicitly define principal and secondary tasks. Furthermore, it has been demonstrated the usefulness of use multiple tasks even if there is not correlation between each others.

The assumption behind MTL is borrowed from the living beings way of learning: while machine learning systems in general learn single, highly-focused tasks, people use to learn multiple and very different tasks. It is right to ask ourselves if "the similarities between the thousands of tasks you learn are what enable you to learn any one of them" [34]. The application demonstrates that learning multiple related tasks simultaneously, sharing features or learning processes, make the learning process more robust then single task learning. A general example in Computer Vision could be to train a neural network to simultaneously train the neural network to recognize objects and their shapes, textures, size, orientation and so on, and in this way improve the learning ability to recognize complex real world objects.

So, briefly, the main advantages MTL gives are:

1. Multitask models resolves multiple tasks using a compact unique structure, that requires less resources (in terms of memory usage, time, etc) than training two different specific models.
2. Despite the aggregate gradient in MTL is flatter, it is often more robust to noise.
3. The model results pointed in MTL will be in general more meaningful compared to single task learning. That's because the aggregate gradient will drive the model to choose solutions that fits well for more tasks, i.e. towards more generally useful features.

The MTL models can be implemented in a variety of ways depending on the architectures, number of shared parameters, the way in which they are shared, etc. In particular, there are two main categories of parameters sharing that can be adopted in MTL:

- **hard parameter sharing**, in which some hidden layers are shared between all tasks, while the final layers are task-specifics. The number of layers and parameters shared or specific depends on the number and the weights of the multiple tasks, and in generally requires inferences on test.
- **soft parameter sharing**, where single models are used for each task. In this settings, the parameters of the model are regularized to maintain similarities

among the nets. Regularization is done by applying typical normalization operations such as $L2$ norm, or with more complex techniques.

A diffuse way to improve the generalization ability via multi-task learning is to setup a self-supervised task, based on source dataset in Domain Generalization setting or based on both source and target when the setting of experiments corresponds to Domain Adaptation [24].

2.4.5 Domain Generalization by Solving Jigsaw Puzzles

It has been demonstrated [10] that, in a standard domain generalization setting problem of computer vision, it is possible to reduce the domain shift problem by adding a secondary task that, briefly, ask to the network to solve a simple jigsaw on the image. So: starting from S domains, where N_i is the number of labeled samples $\{(x_j^i, y_j^i)\}_{j=i}^{N_i}$ of the i-th domain, with x_j^i and $y_j^i \in \{1,..,C\}$ represent respectively the input image and class label of the instance j. To reach the first goal of the model, that is to minimize the distance between label predicted by the model and the ground-truth labels, it is asked to the model to satisfy a secondary condition relative to jigsaw puzzles autonomous solution. Let us define as $L_c(h(x|\theta_f, \theta_c), y)$ the loss that measure the error between ground-truth y and prediction obtained by submitting the input x at the deep network $h(\cdot)$ representing and parameters (Θ_f, Θ_c) the setting of the whole amount of weights in the CNN (f stands for feature extraction weights, c for classification-oriented layers). Together with this, it is introduced another loss related to solving jigsaw: it means that a source image is decomposed in $n \times n$ patches that are then shuffled on the space of the image. The network has the task to find the correct permutation order among a dictionary of $P < n^2!$ permutations, where $n^2!$ is the maximum number of possible permutations. So it is possible to define a second classification-based task on K_i labeled samples, with $\{(z_k^i, p_k^i)\}_{k=1}^{K_i}$ where z_k^i indicates the effectively recomposed instance (original sample before patch shuffling) and $p_k^i \in 1,..,P$ the predicted permutation index among the P possibilities. The loss function related to the task will be

$$L_p(h(z|\theta_f, \theta_p), p)$$

. Looking at the loss definition, it appears clearly that the deep model shares a part of the network θ_f with the net, while the last part, θ_c and θ_p are separated and allow to outputs the two task-related prediction. The training phase is processed by solving the optimization problem

$$\mathrm{argmin}_{\theta_f,\theta_c,\theta_p} \sum_{i=1}^{S} \Big(\sum_{j=1}^{N_i} L_c(h(x_j^i|\theta_f, \theta_c), y_j^i) + \sum_{j=1}^{N_i} \alpha L_c(h(x_k^i|\theta_f, \theta_c), p_k^i) \Big) \tag{2.7}$$

Where the loss functions are standard cross-entropy loss, and α represents the weight of the secondary task.

Implementation requires to define some constraints on the experiments:

- $P = 30$, number of possible permutation
- $\beta = 0.6$ represents the percentage of shuffled samples in each batch size that will orient the secondary task learning.
- $n = 3$, where $n \times n$ represents the number of patches in which the original image is decomposed.

Experiments demonstrate that this secondary task allow to overcomes state-of-the-art results on several domain generalization application such as digits and PACS dataset.

2.4.6 Domain Adversarial Training

The appeal of the domain adaptation approaches is the ability to learn a mapping between the source domain and the target domain in the situation when the target data are fully unlabeled, or in simpler cases, with few labeled instances. The first case, *unsupervised domain adaptation*, requires to drive the model learning to build mappings between the source and the target, so that the classifier learned for the source domain can also be applied to the target domain, without supervised instance of the target. One noticeable approach to exploit representation learning for domain adaptation is the proposed *Domain Adversarial Training* [11]. As training progresses, the approach promotes the choice of features that are both discriminative for the main learning task on the source domain and indiscriminate with respect to the shift between the domains. By doing this, the learning process is oriented to extract the most effective features that allow to solve the main task and to be meaningful also for the target domain. To learn a classifier that can generalize in the best way from one domain to another, it is necessary that the internal representation of the neural network contains minimum or, better, no discriminative information about the origin of the input (source or target), while preserving a low risk on the source samples. Application of this principles can be applied on shallow neural networks or even any general classifiers. In the field of Convolutional Neural Networks, the approach used is to define a CNN with two tasks: the first, the principal, is focused on solving the standard supervised classification task on the source domain, while the second has to answer to the needs of features that align the domains. To achieve this, the domain adversarial networks requires a *Gradient Reversal Layer*, that simply returns the input unchanged during forward propagation while reverses the gradient by multiplying it by a negative scalar during the back-propagation. Particularly the architecture requires three different modules:

- **Feature Extractor**: the module is deputed on extracting the optimal features to perform the representative learning. It represents the first common branch of the networks, that then forks on the two following modules

- **Classifier**: this modules takes in input the features and exploit fully-connected layers to report the prediction on the class the inputs belongs to.
- **Domain Classifier**: the domain classifier module has in charge to discriminate among domains. It means that it will predict at which domain each sample belongs. Simple example of Domain Classifier for two domains (source and target) requires a binary classifier.

It is clear that the contribute of the domain discriminator in the back-propagation phase is to drive the model to learn features that contributes to distinguish the source from the target. Because it is exactly the opposite behavior we want, by inverting the contribute of the loss with the GRL, we are driving the neural networks to extract features that will confound the two domains. By this simple approach, we can get heavy improvements in the generalization ability of the model and strong reduction of the domain-shift problem.

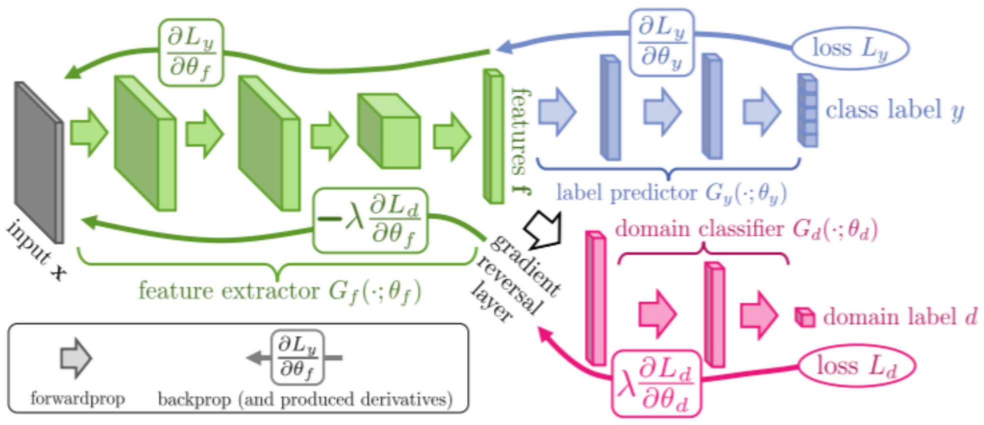
$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features f
label predictor $G_y(\cdot;\theta_y)$
input x
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
gradient reversal layer
domain classifier $G_d(\cdot;\theta_d)$
feature extractor $G_f(\cdot;\theta_f)$
domain label d
loss L_d
$\lambda \frac{\partial L_d}{\partial \theta_d}$
forwardprop
backprop (and produced derivatives)

Figure 2.8: DANN general Neural Network architecture, courtesy of [11]

Chapter 3

Preliminaries on Graph Deep Learning

3.1 Introduction to Graph Theory

In our thesis project, much of the learning process relates to the use of geometric graph structures. It is therefore essential to give an overview on the concepts and properties related to **graphs**.

A graph is a geometrical representation of a set of objects that are connected, two by two, in some ways. This flexible behaviour allow to represent information drawn from various and different domains.

The interconnected entities are called *vertices* or *nodes*, while the connection between the nodes are called *edges*. A graph $G = (V, E)$ can be formalized as the set of the vertices V and the set of edges E.

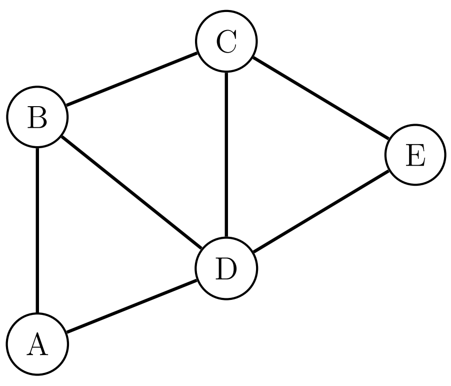

Figure 3.1: Example of simple, general graph.

The figure 3.1 depict a visual representation of a general graph $G_1 = (V_1, E_1)$. In a mathematical way of visualize, G_1 is:

$$G_1 = (V_1, E_1) \text{ where: } \begin{cases} V_1 = \{A; B; C; D; E\} \\ E_1 = \{(A,B); (B,D); (A,D); \\ (B,C); (C,D); (C,E); (D,E)\} \end{cases} \quad (3.1)$$

This definition is so abstract. We can notice that graphs has not an explicit positioning in the space: nodes positional references depends only on the connections between nodes and the representation in 3.1 is just one of the infinite possible designs in which we can depict it.

The edges of the represented graph has not a direction, and thus it is a *undirected graph*. By defining a directed graph, we have to label each edge with an explicit direction, i.e. connection (A,B) just represent the connection *from A to B*, and not vice versa. Directions can also seen as assigning at each edge a sign, where "1" represents the direction from the first node to the second while "-1" the inverse direction. This is the first example of *weighted graph*, where each edge is distinguished by a weight, that will be an index of some information related. Labels, as vertices, do not have a limited format, but can be expressed by numbers, strings, vectors and so on.

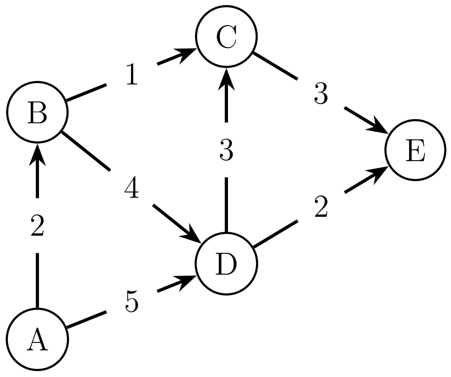

Figure 3.2: Example of weighted, directed graph.

Another typical representation of graphs, particularly familiar for computational purpose is the **Adjacency Matrix**. An adjacency matrix is a square array whose rows are out-node and columns are in-nodes of a graph (so direction is understood): each entry of the adjacency matrix represents an edge with his weight. A zero means no connection between the related vertices. This compact matrix takes care of all the properties we have explained before. We can see a brief example in table 3.1

Lastly, an important concept bounded with deep learning methods on graphs is the *graph walk*: a *path* or a *walk* is a collection of vertices for which it is true:

$$W_1 = \{v_1, v_2, ..., v_n\} \;:\; (v_i, v_{i+1}) \in E \;\forall\; i \in \{1, ..., n\} \quad (3.2)$$

To enrich these concepts,it can be useful to define the *signal function*, or *vertex function*, $f : \mathcal{V} \to \mathbb{R}$, defined on the vertices of the graph. It can be represented as a vector $f \in \mathbb{R}^N$, where the i^{th} component is the function value related to vertex $i^{th} \in V$ of the graph. In this way, we are able to perform deep learning methods based on the input features defined by function f.

	A	B	C	D	E
A	0	0	0	0	0
B	2	0	0	0	0
C	0	1	0	3	0
D	5	4	0	0	0
E	0	0	3	2	0

Table 3.1: Adjacency matrix representation for graph in 3.2.

3.2 Geometric Deep Learning

Since the advent of the Neural Networks, many were the attempts to propose applications on different data representations. The majority of successful applications in this regard are mainly on euclidean data: it includes data representations in one or two dimensions. Examples of euclidean representations involves images, texts, audio records, signals, etc. Each of these is an example of discretization of physical information into euclidean representations, but other types of geometry can be involved in sampling processing. **Non-Euclidean Geometry**, briefly, can reproduce more complex data in comparison with mono-dimensional or bi-dimensional samples: commonly associated to this concept are **Graphs and Manifolds**. **Geometric Deep Learning** involves every attempts of generalizing neural networks methods on non-Euclidean structures.

3.2.1 Deep learning on Euclidean Data

To deeper understand the theoretical concepts about Convolutional Neural Networks, and thus the advantages of introducing a new, non-euclidean, method for data analysis, let us introduce some formal definition of CNN components[1].

Standard CNNs applied in computer vision tasks consists generally on several convolutional layers, passing the input images through a set of filters Γ and then on a non-linearity layer ε. The model generally includes a bias term, equivalent to the addiction of a coordinates constraint to the input.

[1]mathematical formulations are largely inspired from [35]

Defining:

1. Euclidean domain $\Omega = [0,1]^d \subset \mathbb{R}^d$ of d dimensions.
2. $f \in L^2(\Omega)$ as the square-integrable function that represents the input of the network. For image application domain, function f belongs to $\Omega = [0,1]^2$
3. $y : L^2(\Omega) \to Y$ unknown function that describes the association between labels $y \in Y$ in general supervised tasks. For a supervised classification experiments, Y is a discrete set of definite dimension $|Y|$, while for regression tasks, $Y = \mathbb{R}$. It is true that a label is assigned to each inputs. This statements is resumed in the definition:
$$\{f_i \in L^2(\Omega), y_i = y(f_i)\}$$

By starting from these definitions, Convolutional Neural Networks are based on prior geometric knowledge on the function y that constitute a fundamental assumption to validate the relative methods:

- y is *translation-invariant or -equivariant* in relation to the required task. It means that the translation do not perturb the output of the unknown function y or, at least, it translates whenever the input translates.
- Most of the y, according to the Computer Vision task, can be *stable to local deformations or scale mutations.* For translation-invariant tasks, means that the probability prediction is not heavily muted in case of slight deformation of the input image. For translation equivariant, the property of stability is even stronger. This properties make possible to validate another core assumption of the CNN, that leads to the possibility of progressively reduce the dimension of the layer input without losing substantial information of the original image. In other words, "whereas long-range dependencies indeed exist in natural images and are critical to object recognition, they can be captured and down-sampled at different scales."[35]

A **convolutional neural networks** consists of several **convolutional layers**, defined as $\mathbf{g} = C_\Gamma(\mathbf{f})$ acting on an input that works on p dimensions, defined by the vector of functions $\mathbf{f}(x) = ((f_1(x), .., f_p(x)))$ by applying a set of filters (*kernels*) Γ where $\Gamma = (\gamma_{l.l'}); l = 1, .., q; l' = 1, .., p$ and subsequently a predefined point-wise non-linearity ξ. Following this definition, application of convolutional operation over x is represented as:

$$g_l(x) = \xi(\sum_{l'=1}^{p} (f'_l \star \gamma_{l,l'})(x))$$

The **convolutional layers** define the core of the networks: the duty of these layers is to extract meaningful features by applying the convolutional operation on small squared windows of the input volume and returning in output a q-dimensional vector $\mathbf{g}_l(x) = (g_1(x), .., g_l(x))$ that is called **feature map**, meaning the map of features generated by convolutional layer. Standard convolution is generally denoted by:

$$(f \star \gamma)(x) = \int_{\Omega} f(x - x')\gamma(x')dx'$$

In addiction to these operations, one of the fundamental layer in a CNN architectures is the **pooling layer**, that consists of a rational downsampling operation to reduce the amount of data to compute. The pooling layer $\mathbf{g} = P(\mathbf{f})$ can be stated as:

$$g_l(x) = P(f_l(x') : x' \in N(x)), l = 1, .., q$$

with $N(x) \subset \Gamma$ define the neighborhood of x and P define a permutation-invariant function applied over all $x \in N(x)$. Common examples of pooling logics are average pooling, where the pooling layers outputs the average measure of the $N(x)$ set, and max pooling layers, where the output is $x^* : x^* > x \ \forall \ x \in N(x)$. Convolutional Neural Networks are essentially based on the concatenation of several convolutional and pooling layers in a hierarchical disposition that begins to the input and ends with an output that returns the probability of prediction among the possible target labels. The entire processing function exploit in CNNs can be defined by:

$$U_{\Theta}(f) = (C_{\Gamma^{(K)}} \circ ... \circ P \circ ... \circ C_{\Gamma^{(2)}} \circ C_{\Gamma^{(1)}})(f)$$

where the hyper-vector $\Theta = \{\Gamma^{(1)}, .., \Gamma^{(K)}\}$ consists of the networks parameters (or weights, means all the coefficients of the filters of convolutional layers).

these general definitions can be applied in the field of computer vision (*i.e.* image classification, object retrieval, face detection, etc.) by considering the input images as three-dimensional matrix with dimension $h \times w \times d$ where h represents the height, w the width, and d the *dimension* to represent the pixel gradient: $d = 1$ for grayscale images, $d = 3$ in case of RGB format, $d = 4$ in case of RGB-D inputs as in [36].The *depth* of the networks refers to the multiple layers positioned in the hierarchical structure. As more the research goes forward, as more complex the networks becomes, and this is not only related to the deeper *backbones* (meaning the structure of the conv and pooling layers that constitues the network), but also in increasing the efficency and performances of the single layers. Convolutional Neural Networks give us today an incredible space in research, with different directions and new scenarios every day.

3.2.2 Deep Learning on Graphs

Graphs are meaningful data structures, based on nodes and edges, that allow to reproduce individual features, while also providing information regarding relationships and structure. Graphs are even flexible representations, particularly suitable to depict physical interactions among different objects: it can models protein interaction networks, molecules, Feynman diagrams, cosmological maps, etc. Even Neural Networks are represented in both conceptual and effective maps (deep learning frameworks defines graphs to embed neural networks).

Adaptation of traditional Deep Learning methods to graph domain is bounded to some issues that need to be managed:

- **Irregularity of representations**: while euclidean geometry samples belong to continuous and (almost) grids-discreditable representations such as images, audios or texts. Graphs do not lie in a regular domain, making hard to generalize some basic mathematical operations to graphs. To redefine convolution and pooling operation, that constitute core implementations for CNN architectures, for graph data was non-trivial task for developers.

- **Different definitions for graph structures**: as already explained, several graph structures can be defined. Examples of graph discriminations are: heterogeneity (heterogeneous or homogeneous), edge-weights definition (weighted or unweighted), edge direction definitions (directed or undirected),etc. In addition, tasks for graphs also are affected of many different variations, ranging from node-focused problems such as node classification and link prediction, to graph-focused problems such as graph classification and graph generation. The more tasks are defined, the more different architectures are necessary to tackle the specific challenge.

- **Scalability**: to model real world information requires millions edges and nodes, especially in our era. It is necessary to develop scalable architectures. As a result, how to design scalable models, preferably with a linear time complexity, becomes a key problem. In addition, since nodes and edges in the graph are interconnected and often need to be modeled as a whole, how to conduct parallel computing is another critical issue.

- **Interdisciplinary**: potential of graphs could prove also challenging, since it is necessary to leverage domain knowledge to solve specific problems but integrating domain specificities could make designing the model more difficult. For example, to handle loss optimization for molecular graphs is non-trivial due to not-differentiable nature of the objective function and chemical constraints.

3.2.3 Graph Convolutions

We would like, at this point, to introduce some theoretical concept to understand the convolutional neural networks on graphs [35]. We will consider, for the sake of simplicity, only *weighted undirected* graphs, meanings graphs in which a weight, but not a direction, is assigned at each edges. Given a graph $\mathscr{G} = (\mathscr{N}, \mathscr{E})$ where $\mathscr{N} = \{1, .., n\}$ is the set of n vertices (or nodes) and $\mathscr{E} \subseteq \mathscr{N} \times \mathscr{N}$ is the set of edges between two vertices. Since an undirected edge e among nodes i and j connect both i to j and j to i, the undirected nature of graph can be formalized with the assertion $(i, j) \in \mathscr{E} \iff (j, i) \in \mathscr{E}\ \forall\ i, j \in \mathscr{N}$. Furthermore, it is true the assumption that specifies that a weight is defined for all the nodes, so: $a_i > 0\ \forall\ i \in \mathscr{N}$. At the same way, a weight is defined for all the edges $w_{ij}\ \forall\ (i, j) \in \mathscr{E}$. Real functions $f : \mathscr{N} \to \mathbb{R}$ and $F : \mathscr{E} \to \mathbb{R}$ are function defined on nodes and edges of the graphs. To Define the Hilbert spaces[2], respectively $L^2\{\mathscr{N}\}$ and $L^2\{\mathscr{E}\}$, of such this functions is possible by specifying the inner products:

$$\langle f, g \rangle_{L^2(\mathscr{N})} = \sum_{i \in \mathscr{N}} a_i f_i g_i \tag{3.3}$$

$$\langle F, G \rangle_{L^2(\mathscr{E})} = \sum_{(i,j) \in \mathscr{E}} \omega_{(i,j)} F_{(i,j)} G_{(i,j)} \tag{3.4}$$

Hilbert spaces are fundamental prior to define differential over $f \in L^2(\mathscr{N})$ and $F \in L^2(\mathscr{E})$. We can define the *graph gradient* with the operator

$$(\nabla f)_{ij} = f_i - f_j \tag{3.5}$$

that satisfies $(\nabla f)_{ij} = (\nabla f)_{ji}$. On the other side, we can define *graph divergence* as operator $div : L^2(\mathscr{E}) \to L^2(\mathscr{N})$:

$$(\mathrm{div} F)_i = \frac{1}{a_i} \sum_{j:(i,j) \in \mathscr{E}} \omega_{ij} F_{ij} \tag{3.6}$$

Operator *div* represents the reverse operation in comparison to the gradient operator. The *graph Laplacian* is an operator $\Delta : L^2(\mathscr{N}) \to L^2(\mathscr{N})$ that can be represented as the operator $\Delta = -\mathrm{div}\nabla$. By substituing the equation ref 3.5 and 3.6 it is possible to obtain a known expression of the formula:

$$(\Delta f)_i = \frac{1}{a_i} \sum_{j:(i,j) \in \mathscr{E}} \omega_{ij} (f_i - f_j) \tag{3.7}$$

The equation representation 3.7 allow to understand the geometric concept of the Laplacian, that is the difference between the local average of a function applied around a certain point, and the exact value of the function at the point itself.

[2]In mathematics, the *Hilbert space* is an "inner product space that is complete with respect to the norm defined by the inner product. Hilbert spaces serve to clarify and generalize the concept of Fourier expansion and certain linear transformations such as the Fourier transform."

Laplacian operator is a positive-semidefinite operator. In compact domains, meaning domains limited to a finite interval and where the Fourier transform of functions defined on it will be discretes (all practical applications use compact domains), Laplacian operator admits a eigendecomposition in a discrete set of eigenfunctions ϕ_i and real eigenvalues for which it is true $0 = \lambda_0 \leq \lambda_1 \leq \lambda_2 ...$ and $\Delta\phi_i = \lambda_i\phi_i,\ i = 0,1,2, ...$

Fourier Series of a function f defined on $\mathscr{X}$ can be obteined by:

$$f(x) = \sum_{i\geq 0} \langle f, \phi_i \rangle^2_L(\mathscr{X})\phi_i(x) = \sum_{i\geq 0} \Big(\sum_{i\in\mathscr{N}} a_i f_i g_i \Big) \phi_i(x) \tag{3.8}$$

Last representation is the graph-based equation that is obtained by substituing the inner product with the definition in 3.3.

Fundamental component of the Euclidean CNN processing is, obviously, the **convolutional operation** $f \star g$. The convolution operation can be generalized to work in non-Euclidean context with the definition:

$$(f \star g)(x) = \sum_{i\geq 0} \langle f, \phi_i \rangle_{L^2(\mathscr{X})} \langle g, \phi_i \rangle_{L^2(\mathscr{X})} \phi_i(x) \tag{3.9}$$

Even in this case, the inner product can be replaced by 3.3 in graph-based contexts. This adaptation do not support the shift-invariance property of the classical convolution in Euclidean domain; meaning that the filters are dependent on the position and every shift in the input can change the results, even widely.

In a matrix-vector notation, the convolution can be defined by the expression

$$\mathbf{Gf} = \mathbf{\Phi}\mathrm{diag}(\hat{\mathbf{g}})\mathbf{\Phi}^{\mathbf{T}}\mathbf{f}$$

where $\hat{\mathbf{g}} = (\hat{g}_1, \hat{g}_2, ..., \hat{g}_n)$ is the spectral representation of the filter and $\mathbf{\Phi} = (\phi_1, \phi_2, ..., \phi_n)$ is composed by Laplacian eigenvectors.

3.2.4 Graph Coarsening

This is a recall of the concept of pooling defined in euclidean CNN.It is necessary to define, in order to replicate standard convolutional network pipelines, a layer that gives the possibility to:

1. Lighten the overall features to gain in performanes

2. Enrich the generalization power by avoid overfitting

3. Get some invariance to local and global deformations.

Typically the operation of downsampling a given graph $G = (V, E)$ into $G^{red} = (V^{red}, E^{red})$ is referred as **graph coarsening**. The standard approach foresees the search of grouped vertices and perform pooling operations starting from those groups. One common method of coarsening is the *Normalized Cut*, based on the assumption that the dissimilarity between two sets of vertices can be computed as the total weight of the edges that connect nodes of the first set to nodes of the second one, and viceversa. Such this approach has proved effective in various applications, but is characterized by a considerable computational complexity. Another approach often used in modern application is the *Graclus* [37] algorithm, with a more efficient way to perform graph downsampling.

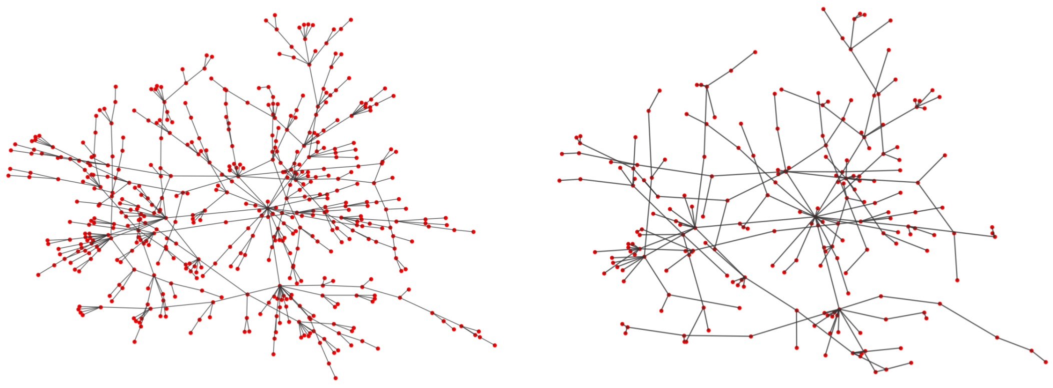

Figure 3.3: Example of graph coarsening (before and after)

3.2.5 Previous Works

Literature in the field of graph prediction models is substantially condensed in two main branches: the first field try to to generalize **signal processing** methods to graph-based data [38]. Spectral analysis techniques were applied to graphs by considering orthogonal eigenfunctions of the Laplacian operator as a generalization of the Fourier basis. Constructions such as wavelets [39]–[43] or other algorithms such as dictionary learning [44], Lasso [45], PCA [46], originally based on Euclidean geometry, were also adapted to non-Euclidean graph-structured data. Then, the second and more recent field regards the deep learning models applied on graph structures. The interest in non-Euclidean deep learning is due to the recent works of Bruna et al. [5], [47] in which the authors presented convolutional-based [48] neural networks on graphs in the spectral domain, leaning on an analogy between Fourier transforms and results deriving from graph Laplacian operator [38]. Lately, [49] proposed an efficient filtering approach that does not require explicit computation of the Laplacian operator by approximate it with recurrent Chebyshev polynomials. Then, the approach was intensely lightened by Kipf and Welling [6], using simple filters operating on one-hop neighborhoods of the graph. Similar methods were proposed in [50] and [51]. Finally, in the network analysis community, several works began to base on graph embedding [52]–[56] methods inspired by the technique mainly known as Word2Vec [57]. A key criticism of spectral approaches such as [5], [47] is the problem that the spectral definition of convolution is dependent on the Fourier basis, which is domain-dependent. It implies that a spectral convolutional neural networks tuned one graph cannot be easily applied to another graph with a different Fourier basis, as it belongs to a complete different domain.

3.2.6 Dynamic Edge-Conditioned Filters

Thanks to the fast spread of Deep Learning on Graph and Manifolds, new architectures for nets and convolutional layers are frequently published. So it was necessary to pick the architecture that fit our problem (image-based graph analysis) as best as possible.
The architecture that we have choosen as backbone of our network is a Convolution Neural Network that works on both directed or undirected graphs, called **ECC** [7]. ECC network is based on **edge-conditioned convolution layers**, that exploits both node and edge labels during convolution over local graphs neighborhoods. We used ECC network essentially to perform graph classification, means identifying label for the whole input graph, but it could be used likewise for local nodes classification.
The Edge-conditioned convolution defined in ECC takes care of both vertex and edge labels to performs the output predictions. Let us present a formal definition of the convolutional layer.

Considering:

- G as a directed or undirected graph, for which:
  - V represents the finite set of vertices componing the graph, where $|V| = n$
  - E is the finite set of edges that connect two vertices, so $E \subseteq V \times V$ and $|E| = m$
  - Exists $X^l : V \to \mathbb{R}^{d_l}$, the function that assign a label (also named in this context signal or feature) at each vertex (vertex-labeled graph). By representing this function as a matrix $X^l \in R^{m \times d_l}$, X^0 shall identify the input signal of the convolutional layer.
  - Exists $L : E \to \mathbb{R}^d$, the function that assign at each edge one label, or attribute (edge-labeled graph). The function L can be represented as a matrix $L \in R^{m \times s}$

- $l \in \{0,..,l_{max}\}$ as the layer index that localize a layer among the feed-forward network in use(*i.e.* Multi-Layer Perceptron).

Then, define:

- $N(i) = \{j; (j,i) \in E\} \cup \{i\}$ define the *neighborhood* of the vertex i, means the set of all adjacent vertices (or, in directed graphs, predecessors) of i, considering i itself.

The ECC approach is based on obtaining the filtered signal $X^l(i)$ by combining all the signals of the neighbors defined by $N(i)$. So the output of an edge-conditioned convolution layer will depend on the linear combination (weighted sum) of the signals $X^{l-1}(j)$ where $j \in N(i)$. To keep in care structural information of the edge attributes, the weight used in the weighted sum is computed by using a dinamic filter networks that, given an edge attributes $L(j,i)$ will return a specific weight $\Theta^l_{ji} \in R^{d_l \times d_{l-1}}$, by a specific function $F^l : R^s \to R^{d_l \times d_{l-1}}$.
The formal definition of the convolutional function, said b^l and F^l the dinamic filter network parametrized by learnable networks weights ω^l as learnable bias:

$$X^l(i) = \frac{1}{|N(i)|} \sum_{j \in N(i)} F^l(L(j,i); \omega^l) X^{l-1}(j) + b^l = \frac{1}{|N(i)|} \sum_{j \in N(i)} \Theta^l_{ji} X^{l-1}(j) + b^l$$

As reported on paper [7].
This conv-layer is proposed, albeit with some variations of the common layers that has been adapted in the context of graphs processing (e.g. a different approach is required for max pooling applied on graph inputs compared to the one used for images) to compose deep networks.
Our experiments has been executed on both original ECC source code and a custom code based on PyTorch Geometric Framework [58].
The ECC-conv layer proposed has some analogies to the standard convolutional layer of the euclidean-based neural networks: as the nets use to apply the kernel to a $n \times n$ set of pixels, which in general is a small number ($i.e.$ 3), we can consider the pixels as nodes and these small frames as the neighborhood of the node $N(i)$. In this way, we allow to overcomes the standard grid application of the convolutional layers, but we can define the neighborhood as we prefer. Furthermore, we can weight each connection among nodes by assigning a value to the edges, that is even considered in the convolution.

Applications of the ECC-based neural networks, since its discovery, on simple images have achieved good results even compared with specific CNN on images.

Chapter 4

Experiments

4.1 Settings and Implementation

4.1.1 Digits Datasets

Since its introduction, *Digit Recognition* has represented one of the most common problems linked to computer vision. The reputation is due to the fast spread of the problem of recognizing handwritten documents and to the simplicity of the task. Indeed, digits recognition is considered the elementary problem for computer vision: generally, digits classification is used as initial experimental setting to inference if a particular predictive model can be properly applied in visual learning tasks. A digits recognition model is used to label unseen instances of handwritten digits with the relative represented number, from 0 to 9. The problem is just a variant of the standard classification task, and the learning process follows the supervised approach. Digits recognition is imputed to be a too easy problem to actually check the learning power of a model, but even for humans recognize digits can be harder than it looks: similarity between digits (i.e. 1 and 7, 5 and 6, 3 and 8, 2 and 5) and visualization problems (i.e. view occlusions, distortions, low quality images, noise) can make the recognition difficult or even impossible.

Furthermore, various digits datasets have been developed or acquired during the last decade, with large difference in representation, style and probability distribution. For this reason, we can work with different domains of digits datasets and experiment techniques of domain adaptation, domain generalization and transfer learning on them. Indeed investigating the gap between digits domains and how to close them has recently attracted a large attention in the scientific community [59]–[62].

MNIST

The MNIST database [63] is composed of black-and-white handwritten digits, with a training set of 60,000 examples, and a test set of 10,000 examples. It was developed by National Institute of Standards and Technology (USA) and is just a subset of the original dataset. The digits have been size-normalized and centered in a fixed-size image: specifically, the original bi-level images were adjusted to fit in a 20×20 pixel box and to preserve the aspect ratio. Images were in turn centered in a 28×28 window by computing the center of mass of the pixels, and translating the image so as to locate this point at the center of the 28×28 field.

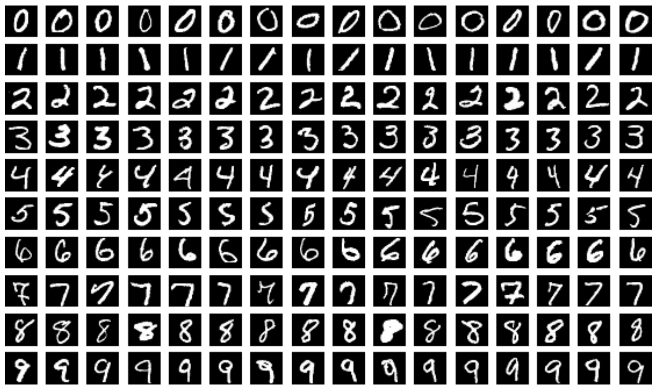

Figure 4.1: MNIST Dataset

.

A random sub-set of instances is available in figure 4.1

It represents a crucial starting point for digits recognition and computer vision challenges. Despite the large number of samples available, the small size and black-and-white format of the instances ensure relatively fast training phase.

MNIST-M

Firstly introduced in [11] in the context of Domain Adaptation, MNIST-M is a digits dataset originally presented to challenge models to deal with a domain shift where the source domain comprehends MNIST. According to the authors of the dataset, MNIST-M is obtained by blending digits from the original MNIST set over patches randomly extracted from color photos belonging to *BSDS500* [64]. Briefly, each output sample is made by extracting a 32×32 patch from a photo and inverting pixel colors at positions corresponding to the digit. While for humans the digit recognition becomes just slightly harder compared to the original MNIST samples (digits are still distinguishable), for a Convolutional Neural Networks this domain largely differs from MNIST, as the background and the strokes are no longer constant. Consequently, CNN trained from MNIST are not able to recognize

MNIST-M digits when introduced. However, MNIST-M preserve the location of the samples, thus the digits remains centered with respect of the center of the mass. Some examples are available in figure 4.2

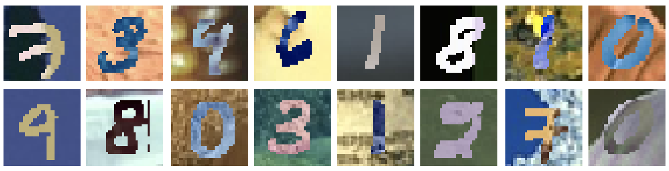

Figure 4.2: MNISTM Dataset

.

USPS

USPS is a digit dataset automatically scanned by the United States Postal Service from envelopes, and contains 9298 samples with 16×16 pixel, in gray-scale format. Instances are centered, normalized and show a highly variable range of orthography styles. For the experiments the standard sub-setting is training/test 7438/1859 samples and the images are upscaled to 28×28. Visualization could be similar to MNIST for an human beings, but the gap between probability distributions of MNIST and USPS is quite large, and the domain shift is not trivial for a Convolutional Network. Random samples are shown in figure 4.3

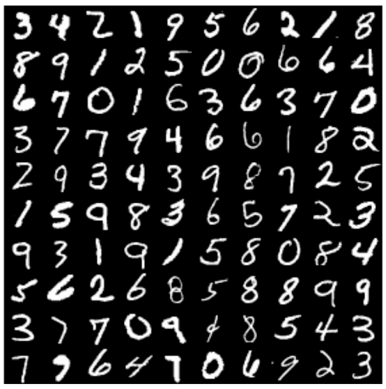

Figure 4.3: USPS Dataset

.

4.1.2 Image-to-Graph Mapping

The first step of our project consists in mapping an image into a graph, so passing from a full pixel matrix to a sparse graph defined by a set of vertices and their connections. This needs a pipeline with two main modules: (1) selecting informative points to identify the vertices and their representation, also indicated as *nodes* in the following; (2) choose the connections between nodes, or *edges*, as well as their strength.

Node Selection and Description

Given an image, we need to extrapolate well-distributed points that allow to summarize its semantic content while discarding non-relevant details which might capture a specific domain bias. Just as intuitive reference, consider this high-level example: from the image of an elephant we would like to get to a simple representation of its skeleton which also neglects the original background.

With this aim we explored existing segmentation algorithms to collect clusters of points. Formally, these algorithms are based on functions that allow to define a *mask*, meaning a vector with the same size of the original input, where at each pixel is assigned a label identifying one of the clusters. A general segmentation function $f : \mathbb{R}^{d\times d} \to \Omega$ where $\Omega \subset \mathbb{N}^{d\times d}$ returns a value contained in the discrete set according to the number of clusters, or *superpixels*, we want. Starting from the obtained segmentation results, we compute the center of mass of each cluster and assign to each center the mean of the colors within the superpixel. In case of gray-scale images, we just compute the average of the intensity values for each cluster; while when RGB inputs is used, we calculate the means of each channel separately. This standard approach provides several different configurations according to the settings of the segmentation algorithm. In particular, *SLIC* segmentation algorithm (in the version provided by *openCV* library) handle the segmentation with three main tunable parameters:

- *Number of Segments (N):* The chosen number of labels (clusters) in the segmented output image.
- *Compactness (C):* Balances color proximity and space proximity. Higher values give more weight to space proximity, making superpixel shapes more squared. This parameter depends strongly on image contrast and on the shapes of objects in the image.
- *Sigma (Σ):* Width of Gaussian smoothing kernel for pre-processing for each dimension of the image. The same Σ is applied to each dimension in case of a scalar value and $\Sigma = 0$ means no smoothing.

Since for digits dataset the common input dimension is 28×28, the maximum number of segments cannot overcome the 784 superpixels (trivial segmentation where each pixel belongs to a different superpixel). We will practically focus on cases with less than 50% of the total number of pixels to check if the overall image content can still be preserved: this means considering always $N < 400$. The parameters C and Σ define the shape of single superpixels generated over the image, perturbing the smoothness and regularity of the borders. There are no pre-defined rules to set these parameters and from an initial set of experiments we observed that extremely irregular superpixels may lead to graph structures that are difficult to understand, even for human eyes. For our analysis we chose $C = [0.1,1,5]$ and $\Sigma = [1,5]$: these ranges allow to cover structure variability and experiment with different final representations while keeping a reasonable visual structure (see figure 4.4 and 4.5).

Regarding the node description, we adopted two basic strategies: we either considered only the 2D, (x, y) position of the node in the original image frame, or we extend this vector with the intensity pixel values. In case of gray-scale images, this means just adding the gray-scale intensity (i_{gray}) element to the position descriptor (x, y, i_{gray}), while in case of colorful images the vector dimension rises to five with the three channels intensity values (x, y, r, g, b). In case of comparison between domains with different color-map, we replicate the i_{gray} intensity three times to obtain the same number of components with respect of *rgb*-mapped samples.

Edge Detection and Description

Starting from the un-ordered set of points obtained from the previous node selection, we need to connect each vertex with one or more of the others through a meaningful criterion that preserves the geometric structure of the original input. Moreover, not all the edges should be created equal: the node similarity will affect their connection with different *weights* as edge descriptors.

Several methods are possible for detecting edges. By adopting approaches from previous works [7], we test two different edge definition methods:

- *Radius*: we choose a value ρ as radius. For each vertex $v_i \in V$, where V is the set of points extracted, we connect all the vertices v_j that satisfy the condition:

$$distance(v_i, v_j) < \rho \;\forall\; v_i, v_j \in V$$

- *K-Nearest Neighbors*: we choose a value K as parameter of the edge definition method, and select the edges according to the standard KNN algorithm, associating each vertex with the (k) nearest vertices according to the distance measure selected.

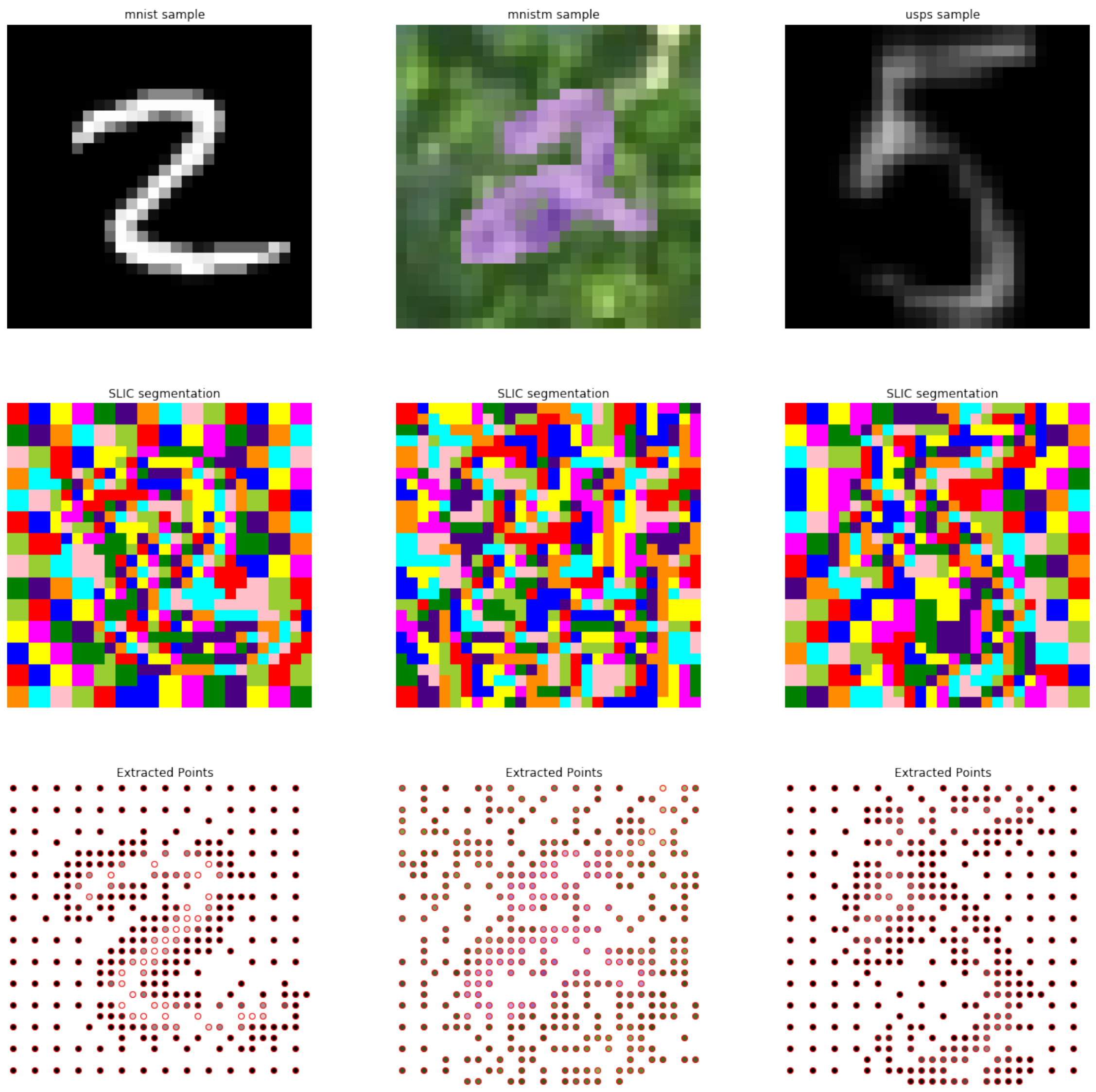

Figure 4.4: Example of points extraction on different datasets. In this case was used SLIC with $sigma = 1,\ compactness = 1,\ Segments = 200$.

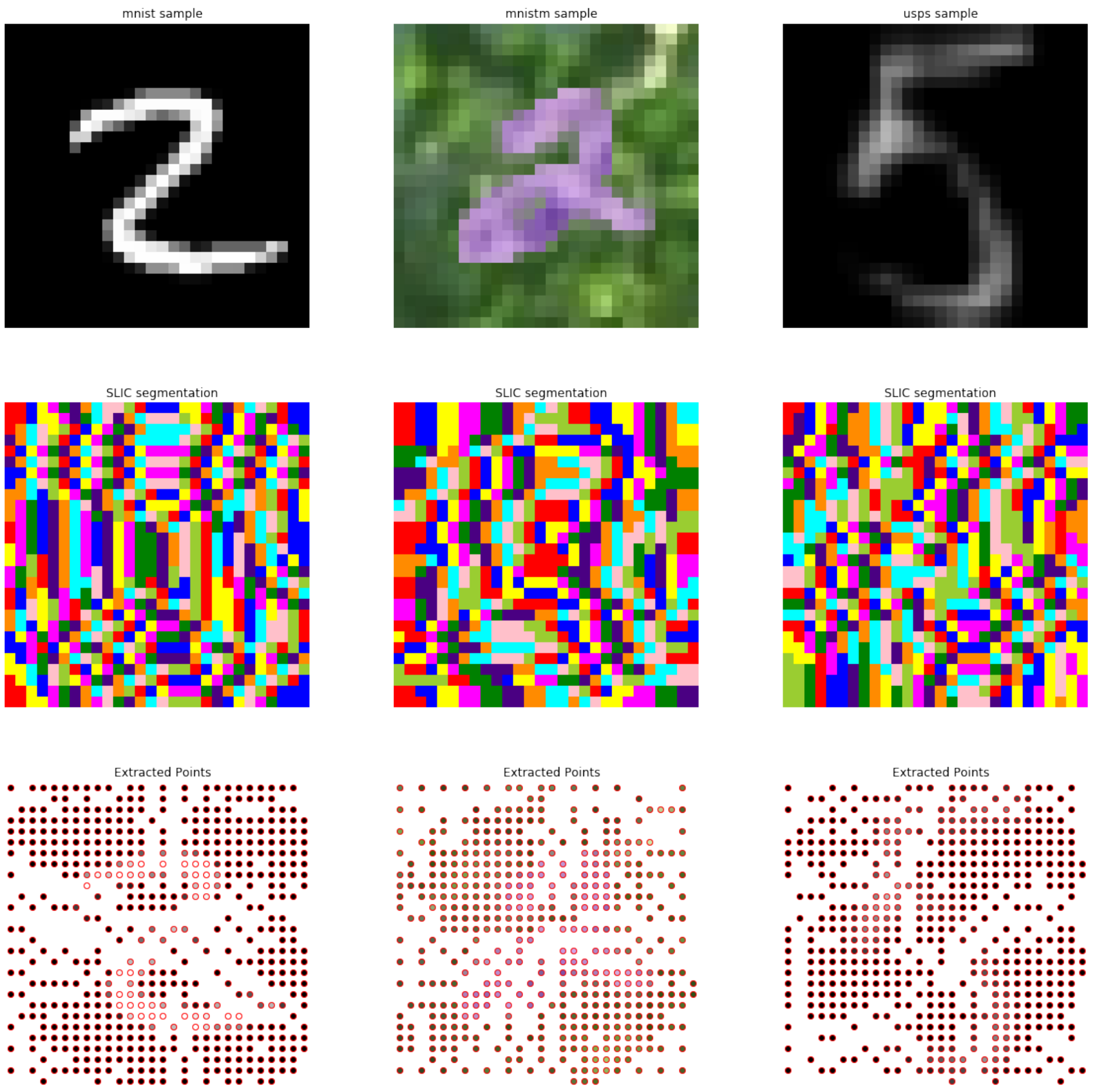

Figure 4.5: Example of points extraction on different datasets. In this case was used SLIC with $sigma = 5,\ compactness = 0.1,\ Segments = 200$.

In both cases, we use the Euclidean distance based on the position of each vertex. Since the input is defined on 2D, the distance is typically computed through the (x, y) components that describe the vertices, or *L2 norm* in vector notation. Moreover, when an edge is present, its weights corresponds to the distance between the nodes, estimated either on the basis of their simple 2D position or exploiting their full descriptors including grayscale or color intensity values. Experimentally we investigate the parameter range $\rho, K \in \{2,6\}$.

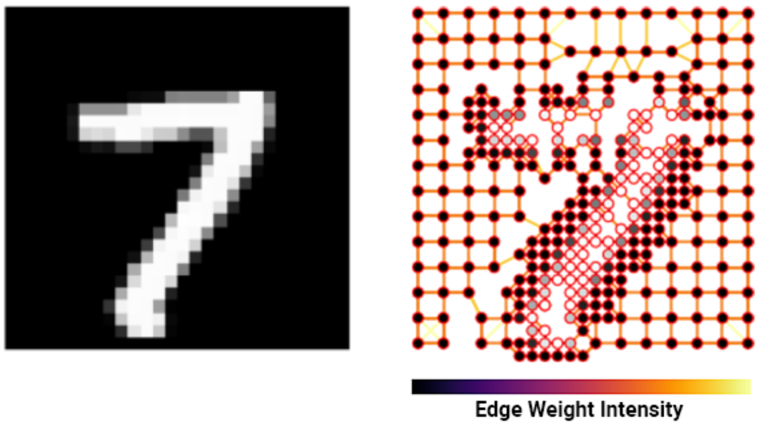

Figure 4.6: Example of full graph embedding. In this case was used SLIC with $sigma = 1$, $compactness = 1$, $Segments = 200$. Edges are detected with *k-nearest neighbors* method ($k = 3$).

4.1.3 Domain Adaptation and Generalization

Although summarizing images through their structural graphs can help keeping only their most relevant semantic content, per-se this choice may not decrease the risk of dealing with domain shift conditions. Indeed, even when the original image distribution of training and test is the same, if samples are pre-processed through different node and edge extraction procedures without a centralized control, we might need to recover coherence among them to get good generalization performance. In the more challenging case of source and target belonging to two different datasets, even admitting full control on the graph extraction, it might not be enough to impose the same structural parameters for all the samples. Overall, as long as the graph creation is considered as an external and separate pre-processing step with respect to the following discriminative learning procedure (i.e. digit classification), a dedicated and adaptive solution will still be needed to get a robust model. We investigate here two different adaptive strategies based on self-supervision and adversarial feature alignment.

Self-Supervised Adaptation: Solving Jigsaw Puzzles

As already discussed in section 2.4.5, optimizing jointly a self-supervised auxiliary task with the main classification objective may help supporting robustness across domains. In particular, [10] focused on solving jigsaw puzzles where the images are decomposed in patches which are then shuffled and re-used to produce disordered images. The network finally identifies the correct class from the original image while also recognizing the correct patch order of the puzzled image version out of a predefined subset of possible permutations. Translating this strategy into a graph-based network is not trivial. A first challenge is to re-elaborate the concept of patch location when dealing with graphs. Indeed position and distance in a graph are defined and measured differently with respect to plain images due to the sparse structure of vertices and edges. Specifically, the distance between a starting and an ending node depends on the number of nodes that has to be crossed between them, as well as on the weight of the connecting edges (typical definition of *walk*). Luckily, Pytorch-Geometric [58] allows to embed an explicit location for each vertex of the graph, expressed in terms of x and y of the original image. In this way, we are able to define the problem in a similar form respect to the computer vision proposed method. We start from the clustered image with pre-selected nodes and divide it into a regular 3×3 grid of patches. From their original positions, the patches are moved according to 30 different random patch permutations. We identify for each patch the original barycenter of the contained nodes and swap two patches by simply moving their barycenters. This operation is repeated as many time as needed to produce every puzzled version of the original graph and overall to get all the 30 different puzzled versions of each original sample. Finally, edge detection and description re-ran on each of the produced puzzles to get a complete permuted graph.

During training in the Domain Generalization setting, each batch of source training samples will contain 60% of original graphs and 40% of puzzled graphs. In the Domain Adaptation setting, the ratio remains the same, but the 40% puzzled graphs are selected from both source and target. Indeed, while the main classification task needs annotated samples, the auxiliary jigsaw puzzle task is trained to recognize the permutation index out of the self-defined 30 classes, thus can easily exploit the unlabeled target samples.

Adversarial Feature Alignment

The adversarial adaptive method DANN, already described in section 2.4.6, can be easily introduced in our graph-based network. Differently from the Jigsaw Puzzle case where several difficulties arise from the definition of the self-supervised task and its dedicated data processing, here the auxiliary objective trained together with the main digit recognition task consists simply in a binary classification on the domain label. Source and target samples are considered as belonging to two different categories that should be correctly annotated to minimize a binary cross-entropy loss. The internal gradient reversal layer inverts the learning trend so that the internal features are updated to make source and target look alike. It is clear that, due to the essential need of target samples, DANN can be used in the Domain Adaptation setting but not in the Domain Generalization case.

4.1.4 CNN Backbones

With the term *Backbone*, we mean the main architecture, in terms of layers and branches, of the Convolutional Neural Networks that has been employed in the project. We tested different convolutional kernels, different hierarchical structures, different construction and connections between layers. Sill, all the network variants follow a general design that can be resumed in two main blocks:

- **Feature Extractor**: represents the part of the network in which the input is processed by convolutional layers and, optionally, pooling layers, to extract meaningful features to support the classification prediction. In this part we often use a sequence of 2 or 3 layers, possibly interspersed with pooling operation. Convolutional layers are mainly based on the ECC kernel, described in section 3.2.6. Some particular skip connections has been tested (concatenations of features or residual connection).
- **Main Classifier**: the main classifier block is the one in charge of producing a final prediction in terms of digit label from the extracted features. It is based on mainly 2 fully-connected layers with a final *Softmax* layer, that outputs as many values as the number of possible labels, representing the probabilities that the input belongs to the related class. The fully-connected layers are separated by a *Dropout Layer* [13] to improve the generalization capability.

Here there is the main backbones used in our work:

BB.1 (ECCNet): $x \rightarrow ECC1(nfeat{,}32) \rightarrow MaxPool(2) \rightarrow ECC2(32{,}64) \rightarrow MaxPool(4) \rightarrow GlobalMeanPool \rightarrow FC(64{,}128) \rightarrow DropOut(.5) \rightarrow FC(128, nclasses)$.

BB.2 (ReinforcedECCNet): $x \rightarrow ECC1(nfeat{,}32) \rightarrow MaxPool(2) \rightarrow ECC2(32{,}64) \rightarrow MaxPool(4) \rightarrow ECC3(64{,}128) \rightarrow MaxPool(8) \rightarrow GlobalMeanPool \rightarrow FC1(128{,}256) \rightarrow DropOut(.5) \rightarrow FC2(256, nclasses)$.

BB.3 (Deep / plain): $x \rightarrow ECC1(nfeat, 64) \rightarrow ECC2(64{,}64) \rightarrow ECC3(64{,}64) \rightarrow GlobalMeanPool \rightarrow FC(64{,}128) \rightarrow DropOut(.5) \rightarrow FC(128, nclasses)$.

BB.4 (Deeper / nopool): $x \rightarrow ECC1(nfeat, 32) \rightarrow ECC2(32{,}64) \rightarrow ECC3(64{,}128) \rightarrow GlobalMeanPool \rightarrow FC(128{,}256) \rightarrow DropOut(.5) \rightarrow FC(256, nclasses)$.

BB.5 (Residual based): $x\oplus \rightarrow ECCConv1(nfeat, 64) \rightarrow ECCConv2(64{,}64)\oplus \rightarrow ECCConv3(64{,}64) \rightarrow GlobalMeanPool \rightarrow FC(64{,}128) \rightarrow DropOut(.5) \rightarrow FC(128, nclasses)$[1].

With *ECC*, we indicate the graph-based edge-conditioned convolutional layer (for a further description, see chapter 3.2.6). *FC* stands for *fully-connected layers*. Numbers in parenthesis indicate respectively the input and the output features. With $-Pool$ we refers to different pooling layer, where number in parenthesis represents the reduction factor with respect of the original input. *DropOut* refers to homonym layer, with dropout probability described between parenthesis. We will refer to them by using symbolic reference (i.e. **BB.1**).

The two tested adaptive methods (JiGraphNet, GraphDANN) are both multi-task variants of the considered backbones and extend the network with the introduction of a third block as illustrated in figure 4.7.

For **JiGraphNet**, implementation of *Jigen* method [10], we introduce a branch at the end of the feature extraction part of the network with two fully-connected layers of the same dimension (64 features in output), interlaced with a dropout layer. Finally a softmax layer computes prediction on the right permutation. We can formalize the side branch as:

$... \rightarrow FC(nfeat{,}64) \rightarrow DropOut(p = 0.5) \rightarrow FC(64, N_p) \rightarrow SoftMax$

For **GraphDANN**, implementation of *DANN* method [11], we introduce the same architecture of previous method, with a branch with two fully-connected layer with same dimension (64 features in output) and an ending softmax layer to compute binary prediction on the right domain. Here the main difference is in the introduction of a *Gradient Reversal Layer* that during the back-propagation inverts the loss and multiplies it for an optimal parameter depending on the iteration, epoch and learning parameters. The side branch can be represented as follows:

$... \rightarrow GRL \rightarrow FC(nfeat{,}64) \rightarrow DropOut(p = 0.5) \rightarrow FC(64{,}2) \rightarrow SoftMax$

[1]With $\oplus$, we refers to the common behaviour of adding the input to the output representation computed by the convolutional layer. It is a novelty introduced with Residual Networks [65]

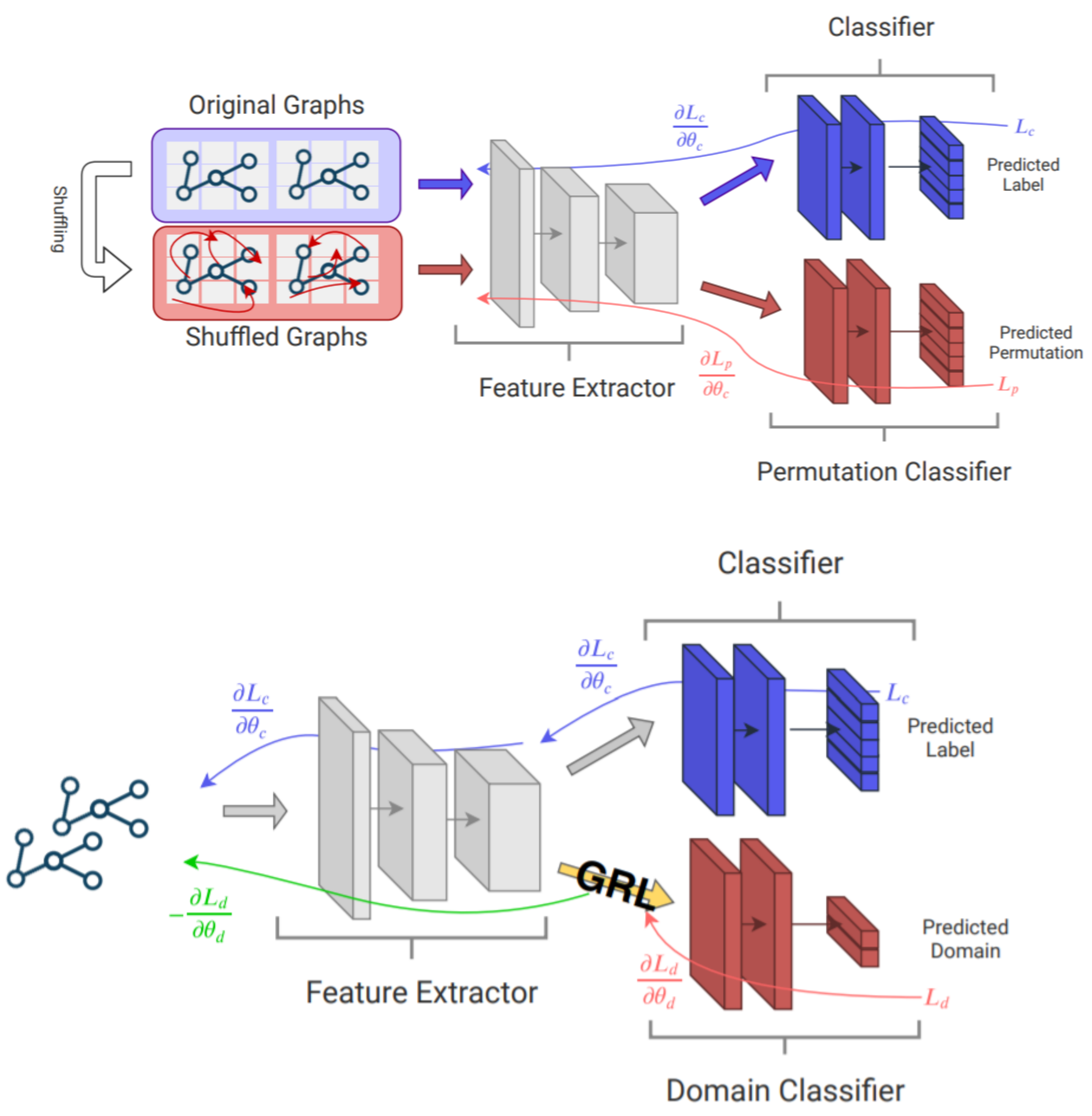
Classifier
Original Graphs
Shuffling
Shuffled Graphs
$\frac{\partial L_c}{\partial \theta_c}$
L_c
Predicted Label
Predicted Permutation
L_p
$\frac{\partial L_p}{\partial \theta_c}$
Feature Extractor
Permutation Classifier
Classifier
$\frac{\partial L_c}{\partial \theta_c}$
$\frac{\partial L_c}{\partial \theta_c}$
L_c
Predicted Label
GRL
$-\frac{\partial L_d}{\partial \theta_d}$
$\frac{\partial L_d}{\partial \theta_d}$
Predicted Domain
L_d
Feature Extractor
Domain Classifier

Figure 4.7: GraphDAN and JiGraphNet, architecture and losses involved

4.2 Single-Domain Classification

In this section, we focus on all the experiments done considering each single dataset separaterly: in other words **train and test samples are drawn from the same domain** as in the most standard supervised setting. Our aim is to evaluate the proposed graph creation procedure and verify the potentiality and possible limits of graph deep learning with respect to the most common CNN solutions when dealing with images.

The different choices included in this experiments can be summarized as follow:

- **Node Selection Algorithm**: *SLIC* segmentation was mainly used. We also ran initial tests with Felzenszwalb's algorithm [66] as baseline, but the results were not competitive. We also consider a further reference with nodes selected randomly on a regular grid.
- **Node Selection Setting**: we considered different parameters for SLIC (N, C, Σ).
- **Node Description**: meaning the features assigned to vertices. We tested scenarios with *rgb* features, or just *gray* intensity (with a coherent translation of rgb to black and white), or incorporating to the color channel also the spatial components x and y.
- **Edge Selection**: the connection between the nodes are mainly selected on the basis of their reciprocal Euclidean distance on the (x, y) location with a threshold ρ. The *KNN* algorithm was less effective but we show anyway the best result we were able to get as reference.
- **Edge Description**: meaning the features assigned to the edges. We tested situations where the edge weight is equal to the Euclidean distance between connected nodes, or using an L2 norm based on the components (r, g, b, x, y)
- **Network Backbone**: we considered different variants for the main network architecture as discussed above. We ran most of the experiments with **BB.1** to evaluate the graph representation. Starting from the best graph structures, we then extended the analysis also to the deeper network structures.

Experiments with BB.1 Network

We trained our basic network backbone with learning rate = 0.05; decay step each 20 epoch of $\times 0.1$; optimizer = SGD; batch size = 64; epochs = 60 and present the results for different dataset in separate tables.

Table 4.1 regards the MNIST dataset. Here most of the parameter variations do not heavily perturb the learning power of the network. Best results are reached with 200 segments, but the difference when using 300 or 100 is limited to 1 or 2 percentage points (pp) in classification accuracy. Small modifications of ρ for edge definition have a similar effect, but its influence seems to increase when this threshold value grows high ($\rho = 6$). The parameter that mostly influences the results is C, at least in combination with limited values of Σ and N. For the edge descriptors, including the pixel intensity is not a good choice as well as using *KNN* for edge selection.

SLIC (N,Σ,C)	Node Feat.	Edge Selection	Edge Feat.	Test Acc (%)
(300,5,5)	gray	$\rho = 3$	xy Distance	96.66
(200,5,5)	gray	$\rho = 3$	xy Distance	**98.38**
(100,5,5)	gray	$\rho = 3$	xy Distance	97.37
(200,1,1)	gray	$\rho = 3$	xy Distance	98.17
(100,1,1)	gray	$\rho = 3$	xy Distance	97.42
(200,1,0.1)	gray	$\rho = 3$	xy Distance	69.75
(100,1,0.1)	gray	$\rho = 3$	xy Distance	65.84
(200,5,1)	gray	$\rho = 3$	xy Distance	94.65
(200,5,0.1)	gray	$\rho = 3$	xy Distance	98.20
(200,5,5)	gray	$\rho = 2$	xy Distance	97.81
(200,5,5)	gray	$\rho = 4$	xy Distance	97.70
(200,5,5)	gray	$\rho = 6$	xy Distance	94.86
(200,5,5)	gray	$\rho = 3$	$rgbxy$ Distance	85.96
(200,5,5)	gray	KNN $k = 3$	xy Distance	73.32

Table 4.1: Results on MNIST dataset with different graph-creation settings and **BB.1** network.

Table 4.2 shows the results on the MNIST-M dataset. Here the task is more challenging because we deal with colorful and confusing images due to their complex background. The learning model is more sensitive to graph parameter variations and the drop in performance is particularly evident for low values of all the elements in the tuple (N, Σ, C). Moving ρ too far from the best value 3 can cause a significant performance loss. Moreover, both ignoring the rgb features for the nodes or extending them with the xy location show bad results.

Finally, table 4.3 focuses on the USPS dataset. In this case we are dealing again with a grayscale dataset, but the results are much more sensitive to variations in the chosen parameters, with respect to MNIST. To better investigate the graph effect on the digit recognition accuracy, we also extended the original parameter ranges (considering ρ = 9,11) and to overcome the SLIC convergence limits for a number of segments over 300 we considered the case of 400 and 600 random nodes as baseline references.

Regardless of the specific trend shown by each dataset, the overall conclusion that we can draw from this initial set of experiments, is that **with a proper graph construction it is possible to discard most of the image pixels while maintaining the image semantic content**.

SLIC (N,Σ,C)	Node Feat.	Edge Selection	Edge Feat.	Test Acc (%)
(300,5,5)	rgb	$\rho = 3$	xy Distance	**86.45**
(200,5,5)	rgb	$\rho = 3$	xy Distance	85.48
(100,5,5)	rgb	$\rho = 3$	xy Distance	61.80
(200,1,1)	rgb	$\rho = 3$	xy Distance	75.84
(100,1,1)	rgb	$\rho = 3$	xy Distance	22.23
(200,1,0.1)	rgb	$\rho = 3$	xy Distance	54.06
(100,1,0.1)	rgb	$\rho = 3$	xy Distance	23.34
(200,5,1)	rgb	$\rho = 3$	xy Distance	25.43
(200,5,0.1)	rgb	$\rho = 3$	xy Distance	82.76
(200,5,5)	rgb	$\rho = 2$	xy Distance	73.20
(200,5,5)	rgb	$\rho = 4$	xy Distance	82.61
(200,5,5)	rgb	$\rho = 6$	xy Distance	73.13
(200,5,5)	rgb	$\rho = 3$	$rgbxy$ Distance	83.09
(200,5,5)	rgb	KNN $k = 3$	xy Distance	76.30
(300,1,1)	gray	$\rho = 3$	xy Distance	30.65
(200,5,5)	gray	$\rho = 3$	xy Distance	53.07
(100,5,5)	gray	$\rho = 3$	xy Distance	28.62
(300,1,1)	rgbxy	$\rho = 3$	xy Distance	47.88
(200,5,5)	rgbxy	$\rho = 3$	xy Distance	11.57
(100,5,5)	rgbxy	$\rho = 3$	xy Distance	26.95

Table 4.2: Results on MNIST-M dataset with different graph-creation settings and **BB.1** network.

Experiments on different backbones

After having identified the best graph settings for each dataset, we focused on evaluating the proposed network variants defined by increasing the depth in different ways. Specifically we started with the **BB.2**. The results in table indicate that for MNIST-M and USPS the network backbone can make a significant difference with a performance gain up to 6 pp.

For MNIST the **BB.2** network does not present any advantage with respect to its simpler version, but further improvement can be observed by considering other graph-parameter combination and the network variants described in 4.1.4, as shown in table 4.5.

All experiments are executed in 60 epochs, with learning rate 0.05 and decay step

SLIC (N,Σ,C)	Node Feat.	Edge Selection	Edge Feat.	Test Acc (%)
(600,random)	gray	$\rho = 3$	xy Distance	41.20
(400,random)	gray	$\rho = 3$	xy Distance	78.00
(300,5,5)	gray	$\rho = 3$	xy Distance	86.44
(200,5,5)	gray	$\rho = 3$	xy Distance	77.62
(100,5,5)	gray	$\rho = 3$	xy Distance	80.53
(300,1,1)	gray	$\rho = 3$	xy Distance	83.86
(200,1,1)	gray	$\rho = 3$	xy Distance	77.68
(100,1,1)	gray	$\rho = 3$	xy Distance	79.83
(300,1,0.1)	gray	$\rho = 3$	xy Distance	74.45
(200,1,0.1)	gray	$\rho = 3$	xy Distance	71.01
(100,1,0.1)	gray	$\rho = 3$	xy Distance	67.13
(300,5,1)	gray	$\rho = 3$	xy Distance	37.51
(300,5,0.1)	gray	$\rho = 3$	xy Distance	38.22
(300,5,5)	gray	$\rho = 2$	xy Distance	63.85
(300,5,5)	gray	$\rho = 4$	xy Distance	82.30
(300,5,5)	gray	$\rho = 6$	xy Distance	86.34
(200,5,5)	gray	$\rho = 2$	xy Distance	53.85
(200,5,5)	gray	$\rho = 4$	xy Distance	74.66
(200,5,5)	gray	$\rho = 6$	xy Distance	81.01
(100,5,5)	gray	$\rho = 6$	xy Distance	80.90
(100,5,5)	gray	$\rho = 9$	xy Distance	86.07
(100,5,5)	gray	$\rho = 11$	xy Distance	83.65
(300,5,5)	gray	$\rho = 3$	$rgbxy$ Distance	**88.49**
(200,5,5)	gray	$\rho = 3$	$rgbxy$ Distance	78.75
(300,5,5)	gray	KNN $k = 3$	xy Distance	82.95
(200,5,5)	gray	KNN $k = 3$	xy Distance	76.33

Table 4.3: Results on USPS dataset with different graph-creation settings and **BB.1** network.

each 20 epochs, even if convergence is reached earlier (epoch 45) for simpler problem (i.e. 200 superpixels).

Domain	SLIC (N,Σ,C)	Node Feat.	Edge Selection	Edge Feat.	Test Acc (%)
MNIST	(200,5,5)	gray	ρ=3	xy	98.85
MNIST-M	(200,5,5)	rgb	ρ=3	xy	92.13
USPS	(300,5,5)	gray	ρ=3	xy	90.71

Table 4.4: Best scores executed with backbone **BB.2**

Backbone	Domain	SLIC (N,Σ,C)	Test Acc (%)
BB.4	MNIST	(200,5,5)	99.07
	MNIST	(100,5,5)	98.46
	MNIST	(100,1,0.1)	84.75
	MNIST	(100,1,1)	98.40
	MNIST	(200,1,1)	99.17
BB.3	MNIST	(200,5,5)	**99.07**
	MNIST	(100,5,5)	84.06
	MNIST	(100,1,0.1)	84.90
	MNIST	(100,1,1)	98.29
	MNIST	(200,1,1)	**99.17**
BB.5	MNIST	(200,5,5)	98.87
	MNIST	(100,5,5)	**98.50**
	MNIST	(100,1,0.1)	**85.04**
	MNIST	(100,1,1)	**98.53**
	MNIST	(200,1,1)	99.05

Table 4.5: Combined analysis on different graph parameters and our deeper graph-based networks on MNIST.

With these experiments, we actually achieve competitive results even in comparison with standard Convolutional Neural Networks and we overcomes results of the original graph network illustrated in [7]. We highlight that their result on MNIST (99.14) was obtained considering only white pixels which largely facilitated the digit recognition out of the black backround. This approach is not replicable for other real-world (rgb) images, while our graph extraction method is not restricted to bi-level color maps.

4.3 Domain shift due to Graph Construction

Due to the variability observed in the previous experiments we can state that the way in which we define the graph may have a relevant effect on the observed data

	Source - SLIC,ρ	Target - SLIC,ρ	*NoAdapt.*	Jig(DG)	Jig(DA)	GDANN
MNIST	(200,5,5),3	(200,5,5),3	**98.38**	98.08	-	-
	(200,5,5),3	(100,1,1),3	51.72	**62.86**	**61.61**	**77.25**
	(200,5,5),3	(200,5,5),6	**30.85**	16.60	29.36	16.04
	(200,5,5),3	(200,1,0.1),3	66.30	49.45	61.73	**67.07**
MNIST-M	(200,5,5),3	(200,5,5),3	**85.48**	76.33	-	-
	(200,5,5),3	(200,1,1),3	42.13	**78.58**	40.09	21.80
	(200,5,5),3	(100,5,5),3	23.13	**29.06**	17.85	**25.10**
	(200,5,5),3	(200,1,0.1),3	66.30	36.26	41.31	33.59
USPS	(300,5,5),3	(300,5,5),3	86.44	**92.08**	-	-
	(300,5,5),3	(100,5,5),3	**61.05**	20.46	39.02	57.34
	(300,5,5),3	(200,1,1),3	84.79	**88.52**	**93.71**	**85.24**
	(300,5,5),3	(100,1,0.1),3	38.65	10.36	12.23	**56.29**

Table 4.6: Domain generalization (DG) and adaptation (DA) with different graph extraction. SLIC parameters are listed as *(NumSegments, Sigma, Compactness).* Jig(DG) and Jig(DA) stand for *JiGraphNet* in its applications in the two considered settings. GDANN stands for *GraphDANN*. Results are compared per-row and appear in bold when improving over the NoAdapt. baseline.

	S - SLIC,ρ	T - SLIC,ρ	*NoAdapt.*	Jig(DG)	Jig(DA)	GDANN
MNIST	(200,5,5),3	(100,1,1),3	69.35	**76.15**	**75.38**	**86.13**
MNISTM	(200,5,5),3	(200,1,0.1),3	48.75	**52.93**	**62.13**	**71.74**
USPS	(300,5,5),3	(200,1,1),3	87.12	86.51	**90.55**	**92.08**

Table 4.7: Domain Adaptation and generalization with different graph extraction, repeating best score on **BB.2**. SLIC parameters are listed as *(NumSegments, Sigma, Compactness)*

distribution. Thus, although starting from a single dataset, the used graph setting may introduce a structural domain shift. We investigate this condition and how to overcome the shift with adaptive learning solutions. Specifically we sub-selected cases with significantly different graph configurations and tested both *JiGraphNet* and *GraphDANN*. The first refer to our graph-based implementation of [10] method, based on solving unsupervised jigsaws based on different domains. We indicate with *Jig DA* the Domain Adaptation experiments when both the source and the target are involved in the unsupervised method. Conversely, *Jig DG* stands for Domain Adaptation experiments, meaning when only the source is involved in the secondary task. For *GraphDANN* only the DA setting is feasible.

The results in table 4.6 confirm a significant performance drop when source and target samples are described with different graph parameters (column *NoAdapt.*). In five out of twelve cases Jig(DG), by just introducing the secondary auxiliary puzzle task, support the main classification performance producing a significant accuracy improvement over the non adaptive baseline. In the DA setting, *GraphDANN* seems more effective than Jig(DA). Our intuition is that the graph quantization needed to define the puzzle may interfere at different extent with source and target when there is a structural shift among them. Thus, simple feature adaptation with *GraphDANN* shows better results. Indeed this reasoning may also justify the Jig(DG) performance tha appear better than the corresponding DA case: in DG we do not observe the target, thus the method deals only with samples of a single graph structure, avoiding any quantization confusion during training.

We re-ran the experiments with top results also with the deeper **BB.2** network, showing the results in table 4.7. Here with DG we get performance almost equal or higher than the NoAdapt. baseline and we always observe an improvement over this reference in the DA settings.

4.4 Domain shift due to Visual Style

The second part of our study focuses on the more standard domain shift problem, due to a visual style variation across datasets. In this context, the gap between domains is larger than the case in the previous section and the effect of the adaptation approach is more evident. We fix several graph embeddings and change the visual domain between source and target considering different dataset pairs. Table 4.8 show the obtained results both without adaptation as well as in the DG and DA settings. As can be observed, almost all the performance improve over the naïve not-adaptive baseline. Moreover here Jig(DA) outperforms GDANN in 8 out of 12 cases, confirming that the proposed self-supervised solution tend to be better than its competitor when the source and target share the same graph structure. This behaviour is also confirmed when using the deeper version of the network, as shown by the results in 4.9.

Source	Target	SLIC	*NoAdapt.*	Jig(DG)	Jig(DA)	GDANN
MNIST	USPS	(100,5,5)	68.81	**71.27**	**69.40**	60.19
MNIST	USPS	(300,5,5)	24.7	**54.71**	**58.22**	**54.61**
MNIST	USPS	(200,5,5)	24.04	**53.09**	**55.69**	**45.55**
MNIST	MNISTM	(100,5,5)	12.63	10.65	9.78	**12.44**
MNIST	MNISTM	(200,1,1)	9.42	**9.79**	**13.76**	**9.97**
MNIST	MNISTM	(200,5,5)	10.76	**10.99**	**10.80**	**14.76**
MNISTM	USPS	(100,5,5)	17.63	16.63	**18.23**	**18.90**
MNISTM	USPS	(300,5,5)	18.69	**27.35**	**28.41**	**23.70**
MNISTM	USPS	(200,5,5)	20.62	**24.29**	**28.91**	**20.96**
USPS	MNIST	(100,5,5)	23.65	**24.02**	8.83	**39.62**
USPS	MNIST	(200,5,5)	11.72	**21.52**	**45.5**	**20.04**
USPS	MNIST	(300,5,5)	12.01	**32.04**	**30.12**	**19.96**

Table 4.8: Domain Adaptation and generalization with different graph extraction. SLIC parameters are listed as *(NumSegments, Sigma, Compactness)*.

Source	Target	SLIC	*NoAdapt.*	Jig (DG)	Jig (DA)	GDANN
MNIST	USPS	(100,5,5)	65.36	60.86	**70.18**	**70.09**
MNIST	MNISTM	(300,5,5)	11.48	10.33	**12.21**	**25.99**
MNISTM	USPS	(200,5,5)	22.71	**32.38**	**32.30**	19.09
USPS	MNIST	(300,5,5)	21.27	**31.48**	**46.38**	20.29

Table 4.9: Domain Adaptation and generalization with different graph extraction, repeating best score on **BB.2**. SLIC parameters are listed as *(NumSegments, Sigma, Compactness)*

Chapter 5

Conclusion and Future Works

In the present work, we experimented a new way of dealing with computer vision challenges. Our first goal was to validate the hypothesis of being able to dealing with visual learning by: 1. using just a subset of relevant points of the original samples; 2. building graph structures on those points to embed meaningful relations between points; 3. exploiting state-of-the-art graph-based Convolutional Neural Networks to analyze those graphs. We considered a simple computer vision task such as *digits recognition*, taking under consideration different domains with a different level of complexity. We mainly looked at 3 datasets, *MNIST, MNISTM* and *USPS*. It should be noted that, apart from MNIST, this was the first time these domains were used in the context of graph deep learning. We created a full framework able to extract relevant points from the input samples and to define edges that connect each other with different criteria and options. The construction of the graph was based on two main stages: extraction of points based on clusters of pixels, obtained in turn by segmentation method *SLIC*; and the definition of the edges, and the relative weight, among nodes.

As a baseline of our research, we found that *Dynamic Edge-Conditioned Filters* [7] convolutional kernel was the best solution to implement convolutions over image-based graphs. Starting from this convolutional layer and the original architecture made available, we researched different neural network structures in order to obtain the optimal solution. Furthermore, we examined different graph extraction methods to validate the most meaningful graph-based representation of the input. As a result of this first part of our project, we were able to obtain competitive results regarding standard convolutional networks, despite using a heavily reduced number of points from the original samples (~ 1 order of magnitude). This means that, even with a strongly reduced set of points, it is possible to approach state-of-the-art results for digits recognition.

The second part of our research focused on the hard challenge of the domain shift problem: this refers to cases where training and test data belong to two different distributions. To handle the gap between domains, we considered two main domain generalization methods. The first one was the adaptation to the graph world of a recent domain generalization method that has shown how combining supervised and self-supervised knowledge can improve generalization [10]. To adjust the method for graph geometry, we passed through group-based shuffling and graph reconstruction. The puzzle tasks consisted in learning and then predicting the right permutation index. Our second method lay on the existing technique [11] of domain adversarial learning. As in the original method, we used a combination of *Gradient Reversal Layer* and domain classifier to avoid the extraction of domain-dependent features.

Although our research mainly focused on these two methods, further methods were originally considered. Furthermore, we defined two different domain shift problems: the first is induced by using different segmentation methods on the same domain to define the source and target domain, while the second one depends on actual different domains used as source and target. Results of the second part of the research are compelling. Since the aim of an experimental setting is to validate the effectiveness of adaptation methods in the context of a graph, it should be stressed that our goal was not to obtain the best scores, but to obtain a sensible increment of learning performances when adaptation methods are involved in the network training. With the present research, we were able to demonstrate that domain adaptation methods work for graph-based learning models, in both single and multiple-domain tasks. Furthermore, we tested with successful outcomes a new adaptation method that had never been applied before in graph context.

Needless to say, this work is only a starting point to a more in-depth analysis and evaluation of the described intuition that connects images and graphs to generalization. One possibility for future developments envisages a smoother integration of all the components analyzed thus far, towards a fully learnable end-to-end model able to perform jointly segmentation and cross-domain visual learning. Another possibility regards the actuation of graph-based generalization methods for other tasks, such as node classification [6], [67] or graph classification with other datasets [68][69].

List of Tables

List of Figures

Bibliography

[1] 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.

[2] R. Geirhos, C. R. Temme, J. Rauber, H. H. Schütt, M. Bethge, and F. A. Wichmann, "Generalisation in humans and deep neural networks", in *Advances in Neural Information Processing Systems*, 2018, pp. 7538–7550.

[3] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, "Multimodal deep learning", in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 689–696.

[4] M. Gori, G. Monfardini, and F. Scarselli, "A new model for learning in graph domains", *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, 729–734 vol. 2, 2005.

[5] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, *Spectral Networks and Locally Connected Networks on Graphs*, 2013. arXiv: `1312.6203 [cs.LG]`.

[6] T. N. Kipf and M. Welling, *Semi-Supervised Classification with Graph Convolutional Networks*, 2016. arXiv: `1609.02907 [cs.LG]`.

[7] M. Simonovsky and N. Komodakis, "Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs", *CoRR*, vol. abs/1704.02901, 2017. arXiv: `1704.02901`. [Online]. Available: `http://arxiv.org/abs/1704.02901`.

[8] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic Graph CNN for Learning on Point Clouds", *ACM Transactions on Graphics*, vol. 38, no. 5, 1–12, 2019, ISSN: 0730-0301. DOI: `10.1145/3326362`. [Online]. Available: `http://dx.doi.org/10.1145/3326362`.

[9] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric Deep Learning on Graphs and Manifolds Using Mixture Model CNNs", *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. DOI: `10.1109/cvpr.2017.576`. [Online]. Available: `http://dx.doi.org/10.1109/CVPR.2017.576`.

[10] F. M. Carlucci, A. D'Innocente, S. Bucci, B. Caputo, and T. Tommasi, "Domain Generalization by Solving Jigsaw Puzzles", in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[11] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-Adversarial Training of Neural Networks", *Advances in Computer Vision and Pattern Recognition*, 189–209, 2017, ISSN: 2191-6594. DOI: `10.1007/978-3-319-58347-1_10`. [Online]. Available: `http://dx.doi.org/10.1007/978-3-319-58347-1_10`.

[12] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *nature*, vol. 521, no. 7553, p. 436, 2015.

[13] 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, pp. 1929–1958, 2014. [Online]. Available: `http://jmlr.org/papers/v15/srivastava14a.html`.

[14] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk, "Slic superpixels", Tech. Rep., 2010.

[15] R. Achanta, A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Susstrunk, "SLIC Superpixels Compared to State-of-the-Art Superpixel Methods", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2274–2282, Nov. 2012.

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

[17] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, "Deeper, broader and artier domain generalization", in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5542–5550.

[18] T. Tommasi, "Learning to learn by exploiting prior knowledge", EPFL, Tech. Rep., 2013.

[19] G. Csurka, *Domain Adaptation for Visual Applications: A Comprehensive Survey*, 2017. arXiv: `1702.05374 [cs.CV]`.

[20] M. Mancini, L. Porzi, S. R. Bulo, B. Caputo, and E. Ricci, "Boosting Domain Adaptation by Discovering Latent Domains", *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018. DOI: `10.1109/cvpr.2018.00397`. [Online]. Available: `http://dx.doi.org/10.1109/CVPR.2018.00397`.

[21] M.-M. Paumard, D. Picard, and H. Tabia, "Image Reassembly Combining Deep Learning and Shortest Path Problem", in *European Conference on Computer Vision (ECCV 2018)*, Munich, Germany, Sep. 2018. [Online]. Available: `https://hal.archives-ouvertes.fr/hal-01869765`.

[22] Z. Cao, L. Ma, M. Long, and J. Wang, "Partial adversarial domain adaptation", in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 135–150.

[23] G. Angeletti, B. Caputo, and T. Tommasi, "Adaptive Deep Learning Through Visual Domain Localization", in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7135–7142. DOI: `10.1109/ICRA.2018.8460650`.

[24] S. Gidaris, P. Singh, and N. Komodakis, "Unsupervised representation learning by predicting image rotations", *arXiv preprint arXiv:1803.07728*, 2018.

[25] A. D'Innocente and B. Caputo, "Domain Generalization with Domain-Specific Aggregation Modules", *Pattern Recognition*, 187–198, 2019, ISSN: 1611-3349. DOI: `10.1007/978-3-030-12939-2_14`. [Online]. Available: `http://dx.doi.org/10.1007/978-3-030-12939-2_14`.

[26] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, "Learning to generalize: Meta-learning for domain generalization", in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[27] R. Gong, W. Li, Y. Chen, and L. V. Gool, "DLOW: Domain flow for adaptation and generalization", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2477–2486.

[28] M. Ghifary, W Bastiaan Kleijn, M. Zhang, and D. Balduzzi, "Domain generalization for object recognition with multi-task autoencoders", in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2551–2559.

[29] S. Motiian, M. Piccirilli, D. A. Adjeroh, and G. Doretto, "Unified deep supervised domain adaptation and generalization", in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 5715–5725.

[30] S. J. Pan and Q. Yang, "A survey on transfer learning", *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[31] R. Raina, A. Battle, H. Lee, B. Packer, and A. Y. Ng, "Self-taught learning: transfer learning from unlabeled data", in *Proceedings of the 24th international conference on Machine learning*, ACM, 2007, pp. 759–766.

[32] L. Torrey and J. Shavlik, "Transfer learning", in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, IGI Global, 2010, pp. 242–264.

[33] R. Caruana, "Multitask Learning", *Machine Learning*, vol. 28, no. 1, pp. 41–75, 1997, ISSN: 1573-0565. DOI: `10.1023/A:1007379606734`. [Online]. Available: `https://doi.org/10.1023/A:1007379606734`.

[34] ——, "Multitask Learning: A Knowledge-Based Source of Inductive Bias", in *ICML*, 1993.

[35] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, *Geometric deep learning: going beyond Euclidean data*, 2016. arXiv: `1611.08097 [cs.CV]`.

[36] F. Cermelli, M. Mancini, E. Ricci, and B. Caputo, *The RGB-D Triathlon: Towards Agile Visual Toolboxes for Robots*, 2019. arXiv: `1904.00912 [cs.RO]`.

[37] I. S. Dhillon, Y. Guan, and B. Kulis, "Weighted Graph Cuts without Eigenvectors A Multilevel Approach", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, pp. 1944–1957, 2007. DOI: `10.1109/TPAMI.2007.1115`.

[38] D. I. Shuman, S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst, "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains", *IEEE signal processing magazine*, vol. 30, no. 3, pp. 83–98, 2013.

[39] R. R. Coifman and M. Maggioni, "Diffusion wavelets", *Applied and Computational Harmonic Analysis*, vol. 21, no. 1, pp. 53–94, 2006.

[40] M. Gavish, B. Nadler, and R. R. Coifman, "Multiscale Wavelets on Trees, Graphs and High Dimensional Data: Theory and Applications to Semi Supervised Learning.", in *ICML*, 2010, pp. 367–374.

[41] D. K. Hammond, P. Vandergheynst, and R. Gribonval, "Wavelets on graphs via spectral graph theory", *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150, 2011.

[42] R. Rustamov and L. J. Guibas, "Wavelets on graphs via deep learning", in *Advances in neural information processing systems*, 2013, pp. 998–1006.

[43] N. Sharon and Y. Shkolnisky, "A class of Laplacian multiwavelets bases for high-dimensional data", *Applied and Computational Harmonic Analysis*, vol. 38, no. 3, pp. 420–451, 2015.

[44] X. Zhang, X. Dong, and P. Frossard, "Learning of structured graph dictionaries", in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2012, pp. 3373–3376.

[45] X. Bresson, T. Laurent, and J. von Brecht, "Enhanced lasso recovery on graph", in *2015 23rd European Signal Processing Conference (EUSIPCO)*, IEEE, 2015, pp. 1501–1505.

[46] N. Shahid, V. Kalofolias, X. Bresson, M. Bronstein, and P. Vandergheynst, "Robust principal component analysis on graphs", in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2812–2820.

[47] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data", *arXiv preprint arXiv:1506.05163*, 2015.

[48] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, “Gradient-based learning applied to document recognition”, *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[49] M. Defferrard, X. Bresson, and P. Vandergheynst, “Convolutional neural networks on graphs with fast localized spectral filtering”, in *Advances in neural information processing systems*, 2016, pp. 3844–3852.

[50] J. Atwood and D. Towsley, “Diffusion-convolutional neural networks”, in *Advances in Neural Information Processing Systems*, 2016, pp. 1993–2001.

[51] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, “Convolutional networks on graphs for learning molecular fingerprints”, in *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[52] B. Perozzi, R. Al-Rfou, and S. Skiena, “Deepwalk: Online learning of social representations”, in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2014, pp. 701–710.

[53] J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei, “Line: Large-scale information network embedding”, in *Proceedings of the 24th international conference on world wide web*, International World Wide Web Conferences Steering Committee, 2015, pp. 1067–1077.

[54] S. Cao, W. Lu, and Q. Xu, “Grarep: Learning graph representations with global structural information”, in *Proceedings of the 24th ACM international on conference on information and knowledge management*, ACM, 2015, pp. 891–900.

[55] A. Grover and J. Leskovec, “node2vec: Scalable feature learning for networks”, in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 2016, pp. 855–864.

[56] Z. Yang, W. W. Cohen, and R. Salakhutdinov, “Revisiting semi-supervised learning with graph embeddings”, *arXiv preprint arXiv:1603.08861*, 2016.

[57] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality”, in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[58] M. Fey and J. E. Lenssen, “Fast Graph Representation Learning with PyTorch Geometric”, in *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

[59] P. Russo, F. M. Carlucci, T. Tommasi, and B. Caputo, “From source to target and back: symmetric bi-directional adaptive gan”, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 8099–8108.

[60] S. Cicek and S. Soatto, "Unsupervised Domain Adaptation via Regularized Conditional Alignment", *arXiv preprint arXiv:1905.10885*, 2019.

[61] F. M. Carlucci, A. D'Innocente, S. Bucci, B. Caputo, and T. Tommasi, "Domain Generalization by Solving Jigsaw Puzzles", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2229–2238.

[62] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation", *arXiv preprint arXiv:1409.7495*, 2014.

[63] Y. LeCun, C. Cortes, and C. Burges, "MNIST handwritten digit database", *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, vol. 2, p. 18, 2010.

[64] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour Detection and Hierarchical Image Segmentation", *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 898–916, May 2011, ISSN: 0162-8828. DOI: `10.1109/TPAMI.2010.161`. [Online]. Available: `http://dx.doi.org/10.1109/TPAMI.2010.161`.

[65] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition", in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[66] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation", *International journal of computer vision*, vol. 59, no. 2, pp. 167–181, 2004.

[67] B. Jiang, Z. Zhang, J. Tang, and B. Luo, *Graph Optimized Convolutional Networks*, 2019. arXiv: `1904.11883 [cs.CV]`.

[68] M. Cho, K. Alahari, and J. Ponce, "Learning graphs to match", in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 25–32.

[69] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning", 2011.

[70] C. Qin, H. You, L. Wang, C.-C. J. Kuo, and Y. Fu, "PointDAN: A Multi-Scale 3D Domain Adaption Network for Point Cloud Representation", in *Advances in Neural Information Processing Systems*, 2019, pp. 7190–7201.

[71] H. Li, S. Jialin Pan, S. Wang, and A. C. Kot, "Domain generalization with adversarial feature learning", in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5400–5409.

[72] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, and et al., "ImageNet Large Scale Visual Recognition Challenge", *International Journal of Computer Vision*, vol. 115, no. 3, 211–252, 2015, ISSN: 1573-1405. DOI: `10.1007/s11263-015-0816-y`. [Online]. Available: `http://dx.doi.org/10.1007/s11263-015-0816-y`.

[73] Y. Li, X. Tian, M. Gong, Y. Liu, T. Liu, K. Zhang, and D. Tao, "Deep domain generalization via conditional invariant adversarial networks", in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 624–639.

[74] N. Courty, R. Flamary, A. Habrard, and A. Rakotomamonjy, *Joint Distribution Optimal Transportation for Domain Adaptation*, 2017. arXiv: `1705.08848 [stat.ML]`.

[75] J. Shi and J. Malik, "Normalized cuts and image segmentation", *Departmental Papers (CIS)*, p. 107, 2000.

[76] Q. Wu and P. Hall, "Modelling Visual Objects Invariant to Depictive Style", in *BMVC*, 2013.

[77] Q. Wu, H. Cai, and P. Hall, "Learning Graphs to Model Visual Objects across Different Depictive Styles", in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 313–328, ISBN: 978-3-319-10584-0.

[78] J. Friedman, T. Hastie, and R. Tibshirani, *The elements of statistical learning*, 10. Springer series in statistics New York, 2001, vol. 1.

[79] P. K. Chan, M. D. F. Schlag, and J. Y. Zien, "Spectral K-way ratio-cut partitioning and clustering", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 9, pp. 1088–1096, 1994. DOI: `10.1109/43.310898`.