Alma Mater Studiorum - Università di Bologna

DOTTORATO DI RICERCA IN

MATEMATICA

Ciclo 33

Settore Concorsuale: 01/A2 - GEOMETRIA E ALGEBRA

Settore Scientifico Disciplinare: MAT/03 - GEOMETRIA

NON-TOPOLOGICAL PERSISTENCE FOR DATA ANALYSIS AND MACHINE LEARNING

Presentata da: Alessandro Mella

Coordinatore Dottorato

Fabrizio Caselli

Supervisore

Massimo Ferri

Co-supervisore

Mattia G. Bergomi

Esame finale anno 2021

Contents

List of Figures

List of Tables

List of Algorithms

Introduction 1

The increased computational power, and possibility of storing safely and rapidly vast amount of high-dimensional data lead to the production of huge amounts of data and therefore the development of techniques for their analysis became a central issue. During the last decades, researchers defined many approaches for data representation and classification, each tanking advantage of different features of the data. Among these techniques, topological data analysis and persistent homology represent flexible techniques, capable of adapting to different data structures.

Originally—and still now in the most concrete applications—persistence was structured considering two sides: data and representation, respectively. Data can be of various origins, provided that the properties one needs to investigate can be expressed in terms of *birth* and *death*. The representation side is actually twofold. Indeed, one needs to define a persistence module, and its *fingerprint*, i.e., either a persistence diagram or a barcode. A persistence module is a functor from an indexing category (typically either $\mathbb{N}$, a finite subset of $\mathbb{N}$, or $\mathbb{R}$, with their usual order) to $\mathbf{FinVec}_{\mathbb{K}}$. Much attention has been recently devoted to generalisations of persistent modules and their associated fingerprints, [1, 2]: accepting any preordered set as indexing category and an arbitrary category as target gives huge freedom to the representation. Conditions on the target category make definition and computation of persistence diagrams possible and effective (e.g., for stability issues). The usual limitation to homology with coefficients in a field might be dropped, and torsion represented in a generalised persistence diagram. The generalisation of the indexing category makes it also possible to merge, somehow, data and their representation as persistence modules: [1, Example 4.1] shows a case where data **is** the indexing category. Still, the great majority of applications follows this algorithmic flow: first, data are mapped into a filtered topological space (often a simplicial complex); then one computes persistent homology (on a field) on the filtered topological space; finally, persistence diagrams are used for analysis, classification, retrieval etc. The papers on which the present thesis relies, namely [3–5], focus on the generalisation of the first part of the classical flow mentioned above answering the question: “What can be represented by a persistence diagram?”. The axiomatization task pursued in those papers consists in finding features of Persistent Betti Number functions that allow for the definition of persistence diagrams, and make them the requirement in a very general definition. Can this generalisation be connected to the one of [1]? Possibly. Can it be linked to the generalisations of [1, 2] on the representation side of the classical process? Hopefully, and this is already the spirit of [3].

We now examine in detail classical persistent homology. It is a topological method that can be summarized in two steps: the construction of a sequence of nested topological spaces, called *filtration*, and the analysis of how homology groups change along this filtration. Given a space X, the filtration procedure consists in the extraction of sublevel-sets of a *filtering function* $f : X \to \mathbb{R}$. Some data structures are already endowed with a filtering function, think about the intensity of pixels in an image or the weights associated with the edges of a graph. In other cases, it is necessary to explicitly define such functions, as for the construction of filtered complexes on point cloud data, [6]. Homology groups are topological invariants that can change along with the defined filtration. To each homology class, it is possible to associate birth and death times, and its lifetime is called *persistence*. For example, in the case of 0-th degree homology classes are connected components. The birth and death times of a connected component respectively correspond to the time of its first appearance and to the time of its merging with an older component. Various visualization systems have been introduced for persistence homology, e.g. barcodes or persistence diagrams [6, 7]. The latter is the most used and consists in the subset of $\mathbb{R}^2$ composed by the diagonal and all the pairs (u, v) where u and v are the birth and death times of a homology class respectively.

As shown in [3, 8–10], the approach just described presents some limitations, principally caused by the categorical framework used. In the aforementioned procedure, the categorical framework can be summarized as follows: a source category **Top**, a target category $\mathbf{FinVec}_{\Bbbk}$ and a homology functor $H_k : \mathbf{Top} \to \mathbf{FinVec}_{\Bbbk}$. In [3] the authors provide a generalization of these concepts, as reported in table 1.1. It is possible to notice that this generalized persistence framework does not require auxiliary constructions as topological spaces. This allows one to use directly the category to which the data belongs as source category, without any transformation. For example, in [11], the authors presented many different ways to associate to a weighted graph a sequence of simplicial complexes. In this generalized scenario, this step is not necessary, and the persistence approach can be applied directly on the category of weighted graphs **WGraph**. Moreover, it is possible to use functors different

Classical framework	**Categorical framework**
Topological spaces	Source category **C**
Vector spaces	Regular target category **R**
Dimension	Rank function on **R**
Homology functor	Arbitrary functor from **C** to **R**

Table 1.1: From the classical to the categorical framework, [3].

from homology, and to have different target categories. In [4] autors show how it is possible to define a categorical persistence function on the category of weakly directed posets, having the category **Set** as target category, and use this to induce a categorical persistence function on other categories. In [4] the authors reports some examples on the category of graphs, such as the analysis of maximal clique communities or of maximal blocks.

The objective of this thesis is to understand how data analysis can benefit from the generalised persistence approach defined in [3]. In chapter 2, we recall the basic notions of category theory and the main theoretical results behind the generalised persistence framework defined in [3].

In chapter 3 we study the category of graphs. After briefly recalling some notation and concepts about graphs, we present how the category of graphs, **Grphs**, can be used as a target category. Recalling that this category is a regular category, we are left with the choice of a rank function. We provide some examples of possible rank functions, although many of them fail in at least one of the conditions required to be rank functions. Moreover, we analyse how the framework defined in [4] applies to connectivity-related notions in the case of directed graphs, also providing some examples highlighting the differences between such notions. In the latter part of the chapter, we present a study about how some features change as we assign different orientations to the same graph.

In chapter 4 we extend the notions introduced in [5] on graphs to the framework of sets. We use the introduced notions to define a novel image operator that enhance the signal intensity of the pixels near to a border. We provide some examples of its effect on a test image, also showing its stability to salt and pepper noise perturbations. Moreover, we use such an operator to define a new pooling layer, which provides an efficient downsampling procedure. We present some examples showing the performances of this new layer in terms of accuracy against some state-of-the-art pooling layers on the image classification task.

Background 2

In this chapter we recall some notions that will be useful in the remainder. Many of the definitions and results are taken from [3, 4]. For further references concerning category theory, see [12, 13].

2.1 Categories

Categories are very general and abstract structures, introduced in [14] with the aim of building a language allowing to provide insight into similar structures through different areas of mathematics by formulating and investigating these structures simultaneously with a high degree of generality and, through functors, to move problems from one area of mathematics to another where solutions may be more straightforward.

Category

Definition 2.1.1 *A* category $\mathbf{C}$ *consists of:*

a collection of objects $Obj(\mathbf{C})$*;*
- *for each pair* X, Y *of objects, a set* $Hom(X, Y)$ *of morphisms from* X *to* Y*. Let us denote a morphism* $f \in Hom(X, Y)$ *as* $f : X \to Y$*;*
- *for each triple of objects* $X, Y, Z \in Obj(\mathbf{C})$ *a binary operation* $\circ : Hom_{\mathbf{C}}(X, Y) \times Hom_{\mathbf{C}}(Y, Z) \to Hom_{\mathbf{C}}(X, Z)$*;*
- *for each object* X *an element* $\mathbf{1}_X \in Hom(X, X)$ *called the identity morphism;*
- *such that the following properties are satisfied:*
  - *composition is associative: for each quadruple* W, X, Y, Z *of objects, if* $f \in Hom(Y, Z)$*,* $g \in Hom(X, Y)$ *and* $h \in Hom(W, X)$*, then* $(f \circ g) \circ$

$h = f \circ (g \circ h)$;

- *composition satisfies the left and right unit laws: for each pair X, Y of objects, if $f \in Hom(X,Y)$, then $f \circ \mathbf{1}_X = f = \mathbf{1}_Y \circ f$.*

Functor

Definition 2.1.2 *Consider two categories $\mathbf{B}, \mathbf{C}$. Then a functor $F : \mathbf{B} \longrightarrow \mathbf{C}$ is a mapping such that:*

F associates to each $X \in \mathrm{Obj}(\mathbf{B})$ an object $F(X) \in \mathrm{Obj}(\mathbf{C})$

- *F associates to each morphism $f : X \longrightarrow Y$ in $\mathbf{B}$ a morphism $F(f) : F(X) \longrightarrow F(Y)$ in $\mathbf{C}$ in such a way that:*
  - *F preserves identities, i.e. $F(id_X) = id_{F(X)}$ for all $X \in \mathrm{Obj}\mathbf{C}$*
  - *F preserves compositions, i.e. $F(g \circ f) = F(g) \circ F(f)$ for all morphisms $f : X \longrightarrow Y, g : Y \longrightarrow Z$ in $\mathbf{B}$.*

Given $X, Y \in \mathrm{Obj}(\mathbf{B})$ we can define the mapping $F_{X,Y} : \mathrm{Morph}_{\mathbf{B}}(X,Y) \longrightarrow \mathrm{Morph}_{\mathbf{C}}(F(X), F(Y))$. We will say that the functor F is faithful if all the $F_{X,Y}$ are injective, full if they are surjective and fully faithful if they are both injective and surjective.

Category theory deals mostly with the properties of morphisms instead of studying objects, as it possible to infer from definitions 2.1.1, 2.1.2. This behaviour reflects in the following definitions, where, differently from other mathematical theories, properties are stated by looking at the relationship between morphisms.

Epimorphism

Definition 2.1.3 *Consider $X, Y \in \mathrm{Obj}(\mathbf{C})$. $f : X \longrightarrow Y$ is an epimorphism if given the following diagram*

$$\begin{tikzcd} X \arrow[r, "f"] & Y \arrow[r, shift left, "g"] \arrow[r, shift right, "h"'] & Z \end{tikzcd}$$

if $g \circ f = h \circ f$ then $g = h$.

Example 2.1.1 (Epimorphism) Let **Set** be the category whose objects are sets and whose morphisms are functions between sets. Let $X, Y \in \mathrm{Obj}(\mathbf{Set})$ and $f : X \longrightarrow Y$, then f is an epimorphism if and only if it is surjective.

Monomorphism

Definition 2.1.4 *Consider $X, Y \in \mathrm{Obj}(\mathbf{C})$. $f : X \longrightarrow Y$ is a monomorphism if* *given the following diagram*

$$\begin{tikzcd} Z \arrow[r, shift left, "g"] \arrow[r, shift right, "h"'] & X \arrow[r, "f"] & Y \end{tikzcd}$$

if $f \circ g = f \circ h$ *then* $g = h$.

Example 2.1.2 (Monomorphism) Let $X, Y \in \mathrm{Obj}(\mathbf{Set})$ and $f : X \longrightarrow Y$, then f is a monomorphism if and only if it is injective.

Isomorphism

Definition 2.1.5 *Consider* $X, Y \in \mathrm{Obj}(\mathbf{C})$. $f : X \longrightarrow Y$ *is an isomorphism if given the following diagram*

$$\begin{tikzcd} X \arrow[r, shift left, "f"] & Y \arrow[l, shift left, "g"] \end{tikzcd}$$

if $g \circ f = 1_X$ *then* $f \circ g = 1_Y$.

Example 2.1.3 (Isomorphism) Let X, Y be sets and $f : X \longrightarrow Y$, then f is an isomorphism if and only if it is an isomorphism in the classical sense (injective and surjective).

In many applications that follow, it will be essential to have a notion of subobjects of a particular object, e.g. subsets, subgroups, subspaces. Since we do not want to deal directly with what is inside an object but work only with morphisms, we will define subobjects as equivalence classes of monomorphisms, [12].

Subobject

Definition 2.1.6 *Consider a category* $\mathbf{C}$ *and two monomorphisms* $f : Y \longrightarrow X$, $g : Z \longrightarrow X$. *We will say that* $f \leq g$ *if* f *factors through* g, *i.e.* $f = g \circ f'$ *for some* $f' : Y \longrightarrow Z$. *If both* $f \leq g$ *and* $g \leq f$ *we will write* $f \equiv g$ *and this defines an equivalence relation among monomorphisms with common codomain* X. *The equivalence classes of* $\equiv$ *are the subobjects of* X.

Limits and colimits

In the remaining part of the chapter uniqueness will be always intended up to isomorphisms.

Definition 2.1.7 *Let us consider a category* **C** *and a category* **I** *used for indexing and called index category. We can define a diagram as a functor* $F : \mathbf{I} \longrightarrow \mathbf{C}$.

Cone, cocone

Definition 2.1.8 *Given a diagram* $F : \mathbf{I} \longrightarrow \mathbf{C}$*, a cone of* F *is a pair* (C, ψ)*, where* $C \in Obj(\mathbf{C})$ *and* ψ *is a family of morphisms with* $\psi_X : C \longrightarrow F(X)$ *for* $X \in \mathbf{I}$*, such that for every morphism* $f : X \longrightarrow Y$ *in* $\mathbf{I}$*, we have* $F(f) \circ \psi_X = \psi_Y$*. An example of cone is the notion of limit, see definition 2.1.9.*

A cocone of F *will dually be a pair object family of morphisms* (C, ψ) *with* $\psi_X : F(X) \longrightarrow C$ *for* $X \in \mathbf{I}$*, such that for every morphism* $f : X \longrightarrow Y$ *in* $\mathbf{I}$*, we have* $\psi_Y \circ F(f) = \psi_X$*. An example of cocone is the notion of colimit, see definition 2.1.10.*

Limit

Definition 2.1.9 *Consider a diagram* $F : \mathbf{I} \longrightarrow \mathbf{C}$ *over* $\mathbf{C}$*. A limit of* F *is a cone* (L, ϕ) *of* F *such that for every other cone* (C, ψ) *there exists a unique morphism* $u : C \overset{!}{\longrightarrow} L$ *such that* $\phi_X \circ u = \psi_X$ *for all* X *in* $\mathbf{I}$.

$$\begin{tikzcd}
& C \arrow[d, dashed, "u"] \arrow[ldd, bend right, "\psi_X"'] \arrow[rdd, bend left, "\psi_Y"] & \\
& L \arrow[ld, "\phi_X"'] \arrow[rd, "\phi_Y"] & \\
F(X) \arrow[rr, "F(f)"] & & F(Y)
\end{tikzcd}$$

Colimit

Definition 2.1.10 *Consider a diagram* $F : \mathbf{I} \longrightarrow \mathbf{C}$ *over* $\mathbf{C}$*. A colimit of* F *is a cocone* (L, ϕ) *of* F *such that for every other cone* (C, ψ) *there exists a unique morphism* $u : L \overset{!}{\longrightarrow} C$ *such that* $u \circ \phi_X = \psi_X$ *for all* X *in* $\mathbf{I}$.

$$\begin{tikzcd}
F(X) \arrow[rr, "F(f)"] \arrow[rd, "\phi_X"] \arrow[rdd, bend right, "\psi_X"'] & & F(Y) \arrow[ld, "\phi_Y"] \arrow[ldd, bend left, "\psi_Y"] \\
& L \arrow[d, dashed, "u"] & \\
& C &
\end{tikzcd}$$

All the following definitions are obtained by considering limits and colimits of some appropriate diagrams.

Definition 2.1.11 *An object* pt *in a category* **C** *is called terminal if there exists a unique morphism* $x \xrightarrow{!} \mathrm{pt}$ *for any object* $x \in \mathbf{C}$. *If it exists, the terminal object is unique, up to unique isomorphism.*

Terminal object

Example 2.1.4 (Terminal object) Let the category **Top** be the category whose objects are topological spaces and whose morphisms are continuos functions. Then, any point space pt is a terminal object in **Top**.

Definition 2.1.12 *An object* $\emptyset$ *in a category* **C** *is initial if for any object* $x \in \mathbf{C}$ *there exists a unique morphism* $\emptyset \xrightarrow{!} x$.

Initial object

Example 2.1.5 (Initial object) The empty set is an initial object in **Set**.

Definition 2.1.13 *An object which is both initial and terminal is said zero object. A category* **C** *equipped with a zero object is said pointed.*

Zero object, pointed category

Example 2.1.6 (Zero object) Let **Grp** be the category whose objects are groups and whose morphisms are groups homomorphisms. Then any trivial group 1 is the zero object in **Grp**. Indeed $1 \hookrightarrow G \twoheadrightarrow G/G = 1$, for every $G \in \mathbf{Grp}$. In the category of $\mathbf{Vec}_{\mathbb{K}}$, whose objects are vector spaces on the field $\mathbb{K}$ and whose morphisms are $\mathbb{K}$-linear maps between vector spaces, the zero object is the 0-dimensional vector space.

Definition 2.1.14 *Let* X, Y *be objects of a category* **C**. *The (binary) product of* X *and* Y *is a triplet* (P, π_X, π_Y) *composed by an object* P *and two morphisms* $\pi_X : P \longrightarrow X$ *and* $\pi_Y : P \longrightarrow Y$, *such that given* (P', π'_X, π'_y), *we have a unique morphism* $P' \longrightarrow P$ *that makes the following diagram commute:*

Product

$$\begin{tikzcd}
 & P' \arrow[dl, "\pi'_Y"'] \arrow[d, dotted, "u"] \arrow[dr, "\pi'_X"] & \\
X & P \arrow[l, "\pi_X"] \arrow[r, "\pi_Y"'] & Y
\end{tikzcd}$$

We denote the product $X \times Y$.

Definition 2.1.15 *Let* X, Y *be objects of a category* **C**. *The coproduct of* X *and* Y

Coproduct

is a triplet (C, ι_X, ι_Y) composed by an object C and two morphisms $\iota_X : X \longrightarrow C$ and $\iota_Y : Y \longrightarrow C$, such that given (C', ι'_X, ι'_Y), we have a unique morphism $C \longrightarrow C'$ that makes the following diagram commute:

$$\begin{tikzcd}
 & C' & \\
X \arrow[ur, "\iota'_Y"] \arrow[r, "\iota_X"'] & C \arrow[u, dashed, "u"] & Y \arrow[ul, "\iota'_X"'] \arrow[l, "\iota_Y"]
\end{tikzcd}$$

We denote the coproduct $X \amalg Y$.

Example 2.1.7 (Product and coproduct) Let $X, Y \in \mathrm{Obj}(\mathbf{Set})$. The product $X \times Y$ is simply the cartesian product. The coproduct $X \amalg Y$ is the disjoint union of X and Y.

Equalizer

Definition 2.1.16 *Let X, Y be objects of $\mathbf{C}$ and consider two morphisms $X \xrightarrow{f} Y$, $X \xrightarrow{g} Y$. A pair object morphism (Q, q) such that $Q \xrightarrow{q} X$ is an equalizer if $f \circ q = g \circ q$. Moreover, the pair (Q, q) must be universal, i.e. given another equalizer (Q', q'), there exists a unique morphism $Q' \xrightarrow{u} Q$ such that the following diagrams commutes.*

$$\begin{tikzcd}
Q \arrow[r, "q"] & X \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & Y \\
Q' \arrow[u, dashed, "u"] \arrow[ur, "q'"'] & &
\end{tikzcd}$$

Thus, equalizers are unique up to isomorphisms. Moreover, every equalizer is a monomorphism.

Example 2.1.8 (Equalizer) Let $A, B \in \mathrm{Obj}(\mathbf{Set})$ and $f, g : A \longrightarrow B$, then the equalizer is

$$\{a \in A \mid f(a) = g(a)\}$$

Coequalizer

Definition 2.1.17 *Let X, Y be objects of $\mathbf{C}$ and consider two morphisms $X \xrightarrow{f} Y$, $X \xrightarrow{g} Y$. A pair object morphism (Q, q) such that $Y \xrightarrow{q} Q$ is a coequalizer if $q \circ f = q \circ g$. Moreover, the pair (Q, q) must be universal, i.e. given another coequalizer (Q', q'), there exists a unique morphism $Q \xrightarrow{u} Q'$ such that the following* *diagrams commutes.*

$$\begin{tikzcd}
X \arrow[r, shift left, "f"] \arrow[r, shift right, "g"'] & Y \arrow[r, "q"] \arrow[dr, "q'"'] & Q \arrow[d, dotted, "u"] \\
 & & Q'
\end{tikzcd}$$

Thus, coequalizers are unique up to isomorphisms. Moreover, every coequalizer is an epimorphism.

Example 2.1.9 (Coequalizer) Let $A, B \in \mathrm{Obj}(\mathbf{Set})$ and $f, g : A \longrightarrow B$, then the coequalizer is the quotient of B with $\sim$, such that $f(x) \sim g(x)$ for every $x \in A$.

Definition 2.1.18 *A category* $\mathbf{C}$ *is finitely complete if it has equalizers, a terminal object and binary products. Analogously, a category* $\mathbf{C}$ *is finitely cocomplete if it has coequalizers, an initial object and binary coproducts.*

Finitely (co)complete category

Definition 2.1.19 *Let* X, Y *and* Z *be objects of a category* $\mathbf{C}$*, and* $X \xrightarrow{f} Z, Y \xrightarrow{g} Z$ *morphisms. A triplet* (P, p_1, p_2)*, where* P *is an object and* $P \xrightarrow{p_1} X$*,* $P \xrightarrow{p_2} Y$ *are two morphisms, is a pullback if the following diagram*

$$\begin{tikzcd}
P \arrow[r, "p_1"] \arrow[d, "p_2"'] & X \arrow[d, "f"] \\
Y \arrow[r, "g"] & Z
\end{tikzcd}$$

commutes and, given another triplet (P', p_1', p_2') *we have a unique morphism* $P' \xrightarrow{u} P$ *that makes the following diagram commute:*

$$\begin{tikzcd}
P' \arrow[dr, dotted, "u"] \arrow[drr, bend left, "p_1'"] \arrow[ddr, bend right, "p_2'"'] & & \\
 & P \arrow[r, "p_1"] \arrow[d, "p_2"'] & X \arrow[d, "f"] \\
 & Y \arrow[r, "g"] & Z
\end{tikzcd}$$

That is to say, the pullback is universal with respect to the diagram, and thus unique up to isomorphism. We denote it $X \times_Z Y$.

Pullback

Example 2.1.10 (Pullback) Given three sets X, Y and Z and functions $X \xrightarrow{f} Z$, $Y \xrightarrow{g} Z$, the coproduct $X \times_Z Y$ is the subset of the cartesian product:

$$X \times_Z Y = \{(x, y) \mid x \in X,\ y \in Y \text{ and } f(x) = g(y)\}$$

Fiber

Example 2.1.11 (Fiber) In the category of sets, let pt be the terminal object. Let $f : X \to Y$ be a map between sets and $y \in Y$. The fiber over y is $f^{-1}(y) \subset X$ realised by the pullback

$$\begin{tikzcd} f^{-1}(y) \arrow[r] \arrow[d] & X \arrow[d, "f"] \\ \mathrm{pt} \arrow[r] & Y \end{tikzcd}$$

Regular, Abelian and semisimple categories

In our investigation we will need to add some assumptions to the general definition of category, in order to guarantee that objects can be factorized, and in some cases, the existence of finite number of irreducible objects.

Regular epimorphism

Definition 2.1.20 *An epimorphism that is the coequalizer of a parallel pair of morphism.*

Regular category

Definition 2.1.21 *A category* **R** *is regular if the following conditions hold:*

R *is finitely complete.*

2. *Given $X \xrightarrow{f} Y$ a morphism and its pullback (P, p_1, p_2), then the coequalizer of p_1 and p_2 exists.*
3. *Given the pullback*

$$\begin{tikzcd} R \arrow[r] \arrow[d, "g"'] & X \arrow[d, "f"] \\ Z \arrow[r] & Y \end{tikzcd}$$

if f is a regular epimorphism, so is g.

Example 2.1.12 (Regular categories) The category **Set** with usual functions between sets as morphisms and the category of groups **Grp** with group homomorphisms are regular categories.

We will now introduce some preliminary concepts to the definition of Abelian category.

Kernels and cokernels Let **C** be a category and $\xi : X \longrightarrow Y$ a morphism. If for every object Z and morphisms $g, h : Z \longrightarrow X$, we have $\xi g = \xi h$, then ξ is said a (left) zero morphism. If **C** is pointed, i.e. it has a zero object 0, then given two objects X, Y there exists a unique zero morphism $\xi : X \longrightarrow Y$ given by the composition $X \longrightarrow 0 \longrightarrow Y$.

Definition 2.1.22 *Let* **C** *be a category with zero morphism* ξ *and* $f : X \longrightarrow Y$ *a morphism. The kernel of* f *is defined as the equalizer of* ξ *and* f*.*

Kernel

Definition 2.1.23 *Let* **C** *be a category with zero morphism* ξ *and* $f : X \longrightarrow Y$ *a morphism. The cokernel of* f *is defined as the coequalizer of* ξ *and* f*.*

Cokernel

Definition 2.1.24 *A category* **C** *is abelian if*

Abelian category

   it is pointed, i.e. **C** *has a zero object;*

2. *has binary products and binary coproducts;*
3. *every morphism has kernel and cokernel;*
4. *each monomorphism is a kernel and each epimorphism is a cokernel.*

In an Abelian category, the binary product and binary coproduct coincide and are sometimes called biproduct. We will sometimes simply call it sum, in analogy with the sum of vector spaces.

Remark 2.1.1 Every Abelian category is a regular category.

Definition 2.1.25 . *Let* **C** *be an Abelian category. An object* $X \in \mathrm{Obj}(\mathbf{C})$ *is simple if its only subobjects are* 0 *and* X*.*

Simple object

Lemma 2.1.1 (Schur Lemma) *Given* S, S' *simple objects in an Abelian category, morphisms from* S *to* S' *are either zero or invertible.*

Definition 2.1.26 *An Abelian category is semisimple if all its objects are semisimple, i.e. each object can be written as a finite sum of simple objects.*

Semisimple category

2.2 Rank-based persistence

Standard persistence theory is based on some fundamental steps. The first is the construction, starting from the data, of a filtration of a topological space. To this filtration, it is then associated a sequence of vector spaces through a homology functor. The dimension of the images of the mappings between these vector spaces describes how homology classes vary along with the filtration, and this information is usually encoded in persistence diagrams. This theory requires the use of topological spaces as source category, vector spaces as target category and homology functor between them. Historically these restrictions are natural because persistence theory was born from homology theory, but they are not necessary, as proven in [3]. In the remainder the domain of the functor used to build categorical persistence will be called *source category* while the codomain will be called *target category*.

In this section, we will sketch the generalisation of persistence theory provided in [3], to which we refer the reader for details. In this approach, persistence functions are defined as the rank of the image of morphisms. For this reason, a good notion of image is achieved by considering regular categories as target categories. In a regular category, see definition 2.1.21, each morphism $X \xrightarrow{\phi} Y$ can be factored as $X \overset{\varepsilon}{\twoheadrightarrow} Z \overset{\mu}{\hookrightarrow} Y$, where μ is a monomorphism and ε is a regular epimorphism. Thus the key ingredients of this formulation will be regular categories endowed with rank functions.

Rank functions

We borrowed the following definitions and text from [3].

Rank function, ranked category

Definition 2.2.1 *Let* **R** *be a regular category. Given a lower-bounded function* $r : \mathrm{Obj}(\mathbf{R}) \longrightarrow \mathbb{Z}$*, we say that* r *is a rank function if:*

For any monomorphism $A \hookrightarrow B$, $r(A) \leq r(B)$

2. *For any regular epimorphism* $B \twoheadrightarrow D$, $r(B) \geq r(D)$
3. *For any pullback square:*

$$\begin{tikzcd}
A \arrow[r, hook, "\iota_1"] \arrow[d, two heads, "\pi_1"'] & B \arrow[d, two heads, "\pi_2"] \\
C \arrow[r, hook, "\iota_2"] & D
\end{tikzcd}$$

where ι_1, ι_2 are monomorphisms and π_1, π_2 are regular epimorphisms, the following inequality holds:

$$r(B) - r(A) \geq r(D) - r(C)$$

We say that a rank function r is strict *if the inequalities in conditions 1 and 2 are strict unless the morphisms are invertible. If furthermore $\mathbf{R}$ has an initial object $\emptyset$ and $r(\emptyset) = 0$, we say that r is* 0-based. *A* ranked category $(\mathbf{R}, r)$ *is simply a regular category $\mathbf{R}$ equipped with a rank function r.*

An important class of rank functions is the one of fiber-wise rank functions, see Example 2.1.11.

Definition 2.2.2 *Given a regular category $\mathbf{R}$ with terminal object pt, we say that a function $r : \mathrm{Obj}(\mathbf{R}) \longrightarrow \mathbb{Z}$ is fiber-wise if, for all regular epimorphism $B \overset{\phi}{\twoheadrightarrow} D$, we have:*

Fiber-wise function

$$r(B) - r(D) = \sum_{\iota \in \mathrm{Hom}(\mathrm{pt}, D)} (r(B \times^{\iota}_{D} \mathrm{pt}) - r(\mathrm{pt})) \tag{2.1}$$

where the $B \times^{\iota}_{D} \mathrm{pt}$ realizes the pullback:

$$\begin{tikzcd}
B \times^{\iota}_{D} \mathrm{pt} \arrow[r, hook] \arrow[d, two heads] & B \arrow[d, two heads, "\phi"] \\
\mathrm{pt} \arrow[r, hook, "\iota"] & D
\end{tikzcd}$$

Proposition 2.2.1 *Let $\mathbf{R}$ be a regular category with terminal object pt and $r : \mathrm{Obj}(\mathbf{D}) \longrightarrow \mathbb{Z}$ a lower-bounded function such that:*

1. *For any monomorphism $A \hookrightarrow B$, $r(A) \leq r(B)$*
2. *For any regular epimorphism $A \twoheadrightarrow \mathrm{pt}$, $r(A) \geq r(\mathrm{pt})$*
3. *r is fiber-wise*

Then r defines a rank function on $\mathbf{R}$.

Proposition 2.2.2 *If a functor $F : \mathbf{Q} \longrightarrow \mathbf{R}$ preserves the image factorization, i.e. it preserves monomorphisms and regular epimorphisms, and r is a rank function on $\mathbf{R}$, then $r \circ F : \mathrm{Obj}(\mathbf{Q}) \longrightarrow \mathbb{Z}$ is a rank function on $\mathbf{Q}$.*

These conditions simplify as we move to the stronger notion of Abelian category. There, indeed, to prove that a function is fiber-wise, it is enough to check that it satisfies the following condition.

Proposition 2.2.3 *Let* $\mathbf{R}$ *be an Abelian category. Then* $r : \mathrm{Obj}(\mathbf{R}) \longrightarrow \mathbb{Z}$ *is fiber-wise if and only if for all short exact sequence* $A \hookrightarrow B \twoheadrightarrow D$, $r(A)+r(D) = r(B)+r(0)$.

Also proving that a fiber-wise function is a rank function is much easier.

Proposition 2.2.4 *Let* $\mathbf{R}$ *be an Abelian category. If* $r : \mathrm{Obj}(\mathbf{R}) \longrightarrow \mathbb{Z}$ *is fiber-wise and for all* $D \in \mathrm{Obj}(\mathbf{R})$, $r(0) \leq r(D)$ *then* r *is a rank. Furthermore, if* $r(0) = r(D)$ *only if* D *is null then* r *is strict.*

Example 2.2.1 Let $\mathbf{C}$ be an Abelian category. As in [15, Sect. 1] we say that an object X in $\mathbf{C}$ has finite length if there exists a series of inclusions

$$0 \simeq X_0 \hookrightarrow X_1 \hookrightarrow \cdots \hookrightarrow X_n \simeq X$$

where all quotients X_i/X_{i-1} are simple. If such series exists, then $length(X) = n$. If in an Abelian category all objects have finite length, we say that the category has finite length.

Proposition 2.2.5 *Given* $\mathbf{C}$ *an Abelian category of finite length, the function*

$$length : \mathrm{Obj}(\mathbf{C}) \longrightarrow \mathbb{Z}$$

is a strict 0*-based fiber-wise rank.*

Categorical persistence

Given a functor $\psi : \mathbf{C} \longrightarrow \mathbf{R}$, between a category $\mathbf{C}$ and a regular category $\mathbf{R}$ endowed with a rank function r, it might be useful to define a rank on $\mathbf{C}$, but this is not possible unless $\mathbf{C}$ is regular and the functor ψ preserves image factorizations, see proposition 2.2.2. These assumptions are not satisfied in usual scenarios. For this reason it is necessary to define *categorical persistence functions* directly on the morphisms of $\mathbf{C}$.

Categorical persistence function

Definition 2.2.3 *Let* $\mathbf{D}$ *be a category. A lower-bounded function* $p : \mathrm{Morph}(\mathbf{D}) \longrightarrow \mathbb{Z}$ *is a categorical persistence function if, for all* $u_1 \longrightarrow u_2 \longrightarrow v_1 \longrightarrow v_2$*, the following* *inequalities hold:*

$$p(u_1 \to v_1) \leq p(u_2 \to v_1) \text{ and } p(u_2 \to v_2) \leq p(u_2 \to v_1).$$

2. $p(u_2 \to v_1) - p(u_1 \to v_1) \geq p(u_2 \to v_2) - p(u_1 \to v_2)$.

Proposition 2.2.6 *Given a functor $F : \mathbf{C} \to \mathbf{D}$ and a categorical persistence function p for $\mathbf{D}$, $p \circ F$ is a categorical persistence function for $\mathbf{C}$.*

Given a regular category $\mathbf{R}$, we denote by $im : \mathrm{Morph}(\mathbf{R}) \to \mathrm{Obj}(\mathbf{R})$ the map associating to each morphism its image.

Proposition 2.2.7 *Given a ranked category $(\mathbf{R}, r)$, $r \circ im$ defines a categorical persistence function on $\mathbf{R}$.*

Proposition 2.2.8 *Given a ranked category $(\mathbf{R}, r)$ and a functor $F : \mathbf{C} \to \mathbf{R}$, the function $r \circ im \circ F : \mathrm{Morph}(\mathbf{C}) \to \mathbb{Z}$ is a categorical persistence function.*

With the notion of categorical persistence function it is possible to give the following definition of *generalized persistence functions*, see [5].

Definition 2.2.4 *An $(\mathbb{R}, \leq)$*-indexed diagram *is any functor F from the category $(\mathbb{R}, \leq)$ to $\mathbf{C}$. The $(\mathbb{R}, \leq)$-indexed diagram F is said to be* monic *if all morphisms of its image are monomorphisms of $\mathbf{C}$. $(\mathbb{R}, \leq)$-indexed diagrams form a category, $\mathbf{C}^{(\mathbb{R},\leq)}$.*

Indexed diagram

The information provided by persistence can be visualised as persistence diagrams.

Definition 2.2.5 *Given $u < v \in \mathbb{R} \cup \{-\infty, +\infty\}$ we define the multiplicity of u, v as the minimum of the following expression, over I_u, I_v disjoint connected neighborhoods of u and v respectively:*

$$\begin{aligned} p_F(\sup(I_u), \inf(I_v)) - p_F(\inf(I_u), \inf(I_v)) \\ -p_F(\sup(I_u), \sup(I_v)) + p_F(\inf(I_u), \sup(I_v)) \end{aligned} \tag{2.2}$$

We denote this quantity by $\mu(u, v)$. Whenever $\mu(u, v) > 0$ we say (u, v) is a cornerpoint. By convention in this definition we consider $p_F(u, v) = \min_{x,y} p_F(x, y)$ whenever either u or v is not finite.

Multiplicity, cornerpoint

Persistence diagram

Definition 2.2.6 *The persistence diagram D_F associated with the persistence function p_F is the multiset of its cornerpoints, along with all the diagonal points $\{(u,u) | u \in \mathbb{R}_{\geq 0}\}$ with infinite (countable) multiplicity.*

Given a persistence diagram D, the coordinates of a point $(u,v) \in D$ with $u \neq v$ correspond to the birth and death times of a certain feature. It is possible to define a distance between persistence diagrams.

Bottleneck distance

Definition 2.2.7 *Given persistence diagrams D, D', let Γ be the set of all bijections between D and D'. We define the* bottleneck distance *as the real number*

$$d(D,D') = \inf_{\gamma \in \Gamma} \sup_{p \in D} \|p - \gamma(p)\|_\infty$$

.

Persistence through posets

Monic persistence function

In what follows we will restrict ourselves to filtrations, i.e. given a category $\mathbf{C}$ we will work in the subcategory $\mathbf{C}_m$, where the only morphisms allowed are monomorphisms. In this framework we call a *persistence function* a categorical persistence function (see definition 2.2.3) on the category $(\mathbb{R}, \leq)$, while we call a *monic persistence function* on $\mathbf{C}$ a categorical persistence function on $\mathbf{C}_m$.

Weakly directed

Upbeat, downbeat

Core

A poset P is *weakly directed* if, whenever $a, b \in P$ have a lower bound, they also have an upper bound. An element $p \in P$ *upbeat* (resp. *downbeat*) if the set of all elements strictly higher (resp.lower) than p has minimum (resp. maximum). It is possible to consider the homotopy type of posets, [16], and moreover determine if two posets have the same homotopy type. Following [17], we can define the *core* of a poset P, core(P), as the deformation retract of P that is minimal. Since the deletion of upbeat and downbeat elements does not change the homotopy type of P, the core of P can be obtained by removing the beat points until none is left.

Theorem 2.2.9 *Two finite posets are homotopy equivalent if and only if they have isomorphic cores.*

Definition 2.2.8 *Let $\mathscr{P} \subseteq \mathrm{Obj}(\mathbf{C})/\simeq$ be a property preserved by isomorphisms. We call $S_{\mathscr{P}}$ the functor $\mathbf{C}_m \longrightarrow \boldsymbol{Poset}_m$ that associates to each object in $\mathbf{C}$ the poset of subobjects that respect the property $\mathscr{P}$. We say that the property $\mathscr{P}$ is* weakly directed *if, for all $X \in \mathrm{Obj}(\mathbf{C})$, $S_{\mathscr{P}}(X)$ is a weakly directed poset.*

Weakly directed property

Proposition 2.2.10 *Let $\mathscr{P}$ be a weakly directed property on $\mathrm{Obj}(\mathbf{C})$. Then $\mathscr{P}$ induces a stable categorical persistence function on $\mathbf{C}_m$ denoted as $p_{\mathscr{P}}$.*

Persistence on graphs | 3

Many real-world problems can be formalized as graphs, i.e. as vertices connected by edges. Think about the emails exchanged by colleagues or the spread of information in a social network. In some contexts, it may be relevant to assign a concept of strength to these edges, for example, associating a number, or an orientation if the information flows only in one direction. In section 3.1 we review some of the basic notions of graph theory, in section 3.2 we provide some examples of how categories of graphs can be endowed with rank functions to obtain target categories. In section 3.3 we show through examples how connectivity related weakly direceted properties work in the framework of directed graphs and compare it with the undirected case. In section 3.4 we will study how some features change when we assign a lot of different orientations to the same undirected graph. For some reference about graph theory, you can refer to [18, 19].

3.1 Background

In this section, we want to introduce some of the principal notions that will be useful throughout this chapter.

Graphs and digraphs

If we ask anybody what a graph is, the most common answer will be some dots connected by arcs. This definition, although primitive, is precisely the intuition laying behind the formal notion of a graph we will state in the next few lines.

Graph, vertex, edge, incidence function, order, size

Definition 3.1.1 *A graph G is an ordered triple $(V(G), E(G), r_G)$ consisting of a set $V(G)$ of* vertices*, a set $E(G)$, disjoint from $V(G)$, of* edges *and an* incidence function *r_G that associates to each edge of G an unordered pair of (not necessarily distinct) vertices of G. If e is and edge and u, v are vertices such that $r_G(e) = \{u, v\}$, then e is said to* join *u and v and u and v are called the ends of e. The number of vertices of G is called the order and denoted as $|G|$, while the number of edges of G is called the size of G and denoted as $||G||$. For notational simplicity we denote the unordered pair $\{u, v\}$ as uv.*

Incident
Adjacent
Neighbours
Degree
Isolated vertex
Maximum, minimum, average degree, independent, loop

The ends of an edge e are said to be *incident* with the edge and vice versa. Two vertices incident with a common edge are said *adjacent* , as are two edges with a vertex in common, and two distinct adjacent vertices are said *neighbours* . The set of neighbours of a vertex v in a graph G is denoted as $N_G(v)$. The degree of a vertex v in a graph G, denoted as $d_G(v)$ is the number of edges of G incident with v, each loop counting as two edges. A vertex of degree zero is called an *isolated vertex* . We denote by $\delta(G)$ and $\Delta(G)$ the *minimum* and the *maximum* degree of the vertices of G, and by $d(G)$ the *average* degree. A set of vertices (edges) is called *independent* if taken any pair of its elements, they are not adjacent. An edge with two identical ends is called a *loop*, while an edge with distinct ends is a *link*. Two edges are said to be *parallel* if they have the same ends.

Subgraph

Definition 3.1.2 *Consider a graph $G = (V(G), E(G), r_G)$. Another graph $G' = (V(G'), E(G'), r_{G'})$ is a subgraph of G if $V(G') \subseteq V(G)$, $E(G') \subseteq E(G)$ and $r_{G'}$ is the restriction of r_G to $E(G')$. We then say that G* contains *G', and write $G' \subseteq G$.*

Given a graph G there are two natural ways of deriving subgraphs, *edge deletion* and *vertex deletion*. Edge deletion consists in the removal of an edge e, leaving all the vertices and the remaining edges intact. Similarly, vertex deletion consists in the removal of a chosen vertex v together with all the edges incident to v. We will denote these subgraphs as $G - \{e\}$ and $G - \{v\}$. A subgraph G' obtained from G by edge deletion only is called *spanning subgraph* , i.e. $V(G') = V(G)$.

Spanning subgraph

If Y is the set of deleted edges, then G' is also denoted as $G - Y$. A graph obtained by vertex deletion only is called *induced subgraph* . If X is the set of vertices deleted, the resulting subgraph is denoted by $G - X$. We will say that this graph is induced by $Y = V(G) \setminus X$.

Induced subgraph

Definition 3.1.3 *A graph G that has no loops and no parallel edges is called simple graph. When no confusion arises, we define the simple graph G as the pair $(V(G), E(G))$, where $E(G) \subseteq (V(G) \times V(G))$, i.e. each edge is uniquely determined by its ends.*

Simple graph

A *path* is a simple graph whose vertices can be arranged in a linear sequence in such a way that two vertices are adjacent if they are consecutive in the sequence and are non-adjacent otherwise. If the sequence is cyclic, then it is a *cycle* .

Path

Cycle

Think about the graph naturally associated with a railway network, where the train stations are vertices and an edge connects two stations if a direct railway line connects them. This representation does not report the strength of such connections (e.g. the number of people travelling between them). In many applications, it is not only useful to know which elements of a set are connected, but also to associate a weight to each connection.

Definition 3.1.4 *A weighted graph is a pair (G, w) where G is a graph and $w : E(G) \longrightarrow \mathbb{R}$ is a weight function associating to each edge a real number.*

Weighted graph

If relevant, graphs can be endowed with a direction. Imagine, for instance, to represent with a graph the streets connecting squares in a city. The edges of this graph the lanes of the streets, need to be oriented.

Definition 3.1.5 *A directed graph D is an ordered triple $(V(D), E(D), d_D)$ consisting of a set $V(D)$ of* vertices, *a set $E(D)$, disjoint from $V(D)$, of* edges *and an* incidence function *d_D that associates to each edge of D an ordered pair of (not necessarily distinct) vertices of D, i.e. if u, v are the ends of e, $d_D(e) = (u, v)$. The vertex u is the* tail *of e and v is the* head *of e; we also say that u dominates v.*

Directed graph

Given a directed graph D, if X and Y are subsets of $V(D)$, we denote set the edges whose tails lie in X and whose heads lie in Y by $E_D(X, Y)$. If $Y = V(D) \setminus X$, the set $E_D(X, Y)$ is called the *outcut* of D associated to X, and denoted by $\partial^+(X)$. Analogously, the $E_D(Y, X)$ is called the *incut* of D associated with X, and denoted by $\partial^-(X)$.

Outcut, incut

Given a directed graph $D = (V(D), E(D), d_D)$, we can associate to D an undirected graph G where $V(G) = V(D), E(G) = E(D)$ and for every edge $e \in E(G)$, if $d_D(e) = (u, v)$ then $r_G(e) = \{u, v\}$, called the *underlying* graph. A *directed path* or *directed cycle* is an orientation of a path or a cycle in which each vertex dominates its successor in the sequence.

Underlying graph

Directed path

Connectivity

In this subsection we will recall different notion of connectivity on graphs and digraphs.

Let $\mathcal{F}$ be a family of subgraphs of a graph G. A member F of $\mathcal{F}$ is *maximal* in $\mathcal{F}$ if no members of $\mathcal{F}$ properly contain F. Given a property p we will say that F is maximal with respect to p if it is maximal in the family of subgraphs of G satisfying p.

Connected graph

Definition 3.1.6 *A graph G is connected if for every partition of its vertex set $V(G)$ into two nonempty sets X and Y, there is an edge with one end in X and one end in Y.*

Connected component

Definition 3.1.7 *Let $\mathcal{C}$ be the family of connected subgraphs of a graph G. We say that a subgraph X of G is a connected component of G if it is maximal in $\mathcal{C}$.*

The notion of connectivity just introduced simply ensures that, if it is satisfied, every vertex is reachable from any other point of the graph. It is possible to define stronger notions of connectivity.

k-connectivity

Definition 3.1.8 *Given a graph G and $k \in \mathbb{N}$, we say that G is k-connected if the subgraph $G - X$ is connected for every $X \subset V(G)$, $|X| < k$. Similarly, we say that G is k-edge connected if $G - Y$ is connected for every $Y \subset E(G)$, $|Y| < k$.*

Blocks

Definition 3.1.9 *Given a graph G we say that a vertex $v \in V(G)$ is a* cut vertex *if its removal increases the number of connected components. Given a subgraph H of G, we say that H is a block of G if it is connected, it does not contain any cut vertex and it is maximal with respect to these properties.*

Definition 3.1.10 *Given a graph G we say that an edge $e \in E(G)$ is a* cut edge *if its removal increases the number of connected components. Given a subgraph H of G we say that H is an edge-block of G if it is connected, it does not contain any cut edge and it is maximal with respect to these properties.*

Edge blocks

All the definitions about connectivity given so far are referred to undirected graphs. Connectivity can be also defined for directed graphs.

Definition 3.1.11 *Consider a directed graph D. We say that D is*

Weak, regular, strong connectivity

weakly connected if its underlying graph G is connected;

- *regularly connected if for every pair of vertices $u, v \in V(D)$, there is either a directed path connecting u to v or a directed path connecting v to u;*
- *strongly connected if for every pair of vertices $u, v \in V(D)$, there are a directed path connecting u to v and a directed path connecting v to u.*

Let us study the relations induced on vertices by the notions of regular and strong connectivities. Consider two vertices $u, v \in V(G)$. We say that u is regularly connected to v, $u \sim_R v$, if there exists in G either a directed path going from u to v or a directed path going from v to u. We say that u is regularly connected to v, $u \sim_S v$, if there exists in G both directed paths going from u to v and from v to u. Then, $\sim_S$ is an equivalence relation, whose equivalence classes are the strongly connected components of G, while $\sim_R$ is not an equivalence relation. For this reason two strongly connected components do not intersect. These concepts extend also the notion of connected component. Set $\mathcal{P} = \{\text{weak, regular, strong}\}$. We say that H subgraph of D is a p-connected component for $p \in \mathcal{P}$ if it is p-connected and maximal with respect to this property. It is possible to extend also the definitions of blocks and edge blocks to the directed framework.

Definition 3.1.12 *Given an oriented graph D we say that a vertex $v \in V(D)$ is a $p-$cut vertex for $p \in \mathcal{P}$ if its removal increases the number of $p-$connected components. Given a subgraph H of D we say that H is a $p-$block of G if it is $p-$connected, it does not contain any $p-$cut vertex and it is maximal with respect to these properties.*

Directed block

Definition 3.1.13 *Given a graph D we say that an edge $e \in E(D)$ is a $p-$cut edge*

Directed edge-block

for $p \in \mathcal{P}$ if its removal increases the number of $p-$connected components. Given a subgraph H of G we say that H is a $p-$edge-block of D if it is $p-$connected, it does not contain any $p-$cut edge and it is maximal with respect to these properties.

The following theorem gives a characterisation of undirected graphs that admit an orientation for which they are strongly connected.

Theorem 3.1.1 (Robbin's theorem) *Given an undirected graph G it is possible to find an orientation for which it is strongly connected if and only if G is 2-edge connected.*

Proof. See [18, thm 5.10]. □

Families of graphs

In the remainder we will sometimes make use of some particular families of graphs. We list them here.

Complete graph

Definition 3.1.14 *A complete graph is a simple graph in which any two vertices are adjacent. We denote them by K_n, where the parameter n corresponds to the order of the graph.*

Bipartite graph

Definition 3.1.15 *A graph G is bipartite if its vertex set can be partitioned into two subsets X and Y so that every edge in $E(G)$ has one end in X and one end in Y. The partition (X, Y) is a* bipartition *of the graph.*

Complete bipartite graph

Definition 3.1.16 *A graph G is called a complete bipartite graph if it is a simple, bipartite graph with bipartition (X, Y) and every vertex in X is joined to every vertex in Y.*

Cycle graph

Definition 3.1.17 *A graph G is a cycle graph if it is composed by a single cycle. It will be denoted as C_n, where n is the order.*

Random graph

Definition 3.1.18 *A random graph of order n is a subgraph of the complete graph K_n, where each edge of K_n is kept with probability $p \in [0, 1]$. This graph is denoted as $G_{n,p}$.*

3.2 Graphs as target category

Categories of graphs

In this subsection, we will present the categories of graphs that will be used in what follows and state their properties. A category is defined by its objects and morphisms. As objects we will consider graphs (see definition 3.1.1) and simple graphs (see definition 3.1.3). Morphisms will be chosen to preserve incidence, map vertices to vertices and edges to vertices or edges, allowing contractions. More formally we have

Graph morphism

Definition 3.2.1 *Let $G = (V(G), E(G), r_G)$ and $H = (V(H), E(H), r_H)$ be two graphs. Then $f : G \longrightarrow H$ is a* graph morphism *if $f(V_F) \subseteq V(H)$, $f(E_F) \subseteq V(H) \cup E(H)$ in such a way that for every $e \in E(G)$ with $r_G(e) = (u, v)$ we can have either $f(e) \in E(H)$ with $r_H(f(e)) = (f(u), f(v))$, or $f(e) = f(u) = f(v) \in V(H)$. Thus, if there is an edge between two vertices in G we want either an edge between the images of these two vertices or that u, v, e are all mapped to the same vertex.*

Combining the classes of objects and the morphism listed above, we can define the following categories:

- **Graphs**: category of graphs with graph morphisms;
- **SiGraphs**: category of simple graphs with graph morphisms;

From literature we know that the two categories introduced are regular.

Theorem 3.2.1 *The category **Graphs** is regular (see [20] theorem 2.34).*

Theorem 3.2.2 *The category **SiGraphs** is regular (see [21], [22]).*

Rank functions

We will provide some examples of functions defined on graphs and show whether or not they satisfy the conditions for being rank functions. Although for simplicity we will work in **SiGraphs**, unless differently stated, these functions perform in the same way also in **Graphs**.

Coloring

Definition 3.2.2 *Let us consider a simple graph G. We define a coloring on G to be the assignment of a label to each vertex of the graph such that two adjacent vertices have different labels. The minimal number of labels needed to color the graph will be called the chromatic number of G, $\chi(G)$.*

Genus

Definition 3.2.3 *The genus of a graph G is the minimal integer n such that there is an embedding of G into a surface of genus n.*

Crossing number

Definition 3.2.4 *Given a graph G, the crossing number cr(G) is the smallest number of edge crossing of a planar representation of the graph G.*

Circuit number

Definition 3.2.5 *Given a graph G, its circuit number cn(G) is the minimal number of edges that must be removed from the graph to break all its cycles.*

Thickness

Definition 3.2.6 *Given a graph G, we define its thickness $\theta(G)$ to be the minimum number of planar graphs whose union is G.*

k-clique, clique number

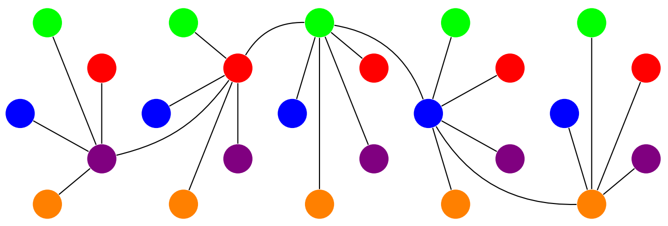
Definition 3.2.7 *Given a graph G we say that a subgraph H of G is a k-clique if it is a complete graph of order k. The maximal k for which G contains a k-clique will be called the clique number of G and denoted $\omega(G)$.*

Spanning tree, number of spanning trees

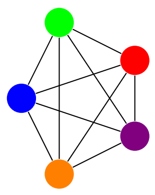
Definition 3.2.8 *Given a graph G, a spanning tree for G is a subgraph of G which is a tree and contains every vertex of G. The number of spanning trees of a graph G will be denoted by $t(G)$.*

Proposition 3.2.3 *The functions chromatic number, genus, crossing number, circuit rank, connectivity, thickness, clique numbers and the number of spanning trees do not satisfy condition 2) of the definition of rank function.*

Proof. Figure 3.1 contains the graphs used to provide the epimorphism contradicting condition 2). The domain of such epimorphism if depicted in panel a). This graph G is composed by five copies of $K_{1,4}$, connected each other as in figure, while the codomain is a graph H isomorphic to K_5. The epimorphism is obtained mapping the vertices in G to the vertex in H sharing the same color. The values of the functions listed in the proposition on these two graph will not satisfy the inequality $r(G) \geq r(H)$, as required in definition 2.2.1.



(a) The graph G



(b) The graph H

Figure 3.1: The graphs G and H needed to define the epimorphism $f : G \longrightarrow H$ used as counterexample in proposition 3.2.3. The morphism sends each vertex of G to the vertex of H with the same color.

□

Proposition 3.2.4 *The number of connected components (and similarly blocks and edge blocks), does not satisfy condition 1) of the definition of rank function.*

Proof. Let us consider as counterexample for the connected components a graph G composed by two connected components G_1 and G_2. We can consider a graph H obtained from G by adding an edge between G_1 and G_2. Then H is connected and the monomorphism $m : G \longrightarrow H$ provides a counterexample. □

Diameter

Definition 3.2.9 *The shortest path distance $d(u, v)$ between two vertices $u, v \in V$ is the number of edges belonging to the shortest path P connecting u and v. The diameter of a graph G is defined as*

$$diam(G) = \max_{u,v \in V} d(u, v)$$

Girth

Definition 3.2.10 *The girth of a graph G is defined to be the length of the smallest cycle contained in G. If the graph is acyclic the girth is defined to be $+\infty$.*

Proposition 3.2.5 *Diameter and girth do not satisfy condition 1) of the definition of rank function.*

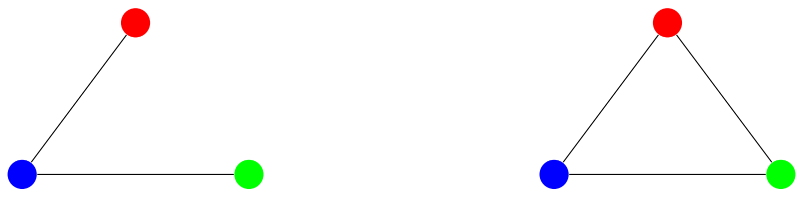

(a) The graph G. **(b)** The graph H.

Figure 3.2: The graphs G and H needed to define the monomorphism $f : G \to H$ used as counterexample to condition 1) of the rank functions definition in proposition 3.2.5.

Proof. Figure 3.2 provides two graphs that define a monomorphism contradicting condition 1) of definition 2.2.1. The monomorphism is defined mapping the vertices of G to the vertex with the same color in H. Both diameter and girth applied to the graphs G an H do not satisfy the inequality defined in definition 2.2.1 condition 1), leading to the sought counterexample. □

Dissociation number

Definition 3.2.11 *Given a graph G, a subset of vertices $D \subset V_G$ is a dissociation if in the induced subgraph all the vertices have at most degree* 1*. The cardinality of a maximal dissociation set is called the dissociation number $dn(G)$.*

Proposition 3.2.6 *The dissociation number of a graph is not a rank function.*

Proof. Let us consider the monomorphism $i : C_5 \to K_5$, where C_5 is the cycle graph with 5 vertices. If we consider in C_5 the subset D containing two adjacent vertices and the antipodal one, this is a maximal dissociation for C_5 and $\mathrm{dn}(C_5) = 3$. In K_5 we have that whenever we pick 3 vertices, in the induced subgraph every vertex has degree 2; so a maximal dissociation set is composed only by 2 vertices and $\mathrm{dn}(K_5) = 2$, contradicting condition 2). □

Proposition 3.2.7 *The maximum degree $\Delta(G)$ is not a rank function.*

Proof. Let us consider the following epimorphism $e : C_8 \to H$, where H is the graph obtained from C_8 by gluing two antipodal edges (obtaining two triangles connected by an edge). Then $\Delta(C_8) = 2$, while $\Delta(H) = 3$ contradicting condition 2). □

Stability number

Definition 3.2.12 *Given a graph G, we say that a set of vertices $D \subset V_G$ is a stable set if its elements are pairwise not adjacent. We say that it is maximum if there* *are no larger stable sets. The cardinality of the maximum stable set is called stability number of G and denoted by $\alpha(G)$.*

Proposition 3.2.8 *The stability number is not a rank function.*

Proof. Let us consider a graph G containing at least one edge. Then let us consider the monomorphism between the graph H, obtained removing from G all the edges, and G. Then $\alpha(H) = |V_H| = |V_G|$, while, since we have at least two adjacent vertices in G, $\alpha(G) < |V_G|$, contradicting condition 1). □

Definition 3.2.13 *Given a graph G, a* matching *M is a set of pairwise non-adjacent edges, none of which are loops. A* maximal matching *is a matching M that is not a subset of any other matching. A* maximum matching *is a matching M whose cardinality is maximal in G. The* matching number *is the cardinality of a maximum matching.*

Matching, maximal matching, maximum matching, matching number

The following two functions are the only ones we found that satisfy conditions 1) and 2) but not condition 3).

Proposition 3.2.9 *The matching number of a graph $\nu(G)$ satisfies conditions 1) and 2) but does not satisfy condition 3).*

Proof. To prove condition 1) let us consider a graph monomorphism $i : G \longrightarrow H$. Let us consider a maximum matching M_G on G. By construction, if two edges are in M_G they are not adjacent in G and then also their images are not adjacent in H. Thus the image of M_G through i is a matching for H and may not be a maximum matching. So $\nu(G) \leq \nu(H)$.

Now consider an epimorphism $e : G \longrightarrow H$, and take a maximum matching M_H on H. We can consider the counterimage of M_H through e. Again by construction this is a matching in G which may not be a maximum matching. So we have that $\nu(G) \geq \nu(H)$, and also condition 2) is satisfied.

For condition 3) let us consider the counterexample provided in figure 3.3. If we label the graphs as in definition 2.2.1 we can see that $\nu(A) = 2$, $\nu(B) = 2$, $\nu(C) = 1$ and $\nu(D) = 2$, contradicting condition 3).

□

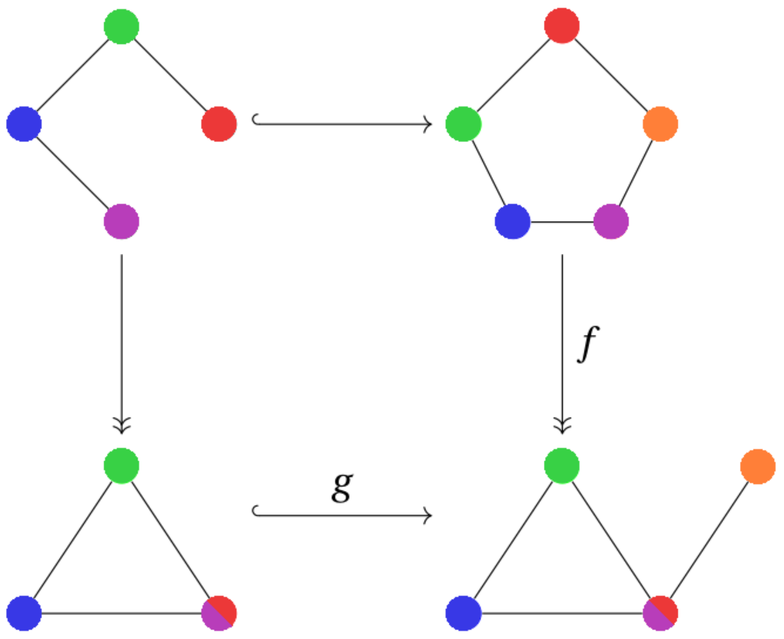


Figure 3.3: This commutative diagram is a counterexample for the matching number function in proposition 3.2.9. The morphisms associates vertices with the same colors and the graph on the top-left is the pullback of the morphisms f and g.

Proposition 3.2.10 *Given a graph G , the cardinality of the set containing all the maximal matchings of the graph $M(G)$ satisfies conditions 1) and 2) but does not satisfy condition 3).*

Proof. To prove condition 1) let us consider a graph monomorphism $i : G \longrightarrow H$. Let us consider a maximal matching M_G on G. By construction, if two edges are in M_G they are not adjacent in G and then also their images are not adjacent in H. Thus the image of M_G through i is a matching for H and it can be extended to a maximal matching M_H. So $\nu(G) \leq \nu(H)$.

Now consider an epimorphism $e : G \longrightarrow H$. Now consider a maximal matching M_H on H. We can consider the counterimage of M_H through e. This is a matching in G which can be extended to a maximal matching M_G. So we have that $m(G) \geq m(H)$, and also condition 2) is satisfied.

Consider the commutative diagram in figure 3.4. Labelling the graphs as in definition 2.2.1 we have that $m(A) = 1$, $m(B) = 2$, $m(C) = 1$ and $m(D) = 5$, and thus this pullback diagram does not satisfy condition 3).

□

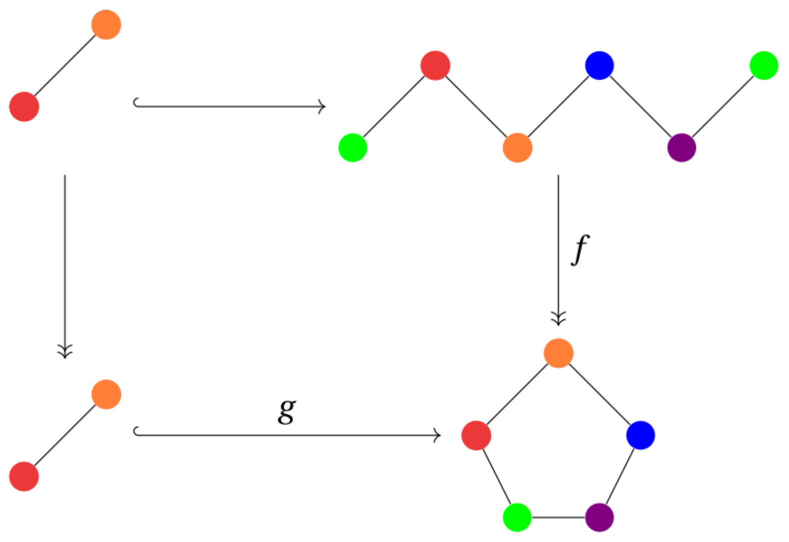


Figure 3.4: This commutative diagram is a counterexample for the function mapping a graph G to the number of maximal matchings of G in proposition 3.2.10. The morphisms associate vertices with the same colors and the graph on the top-left is the pullback of the morphisms f and g.

In the last part we will present some simple functions which are rank functions.

Theorem 3.2.11 *The order function, mapping a graph G to its order $|G|$, is a fiber-wise function (see definition 2.1.11) in both* ***SiGraphs*** *and* ***Graphs****.*

To prove the previous theorem we will need the following results.

Lemma 3.2.12 *Regular epimorphisms in* ***SiGraphs*** *are surjective on vertices and edges (see [21]).*

Lemma 3.2.13 *Epimorphisms in* ***Graphs*** *are surjective on vertices and edges (see [20], proposition 2.25)*

Proof. (Of theorem 3.2.11) From lemmas 3.2.13, 3.2.12, regular epimorphisms are surjective on vertices in both **SiGraphs** and **Graphs**. In both categories the terminal object pt is the graph with only one vertex and no edges. Given regular epimorphism $\phi : G \to H$, we have that $\{\phi^{-1}(v)\}_{v \in V(H)}$ is a partition of $V(G)$ and from surjectivity that $\phi^{-1}(v) \neq \emptyset$ for all $v \in V(H)$. Moreover $|\mathrm{pt}| = 1$.

So we have that

$$\sum_{v_h \in V(H)} \left(|\phi^{-1}(v_h)| - |\mathrm{pt}|\right) =$$
$$= \sum_{v_h \in V(H)} |\phi^{-1}(v_h)| - \sum_{v_h \in V(H)} 1 = |G| - |H|$$

and the function order is a fiber-wise function. □

Theorem 3.2.14 *The function order is a rank function in both* ***SiGraphs*** *and* ***Graphs****.*

Proof. Since the function order is fiber-wise, it is sufficient to prove conditions 1) and 2) of proposition 2.2.1. Consider a monomorphism $\psi : G \hookrightarrow H$. Monomorphisms in both categories are injective on vertices, thus $|G| \leq |H|$ and condition 1) is proven. Consider now a regular epimorphism $\phi : G \twoheadrightarrow \mathrm{pt}$. Since $V(G)$ is cannot be empty, we have that $|G| \geq 1 = |\mathrm{pt}|$. □

Theorem 3.2.15 *The function size, mapping a graph G to its size $\|G\|$ is a rank function.*

In the following proof, given A subgraph of B, we will define $B -_E A$ to be the graph obtained from B removing the edges in A.

Proof. Since we know that monomorphisms are injective in both edges and vertices and regular epimorphisms are surjective on edges and vertices, the first two conditions of the definition are satisfied. For the last condition, consider a commutative diagram of the form

$$\begin{tikzcd}
A \arrow[r, hook, "\iota_1"] \arrow[d, two heads, "\pi_1"'] & B \arrow[d, two heads, "\pi_2"] \\
C \arrow[r, hook, "\iota_2"] & D
\end{tikzcd}$$

where ι_1, ι_2 are monomorphisms and π_1, π_2 are regular epimorphisms. Since ι_1 is injective on edges, $\|A\| = \|\iota_1(A)\|$ and thus $\|B\| - \|A\| = \|B -_E \iota_1(A)\|$. The fact that π_1 is surjective and ι_2 is injective implies that $\iota_2(\pi_1(A)) \subset D$ has size

$\|C\|$. Moreover, since π_2 is surjective, every edge in $D -_E \iota_1(A)$ has nonempty preimage in $B -_E \iota_1(A)$. Thus we have that

$$\|D\| - \|C\| = \|D -_E \iota_2(C)\| \leq \|B -_E \iota_1(A)\| = \|B\| - \|A\|$$

and condition 3) is proven.

□

3.3 Connectivity in directed graphs

In this section, we will analyse the behaviour of connected components, blocks and edge-blocks in directed graphs. We would like to recall the existence of three notions of connectivity for directed graphs, as explained in definition 3.1.11, and the consequent existence of different notions of blocks and edge-blocks, see definitions 3.1.12,3.1.13. These notions of connectivity are very different from one another. In the case of weak connectivity, the analysis of connectivity is reduced to the one made in the undirected case. In the other two cases, orientation is considered, and the results change considerably.

Consider the graph depicted in figure 3.5. It is composed of three disjoint subgraphs A, B, C all connected to a central vertex v by a single edge. Since we have three edges incident to v, at least two of them must have the same orientation with respect to v, outward or inward. Suppose that the two having the same direction are the ones corresponding to the components A and B. Then it is not possible to travel from A to B or from B to A, and thus this graph, although weakly connected, cannot be regularly connected with any orientation.

Strong connectivity is the most difficult of the three kinds of connectivity to achieve. Consider the graphs depicted in figure 3.6, where A and B are two strongly connected components. In panel a) of the figure it is possible to notice that whatever orientation we choose for the edge e the entire graph is regularly connected. However, it is impossible to obtain strong connectivity. Even the addition of many edges between the two components depicted in panel b) will not guarantee strong connectivity. Consider, for example, the case where all the edges share the same orientation, say from A to B. In this scenario it is impossible to reach A from B, contradicting the strong connectivity condition.

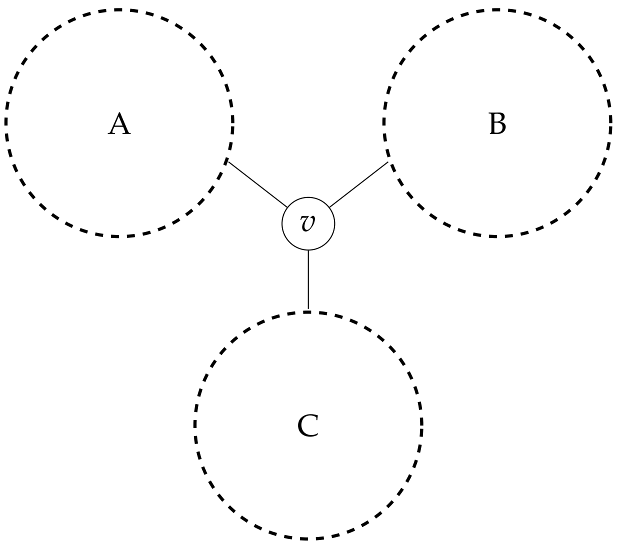
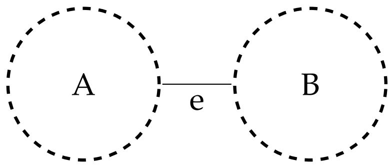
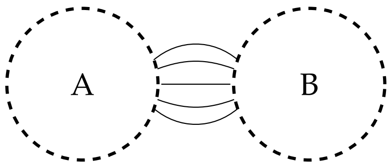


Figure 3.5: An example of a graph that cannot be regularly connected



(a) A graph that cannot be strongly connected.



(b) A graph that can or not be strongly connected.

Figure 3.6: Two examples of graphs that can be non strongly connected.

Since weak connectivity behaves like standard connectivity in undirected graphs, in what follows, we will discuss only regular and strong connectivity. In the remainder, strongly connected components will be denoted by SCC.

Proposition 3.3.1 *Let RC be the property of being regularly connected, i.e. a graph G is RC if it is regularly connected. The property RC is not weakly directed.*

Proof. Consider again the graph in figure 3.5. We have three disjoint subgraphs A, B, C connected to a central vertex v. Suppose that the orientation has an arrow going from A to v, an arrow going from v to B and one from v to C. Suppose moreover for simplicity that the subgraphs A, B and C are strongly connected.

The two regularly directed components C_1 and C_2, respectively induced by the vertices $V(A) \cup \{v\} \cup V(B)$ and $V(A) \cup \{v\} \cup V(C)$, which are maximal, belong to the poset associated to the property of being regularly connected, where given two graphs $H, G, H \leq G$ if H is a subgraph of G. They have lower bound since the subgraph A is regularly connected and $A \subset C_1 \cap C_2$ but by the maximality of C_1 and C_2 they will not have an upper bound, contradicting definition 2.2.8.

$\square$

Notice that this problem is related to the fact that regular connectivity does not induce an equivalence relation on the vertex set $V(G)$. For this reason the notion of strong connectivity does not share the same problem.

Let SC be the property of being strongly connected, i.e. a graph G is SC if it is strongly connected.

Proposition 3.3.2 *SC is a weakly directed property.*

Proof. Consider a graph G and two strongly connected subgraphs G_1 and G_2. If their intersection $G_1 \cap G_2$ is strongly connected, and thus G_1 and G_2 have a lower bound, their union $G_1 \cup G_2$ is strongly connected as well, since, by strong connectivity, if $G_1 \cap G_2 \neq \emptyset$, then $G_1 \cap G_2 = G_1 = G_2 = G_1 \cup G_2$. $\square$

For this reason SC induces a persistence function p_{SC} on graph filtrations, called the *strongly connected component number*. For a given graph filtration F, $p_{SC}(u, v)$ equals the number of SCCs in $F(v)$ that contains at least one SCC when restricted to $F(u)$.

This approach can be generalized to a wider class of properties deriving from equivalence relations defined on graphs.

Proposition 3.3.3 *Consider an equivalence relation $\sim$ defined on the vertex set of a graph. Then the property of being an equivalence class with respect to $\sim$ is a weakly directed property.*

Proof. Straightforward. $\square$

Examples of persistence diagrams on digraphs

In what follows, we provide examples showing the behaviour of strongly connected components, strong blocks and strong edge blocks on a given underlying weighted graph. We will show how these features modulate when assigning different orientations to the graph. The graph we used has a high minimum

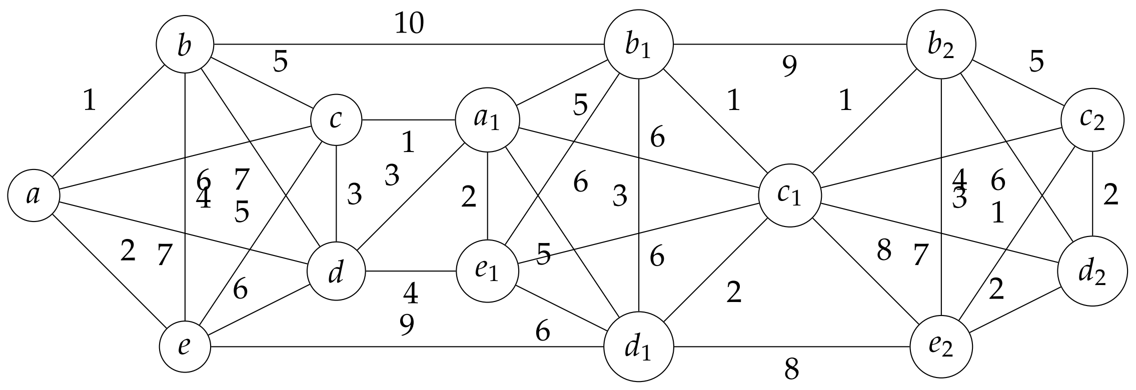


Figure 3.7: The underlying weighted graph G used in the next examples

vertex degree, since theorem 3.1.1 states that it is necessary, but not sufficient, to have a 2-edge connected subgraph to get a strongly connected component, and stronger notions of connectivity are required for strong blocks and edge blocks.

The weighted graph depicted in figure 3.7 is the undirected graph used in the various examples. It is composed by three subgraphs K^0, K^1, K^2, spanned by the set of vertices $\{a, b, c, d, e\}$, $\{a_1, b_1, c_1, d_1, e_1\}$ and $\{a_2, b_2, c_1, d_2, e_2\}$ respectively, which are isomorphic to K_5. Edges have been added to connect these subgraphs. In this way we obtained a graph with minimum degree $\delta(G) = 4$. The choice of having subgraphs isomorphic to K_5 is related to the existence of orientations for which it is a strong block or edge-block. In this way it is possible to get blocks and edge-blocks easily by choosing appropriate orientations on these subgraphs.

We will study 4 different orientations of G, denoted by $G_i = (V, E_i)$ for $i \in \{1, \ldots, 4\}$, where each E_i is an orientation defined on E. The oriented graphs are depicted in figures 3.8,3.10,3.12,3.14.

The orientation of G_1 is chosen in such a way that strongly connected components, strong blocks and strong edge-blocks appear along with the filtration. The orientations are chosen in such a way that every vertex has at least two inward and two outward edges. This choice is dictated by the fact that to obtain, for example strongly connected components, since it is necessary, but not sufficient, to have for each vertex at least one inward and one outward edge. This decision reflects on the persistence diagrams depicted in figure 3.9. In panel a) is reported the strongly connected components persistence diagram. It is possible to notice that two SCCs are born for $t = 3$, corresponding to the subgraphs $\{c, d, a_1\}$ and $\{b_1, c_1, d_1\}$ which merge for time $t = 5$. Panel

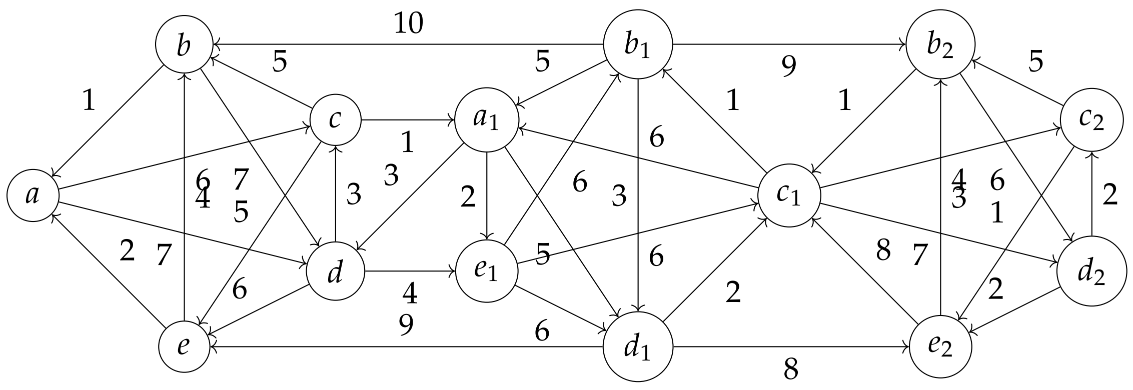
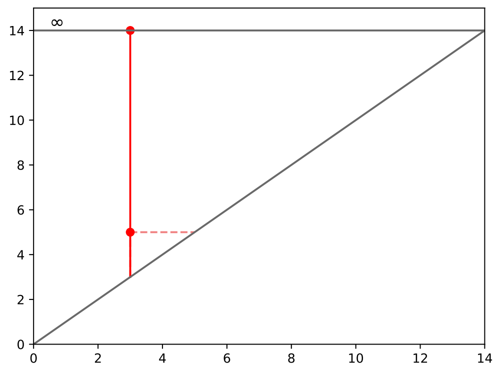


Figure 3.8: The graph G_1, i.e. the one with the first orientation

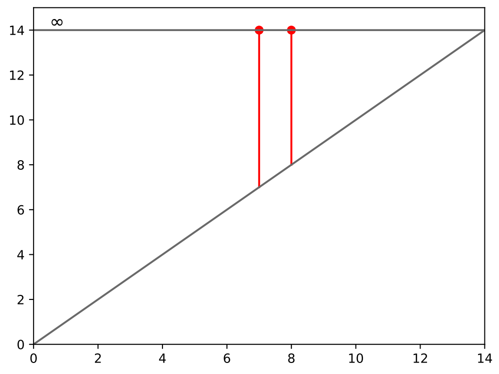
b) reports the strong blocks persistence diagram. There are two strong blocks which never merge, one born for $t = 7$ and one for $t = 8$. These correspond to K^0 and K^2, respectively. We can notice that K^1 cannot be a strong block, since the removal of c_1 leave d_1 with only inward edges. The final blocks are formed by $K^0 \cup K^1$ and K^2. The entire graph is not a block since removing c_1 there are no edges going from K^2 to $K^0 \cup K^1$. In panel c) is depicted the strong edge-blocks persistence diagram. As for the case of the strong blocks, two strong edge-blocks are born, K^0 and K^2 respectively at time $t = 7$ and $t = 8$. In this case they merge for $t = 10$.

The orientation of G_2 is chosen to be similar to the one of G_1, showing how such a minimal difference can affect the final result. For this reason the only change made on G_1 is the orientation of one edge, namely (d_1, e_2). In G_1 the two blocks composed by $K^0 \cup K^1$ and K^2 are not merging because the edges (b_1, b_2) and (d_1, e_2) pointed in the same direction, thus it is not possible to have a directed path going from K_2 to $K^0 \cup K^1$. In G_2 this does not happen because (b_1, b_2) and (e_2, d_1) are pointed in opposite direction, so the two blocks $K^0 \cup K^1$ and K^2 merge.

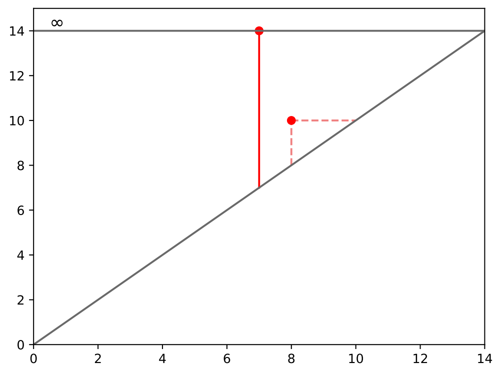
The orientation of G_3 is chosen to be more balanced with respect to the ones of G_1 and G_2, in such a way that for every subgraph X of G_3 the outcut $\partial^+(X)$ and the incut $\partial^-(X)$ have similar cardinality. This has been done properly balancing the number of inward and outward edges for each vertex of G_3. This choice makes it easier to obtain strongly connected components than in G_1 or G_2. Figure 3.13 depicts the persistence diagrams for strongly connected components, strong blocks and strong edge-blocks. The choice of such orientation provides some strongly connected components already at the beginning of the filtration, for time $t = 2$, and the merging of all these components for time $t = 5$, as pictured in the corresponding persistence diagram (see figure 3.13,



(a) The PD relative to the strongly connected components



(b) The PD relative to the strong blocks



(c) The PD relative to the strong edge blocks

Figure 3.9: The persistence diagram computed with respect to the first orientation.

panel a)). For the same reason, each of the three K_5-shaped subgraphs becomes a strong block starting from $t = 6$, as we can see in figure 3.13, panel b).

The orientation assigned to G_4 is defined in such a way that many subgraphs X of G_4 present big differences between the cardinality of the outcut $\partial^+(X)$

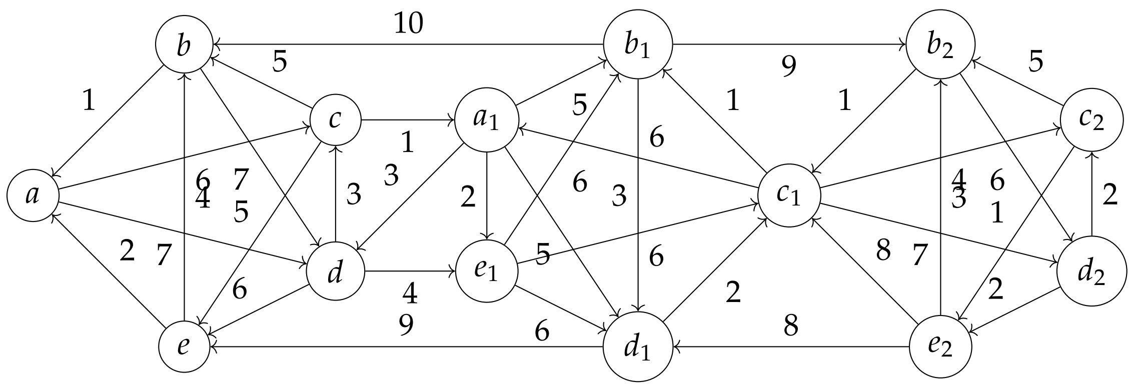
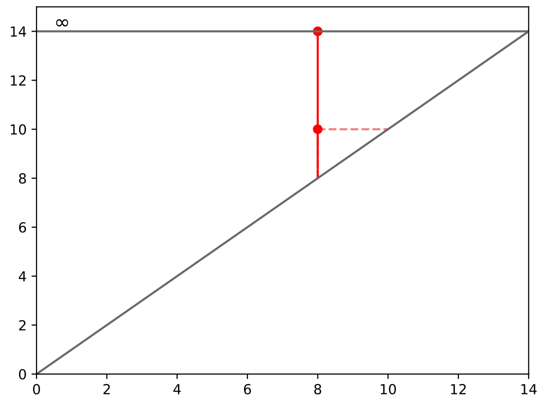


Figure 3.10: The graph G_2, i.e. the one with the second orientation



(a) The PD relative to the strong blocks

Figure 3.11: The persistence diagrams computed with respect to the second orientation.

and the cardinality of the incut $\partial^{-}(X)$. This choice gives a connectivity which is completely different from the one provided in G_1, G_2 and G_3. In figure 3.15 is depicted the strongly connected components persistence diagram. The components born along the filtration are $\{c_2, d_2, e_2\}$ for $t = 2$, $\{a, b, e\}$ and $\{d, a_1, e_1\}$ for $t = 4$ and $\{b_1, b_2, c_1\}$ for $t = 9$. Due to the choice of the orientation, these components never merge. Moreover no strong blocks or strong edge-blocks are ever born.

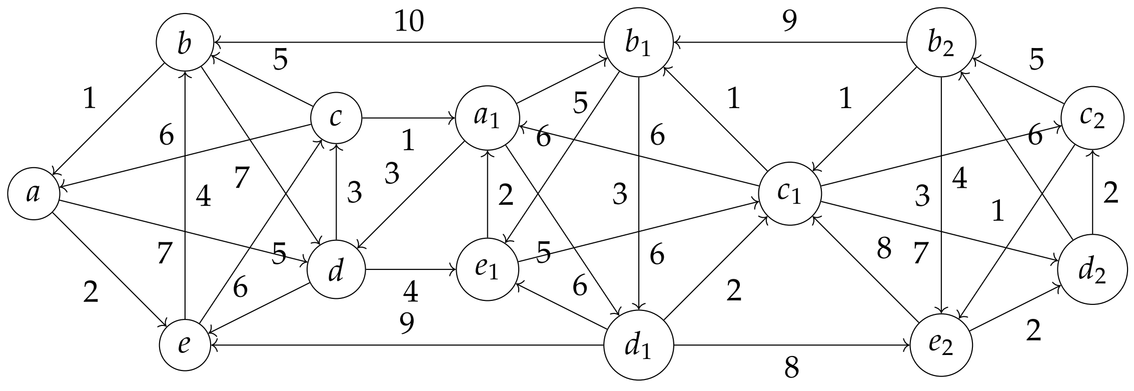
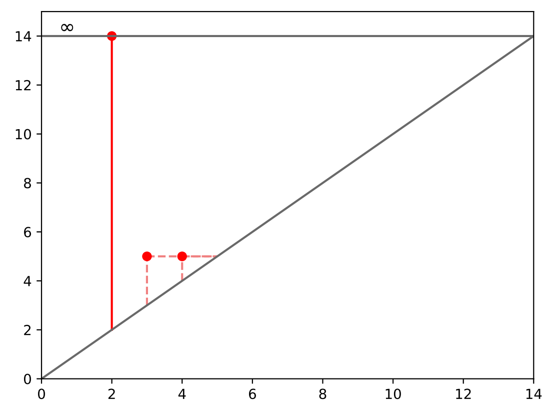
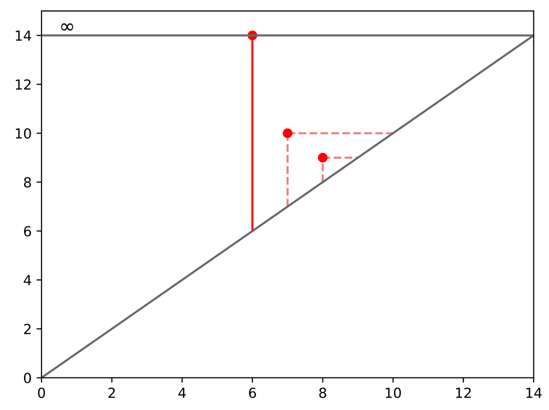


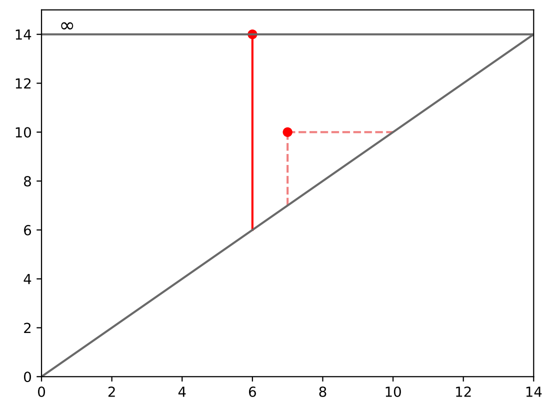
Figure 3.12: The graph G_3, i.e. the one with the third orientation



(a) The PD relative to the strongly connected components



(b) The PD relative to the strong blocks



(c) The PD relative to the strong edge blocks

Figure 3.13: The persistence diagrams computed with respect to the third orientation.

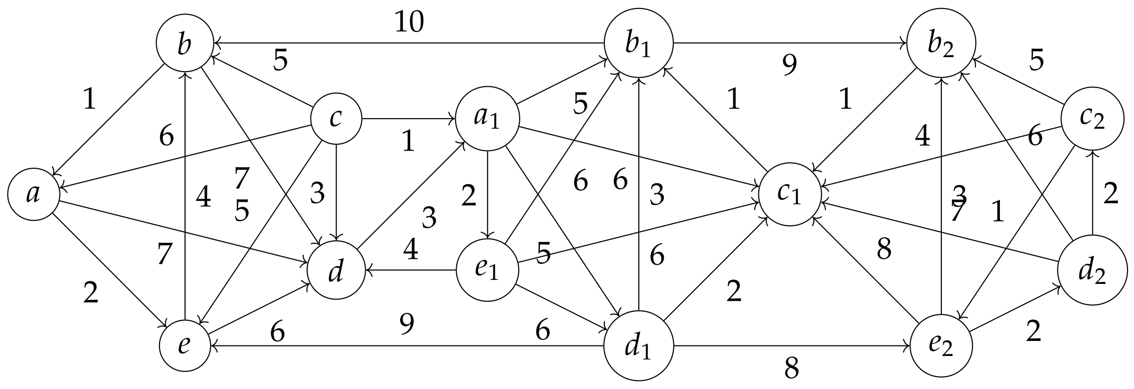
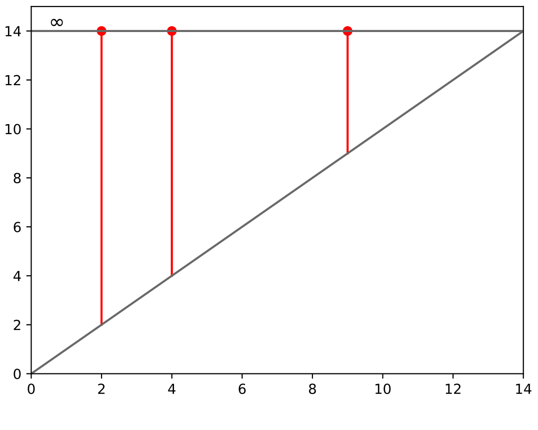


Figure 3.14: The graph G_4, i.e. the one with the fourth orientation



(a) The PD relative to the strongly connected components

Figure 3.15: The persistence diagrams computed with respect to the fourth orientation.

3.4 Cornerpoints distributions for directed graphs

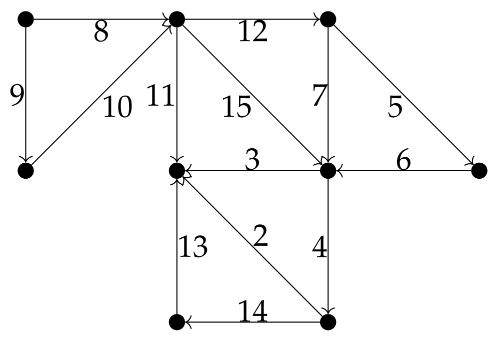
In [4] persistence is used for analysing how certain graph-theoretical properties (e.g. blocks, edge-blocks, clique communities) of a given undirected graph $G = (V, E, r)$ change along the filtration induced by a set of weights $\{w_e\}_{e \in E}$. As shown in the previous sections, by changing the notion of connectivity, it is possible to extend such analysis to the case of oriented graphs.

Given an undirected graph $G = (V, E, r)$ along with a set of weights $\{w_e\}_{e \in E}$, we can consider G as an underlying graph and endow it with an orientation. The analysis of the changes of some properties along the filtration leads to a persistence diagram. Since the set of orientation we can choose from has cardinality $2^{|E|}$, it is possible to compute the persistence diagrams for all these orientations and analyse the distribution of cornerpoints. We may get some information about the structure of G by analysing the distribution of such diagrams, and it may be interesting to relate this to the persistence diagram of the underlying graph. Since theorem 3.1.1 exhibits a connection between the notions of strongly connected components and edge blocks, in this experiments, we will try to highlight any possible connection between the persistence diagrams related to these two objects.

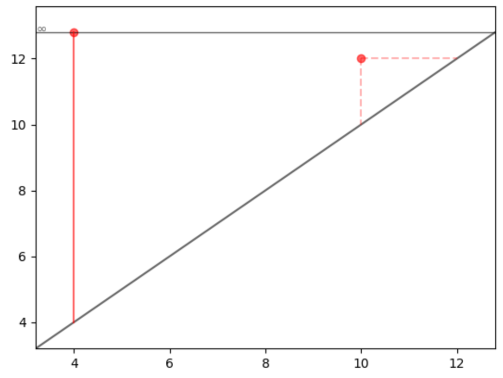
Remark 3.4.1 Since the number of orientations grows quickly as the number of edges increases, it is not computationally feasible to analyse all possible of orientations of G. For this reason we chose to work on a randomly sampled subset of orientations.

Example 3.4.1 Let us consider the undirected graph $G = (V, E, r)$ endowed with weights $\{w_e\}_{e \in E}$ used in [4] Sec. 4.

The fact that the size of G is quite small, $|G| = 15$, implies that only a small number of possible strongly connected components can appear along with the filtration. This leads to a scattered distribution of cornerpoints because only a few cornerpoints can exist. Despite the roughness of the distribution, this experiment can give us an insight into the heuristic effect of the subsampling. In fact, for graphs of this size, it is still feasible to compute the distribution considering the entire set of orientations and compare it with the ones performed subsampling the set of orientations. Figure 3.16 panel a) shows the graph G with the relative weights, while in panel b) is reported its



(a) The graph G



(b) The edge-blocks PD of G

Figure 3.16: The graph G used in the first experiment and the corresponding edge-blocks PD

edge-blocks persistence diagram.

Figure 3.17 depicts the distributions of cornerpoints performed with different subsample. Due to the huge difference between the number of occurring of different cornerpoints, they are represented as the heatmap of the logarithm of the distribution. The distribution computed with respect the entire dataset is reported in panel a), while panels b), c) d) and e) report the distributions computed with subsamples of different size, $10000, 1000, 500, 100$ orientations respectively. Subsample distributions are coherent with the ground truth in most cases, and visible changes appear only when considering the 100 orientations case. This example provides empirical proofs that the subsampling procedure does not affect the distribution strongly.

Moreover, as expected from theorem 3.1.1, the edge blocks persistence diagram is contained in all the distributions of persistence diagrams. The first cornerpoint on the top left corner and the lowest cornerpoint of the distributions correspond to the cornerpoints of the edge blocks persistence diagram.

Example 3.4.2 In the following examples, due to the size of the chosen graphs, it will not be possible to analyse the entire set of all orientations and thus we will not have a ground truth distribution. For this reason, it is necessary to understand for which cardinality of the performed subsampling we can obtain a good approximation of the ground truth distribution. In this experiment we will consider the same random graph $G^{30}_{0.35}$ and compare the distributions obtained considering subsamples of size 100, 200, 500, 1000, 2000, 5000, 10000, 20000, 50000, 100000, 200000, 500000, 750000, 1000000. Results are reported in figure 3.18. It is possible to notice that as the number of orientations increases the distributions stabilise, and we will assume that it is converging to the ground truth distribution. For the scope of future experiments, we consider the distribution obtained with 100000 orientations a good approximation of the ground truth.

Example 3.4.3 In the following example, we will consider a sequence of random graphs G^n_p with $n = 30$. We will associate to this graph a set of random integer weights between 0 and 50. We will see how the distribution of cornerpoints changes as we change the probability p and thus the number of edges appearing in G^n_p. For comparison purposes, we will fix the seed used for generating the random values, in order to have that G^n_p is a weighted subgraph of G_q whenever $p \leq q$. As highlighted before for these experiments will be used only 100000 orientations.

Figure 3.19 shows the distribution of cornerpoints for $p \in \{0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, 0.45\}$. It interesting to notice how the distribution changes as the probability varies. For the smallest values of p, the number of edges belonging to G^n_p is small and so is the number of strongly connected components. SCCs will not appear until the last step of the filtration, and thus the distribution is centred in the top right part of the diagram. As p increases the number of edges in G^n_p increase, making easier the appearance of strongly connected components already in the beginning of the filtration. As a consequence, the distribution shifts close to the origin.

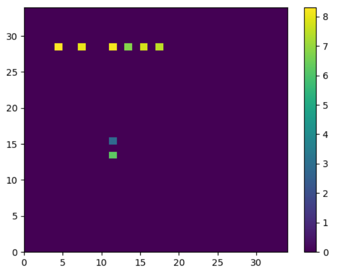
Example 3.4.4 The previous example analysed the relationship between the distribution of cornerpoints and the probability p. In this experiment, we want to analyse the effect of the order of the graph on the distribution. For this experiment we considered orders $n \in \{20, 30, 40, 50\}$ and probability

$p = 0.35$. The results are depicted in figure 3.20. Although the probability p is fixed, whenever we increase the order of the graph, we should expect a higher probability of obtaining edge-blocks and so strongly connected components. This effect reflects in the appearance of strongly connected components already in the beginning of the filtration, shifting the distribution closer to the origin.

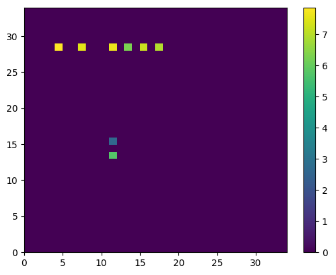
In the experiments performed in this last section, we show how the connectivity of random graphs changes while varying some parameters, e.g., the probability of appearance of an edge, or the order of the graph. To perform such analysis, we consider a graph G as the underlying graph and study the cornerpoint distribution of strongly connected components obtained by assigning to G different orientations. Computationally, it would be arduous to consider all possible orientations, thus in example 3.4.1, we empirically prove that by considering an adequately large subset of possible orientations, we obtain a good approximation of the distribution realised by the whole set of orientations.

In examples 3.4.2 and 3.4.3, we show how changing the probability of appearance of edges and the order of the graph can affect the cornerpoint distribution. As expected, for parameters associated with more complex graphs, i.e. with high order or high probability, cornerpoints cluster near the origin. This effect is due to the fact that more complex structures lead to faster appearance of strongly connected components.

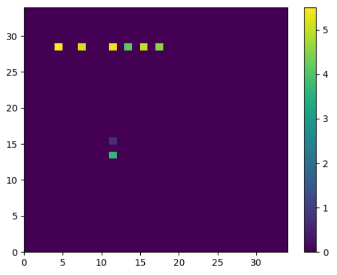
We expect that cornerpoint distributions could be used to compare graph connectivity because they not only capture the presence of connected regions as regular persistence analysis, but also provide an evaluation of the strength of such connectivity. Consider, for example, two graphs G and H, both having an edge block with birth and death times b and d. Suppose that the edge block in G is a loop C_n, while the one in H is a complete graph K_n, with $n > 3$, and that the weights of the edges are randomly chosen to satisfy the birth and death times. Whereas standard persistence would report in both cases only one cornerpoint at coordinates (b, d), the cornerpoint distribution is capable of distinguishing these two scenarios, reporting a unique cornerpoint for G and a more widely spread distribution for H.



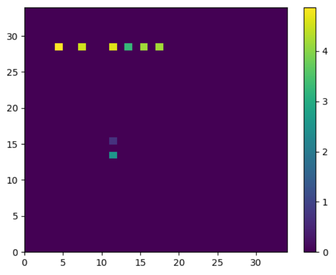
(a) All orientations



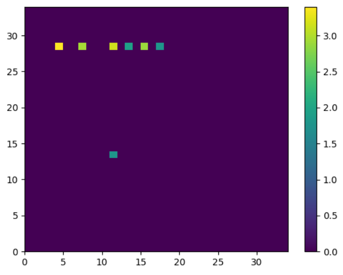
(b) 10000 orientations



(c) 1000 orientations



(d) 500 orientations



(e) 100 orientations

Figure 3.17: The logarithmic heatmap of the distribution of cornerpoints

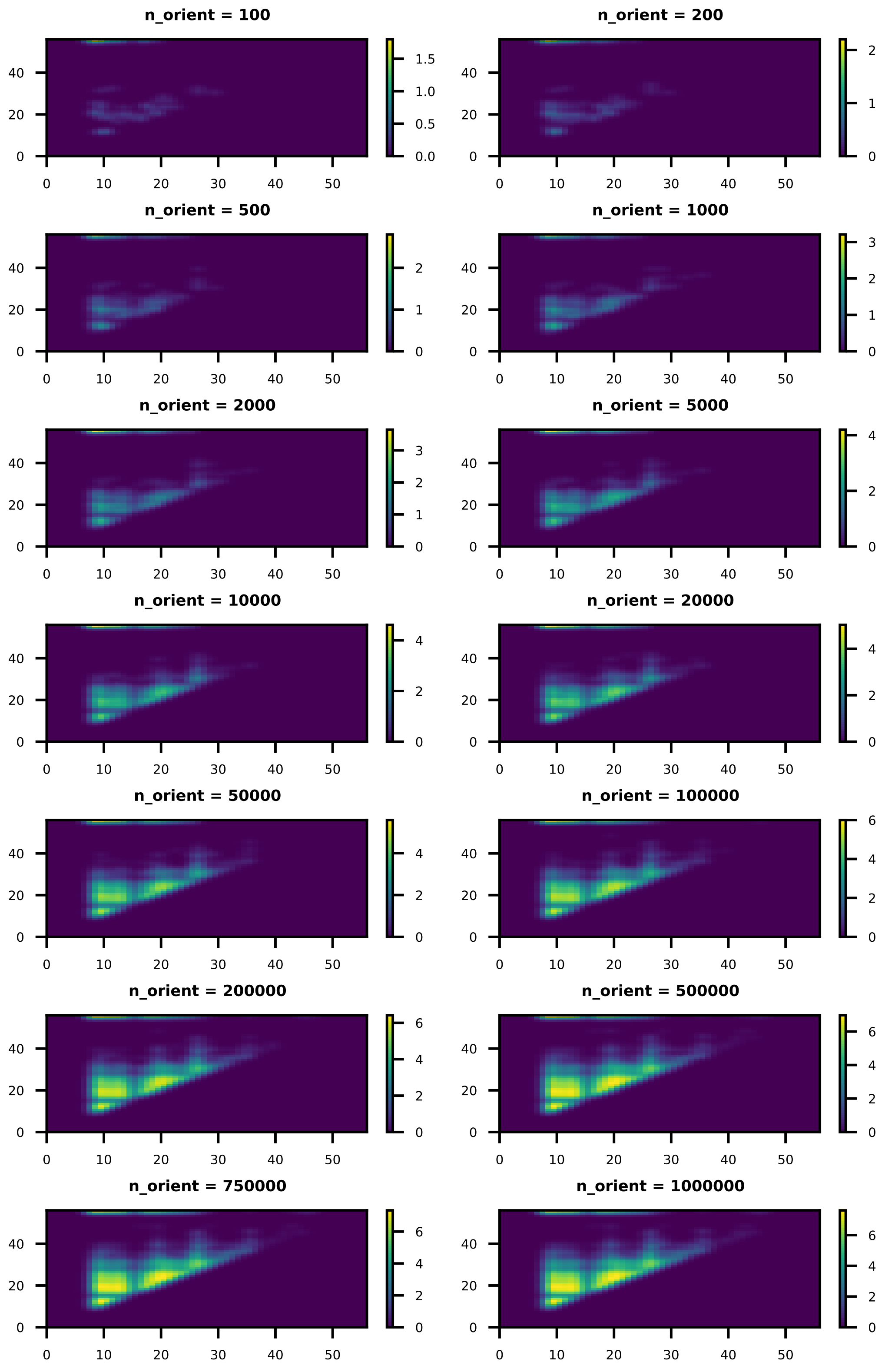


Figure 3.18: The logarithmic heatmap of the distribution of cornerpoints for a random graph with 30 vertices

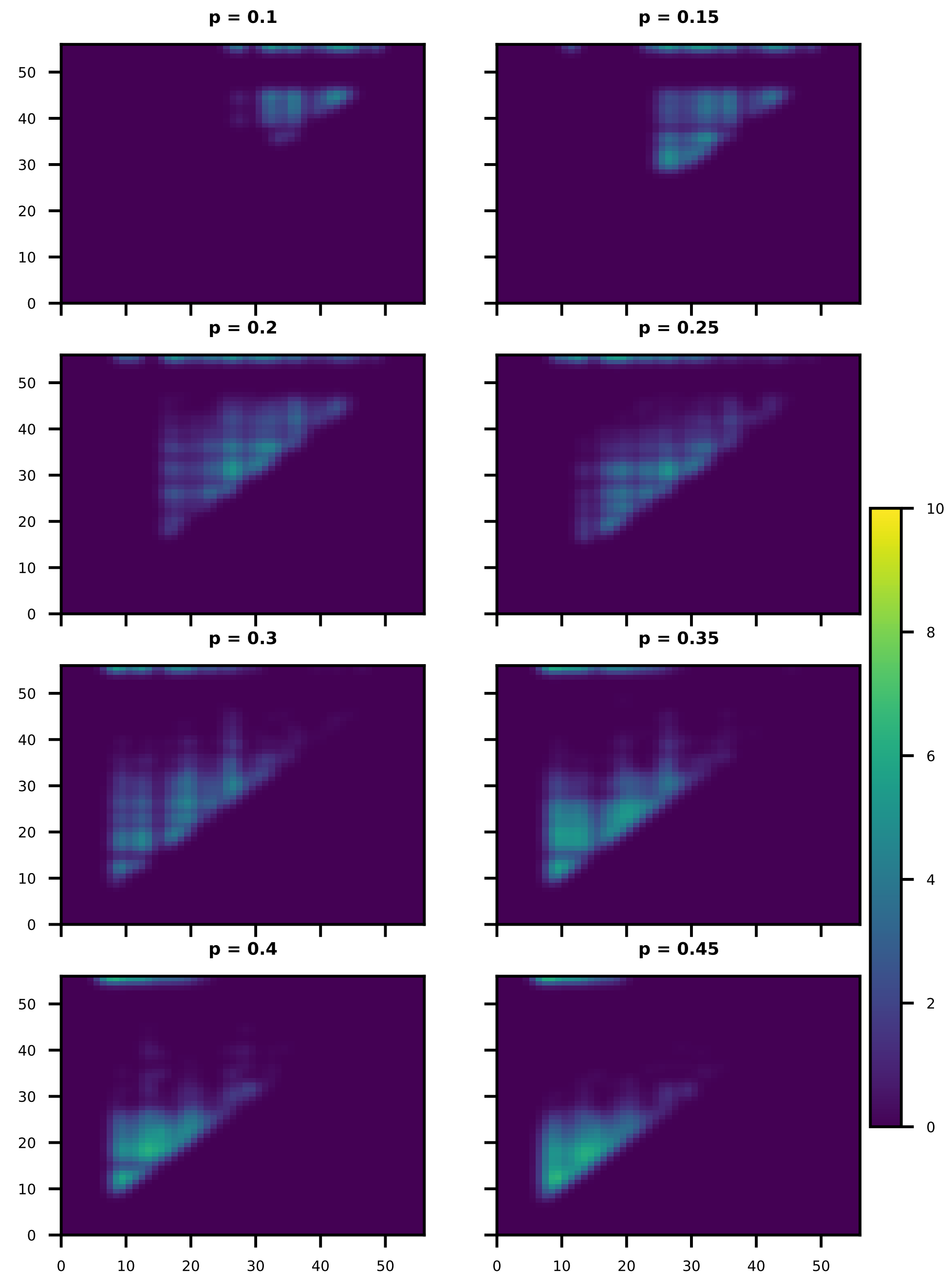


Figure 3.19: The logarithmic heatmap of the distribution of cornerpoints for a random graph with 30 vertices

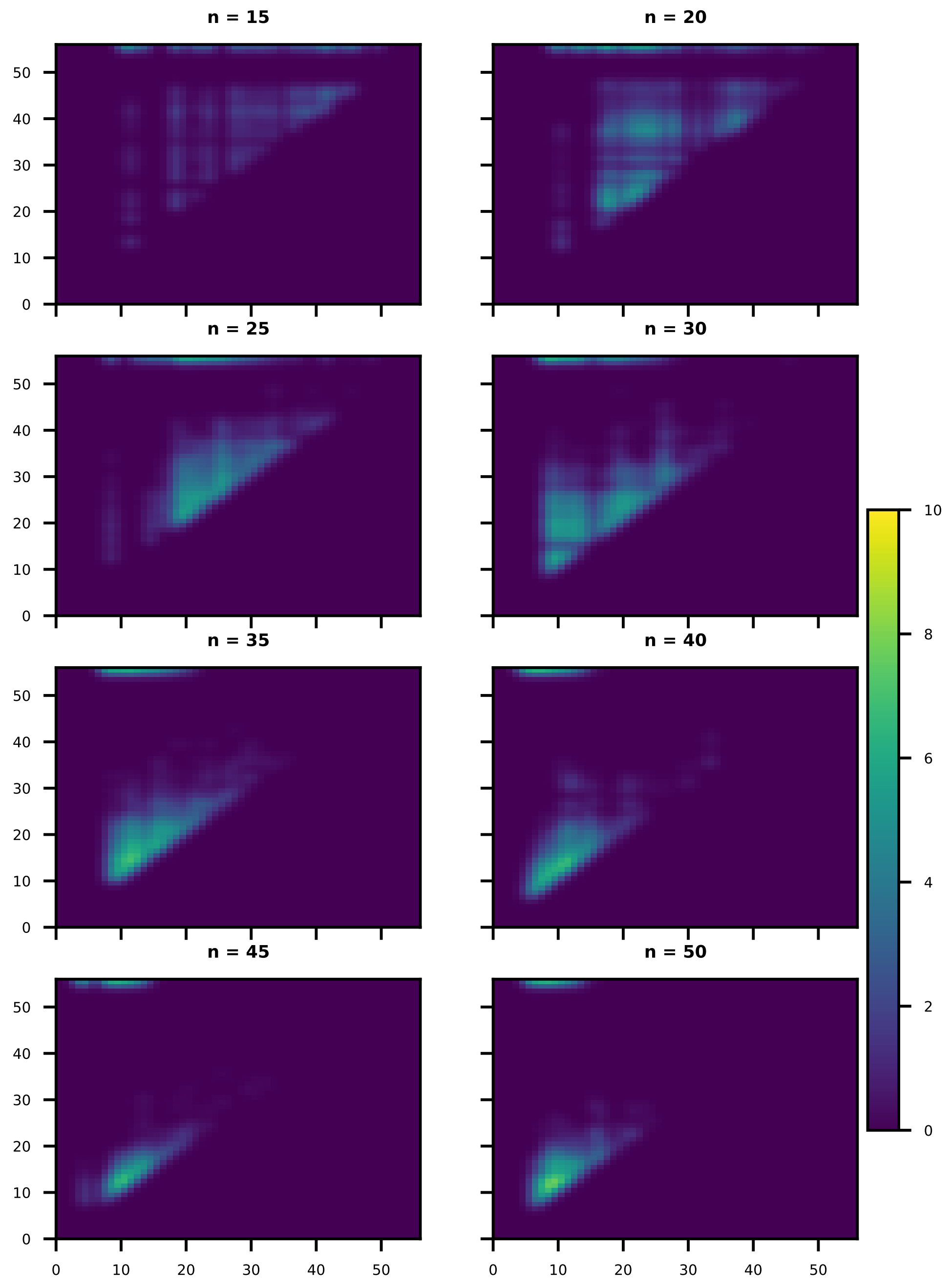


Figure 3.20: The logarithmic heatmap of the distribution of cornerpoints for a random graph with 30 vertices

3.5 Conclusions

The generalized persistence framework introduced in [3] allows us to analyse the persistence of certain features directly, without the need of building auxiliary transformations which map data points to topological spaces. In this chapter, we studied how the category of graphs, **Graphs**, behaves with such generalization, both as source and target category. In 3.1 we recalled some basic notions of graph theory. In section 3.2, after noticing that **Graphs** is a regular category, we proved that many of the classical graphs invariants do not satisfy the conditions for being rank functions. The only exceptions we found are the size and the order functions. In section 3.3, we extended the notions of connected components, blocks and edge-blocks to directed graphs and studied how these behaves along a given filtration. The last section is devoted to the analysis of how different orientations can affect the persistence diagrams obtained studying the strong connectivity of the graph. By analysing many orientations it is possible to obtain a distribution of cornerpoints. The experiments performed considered both deterministic and random graphs, providing an empirical analysis of the behaviour of the distribution as we change the number of vertices or the probability of appearance of an edge.

In future works, we will try and extend these last studies about cornerpoint distributions, trying to capture additional information concerning the connectivity of the underlying graph, for instance by identifying communities. Moreover, it would be interesting to bring more examples of rank functions on **Graphs** and other categories, as **Set** or **Grp**.

Persistence pooling 4

Deep learning is a branch of machine learning which has proven its effectiveness in the last 15 years, becoming a central tool in many fields. Despite its recent development, the first model of a neural network, the perceptron, was introduced in the 40s, [23, 24]. In order to reach high accuracy, it is necessary to use large networks, and, since the training of such networks is computationally expensive, the research in this field was almost abandoned. The recent advances in hardware technology, as the development of faster CPUs and the advent of GPUs, overcame the computational cost issues and gave a boost to the research and applications in deep learning. Whereas the first perceptrons were used just in binary classification tasks, nowadays applications spread in many different fields, for examples image processing, [25–27], medical image analysis, [28], speech analysis, [29, 30], finance, [31, 32]. In section 4.1 we recall basic notions of deep learning. In section 4.2, we extend the notions of steady and ranging sets introduced in [5] to the category of **Set**. In section 4.3, a novel operator for the analysis of images is introduced, and it is used in section 4.4 to define a novel pooling layer. In section 4.5, we present some experiments showing the performance of this new layer in a classification task and comparing it to other state-of-the-art pooling layers. For some reference about this topic, the reader can refer to [33, 34].

4.1 Background

Deep learning

The goal of deep learning algorithms is to approximate some function $f^* : \mathbb{R}^n \to \mathbb{R}^m$, mapping each input $\mathbf{x}$ in the dataset to the corresponding output $\mathbf{y}^*$. The operations performed by the network can be summarized as a function $\mathbf{y} = f(\mathbf{x}, \theta)$ depending on the parameter set θ. The goal of a deep learning algorithm is to find the optimal parameter set θ that best approximate f^*.

Neural networks use perceptrons, or neurons as basic units. For this reason, they are also called multilayer perceptrons. A neuron is the composition of an affine transformation and a nonlinearity. The affine transformation is performed applying a weights vector $\mathbf{w} \in \mathbb{R}^n$ and a bias value $b \in \mathbb{R}$ to an input vector $\mathbf{x} \in \mathbb{R}^n$, as $\mathbf{w}^T\mathbf{x} + b$. The final operation providing the neuron output is the nonlinearity, called the activation function . Being the only source of nonlinearity in the model, this is needed in order to approximate nonlinear target functions f^*. The output of the neuron is then $\mathbf{y} = \phi(\mathbf{w}^T\mathbf{x} + b)$, where ϕ is the activation function. Although many activations have been introduced, the sigmoid and the rectified linear unit (ReLU) are the most used. They are respectively defined as

Activation function

$$\phi(x) = \frac{1}{1 + e^{-x}},$$

$$\phi(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0. \end{cases}$$

Feedforward deep neural networks, in their simplest formulation, are built as a sequence of fully connected layers, see figure 4.1. Thus a neural network can be seen classically as a composition of functions $f_n \circ f_{n-1} \circ \cdots \circ f_0$, where f_0 is the input function and f_n provide the final output of the model $\mathbf{y}$. To each f_i corresponds a layer. Thus the operations performed by f_i can be written as:

$$\mathbf{x}^{(i+1)} = \phi(\mathbf{W}^i\mathbf{x}^i + \mathbf{b}^i)$$

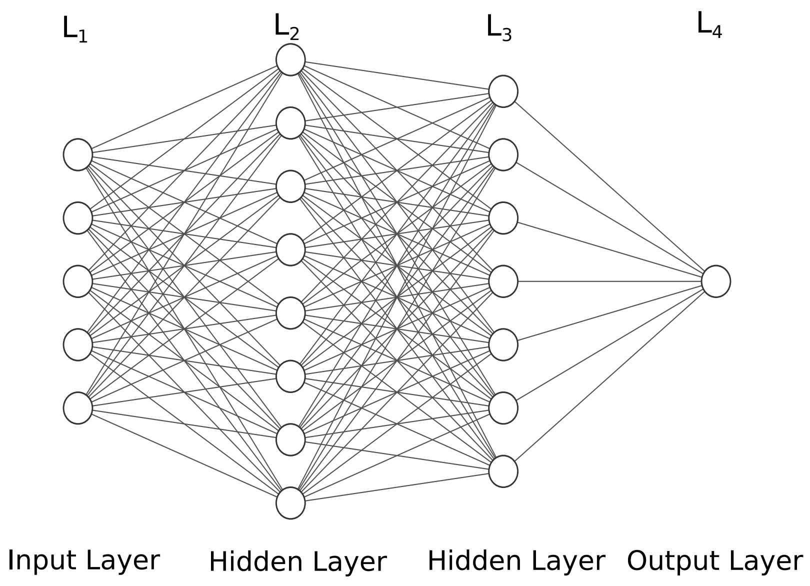


Figure 4.1: A simple example of multilayer perceptron with two hidden layers.

where ϕ is the activation function, usually chosen to be one of the previous one-dimensional functions applied component-wise, $\mathbf{W}^i$ and $\mathbf{b}^i$ are respectively the weights matrix and the bias vector. The j-th row of the matrix $\mathbf{W}^i$ and the j-th element of the bias vector $\mathbf{b}^i$ are respectively the weights vector $\mathbf{w}^i_j$ and the bias $\mathbf{b}^i_j$ of the j-th neuron of the i-th layer.

Following the notation given in the beginning of this section, $\mathbf{y} = f(\mathbf{x}, \theta)$, the entire network is denoted as a function f mapping the input $\mathbf{x}$ to the predicted output $\mathbf{y}$, whereas the set of parameters θ contains all the weight matrices $\mathbf{W}^l$'s and all the bias vectors $\mathbf{b}^l$'s. The learning procedure consists in the optimization of the introduced set of parameters, aiming to the best possible approximation of a target function f^*. To evaluate the performances of the model it is necessary to introduce a cost function, or loss function , $\mathscr{L}$, which measures the distance between the predicted and expected outputs. The optimization of the parameter set is then performed by finding the ones that minimize the loss function:

Loss function

$$\theta^* = \arg\min_\theta \left\{ \mathscr{L}(f(\mathbf{x}, \theta), \mathbf{y}^*) \right\}.$$

Usual choices for $\mathscr{L}$ are the mean square error and the cross entropy. The first one computes the square distance between the predicted and the target values

Mean square error

$$\mathscr{L}(\mathbf{y}) = \frac{||\mathbf{y} - \mathbf{y}^*||^2}{|\mathbf{y}|}.$$

Cross entropy function The cross entropy function , instead, considers the target values $\mathbf{y}^*$ and the predicted values $\mathbf{y}$ as probability distributions over the set of possible outcomes of the model, and define the distance between them as

$$\mathscr{L}(\mathbf{y}) = -\sum_k \mathbf{y}_k^* \ln(\mathbf{y}_k).$$

The minimization of such functions, with respect to the weights of the model, can be performed using the gradient descent method, [35]. The idea is that the gradient computed in a point provides the local direction of the steepest ascent. Then, iteratively computing the gradient and making a step in the opposite direction, should return a path for the minimum. To compute the gradient with respect to all the weights in a neural network, a fast algorithm, based on the chain rule of calculus and called *back-propagation* , was introduced, [36]. The so called forward pass gives the output of each layer, providing the final estimate for the given input, while the backward pass propagates the gradient back from the loss function to the input layer. The forward step can be written as

Back-propagation

$$s_j^l = \sum_i w_{ij}^l x_i^{(l-1)} - b_j^l, \qquad x_j^l = \phi(s_j^l)$$

where we distinguish between the output of the neuron before the activation function s_j^l and after the activation function x_j^l. With this notation, the backward step becomes

$$\frac{\partial \mathscr{L}}{\partial w_{ij}^l} = \frac{\partial \mathscr{L}}{\partial s_j^l}\frac{\partial s_j^l}{\partial w_{ij}^l} = \frac{\partial \mathscr{L}}{\partial s_j^l} x_i^{(l-1)}$$

and a similar equation holds for the biases. Let us call $\delta_i^l = (\partial \mathscr{L})/(\partial s_j^l)$. From [34] pag. 173, we have

$$\delta_i^{(l-1)} = \phi\left(s_i^{(l-1)}\right) \sum_j \delta_j^l w_{ij}^l.$$

Thus it is possible to compute the gradient for the weights belonging to the l-th layer knowing only the input vector $\mathbf{x}^{(l-1)}$ and the already computed δ's from the $(l+1)$-layer.

Convolutional neural networks

According to the fully-connected approach presented in the previous section, data containing spatial information, e.g. images, have to be vectorized in order to be considered a valid input for the first layer. This procedure leads to the loss of spatial information, forgetting, for example, which pixels were close to each other in the original image. Convolutional neural networks (CNNs) overcome this issue, among others, since they use as input the original data without transformations, becoming the perfect tool for analyzing grid-like data as images.

In practice, an input image can be represented as a three-dimensional matrix $X = \{x_{ijk}\}$, whose shape $H_X \times J_X \times K_X$ refers to the width, height and number of channels of the image. For example, grayscale images have only one channel, while RGB images have three. For simplicity, let us focus on the case $K_X = 1$. Then, a filter is two-dimensional and can be represented as a matrix $w = \{w_{ij}\}$ of size $H_w \times J_w$. The shape of the output of the convolutional layer is $(H_X - H_w + 1) \times (J_X - J_w + 1)$. The output Y is computed as

$$Y_{ij} = \sum_{p=0}^{H_w-1} \sum_{r=0}^{J_w-1} X_{i+p,j+r} w_{pr} + b \tag{4.1}$$

where b is a learnable bias. This operation is a cross-correlation and not a real convolution, since, in the latter case, the filter should be flipped. However, since in deep learning applications the filter is learnt by the model, these two operations are equivalent. For this reason, networks relying on this operation are called convolutional. Figure 4.2 depicts an example of convolution on a two-dimensional image.

$$\underset{\text{Original image}}{\begin{array}{|c|c|c|c|c|c|c|}\hline 1&3&2&3&4&8&5\\\hline 4&5&0&2&1&0&4\\\hline 1&2&2&5&3&5&4\\\hline 1&6&1&0&7&4&4\\\hline 7&2&1&0&6&4&1\\\hline 4&0&4&5&2&0&1\\\hline 1&3&0&7&5&7&5\\\hline\end{array}} * \underset{\text{Filter}}{\begin{array}{|c|c|c|}\hline 1&0&-1\\\hline 1&0&-1\\\hline 1&0&-1\\\hline\end{array}} = \underset{\text{Output}}{\begin{array}{|c|c|c|c|c|}\hline 2&0&-4&-3&-5\\\hline 3&6&-8&2&-1\\\hline 5&5&-7&-8&2\\\hline 6&3&-4&-3&4\\\hline 7&-7&-3&1&1\\\hline\end{array}}$$

Figure 4.2: A simple example of how convolutional filters work in the two dimensional case.

The case with $K_X = 1$ just presented deals with a simplified scenario in which both the input and the output are two dimensional. In the extension of this framework to the general case, i.e. when more input and output channels are required, each entry in the output image will be generated by considering all the input channels. For this reason, the filter w has to be of shape $H_w \times J_w \times K_X \times K_Y$, where K_X and K_Y are the number of the input channels and output channels, respectively. Then, the output is computed as

$$Y_{ijk} = \sum_{q=1}^{K_X} \sum_{p=0}^{H_w-1} \sum_{r=0}^{J_w-1} X_{i+p,j+r,q} w_{prqk} + b_k$$

where b_k is the bias corresponding to the $k-$th output channel.

In many cases, convolutional layers have to preserve the input size. In order to achieve this task, a frame around the image, called padding, is added. In most of the cases, the padding is composed of zeros or obtained by replicating the boundary pixels of the image.

Convolutional neural networks are composed by a sequence of convolutional and pooling layers (see section 4.1) and a final part consisting of fully-connected layers. A flattening layer reshapes the feature maps into a unique column, connecting the convolutional and pooling layers to the fully-connected ones.

Pooling

In CNNs pooling layers are used to downsample the information provided by convolutional layers. Pooling layer first subdivides of the input data in patches and then assigns to each patch of the best representative value through a pooling function $\mathcal{P}$. Patches subdivision is usually performed by considering each channel individually as a two-dimensional signal; as a consequence, each patch will be referred to as a two-dimensional object. Given a filter K of fixed size (k_1, k_2), each patch is obtained by sliding K by some integer multiples of each of the so-called stride values s_1, s_2 along the respective direction. The patch P_{ij} mapped to the entry (i, j) of the output is:

$$P_{ij} = I[(s_1 i) : (s_1 i + k_1), (s_2 j) : (s_2 j + k_2)].$$

After the subdivision in patches, each patch P_{ij} is mapped through $\mathcal{P}$ to the final output. The most used pooling functions are the maximum and the average.

The use of pooling layers in CNNs has multiple effects. For example, data size reduction reflects on the computation timing, since the smaller the data, the faster the processing, and it also enlarges receptive fields, i.e. the region of the input space used by a particular CNN feature. Moreover, it provides some stability to minor deformations, see [27], even if some debate about it is still ongoing [37].

On the other side, the main drawback of downsampling, and thus of pooling layers, is the loss of information. The reduction of spatial resolution in pooling layers leads to the inevitable loss of details. Another drawback regarding the most used pooling layer, i.e. max pooling, is that it is not trainable. This implies that the information passed by the pooling is not optimized in order to minimize the training error. Moreover, as highlighted by Hinton:

> *The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster. If the pools (i.e. patches) do not overlap, pooling loses valuable information about where things are.*

Different approaches have been proposed, overcoming such problems. In [38], the authors introduced a stochastic pooling operator. They divided the input into patches, and for each patch, computed a probability distribution based on the values of the patch . The final output is a pixel chosen randomly according to such a distribution. Another stochastic approach, called fractional max pooling, has been introduced in [39]: through a random selection of the size of the patches, the spatial reduction of the input data can be fractional.

The methods just presented do not contain learnable parameters, thus they cannot be optimized to minimize the loss function. Another approach proposed is the introduction of learnable parameters in the pooling, which allow learning optimal downsampling [40]. The method proposed in [40] consists of applying a standard convolutional layer with a stride larger than 1. In this way, the output size will be smaller than the input. As explained in section 4.1, convolutional layers require the introduction of four-dimensional matrices of weights, which considerably increase the number of parameters of the network. To overcome such an issue, in [41], the authors proposed the LEAP pooling layer. For each channel in input, a two-dimensional convolutional filter is introduced. Similarly to the case presented in equation 4.1, the output is obtained by convolving each patch with the filter. For computing each feature channel, the LEAP operator uses only the information from the corresponding channel. Thus, this approach introduces fewer parameters if compared with the one proposed in [40], since there are no intra-channel connections.

Persistence in Deep Learning

Flexibility and modularity make of persistence an ideal candidate method to interact with machine learning. In particular, deep learning models can optimise million of parameters to achieve a given task, however understanding which data transformations and features contributed to the minimization of the error function in a given task remains an open problem. A principled approach could provide new insight, as well as a topological approach could allow deep learning to *see* not only local, but also global structures of a given dataset. The interaction between deep learning and persistence theory is on three levels: the usage of persistence to gain mechanistic understanding of deep learning, the usage of deep learning to make persistence *learnable*, and the integration of the two approaches.

Despite providing surprisingly good performances, the complexity of neural network architectures and the huge number of parameters cause a lack of interpretability. A key challenge in the machine learning field is to understand how deep learning works. Among other approaches, researchers used persistence theory to study certain properties of neural networks. One method is to see neural networks as weighted graphs, [42], where the weight of each edge between layer l and $l+1$ is obtained as the activation values of the starting node, x_i^l times the weight w_{ij}^l. The analysis of the obtained persistence diagrams provided not only information about the functionality of the network but also the distance between persistence diagrams allowed them to identify adversarial examples. Another mysterious aspect about deep learning is the fact that non-smooth activation functions, as ReLu, outperform other smooth activations, as sigmoids. In [43], the authors tackle such an issue. Since, in order to correctly classify the input data, the algorithm has to find a way to unfold the input distribution, trying to separate the classes, the authors assumed that the topological complexity of data distribution should decrease, layer by layer. By using persistent homology to compute the topological complexity, they were able to show that non-smooth activations reduce the complexity faster and more efficiently than smooth ones. On the other side, it is possible to use deep learning to estimate persistence homology outputs. For example it is possible to train neural networks to estimate Betti numbers, [44], persistent images, [45] given a data sample.

The last approach is to integrate persistence theory and deep learning. The topological and geometrical information provided by the persistence analysis can be fed to neural networks layers. Persistence information is usually encoded in persistence diagrams (PDs) or equivalent representation systems. The lack of structure and well-defined basic operations in the space of persistence diagrams makes it impossible to use PDs in machine learning pipelines directly. One possible solution is to vectorise the information provided by a PD and feed such vector to the neural network, [46], or it is possible to use a learnable function to reduce the information carried by a PD to a unique value, [47]. On the other side, persistence landscapes provide an efficient way to encode persistence information in vector spaces of functions, [48]. A weighted average of persistence landscapes can be used to define a vectorization of persistence information that can be fed to a neural network layer, [49].

Loss functions measure the distance between the obtained output and the expected output. Different loss functions focus on different aspects of the output. Persistence can be used to define loss functions measuring the distance between the topology of the expected output and the obtained output, and thus optimise the network parameters in order to preserve topological properties, for example, measuring the distance between PD [50, 51], or to penalise topological complexity of the output, [52].

4.2 Persistence on Set

In this section, we adapt the framework defined in [5] on **Graph** to the category **Set**. We will denote as **Set** a subcategory of the usual set category, in which the only allowed morphisms are monomorphisms. With this choice, all the $(\mathbb{R}, \leq)$-indexed diagrams correspond, up to isomorphisms, to filtrations of sets. Given a set X, a filtration of subsets can be defined as the sublevel set of a filtering function $f : X \to \mathbb{R}$. We will denote the pair set, filtering function as (X, f). Notice that such filtrations are monic diagrams, introduced in definition 2.2.4.

Generalized persistence functions, gp-function generator

Definition 4.2.1 *Assume that a correspondence p is given, which assigns to each monic $(\mathbb{R}, \leq)$-indexed diagram F in a category* **C** *a categorical persistence function p_F on $(\mathbb{R}, \leq)$, such that $p_F = p_{F'}$ for F' naturally isomorphic to F. All the resulting categorical persistence functions p_F are called* generalised persistence functions *(or shortly gp-functions) in* **C***. The map p is called a* gp-function generator.

Let us consider a feature map $F : 2^X \longrightarrow \{true, false\}$ defined for all sets X. We say that $Y \subset X$ is an F-set if $F(Y) = true$. Denote with X_u the sublevel set $f^{-1}(-\infty, u]$. We say that $Y \subset X$ is an F-set at level w if it is an F-set of X_w.

Steady set, ranging set

Definition 4.2.2 *Call $Y \subset X$ a* steady *F-set (or simply an s-F-set) at (u, v) ($(u, v) \in \Delta^+$) if it is an F-set for all levels w such that $u \leq w \leq v$. We call X a* ranging *F-set (or simply an r-F-set) at (u, v) if there exist levels $w \leq u$ and $w' \geq v$ at which it is an F-set.*

Let $SF_{(X,f)}(u, v)$ be the set of s-F-sets at (u, v) and let $RF_{(X,f)}(u, v)$ be the set of r-F-sets at (u, v).

Let F be a feature defined on every set X. Consider the pair (X, f) composed by a set X and a filtering function f. We will say that the triplet (X, f, F) is admissible if both $|SF_{(X,f)}(u, v)|$ and $|RF_{(X,f)}(u, v)|$ are finite for all $(u, v) \in \Delta^+$. This condition is obviously satisfied by triplets containing finite sets.

Proposition 4.2.1 *Consider an admissible triplet (X, f, F). Then the functions $\sigma_{(X,f)}$ which assigns to $(u, v) \in \Delta^+$ the number $|SF_{(X,f)}(u, v)|$ and $\varrho_{(X,f)}$ which assigns to $(u, v) \in \Delta^+$ the number $|RF_{(X,f)}(u, v)|$ are generalized persistence functions.*

The proof works similarly to [[5], Prop 1,2].

As gp-functions are not always stable, we introduce a condition that guarantees stability (see theorem 4.2.2).

Balanced gp-function

Definition 4.2.3 *Let p be a gp-function generator in* **Set***. Then the map p and the resulting gp-functions are said to be* balanced *if, for any two filtered sets (X', f') and (X'', f'') with associated gp-functions $p_{(X',f')}, p_{(X'',f'')}$, the following condition holds. If an isomorphism $\psi : X' \longrightarrow X''$ exists such that* $\sup_{x \in X'} |f'(x) - f''(\psi(x))| \leq h$, $h > 0$, *then for all $(u, v) \in \Delta^+$ the inequality $p_{(X',f')}(u - h, v + h) \leq p_{(X'',f'')}(u, v)$ holds.*

Let (X', f'), (X'', f'') be as above. Consider also the set H of all possible isomorphisms between X' and X''. We can define the following distance among filtered sets.

Definition 4.2.4 *The* natural pseudodistance *of* (X', f') *and* (X'', f'') *is*

Natural pseudodistance

$$\delta\big((X', f'), (X'', f'')\big) = \begin{cases} +\infty & \text{if } \; H = \emptyset \\ \inf_{\phi \in H} \sup_{x \in X'} |f(x) - g(\phi(x))| & \text{otherwise} \end{cases}$$

As in [5], it is possible to prove the following stability result.

Theorem 4.2.2 (Stability) *Let p be a balanced gp-functions generator in* ***Set*** *and $(X', f'), (X'', f'')$ be any two filtered sets; then*

$$d\big(D(f'), D(f'')\big) \leq \delta\big((X', f'), (X'', f'')\big)$$

where d is the bottleneck distance defined in definition 2.2.7 and D are the persistence diagrams associated to the gp-functions $p_{(X', f')}$ and $p_{(X'', f'')}$.

4.3 Image operator

In this section, we present a novel operator for the analysis of images based on the concept of steady sets, see definition 4.2.2. We will use this operator in section 4.4 to define a novel pooling layer. Let us consider a grayscale image I. This image can be represented as a $\mathbb{R}^{m \times n}$ matrix, whose entries correspond to the pixel values. Pixel values naturally define a filtration on the image.

This operator is based on the notion of steady set introduced in definition 4.2.2. The feature map used in such operator deals with each pixel individually. Since images are finite sets, every feature map we choose leads to an admissible triplet.

Definition 4.3.1 *Let $k \in \mathbb{N}$. We define the neighbour set of a pixel $\boldsymbol{x} = (x, y) \in I$ of size k to be*

Neighbour set

$$N_k(\boldsymbol{x}) = \{\boldsymbol{x}' = (x', y') : x' = x + s, y' = y + t, s, t \in [-k, k]\}$$

Definition 4.3.2 *Let $m, n, k \in \mathbb{N}$ be such that $|N_k(\boldsymbol{x})| \geq n > m$ for any pixel $\boldsymbol{x} \in I$. We say that a pixel $\boldsymbol{x} \in I$ is* active *at level $l \in \mathbb{R}$ if the following conditions are satisfied:*

1. $|N_k(\boldsymbol{x}) \cap f^{-1}([-\infty, l])| \geq m$
2. $|N_k(\boldsymbol{x}) \cap f^{-1}([-\infty, l])| \leq n.$

This operator considers a pixel active at time t, with respect to the filtration induced by the pixels values, if at least m and less than n pixels in its neighbour set have a value smaller than t. Thus a single pixel can be activated only one time along the filtration.

Given an image I, the function σ_I that associates to the pair (u, v) the number of pixels that are steady sets at (u, v) is a generalized persistence function. Moreover,

Theorem 4.3.1 *The function σ_I is balanced.*

In this framework, we consider as isomorphic only images with the same number of rows and columns. An isomorphism $\psi : I \longrightarrow J$ associates to each entry (i, j) in the image I the entry (i, j) in J.

Proof. Consider two isomorphic images I and J and the corresponding filtering functions f_I and f_J such that $\sup_{\boldsymbol{x}} |f_I(\boldsymbol{x}) - f_J(\boldsymbol{x})| \leq h$ for $h > 0$. Consider a pixel $\boldsymbol{x}$ that is active in J between levels $u - h$ and $v + h$. This is equivalent to say that $|N_k(\boldsymbol{x}) \cap f_J^{-1}([-\infty, u - h])| \geq m$ and $|N_k(\boldsymbol{x}) \cap f_J^{-1}([-\infty, v + h)]| \leq n$. We need to show that $|N_k(\boldsymbol{x}) \cap f_I^{-1}([-\infty, u])| \geq m$ and $|N_k(\boldsymbol{x}) \cap f_I^{-1}([-\infty, v])| \leq n$. Let $\boldsymbol{y} \in N_k(\boldsymbol{x})$ be a pixel such that $\boldsymbol{y} \in f_J^{-1}([-\infty, u - h])$. Since $|f_I(\boldsymbol{y}) - f_J(\boldsymbol{y})| \leq h$ we have that $\boldsymbol{y} \in f_I^{-1}([-\infty, u])$, and $\boldsymbol{y} \in f_I^{-1}([-\infty, u'])$ for $u' \geq u$. For the same reason if $\boldsymbol{z} \in N_k(\boldsymbol{x})$ is such that $\boldsymbol{z} \notin f_J^{-1}([-\infty, v + h])$, then $\boldsymbol{z} \notin f_I^{-1}([-\infty, v])$, and $\boldsymbol{z} \notin f_I^{-1}([-\infty, v'])$ for $v' \leq v$. So all the pixels $\boldsymbol{y} \in f_J^{-1}([-\infty, u - h])$ are also in $f_I^{-1}([-\infty, u])$ and all the pixels $\boldsymbol{z} \notin f_J^{-1}([-\infty, v + h])$ do not belong to $f_I^{-1}([-\infty, v])$. Thus $|N_k(\boldsymbol{x}) \cap f_I^{-1}([-\infty, u])| \geq |N_k(\boldsymbol{x}) \cap f_J^{-1}([-\infty, u - h])| \geq m$ and $|N_k(\boldsymbol{x}) \cap f_I^{-1}([-\infty, v])| \leq |N_k(\boldsymbol{x}) \cap f_J^{-1}([-\infty, v + h])| \leq n$, and the pixel $\boldsymbol{x}$ is active in I for all the levels between u and v. □

As an image can be thought as the grid-like sampling of a smooth function $f : \mathbb{R}^2 \longrightarrow \mathbb{R}$, it is possible to consider its gradient, which represents the local directional change of intensity in the image. Thus, it is possible to use the norm of the gradient to identify the borders of objects inside an image. The proposed strategy behaves similarly to the gradient, enhancing the signal of the pixels which are close to a discontinuity in the image.

For example, consider a discontinuity inside the image. A discontinuity can be defined as a big change of intensity in the image, and thus there will be two regions, one with low and one with high intensity. Consider now a pixel $\boldsymbol{x}$ close to such discontinuity. Then, the neighbour set $N_k(\boldsymbol{x})$ contains pixels from both regions. Assume that the number of pixels from the lower region that belong to $N_k(\boldsymbol{x})$ is m_1. By setting $m < m_1 < n$, the pixel **x** will be active for a time that is proportional to the change of intensity, and the value associated to x by our operator will be high.

Figures 4.3, 4.4 depict examples of how such a operator behaves on images. There are some effects to notice, related to the choice of the parameters k, m, n. Let us start considering the effect of the choice of the filter size k. Consider a discontinuity inside an image. To detect such a discontinuity, the neighbour set of the pixel have to contain at least $m_1 > m$ pixels from the lower intensity and at least $n_1 > n$ pixels from the higher intensity. Thus, to activate and deactivate a pixel close to this discontinuity, the size of the filter has to be large enough to include enough pixels from both intensities. For this reason, as we can see in figure 4.4, as the parameter k increases, the borders detected are thicker and the details blurrier. On the other side, the choice of the parameters m and n reflects on the kind of features we can detect. For example, by choosing m and n too small or too high we may identify only the corners of objects, while by setting them to be near $K/2$, we may loose certain certain angular borders, see panels a-d) in figure 4.3. This shows how this filter depends on the ratio between the number of high and low intensity pixels m_1 and n_1. One way to overcome such an effect can be to choose the parameters m and n to be far apart and center them around the value $K/2$, as in figure 4.4. Notice, moreover, that as m and n get closer, the details preserved are lesser.

In figure 4.3, panels e-h) we report the effects of the filter when in presence of different intensities in a controlled environment. We normalized the image, in such a way that the brighter square has intensity 1, the darker square has intensity 0.5 and the background 0. The edges found by the filter have different lifetimes and thus different intensities. Notice that the whole border of the darker square have the same gray level, despite the fact that one part is shared with the brighter square and the other is shared with the black background. This phenomenon is due to the fact that the difference between intensities, and

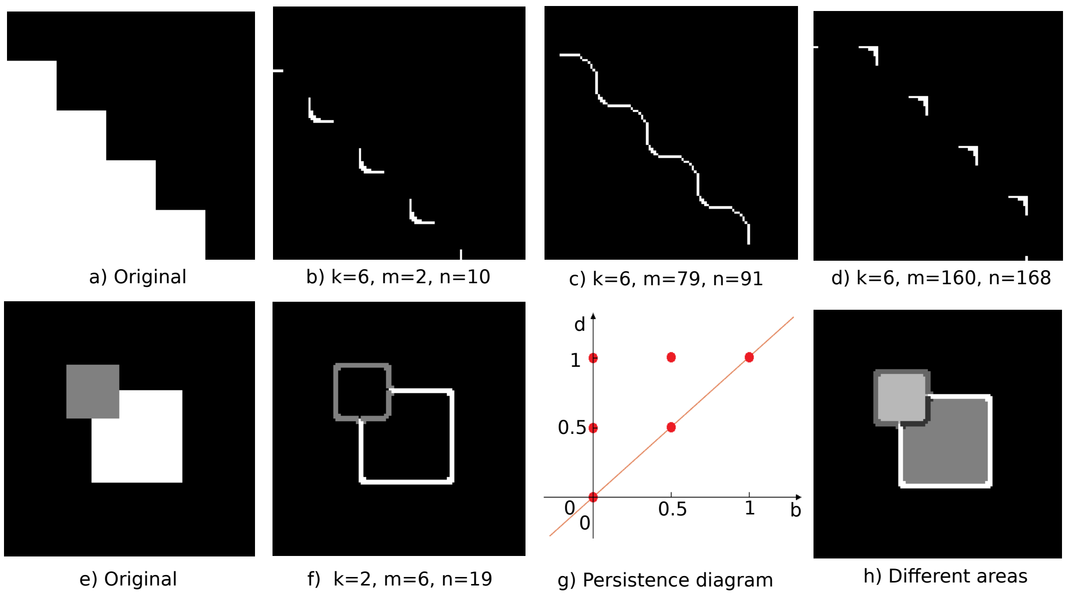


Figure 4.3: The effects of the operator introduced in definition 4.3.2 in some controlled scenarios. Panels a-d) show the effects of different choices of the parameters m, n, and the dependence of the filter on the ratio between the number of high and low intensity pixels m_1 and n_1. Recall that to activate a pixel we need $m < m_1 < n$. If we choose m and n to be small, we need a small m_1 to activate a pixel, and we can detect only edges with unbalanced ratios as corners, b). As a reflection of this scenario, the same happens with high values of m and n, see panel d). Setting m and n close to $K/2$ we detect edges where the ratio is close to 1, for example flat borders, while the corners are not detected, c). The second row depicts an example of how lifetime may not be sufficient to encode all the information provided by the filter. Let us consider two objects S_1 of intensity 0.5 and S_2 of intensity 1 overlapping inside an image, e). Although lifetimes allow to identify the edges of the squares, f), they fail to distinguish all the 6 regions identified by the persistence diagram, g). By considering also the birth time in the analysis it is possible to overcome this issue, h).

thus the lifetime, is equal to 0.5 in both cases. It is still possible to distinguish the difference between the two parts by considering not only the lifetime, but also the birth timing. As highlighted by the persistence diagram, panel o), the filter can identify six regions, and not only three as the lifetime does. Considering also the birth times it is possible to recover all this information, obtaining the partition depicted in panel h).

Figure 4.5 compares the proposed method with other state-of-the-art edge detection methods, namely Canny and Sobel edge detection, in presence of salt and pepper noise. The experiments were performed on the noisy image and on the image preprocessed using a median filter. Panel a) reports the performances in terms of peak signal-to-noise ratio (PSNR) and mean squared error (MSE) of the different algorithms. Panel b) and c) show some sample images from the two experiments. In the first experiment we wanted to test the stability of the algorithms in presence of noise. We used each algorithm to identify the edges of the original Lena image, obtaining for each of them its

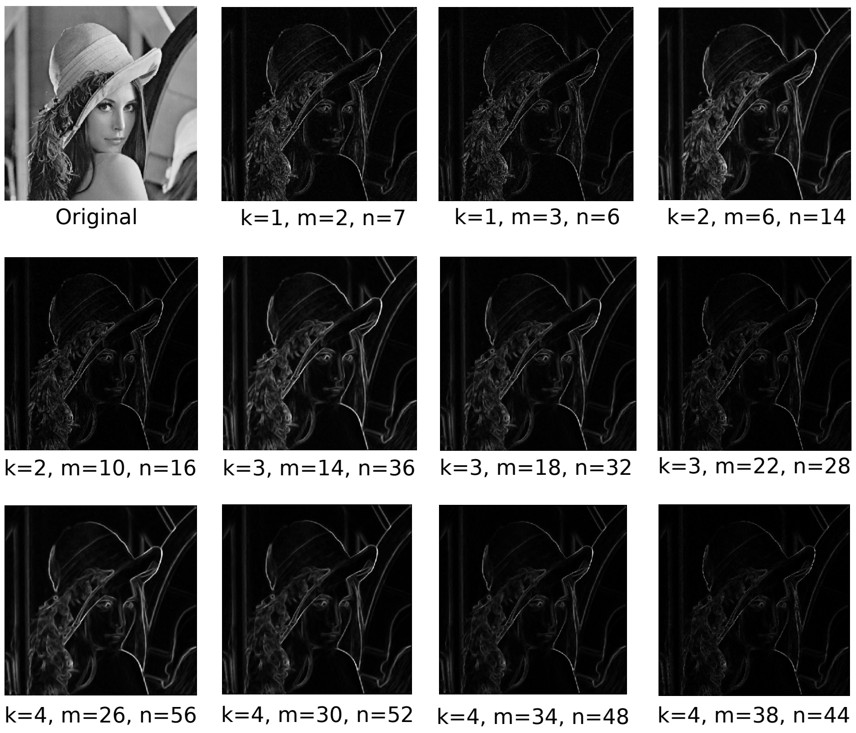


Figure 4.4: The effect of the operator introduced in definition 4.3.2 on the Lena image.

own ground truth (GT), reported in the first row of panel b). We added salt and pepper noise to the Lena image, with noise levels between 5% and 50%. For each noise level we compared the edges found by each algorithm with the edges detected on the original image, computing PSNR and MSE. The second and third rows in panel b) show the results when in presence of noise levels 20% and 40%. From panel b) it is possible to notice that the proposed method guarantees higher stability to noise, also without preprocessing. This empirical analysis is supported by the quantitative measures reported in panel a), first row.

In the second experiment we wanted to test not only the stability to noise perturbations, but also the ability to detect the actual edges present in the image. In this experiment we took as ground truth edges the pixels whose gradient was not zero. Panel c) in figure 4.5 shows the results on the different algorithms in presence of noise levels 20% and 40%. As in the previous experiment, Canny and Sobel without preprocessing lead to noisy images, while with some preprocessing it is possible to clearly recover the original edges, despite the presence of some artifacts. Notice that the proposed filter detects the real edges without the need of any preprocessing. The second row of panel a) depicts the quantitative performances of the algorithms, when compared to the GT edges. Notice how, without noise, all the algorithms obtain comparable results, both in terms of PSNR and MSE, while in presence of noise the persistence based methods outperform the competitors.

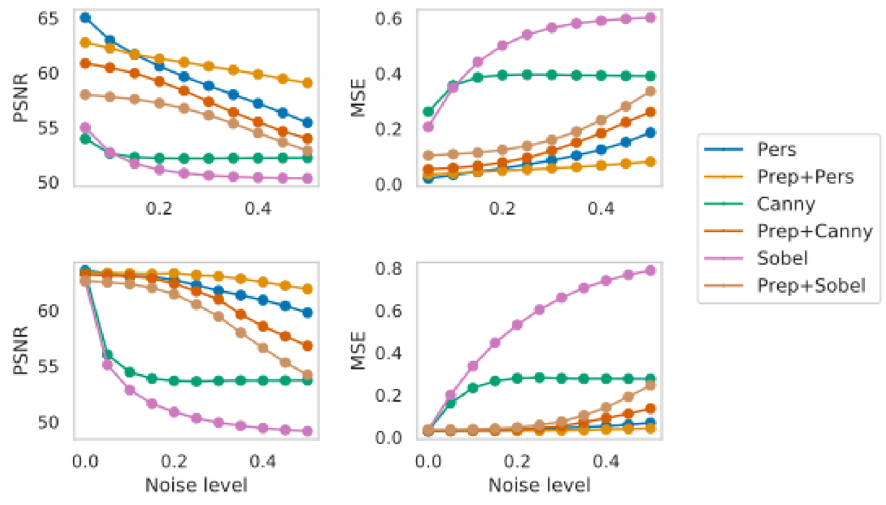
Remark 4.3.1 Let p be the probability of a pixel of being a pepper pixel and q be the probability of being a salt pixel. Suppose, moreover, that the thresholds m and n and the filter size $K = (2k+1)^2$ are fixed. Denote with M and N respectively the number of pepper and salt pixels covered by the filter. Then the probability of $M > m$ is

$$\mathbb{P}(M > m) = 1 - \sum_{i=1}^{m} \binom{K}{i} p^i (1-p)^{K-i}.$$

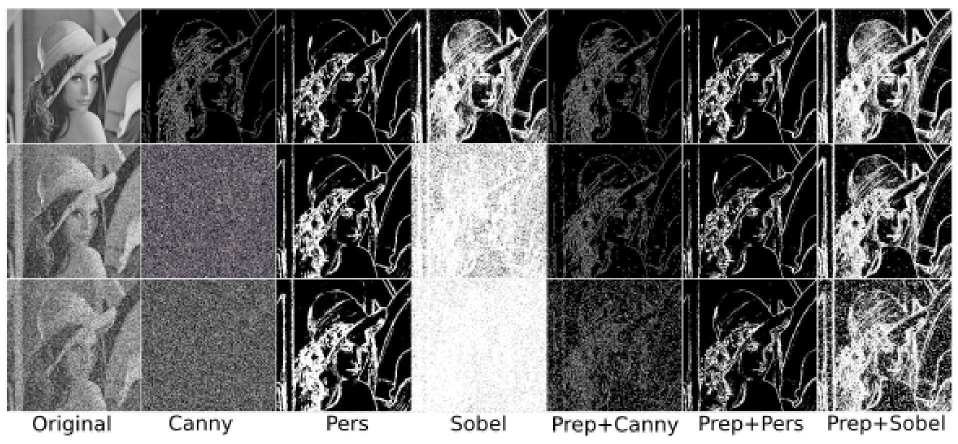
and of $N > K - n$ is

$$\mathbb{P}(N > K - n) = 1 - \sum_{i=1}^{(K-n)} \binom{K}{i} q^i (1-q)^{K-i}.$$

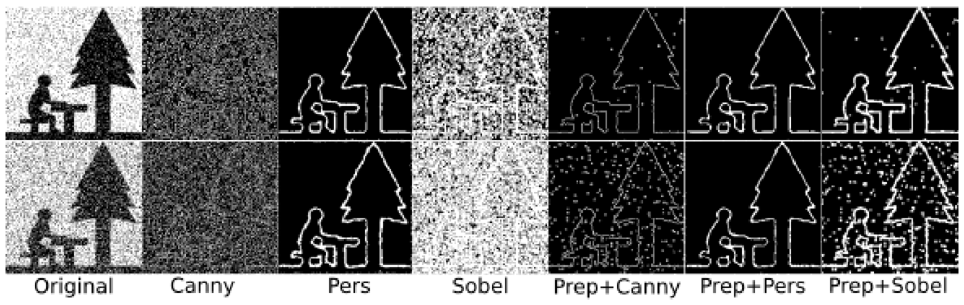
Thus, having an estimate of the noise levels p and q, [53, 54], it is possible to have an estimate of the performances of the operator, and thus to optimize the parameters in order to achieve the desired results.



a)



b)



c)

Figure 4.5: We compare the proposed persistence based filter with Canny and Sobel edge detection algorithms. Panel a) reports a quantitative study of the performances of the different algorithms, both with or without preprocessing through median filter. The first row in panels b) contain the original image and the ground truth images for each algorithm. The second and the third rows in panel b) and the two rows in panel c) show the edges detected in presence of salt and pepper noise, respectively with levels 20% and 40%.

4.4 Persistence pooling

As mentioned above, the most used pooling operators suffer of some drawbacks that may undermine their performances in terms of accuracy and of information preservation. A possible way to overcome such issues is to integrate learnable filters in the pooling operation. This approach has been successfully used in [41], [40], where convolutional filters have been used to reduce the dimension of the input dataset. In this section we present a pooling layer, based on the operator described in section 4.3.

The persistence transform

The main idea of the proposed method is to transform the initial patch by associating a new value to each pixel in the domain. It is necessary to define the hyperparameters, i.e. parameters that are not learnable and thus do not change during the training, $m, n, k \in \mathbb{N}$ following the notation introduced in definition 4.3.2. According to these parameters, the operator can be defined as in definition 4.3.2, and for each pixel activation and deactivation times can be computed. By taking the difference between these two timing, it is possible to compute the persistence of each pixel. The output is obtained by substituting each pixel value with the correspondent persistence.

In order to preserve the size of the original image, it is necessary to define a padding of proper size around the image. This operation is necessary in order to extract the filter of the right size around each pixel, also at the boundary of the image. Algorithm 1 presents the pseudo-code for computing the persistence transform of the input image.

The pooling operator

Input data in CNNs are usually four-dimensional batches of images of fixed size, where each image is a three-dimensional matrix, whose dimensions correspond to height, width and number of channels. Through convolutional operations, the height and width of an image are preserved, thus the input of a pooling layer will be a four-dimensional matrix. Like the max-pooling layer,

Algorithm 1 Persistence transform of an image I

```
1:  Input two dimensional image $I$, parameters $k, m, n$
2:  Output transformed image $T$
3:  $I_{pad}$ = pad_image($I$,$k$) # create a frame of $k$ pixels around $I$
4:  $T$ = zeros(size($I$)) # matrix that will contain the transformed image
5:  for pixel $i, j$ do
6:      $N_{ij}$ = compute the set of neighbors of the pixel $i, j$ in $I$ using $I_{pad}$
7:      $F_{ij}$ = flatten $N_{ij}$
8:      $F_{ij}$ = sort $F_{ij}$
9:      $b = F_{ij}[m]$ # compute the activation time
10:     $d = F_{ij}[n]$ # compute the deactivation time
11:     $T(i, j) = d - b$ # assign the persistence value to the correspondent output
    pixel
12: end for
```

the proposed layer treats each channel individually as a two-dimensional image, not taking advantage of the possible relations between different channels as in [40].

To perform the pooling, we consider a channel of the input image and divide it into patches P_{ij} of size $(x_{\text{patch}}, y_{\text{patch}})$, as in standard pooling layers. Then we associate to each patch P_{ij} its persistence transform, PT_{ij}, as defined in subsection 4.4. As in convolution operations, it is necessary to introduce a set of learnable weights W, of the same size of each PT_{ij}. The $(i, j)-$th pixel of the output image is obtained by using the weights in W to compute the weighted average of PT_{ij}. In practice, for each channel k in the input, a set of weights W_k is needed. The output O is then computed as:

$$O_{kij} = \sum_{rs} PT_{kijrs} W_{krs}.$$

Since the persistence transform has to preserve the size of the input image, it is necessary to add a padding operation. This can be done in different ways. A possible choice is to pad each patch P_{ij} just before computing its persistence transform, as in algorithm 1, obtaining patches of size $(x_{\text{patch}} + 2k, y_{\text{patch}} + 2k)$. In the applications proposed below, we made another choice. We pad the input channel with a frame of size k. Then the patches P_{ij} were directly extracted with shape $(x_{\text{patch}} + 2k, y_{\text{patch}} + 2k)$. In this way, the artificial insertion of pixels affects only the pixels at the boundary of the image, while for the internal ones the information provided is the one of the original image. See algorithm 2.

Algorithm 2 Pooling Layer

```
1: Input three dimensional image $I$, parameters $k, m, n$, weights $W$
2: Output the pooled image $Pool$
3: $I_{pad}$ = pad_image($I$,$k$) # create a frame of $k$ pixels around each channel of $I$
4: $P$ = extract the patches of the image
5: $T$ = compute the persistence transform of each patch
6: $O$ = compute the weighted sum of each transformed patch with weights $W$
```

4.5 Computational experiments

In this section we present some experiments that were performed in order to test the proposed layer, trying to highlight the differences with the already existing methods.

Datasets

The datasets used to perform these experiments are MNIST, [55], Fashion-MNIST, [56], and CIFAR-10, [57]. The MNIST dataset is made of grayscale images of hand-written digits of size $28 \times 28 \times 1$. The dataset is composed of 70000 images, 60000 for training and 10000 for testing. The Fashion-MNIST dataset is made of grayscale images of ten classes of clothes of size $28 \times 28 \times 1$. The dataset is composed of 70000 images, 60000 for training and 10000 for testing. The CIFAR-10 dataset is made of RGB images of ten classes of animals and transportations of size $32 \times 32 \times 3$. The dataset is composed by 60000 images, 50000 for training and 10000 for testing.

Architectures

The architectures chosen for the experiments are simple, but efficient in order to highlight the main differences between the proposed persistence pooling layer and the traditional ones. Figure 4.6 depicts the two architectures used in the experiments with the MNIST and Fashion-MNIST datasets. For the CIFAR-10 experiments the architectures differ only for the height and width of the layers before the flattening.

The first one is a simple architecture, where first the input image is down-sampled via a pooling layer, and then the last two dense layers perform the

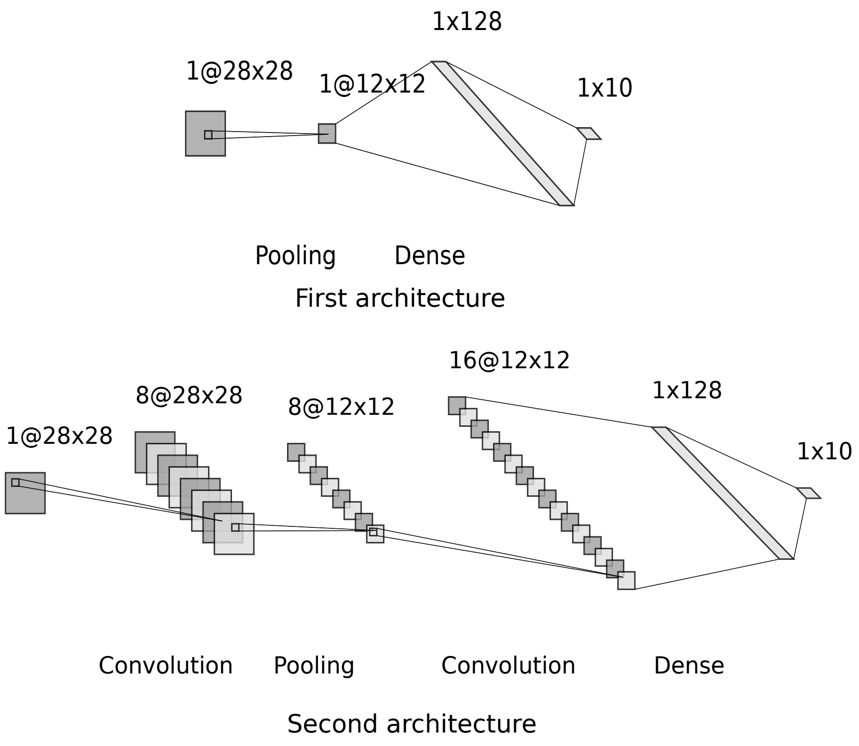


Figure 4.6: The two architecture used in the experiments.

classification. This architecture is used to compare the downsampling performances of the different layers. The second architecture is composed by two convolutional layers separated by a pooling one, and finally two dense layers return the output. This architecture is used to compare the different pooling layers in a more standard scenario, including convolutional layers. Computational limitation dictated the choice of a small model, but such architecture still guarantees a fair comparison between the different pooling layers. The loss function used is the sparse categorical crossentropy provided by the Keras library, see [58]. The networks are trained with batches of size 32 and for the optimization the Adam algorithm has been used, see [59], with learning rate $\lambda = 3e - 4$. The number of epochs is set to 100, but the effective number is smaller due to early stopping, [60], which ends the training if the validation error increase for k consequent steps, in order to avoid overfitting.

Results

In the experiments we tested the two architectures presented in subsection 4.5 on the three datasets presented in 4.5. For each architecture we compared the proposed persistence pooling layer with two states of the art layers: the max pooling and the LEAP defined in [41]. Moreover, we tried to combine the proposed pooling layer with the max-pooling, considering the weighted average of the two layers.

Table 4.1 shows the performances of the aforementioned architectures in each experiment in terms of accuracy, i.e. the ratio between the number of correct guesses and the total number of predictions. The first architecture was developed to highlight how the different layers on the input dataset downsampling task. The absence of convolutional layers penalizes the max-pooling layer, because the downsampling results rough. The performances of the other layers are similar on the Fashion-MNIST dataset, while on the other datasets the proposed layer outperform the others.

The experiments performed on the second architecture shows that the proposed layer perform better than the other state-of-the-art layers, max-pooling and LEAP. Moreover on the MNIST dataset it is the one with the best performances. On the more complex data provided by the Fashion MNIST and the CIFAR10 datasets the combined layer outperform the others, improving the accuracy of the max-pooling layer of 1.40% on the Fashion-MNIST and of 3.44% on the CIFAR10.

Deep learning outstanding results in term of accuracy come at the price of loss of interpretability. The increasing complexity of deep learning models made it almost impossible to understand the relation between inputs and outputs of a neural network. To overcome such an issue, some methods trying to highlight the most relevant features have been proposed. Grad-CAMs, specifically introduced for CNN, use the gradient information flowing into the last layer of the architecture to evaluate the relevance of each pixel in the input image. By iterating this approach for each pixel, Grad-CAMs provide a heatmap showing the significance of each pixel in the classification pipeline. This piece of information is not only useful to understand what a model relies upon to solve a given task, but it can be used to know which information your model relies on not only gives a better understanding of its behaviour but can also assure that the patterns used by the algorithm in the classification procedure are related to the identified class. As an example, in [61] the authors trained the model using a dataset where the horses' images had the same tag in the background. The heatmaps showed that the classification of horses only relied on the presence of the tag and not on the actual presence of a horse in the image.

We used Grad-CAMs to better understand which features were considered relevant by the different models in the classification procedure. Figures 4.7,

	max	LEAP	PL	PML
First arc. MNIST	0.8905	0.9238	**0.9472**	0.9087
Second arc. MNIST	0.9886	0.9848	**0.9908**	0.9880
First arc. FMNIST	0.7978	0.8385	0.8424	**0.8435**
Second arc. FMNIST	0.8845	0.8776	0.8930	**0.8985**
First arc. CIFAR-10	0.3226	0.3291	**0.4185**	0.3869
Second arc. CIFAR-10	0.6145	0.5217	0.6355	**0.6499**

Table 4.1: The accuracy for the three datasets and the two architectures dataset

4.8, 4.9 depict the Grad-CAMs on some of the data used in the training of the models, comparing which features are more relevant for each model. It is interesting to notice how Grad-CAMs coming from the persistence pooling and the max pooling focus on different parts of the image and, expecially on MNIST dataset, they sometimes result complementary. This phenomenon shows how the novel layer and max pooling take advantage of different features of the image, suggesting that a clever interaction between them can lead to more accurate results.

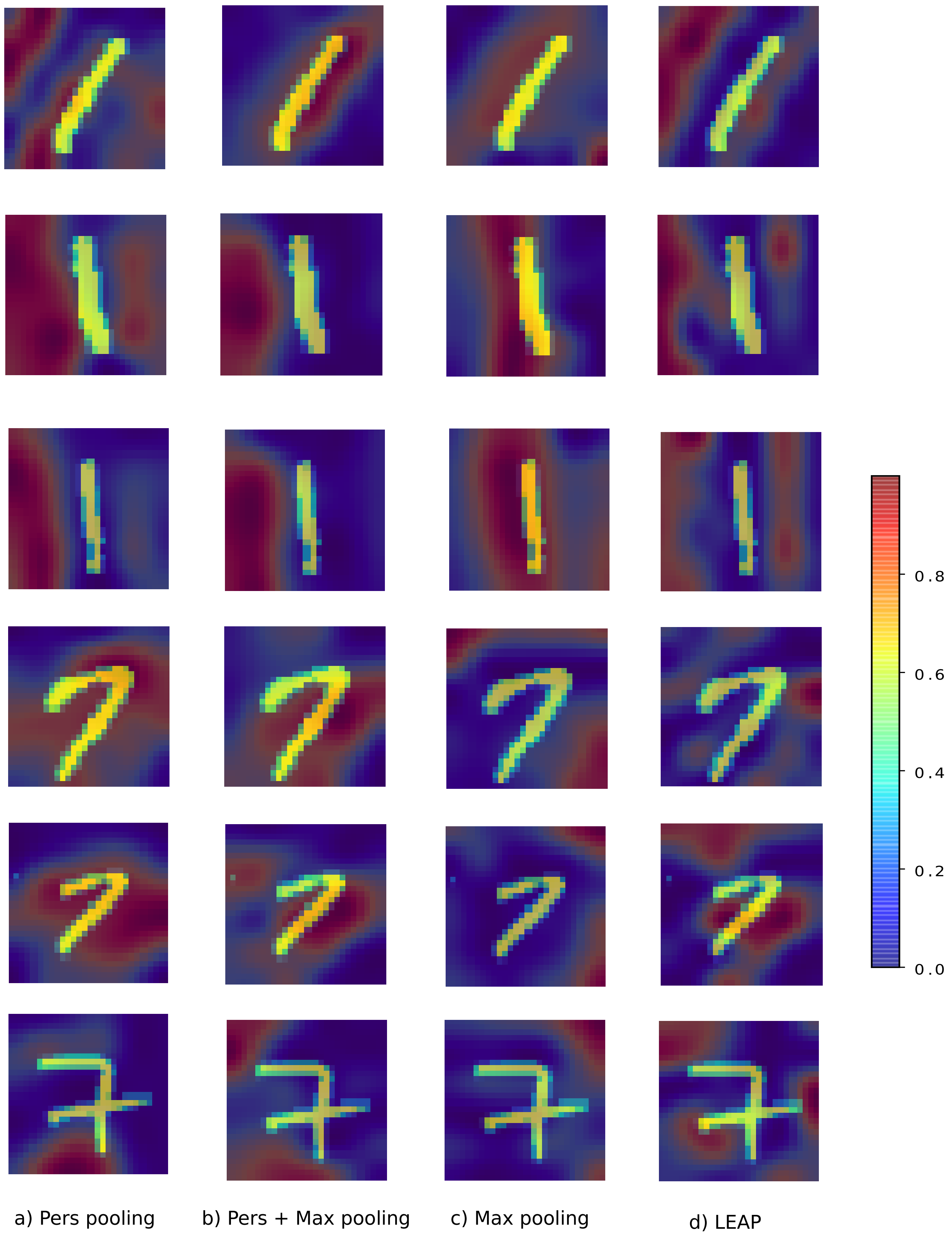


Figure 4.7: Grad-CAMs on the MNIST dataset. The first three rows show the Grad-CAMs on the digit 1, while the second three the Grad-CAMs on 7.

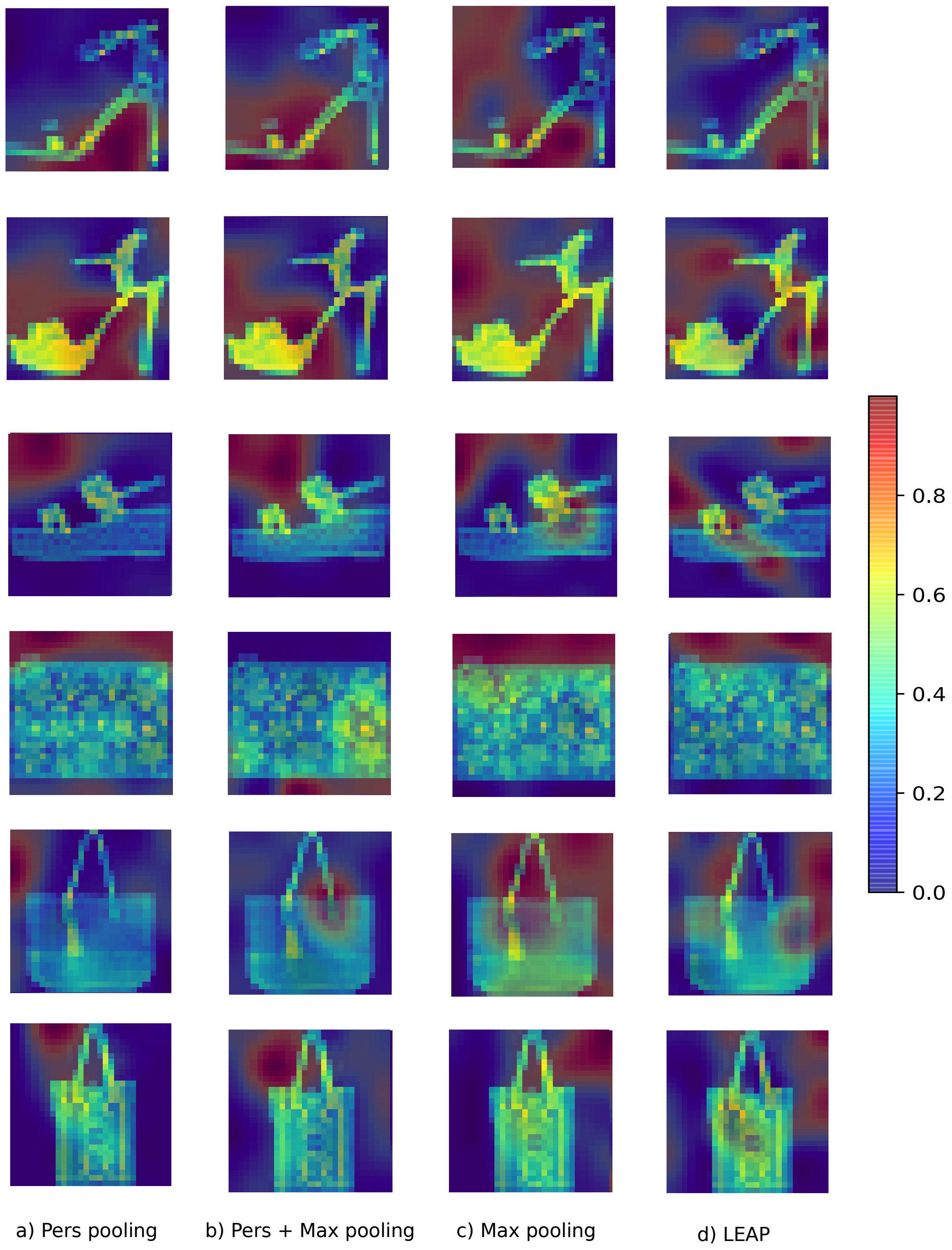


Figure 4.8: Grad-CAMs on the Fashion-MNIST dataset. The first three rows show the Grad-CAMs on the class *sandal*, while the second three the Grad-CAMs on the class *bag*.

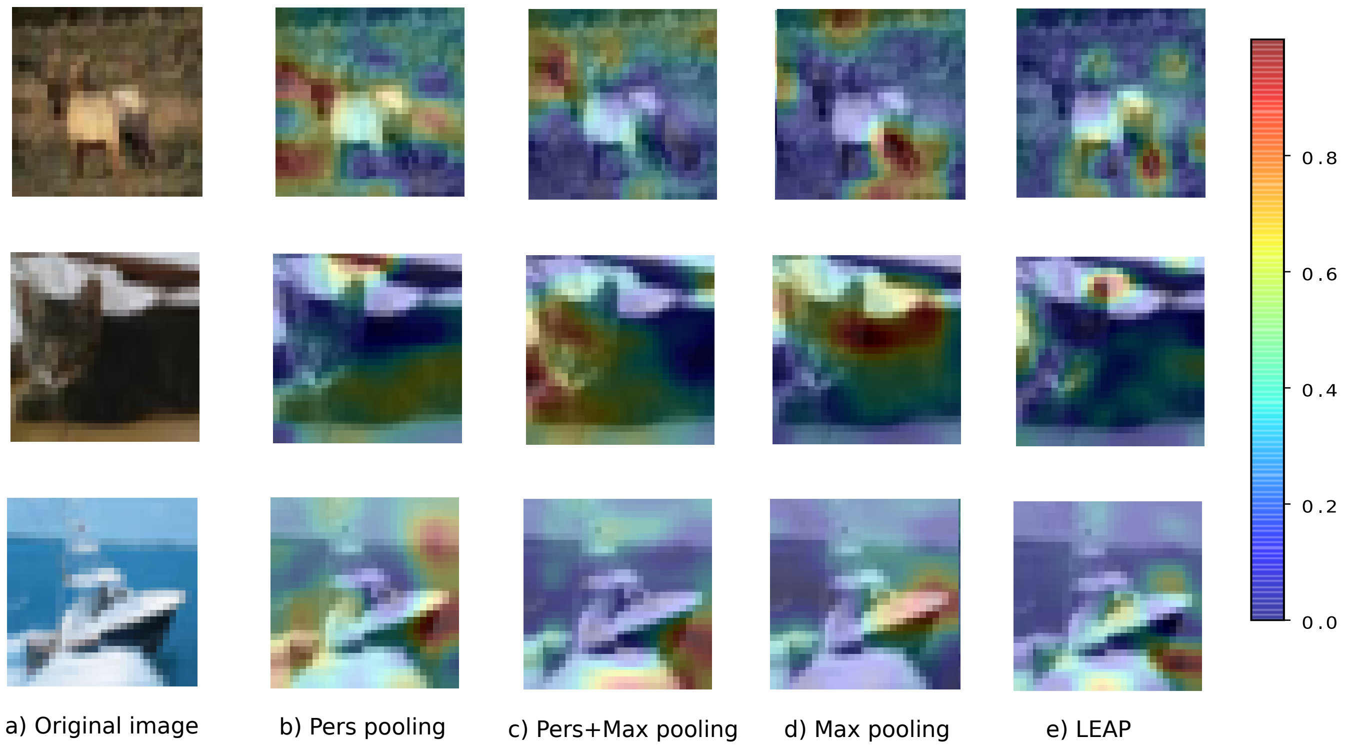


Figure 4.9: Grad-CAMs on the CIFAR10 dataset. The first row shows the Grad-CAMs on a *deer* image, the second on a *cat* and the third on a *boat*.

4.6 Conclusions

In this chapter, we analyze some effects of the categorical persistence introduced in [3], [5] in the image processing field. In section 4.2 we extended the steady and ranging concept introduced in [5] to the category **Set**. In section 4.3, we introduced a new operator, based on the notion of steady set, that enhances the signal of the pixels around the boundary of an object. We provided some examples of its effects on a test image and showed that its performances are stable with respect to the addition of salt and pepper noise to the image. Moreover, we provided a probabilistic framework that optimizes the parameters defining the operator in order to avoid noise artefacts.

In section 4.4, we embedded such an operator in deep convolutional neural networks, by defining a novel pooling layer. This pooling layer consists in the following main steps: the patches subdivision, the application of the persistence transform, defined using the new operator, to each patch and the final weighted average of the transformed values. As the pooling layers introduced in [41], [40] this layer contains a set of learnable weights, allowing to find the optimal downsampling. In section 4.5, we presented some experiments, comparing the performances of the proposed layer with some of the other state-of-the-art pooling layers. The comparison was performed on the task of classification, using two simple architectures. In the experiments, we tested the proposed layer, a mixed layer obtained as the weighted average of the proposed and the max-pooling layers, the max-pooling and the LEAP pooling. These experiments showed that the signal enhancing provided by the persistence transform highlight the relevant features, and that including the proposed layer increases the accuracy of the model. In fact, the best results were obtained with the persistence pooling layer on the MNIST dataset and with the mixed layer on the other datasets.

Future works could be to test the efficiency of the proposed layer in other frameworks, like generative models, or to use this layer not as a pooling layer, but as a simplified convolutional layer, as done for the LEAP layer in [41]. Another topic could be to extend the generalized persistence approach to other kinds of data, like graphs, studying operators that could be used in graph convolutional neural networks.

Conclusions 5

Persistent homology is a useful tool in tackling many problems in data analysis, providing an original perspective on of the structure of the data and, with persistence diagrams, an easy way to summarize and visualize the results of such analysis. For years, the classical approach followed in formalizing and applying persistence was to associate a filtration of topological spaces to the data, usually spaces of simplicial complexes, and then to study how topological features changed along with the filtration. This approach presented some categorical limitations, as highlighted in [3, 4], in the compulsory use of topological spaces and homological functors. In [3], the authors provided an original approach to persistence theory, extending the persistence framework to a more general categorical setting. This method allows working directly in the category where the data belong, allowing one to take advantage of features that could not be studied with standard persistence theory, see [4], [5].

In this thesis, we analyzed how a broader formalization of persistence can be integrated and adapted to machine learning and data analysis. In chapter 2 we collected some basic notions of category theory. Moreover, from [3, 4][5], we recalled some results concerning the novel framework that would be useful in the remaining part of the thesis.

In chapter 3 we firstly addressed the problem of finding functions that can be used as rank functions on the category of graphs. In this analysis, we showed how many of the tested functions fail some of the requirements, leaving us with only two rank functions, namely the size and the order of the graph. In the following sections, we used the approach introduced in [4] to study graphs properties through weakly directed posets. We analyzed connectivity features in directed graphs, highlighting the differences between the different notions of connected components. The last part of the chapter was devoted to the analysis of how strongly connected components distributes as we change the orientation over a chosen underlying graph.

In chapter 4, after a brief recall of the basic concepts of deep learning, we extended the notions of steady and ranging sets introduced in [5] to the category **Set**. The notion of steady set was used to define a new operator on images, whose effect is to enhance the signal of the pixels that are close to a discontinuity. We showed with some experiments the effects of such an operator on a test image. Moreover, the performances of this operator are not strongly affected by salt and pepper noise, and it is possible to find an optimal parameter setting in order to avoid noise artefacts. Another advantage of this operator is that it is possible to use it in neural networks. In the second part of the chapter, we showed how this operator can be embedded in a pooling layer. Standard pooling layers rely only on simple features of the data analyzed, e.g. the maximum or the average. In [40] and [41] the authors attempted to use more information, allowing the network to learn the optimal downsampling. The persistence pooling layer introduced not only allows to optimize the downsampling through a set of learnable parameters but also performs a transformation of the input signal that enhance some relevant features. In the last part of the chapter, we reported some examples showing the performances of the proposed layer on the task of image classification and comparing it with other state-of-the-art pooling layers. These experiments showed that the transformation performed by the proposed layer is a good choice as downsampling step; in fact, this layer outperformed the other state-of-the-art methods tested in terms of accuracy.

Bibliography

[1] Peter Bubenik, Vin De Silva, and Jonathan Scott. 'Metrics for generalized persistence modules'. In: *Foundations of Computational Mathematics* 15.6 (2015), pp. 1501–1531 (cited on pages 1, 2).

[2] Amit Patel. 'Generalized persistence diagrams'. In: *Journal of Applied and Computational Topology* 1.3 (2018), pp. 397–419 (cited on pages 1, 2).

[3] Mattia G Bergomi and Pietro Vertechi. 'Rank-based persistence'. In: *Theory and Applications of Categories* 35 (2020), pp. 228–260 (cited on pages 1–3, 5, 14, 53, 81, 83).

[4] Mattia G Bergomi et al. 'Beyond topological persistence: Starting from networks'. In: *arXiv preprint arXiv:1901.08051* (2019) (cited on pages 1, 3, 5, 45, 83).

[5] Mattia G Bergomi, Massimo Ferri, and Antonella Tavaglione. 'Steady and ranging sets in graph persistence'. In: *arXiv preprint arXiv:2009.06897* (2020) (cited on pages 1, 3, 17, 55, 63–65, 81, 83).

[6] Herbert Edelsbrunner, David Letscher, and Afra Zomorodian. 'Topological persistence and simplification'. In: *Proceedings 41st annual symposium on foundations of computer science*. IEEE. 2000, pp. 454–463 (cited on page 2).

[7] Robert Ghrist. 'Barcodes: the persistent topology of data'. In: *Bulletin of the American Mathematical Society* 45.1 (2008), pp. 61–75 (cited on page 2).

[8] Peter Bubenik and Jonathan A Scott. 'Categorification of persistent homology'. In: *Discrete & Computational Geometry* 51.3 (2014), pp. 600–627 (cited on page 2).

[9] Vin de Silva, Elizabeth Munch, and Anastasios Stefanou. 'Theory of interleavings on $[0, \infty)$-categories'. In: *arXiv preprint arXiv:1706.04095* (2017) (cited on page 2).

[10] Michael Lesnick. 'The theory of the interleaving distance on multidimensional persistence modules'. In: *Foundations of Computational Mathematics* 15.3 (2015), pp. 613–650 (cited on page 2).

[11] Mattia G Bergomi, Massimo Ferri, and Lorenzo Zuffi. 'Topological graph persistence'. In: *Communications in Applied and Industrial Mathematics* 11.1 (2020), pp. 72–87 (cited on page 2).

[12] Saunders Mac Lane. *Categories for the working mathematician*. Vol. 5. Springer Science & Business Media, 2013 (cited on pages 5, 7).

[13] Jíři Adámek, Horst Herrlich, and George E. Strecker. *Abstract and Concrete Categories. The Joy of Cats*. 2004 (cited on page 5).

[14] Samuel Eilenberg and Saunders MacLane. 'General theory of natural equivalences'. In: *Transactions of the American Mathematical Society* 58.2 (1945), pp. 231–294 (cited on page 5).

[15] Pavel Etingof et al. *Tensor categories*. Vol. 205. American Mathematical Soc., 2016 (cited on page 16).

[16] Robert E Stong. 'Finite topological spaces'. In: *Transactions of the American Mathematical Society* 123.2 (1966), pp. 325–340 (cited on page 18).

[17] George Raptis et al. 'Homotopy theory of posets'. In: *Homology, homotopy and applications* 12.2 (2010), pp. 211–230 (cited on page 18).

[18] John Adrian Bondy, Uppaluri Siva Ramachandra Murty, et al. *Graph theory with applications*. Vol. 290. Macmillan London, 1976 (cited on pages 21, 26).

[19] Reinhard Diestel. 'Graph theory 5th ed'. In: *Graduate texts in mathematics* 173 (2017) (cited on page 21).

[20] Will Grilliette and Lucas J. Rusnak. *Incidence hypergraphs: The categorical inconsistency of set-systems and a characterization of quiver exponentials*. 2018 (cited on pages 27, 33).

[21] URL: https://ncatlab.org/nlab/show/category+of+simple+graphs (visited on 09/11/2019) (cited on pages 27, 33).

[22] Jiřı Adámek and Horst Herrlich. 'Cartesian closed categories, quasitopoi and topological universes'. In: *International Conference on Mathematical Foundations of Programming Semantics*. Springer. 1985, pp. 20–41 (cited on page 27).

[23] Warren S McCulloch and Walter Pitts. 'A logical calculus of the ideas immanent in nervous activity'. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133 (cited on page 55).

[24] Frank Rosenblatt. 'The perceptron: a probabilistic model for information storage and organization in the brain.' In: *Psychological review* 65.6 (1958), p. 386 (cited on page 55).

[25] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. 'Imagenet classification with deep convolutional neural networks'. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105 (cited on page 55).

[26] Ian Goodfellow et al. 'Generative adversarial nets'. In: *Advances in neural information processing systems*. 2014, pp. 2672–2680 (cited on page 55).

[27] Yann LeCun et al. 'Handwritten digit recognition with a back-propagation network'. In: *Advances in neural information processing systems*. 1990, pp. 396–404 (cited on pages 55, 61).

[28] Geert Litjens et al. 'A survey on deep learning in medical image analysis'. In: *Medical image analysis* 42 (2017), pp. 60–88 (cited on page 55).

[29] Li Deng et al. 'Recent advances in deep learning for speech research at Microsoft'. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 8604–8608 (cited on page 55).

[30] DeLiang Wang and Jitong Chen. 'Supervised speech separation based on deep learning: An overview'. In: *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 26.10 (2018), pp. 1702–1726 (cited on page 55).

[31] James B Heaton, Nick G Polson, and Jan Hendrik Witte. 'Deep learning for finance: deep portfolios'. In: *Applied Stochastic Models in Business and Industry* 33.1 (2017), pp. 3–12 (cited on page 55).

[32] Robert Culkin and Sanjiv R Das. 'Machine learning in finance: the case of deep learning for option pricing'. In: *Journal of Investment Management* 15.4 (2017), pp. 92–100 (cited on page 55).

[33] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016 (cited on page 56).

[34] Ovidiu Calin. *Deep Learning Architectures*. Springer, 2020 (cited on pages 56, 58).

[35] Haskell B Curry. 'The method of steepest descent for non-linear minimization problems'. In: *Quarterly of Applied Mathematics* 2.3 (1944), pp. 258–261 (cited on page 58).

[36] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. 'Learning representations by back-propagating errors'. In: *nature* 323.6088 (1986), pp. 533–536 (cited on page 58).

[37] Avraham Ruderman et al. *Pooling is neither necessary nor sufficient for appropriate deformation stability in CNNs*. 2018 (cited on page 61).

[38] Matthew D Zeiler and Rob Fergus. 'Stochastic pooling for regularization of deep convolutional neural networks'. In: *arXiv preprint arXiv:1301.3557* (2013) (cited on page 61).

[39] Benjamin Graham. 'Fractional max-pooling'. In: *arXiv preprint arXiv:1412.6071* (2014) (cited on page 61).

[40] Peng Wang et al. 'Order-aware convolutional pooling for video based action recognition'. In: *Pattern Recognition* 91 (2019), pp. 357–365 (cited on pages 61, 62, 72, 73, 81, 84).

[41] Manli Sun et al. ‘Learning pooling for convolutional neural network’. In: *Neurocomputing* 224 (2017), pp. 96–104 (cited on pages 61, 72, 75, 81, 84).

[42] T. Gebhart, P. Schrater, and A. Hylton. ‘Characterizing the Shape of Activation Space in Deep Neural Networks’. In: *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*. 2019, pp. 1537–1542. DOI: 10.1109/ICMLA.2019.00254 (cited on page 62).

[43] Gregory Naitzat, Andrey Zhitnikov, and Lek-Heng Lim. ‘Topology of deep neural networks’. In: *Journal of Machine Learning Research* 21.184 (2020), pp. 1–40 (cited on page 62).

[44] Rahul Paul and Stephan Chalup. ‘Estimating Betti Numbers Using Deep Learning’. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2019, pp. 1–7 (cited on page 63).

[45] Anirudh Som et al. ‘Pi-net: A deep learning approach to extract topological persistence images’. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 2020, pp. 834–835 (cited on page 63).

[46] Mathieu Carrière et al. ‘PersLay: a neural network layer for persistence diagrams and new graph topological signatures’. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 2786–2796 (cited on page 63).

[47] Christoph Hofer et al. ‘Deep learning with topological signatures’. In: *arXiv preprint arXiv:1707.04041* (2017) (cited on page 63).

[48] Peter Bubenik. ‘Statistical topological data analysis using persistence landscapes.’ In: *J. Mach. Learn. Res.* 16.1 (2015), pp. 77–102 (cited on page 63).

[49] Kwangho Kim et al. ‘Efficient topological layer based on persistent landscapes’. In: *arXiv e-prints* (2020), arXiv–2002 (cited on page 63).

[50] James Clough et al. ‘A topological loss function for deep-learning based image segmentation using persistent homology’. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020) (cited on page 63).

[51] Michael Moor et al. ‘Topological autoencoders’. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 7045–7054 (cited on page 63).

[52] Rickard Brüel Gabrielsson et al. ‘A topology layer for machine learning’. In: *International Conference on Artificial Intelligence and Statistics*. PMLR. 2020, pp. 1553–1563 (cited on page 63).

[53] Z. Cheng-Jun. ‘Entropy-based estimation of salt-pepper noise in wavelet domain’. In: *2013 10th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP)*. 2013, pp. 366–370 (cited on page 70).

[54] Kanna B Rajesh et al. 'Model to estimate the salt and pepper noise density level on gray-scale digital image'. In: *EAI Endorsed Transactions on Energy Web* 5.18 (2018) (cited on page 70).

[55] Yann LeCun and Corinna Cortes. 'MNIST handwritten digit database'. In: (2010) (cited on page 74).

[56] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017 (cited on page 74).

[57] Alex Krizhevsky et al. 'Learning multiple layers of features from tiny images'. In: (2009) (cited on page 74).

[58] François Chollet et al. *Keras*. `https://keras.io`. 2015 (cited on page 75).

[59] Diederik P Kingma and Jimmy Ba. 'Adam: A method for stochastic optimization'. In: *arXiv preprint arXiv:1412.6980* (2014) (cited on page 75).

[60] Lutz Prechelt. 'Early stopping-but when?' In: *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69 (cited on page 75).

[61] Sebastian Lapuschkin et al. 'Unmasking clever hans predictors and assessing what machines really learn'. In: *Nature communications* 10.1 (2019), pp. 1–8 (cited on page 76).

Alphabetical Index