

Victor W. Wang

Cartographic Generalisation with Deep Learning

Master's thesis in Engineering & ICT

Supervisor: Terje Midtbø

June 2020

Victor W. Wang

Cartographic Generalisation with Deep Learning

Master's thesis in Engineering & ICT
Supervisor: Terje Midtbø
June 2020

Norwegian University of Science and Technology
Faculty of Engineering



Master thesis

(TBA4925 - Geomatics, Master thesis)

Spring 2020

for

Victor W. Wang

Cartographic Generalisation with Deep Learning

BACKGROUND

On every map, information must be curated. In making maps of smaller scales, generalisation is applied, but there are many solutions to each problem. At the same time, new research from Deep Learning has provided solutions to tasks previously considered solvable by humans only. It is then natural to ask if technology from Deep Learning can perform map generalisation.

TASK DESCRIPTION

The thesis uses milestones within Deep Learning (DL) as starting points for attempting map generalisation. From a DL perspective, the idea is to make a «generative adversarial network» (**GAN**) understand the contents of an image and simplify the representation. From a cartographic perspective, it is generalisation on raster data.

Specific tasks:

- Study and present related literature and theory; motivate usage of this technology
- Perform experiments: train «famous» GANs to perform generalisation
- Discuss, evaluate, and compare results. Especially w.r.t. articles with similar goals and different methods, such as algorithmic methods.

ADMINISTRATIVE/GUIDANCE

The work on the Master Thesis starts on January 15th 2020.

The thesis report as described above shall be submitted digitally in INSPERA at the latest at June 30th, 2020.

External supervisor:

N/A

Supervisors at NTNU and professor in charge:

Terje Midtbø

Trondheim, February, 2020

Preface

This thesis is a continuation of the specialisation project from the preceding semester, provided by my supervisor, prof. Terje Midtbø, marking the last and single largest piece of my studies. Despite various obstacles, this has been my greatest work yet. In that regard, my solemn thanks goes to everyone whose help or good company I could call upon, even if I chose not to.

I wish to extend a more personal thanks to those who put up with my occasional gaffes and bloopers. You know who you are.

— Victor W. Wang

Abstract

This thesis tests usage of research milestones from Deep Learning, i.e. **neural networks**, to perform cartographic generalisation. Generalisation is typically performed with defined algorithms on maps (vector data). Neural networks learn to perform tasks that are represented by the start and end product, and two particular neural networks are used, each with one added variation. Five different tasks are given, represented by five datasets of maps. By representing generalisation problems with maps, four neural networks are trained to perform cartographic generalisation. One of them — CycleGAN — shows competitive results. This thesis contributes evidence to support the quality and robustness of said network, among other contributions, and makes comparisons with related work.

Sammendrag

Denne oppgaven eksperimenterer med bruk av milepæler innenfor «Dyp læring» (*Deep Learning*), dvs. **nevrale nettverk**, for å utføre kartografisk generalisering. Dette utføres typisk med veldefinerte algoritmer for kart (vektordata). Nevrale nettverk lærer å utføre oppgaver ved hjelp av eksempler på start og slutt i utførelsen, og to nevrale nettverk brukes, med én ekstra variant per stykk. Fem typer oppgaver blir gitt, representert av fem datasett av kart. Ved å representere et generaliseringsproblem med kart, trenes fire nevrale nettverk for å utføre kartografisk generalisering. Én av disse — CycleGAN — gir konkurransedyktige resultater. Oppgaven bidrar med bevis for kvalitet og robusthet til nettopp denne, blant andre bidrag, og sammenligner med relatert arbeid.

Contents

Preface	i
1 Introduction	1
1.1 Thesis structure	1
1.2 Motivation	1
1.3 Objectives	4
2 Theory	7
2.1 Cartographic generalisation	7
2.1.1 Map traits	9
2.1.2 Visual variables	11
2.1.3 Methods & Workflow	15
2.2 Deep Learning	19
2.2.1 Neural Networks basics	19
2.2.2 Convolutional neural networks (CNNs)	25
2.2.3 Generative Adversarial Networks (GANs)	31
3 Related Work	35
3.1 Map generalisation	35
3.1.1 Transferring multiscale map styles using GANs [KGR19]	35
3.1.2 Building generalization using Deep Learning [SFT18]	36
3.1.3 A heuristic approach to the generalization of complex building groups in urban villages [YZZ19]	37
3.1.4 Specifying Map Requirements for Automated Generalization of Topographic Data [Sto+09]	38
3.1.5 Fully automated generalization of a 1:50k map from 1:10k data [Sto+14]	40
3.2 GANs	40

3.2.1	Conditional GANs (cGANs) [MO14]	41
3.2.2	Pix2Pix [Iso+17]	41
3.2.3	CycleGAN [Zhu+17]	42
3.2.4	Wasserstein GAN (WGAN) [ACB17]	43
3.2.5	Improved Training of Wasserstein GANs [Gul+17]	44
4	Methodology	47
4.1	GANs & specifications	48
4.2	Datasets	48
4.2.1	Dataset properties.	50
4.3	Hardware & software environment	52
4.4	Performance evaluation	52
5	Results	55
5.1	Pix2Pix	56
5.2	Pix2pix-GP	58
5.3	CycleGAN	61
5.4	CycleGP	63
6	Discussion	65
6.1	Generalisation capabilities	65
6.2	Limitations	66
7	Conclusion	69
7.1	Study objectives	69
7.2	Future work	70
8	Acknowledgements & Legalities	71
	Bibliography	73

List of Figures

1.1	Demonstration: how Tesla Autopilot AI sees.	2
-----	---	---

1.2	Faces generated by Nvidia’s ProGAN	3
1.3	Style transfer of <i>Starry Night</i>	3
1.4	Image-to-image translation	4
1.5	Image translation problem 1	5
1.6	Image translation problem 2	5
2.1	Trondheim in Google Maps	7
2.2	Church map, Poland	8
2.3	Population dot map of the USA	8
2.4	Berlin metro map, 2020	10
2.5	Optical illusion.	11
2.6	An optical illusion relevant to map design.	11
2.7	Partial ISOM 2017 legend.	12
2.8	Choropleth map giving misleading impression.	13
2.9	Adjusted, accurate choropleth map.	13
2.10	Symbolisation failure case	14
2.11	Omission example	16
2.12	Exaggeration example	16
2.13	Classification example.	17
2.14	Neural network analogy	19
2.15	Computational graph for a neuron unit.	21
2.16	Basic NN: perceptron.	22
2.17	Human perception vs. convolution	25
2.18	Convolution operation example	26
2.19	Pooling methods	26
2.20	Generalisation cases	29
2.21	Loss during NN training.	30
2.22	Batch normalisation algorithm	30
2.23	Conv. filter activations	31
2.24	GAN convergence explanation	33
2.25	Interpolating generator outputs.	33
2.26	Semantic image synthesis with GANs	34
3.1	Map generalisation with Pix2Pix	35
3.2	Generalisation by U-Net-like NN	36

3.3	Urban village example	37
3.4	Heuristic-based generalisation.	38
3.5	Generalisation example	39
3.6	Generalisation example: 1:10k to 1:50k	40
3.7	Diagram for conditional GAN	41
3.8	NNs used to develop Pix2Pix	42
3.9	CycleGAN concept	43
3.10	WGAN vs. standard GAN	43
3.11	WGAN vs. DCGAN	44
3.12	4 GANs compared	45
4.1	Image translation problem 1	47
4.2	Image translation problem 2	48
4.3	Tileset samples.	49
5.1	Pix2Pix cost graphs	57
5.2	Pix2Pix on Mapnik-Michelin	57
5.3	Artifacts in Pix2Pix	57
5.4	Pix2Pix: worst result	57
5.5	Pix2Pix: misclassification.	58
5.6	Pix2Pix-GP cost graphs	58
5.7	Pix2Pix-GP on Mapnik-Michelin	59
5.8	Pix2Pix-GP artifacts on OSM-no-labels	59
5.9	Pix2Pix-GP on Michelin	59
5.10	Pix2Pix-GP: misclassification.	59
5.11	CycleGAN cost graphs	62
5.12	CycleGAN on Mapnik-Michelin	62
5.13	CycleGAN on OSM-no-labels	62
5.14	CycleGP: cost during training	63
5.15	CycleGP: failure on Michelin	64
5.16	CycleGP: failure on OSM-no-labels	64
6.1	Misapplied CycleGAN example	67
6.2	Pix2Pix: bad omission	67

List of Tables

2.1	Visual variables' properties	12
4.1	Tilesets and zoom levels used.	49
4.2	Tileset locations.	49
4.3	Datasets made and used for testing.	50
4.4	NN performance metrics	53
5.1	Performance for Pix2Pix.	56
5.2	Performance for Pix2Pix-GP.	60
5.3	Performance for CycleGAN.	61

1. Introduction

1.1 Thesis structure

This chapter begins with motivations, and objectives, research questions resulting from these.

Ch. 2: Theory describes prerequisite knowledge required for the thesis, divided into two main parts: cartographic generalisation and Deep Learning.

Ch. 3: Related Work is largely concerned with generalisation methods from cartography, and research papers from Deep Learning.

Ch. 4: Methodology describes the specific applications used to perform experiments, other material required, equipment used, and additional technical details.

Ch. 5: Results describes the observed performance during and after experiments.

Ch. 6: Discussion discusses the experiments, observations made, and limitations on conclusions.

Ch. 7: Conclusion makes a statement on answering the research questions asked, and suggestions for future work.

1.2 Motivation

In the last two decades, artificial **neural networks** (NNs) have gained increasing amounts of attention from academics and industries. This computational technique has many applications, including classification and generation of arbitrary types of content, all with surprisingly high levels of realism or precision. In recent years, these have

become so powerful that related ethical concerns are no longer hypothetical, but very real problems to consider. NNs are still progressing in solving tasks previously considered solvable by humans only. Already, medical applications have been found [RFB15], and in the car industry, Tesla’s autopilot function [Tes20] has become a prominent feature that is at least on par with human performance, if not clearly superior [Ele19].



Figure 1.1: Demonstration of how Tesla Autopilot AI observes objects; screenshot from [Tes20]. Objects such as cars are marked with boxes, and the boundaries on the road are clearly marked with green, yellow and red lines. The stop sign is also detected, indicating a restricted area, as evidenced by the text on the left part of the image.

The driving forces behind NNs are 1) large amounts of data available, and 2) computation speed, from development of computer hardware. As research on NNs progresses, novel architectural approaches have proven to be essential for solving increasingly difficult problems or radically improving performance. As one of many toolboxes found in the field of Machine Learning, NNs notably present a very different paradigm for problem-solving: this method produces a model that translates input to desired output, fulfilling a desired process $A \rightarrow B$, where humans might not be able to formulate any such algorithm at all. Rather than requiring humans to provide strict instructions about what to do, it requires only that humans implement methods for learning arbitrary processes and provide relevant examples.

Meanwhile, in cartography, there is an ongoing development of methods for cartographic (map) generalisation — how to best present data in a meaningful, visually informative way that provides the user with easily available and useful information,



Figure 1.2: Human faces generated by Nvidia's ProGAN. Figure from the article [Kar+17].

filtering away insignificant details and prioritising visibility of interesting data. User tests, development methods, and finalised tools resulting from this field of research are frequently based on traditional methods such as algorithm design, software development frameworks, and other approaches that rely on rules and conventions, with a the common theme being that rules are already made, unlike NNs where rules are instead developed. It is constantly adapting to a new societal and technological landscape.

With this in mind, a certain type of NNs is of particular interest for the purpose of applying cartographic generalisation: **generative adversarial networks** (GANs). One impressive feature of GANs is that they can generate objects of given types with high authenticity, such as human faces. Another is that these can apply an arbitrary style from a source image to a target image, and yet retaining the content; a task called style transfer.

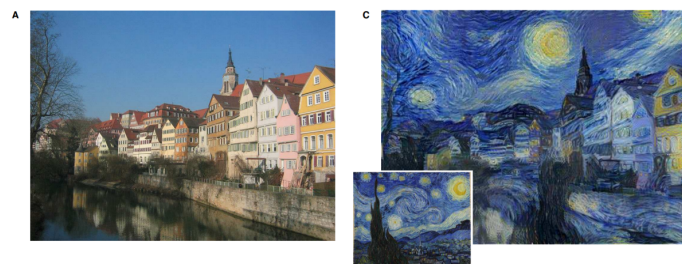


Figure 1.3: The style from Vincent van Gogh's *Starry Night*, applied to an image of some buildings. Figures from [Iso+17]

W.r.t. cartography and abstraction of any kind of data, it is also particularly interesting to see how satellite images can seemingly be converted to maps on the fly, as seen in fig. 1.4.

At a glance, it seems that fulfilment of such tasks requires some level of understanding that separates underlying content from presentation. With these technological



Figure 1.4: Image-to-image translation: satellite image vs. map, using Pix2Pix. Figure from [Iso+17].

achievements in mind, this thesis will investigate the capability to perform the following task: rather than using any methodical, algorithmic design philosophy for cartographic generalisation, the focus is instead on using the data-driven, computational method of NNs to generalise maps.

1.3 Objectives

The main objective of this thesis is to perform cartographic generalisation exclusively through the use of NNs, providing results, insights and problems for future research on this particular generalisation method.

There are various limitations and restrictions for this project further outlined later in the thesis. Two types are explored: «zoom-out generalisation» and «content generalisation». The following research questions can be asked and will be focused on.

- How do selected NNs perform, given the generalisations that are found in a given dataset?
- How robust are the specific methods used? I.e. to what extent is performance largely independent of properties for any given dataset, such that even diverse information can be handled without errors?

This thesis is best read on a computer, where figures can be magnified. Certain figures may be hard to inspect visually on paper.

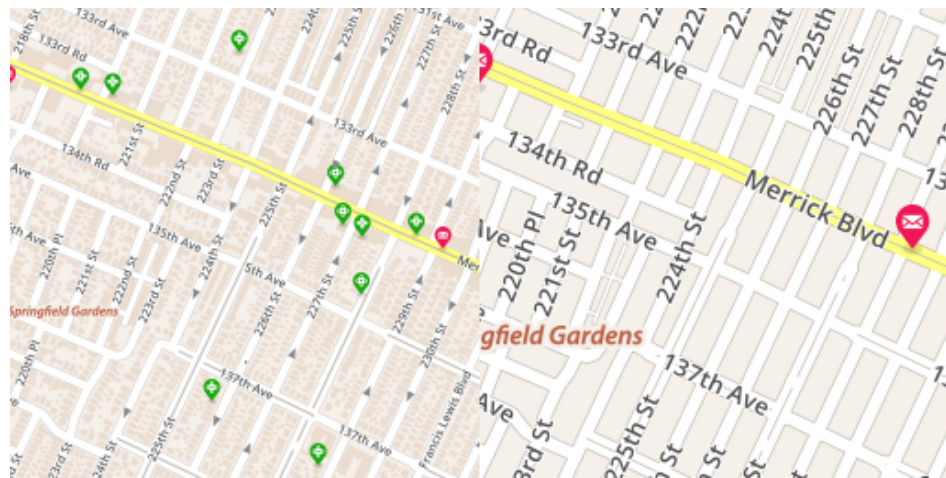


Figure 1.5: Image translation problem: going from a larger scale to a smaller scale. Michelin maps. Details from (zoom level) $z = 17$ on the left vs. details from $z = 16$ on the right. Note how most buildings are no longer rendered at all, and that only a handful of highlighted locations remain.

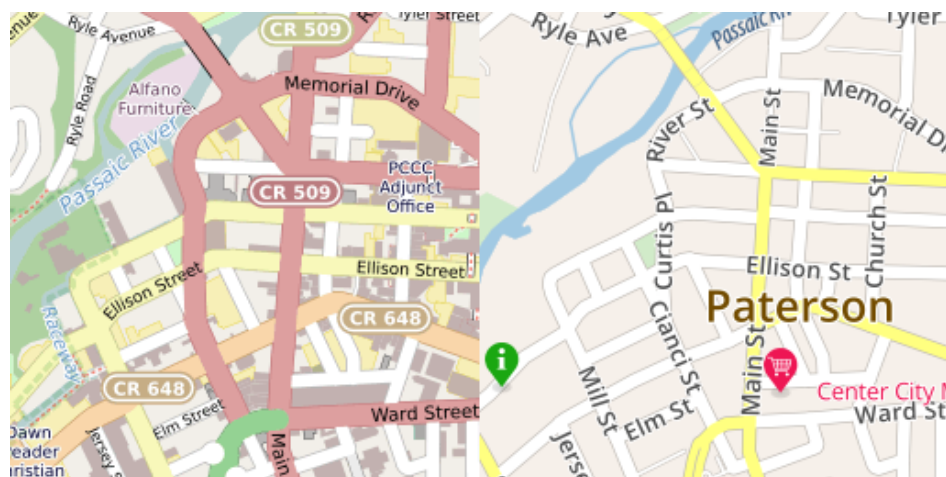


Figure 1.6: Image translation problem: generalising elements presently in a map. OSM Mapnik (left) vs. Michelin (right). Many details are removed altogether such as buildings and road labels, and various road/streets in the left image are classified but not consistently.

2. Theory

2.1 Cartographic generalisation

Before outlining cartographic generalisation, the prerequisite should be defined, namely maps.

Like many other ubiquitous products and services, maps are subject to design principles and pragmatism, and must be adapted to specific requirements. According to the International Cartographic Association (ICA), «a map is a symbolised representation of geographical reality, representing selected features or characteristics, resulting from the creative effort of its author's execution of choices, and is designed for use when spatial relationships are of primary relevance.» [Int14]

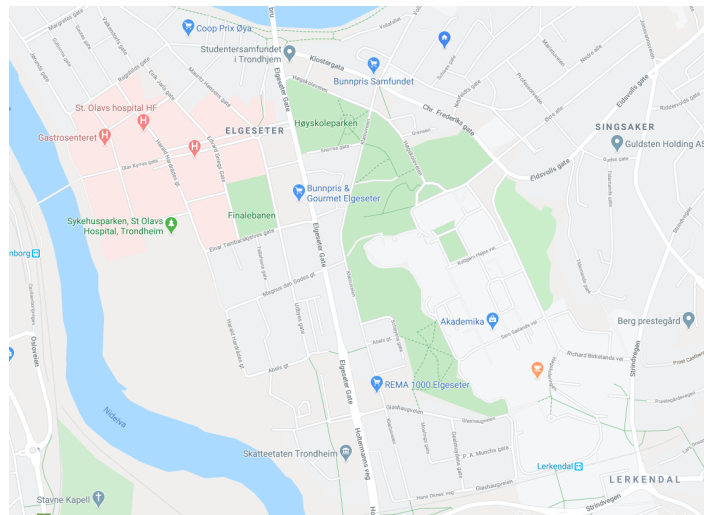


Figure 2.1: Trondheim in Google Maps, a very recognisable map design (copyright by Google).

The principles and methods that govern map design are many, but it is easy enough for anyone to recognise when a map is satisfying or not. A good map can be used instantly for its intended purposes, but a bad one may be messy, requiring too much time from the reader to find the objects of interest, leading users to discard such maps. ICA defines cartography as «the discipline dealing with the conception, production,

dissemination and study of maps» [Int14], and one thing in particular is clear with the average person’s idea of a map: it is not a perfect representation of reality. Most maps for practical uses are not satellite images, focusing instead on zoomed-out, simple representations of the world, such as in fig. 2.1. «A pure photographic reduction of the original scale leads to an illegible map.» [All13].

Map design has changed over the last decades. In the past it was a specialised skill, but with computer technology, the methods and the objects have changed. Map design can be performed with algorithms, or specified by users on user-friendly websites. The underlying data has exploded in quantity too, following the general trend that digital data in the world is growing at an exponential rate.

Cartographic generalisation is motivated by various problems: excessive details, overlap, irrelevance, visibility, perception, and so on. The extent of generalisation and the methods employed depend on the task at hand. E.g. a map for travelling across distant regions has no good reason to highlight particular buildings in any populated area, rather than highlighting said area. Basic questions may have basic answers but representing them is a different matter. Quantity alone can lead to major problems if the density of some object is much too high; see fig. 2.2. But even representational methods such as dot density maps may be misleading. Figure 2.3 is useful in showing dense population centres but makes parts of the USA seem devoid of human life, demonstrating the need for gradual representation rather than binary, simplistic methods.

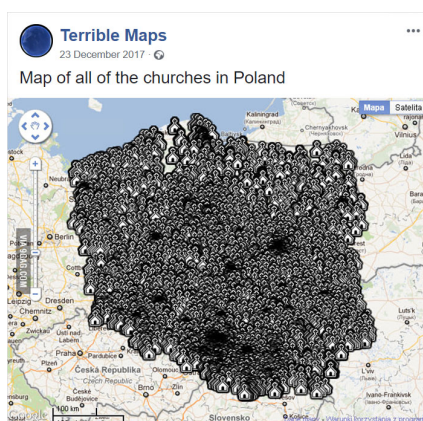


Figure 2.2: Churches in Poland. Each individual church is represented. Figure from [Twi17].

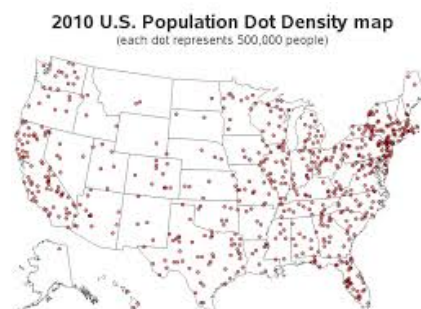




Figure 2.3: Population in USA shown by a dot map. Figure from [All10].

2.1.1 Map traits

In measuring the design of a map, or acquiring a notion of its utility, some traits can be distinguished and evaluated separately, even if the requirements for these vary based on the specific map and the cartographer's purposes. According to [All13], the following should be in mind when designing a map:

Structure. Contents must be prioritised with respects to the map's purpose, scaled to the size of the map itself and the content. Object grouping, clustering and such must make sense, while also distinguished in reasonable ways. E.g. buildings may be joined when sufficiently zoomed-out and distances between them approach negligible values, especially if individual buildings are of minimal interest.

Legend. Symbols and textures used to represent spatial objects and information should be expressive and associative, such that these are recognisable and familiar to the user. E.g. Christian churches may be represented by a Latin cross , while drugstores may be represented by green Greek crosses such as . By common convention, green surfaces typically denote forests, parks and other naturally green scenes. All at the same time, colours must respect rules surrounding perception. Strong, saturated colours typically demand more attention than background colours.

For some maps, an explicit legend is not needed. In this case, maps should adhere to common designs and principles found in publicly available domains; e.g. all kinds of waters should be coloured blue, green areas simply given the same green colour, and roads marked with background colours. Populated areas may be marked simply with some background colour such as a gentle yellow or beige, with no particular buildings drawn.

Generalisation level. A map becomes increasingly generalised when information is removed or abstracted into simpler forms. The extent to which a map should be generalised depends greatly on the map's scale and purpose. E.g. for a greater city map, all kinds of forestry and parks may be grouped into a single category as green leisure areas. An example of heavily generalised maps are metro maps, e.g. fig. 2.4, where the only information presented is topological relationships between the stations, keeping a very weak relationship to the geographical reality.

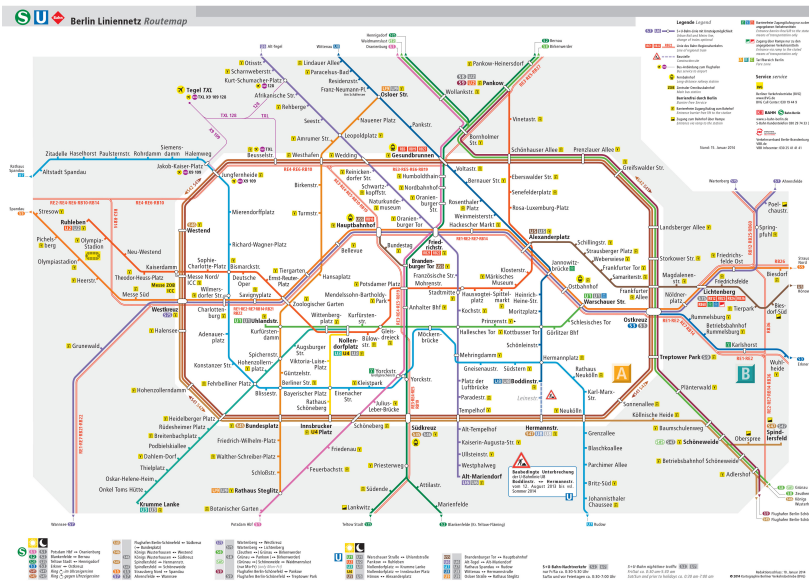


Figure 2.4: The 2020 Berlin metro map, Figure from [map20]. An entirely topological map.

Selection/filtering. Maps have purpose, and any information that is totally irrelevant or useless in fulfilling said purpose should usually be discarded. With that in mind, landmarks still remain useful.

Accuracy. When possible, the position on the map should correspond with real life position. However, geometric accuracy is less important than informative positioning on the map; if accuracy comes at the price of usefulness, that would defeat the purpose of a map. This is best demonstrated by how roads and highways are marked on smaller scale maps: these are typically drawn wide enough to be visible at a glance, even if they only have two lanes. Highways are typically drawn so wide that one could mistakenly infer that there are more than six, if taken at face value.

Realism. As time passes, some facts of reality may change; an intersection may have turned into a roundabout, roads may be rearranged entirely, objects may be demolished and re-purposed. Outdated maps are of course less useful. Labels and legends should be appropriate and self-explanatory or described.

Legibility. Anything else than the map, should not be needed for reading the map, such as magnifying glasses. To this end, objects in the map cannot be so small that they cause clutter, or show details that can be ignored or lack informative value. There is clearly a trade-off between accuracy and legibility, but the latter still comes first.

Graphical representation. Legends should be short and concise. Unusual objects should typically be highlighted to give the correct impression, whereas common, superfluous objects (e.g. buildings in urban areas) should be generalised by means such as grouping them together into larger objects. All at the same time, the means for generalisation should remain consistent. Relations, dependencies (e.g. overlapping objects, symbols) should be considered in designing a map.

2.1.2 Visual variables

Information can be represented in various ways but there are preferences for how this is done, also outside of cartography. Deceptive representations, illusions, indistinguishable representations, may well be used.

And in much the same way as outlined in the previous section, cluttering can also be an issue. Pie charts with many, many categories may be impossible to learn anything useful from, especially if the distribution among these is rather even [Kry18].

According to (sources in) [Bjø05], there are eight visual variables to consider in designing a map: x coordinate, y coordinate, size, brightness, texture, colour (hue), direction, and symbolisation (shape). The latter six variables usually do not influence each other, with some exceptions, such as fig. 2.5 and fig. 2.6. The human mind observes with some kind of context, not necessarily with objectivity, and it is prone to being misled.

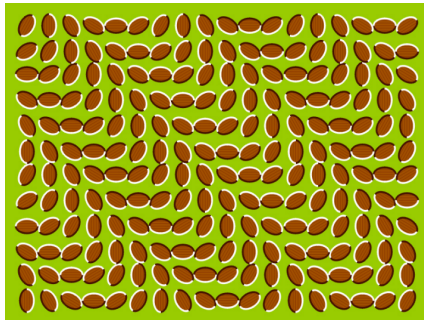


Figure 2.5: This still image seems to move. Figure from [Wor11].

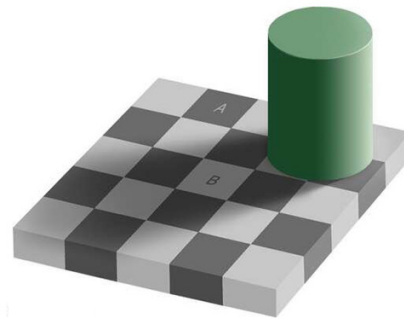


Figure 2.6: A and B seem to be different colours, but they are not. Figure from [Wor11].

There are various properties in these variables; they may be quantifiable, ordered, selective, distinct, and associative, all to varying degrees.

All the variables are selective to some degree, with difficulty arising from smooth sequences and small differences. (Otherwise they would not be worth mentioning as

Variable	Quantifiable	Ordered	Selective	Distinct	Associative
Location (x,y)	+++	+++	+++	✓	
Size	+++	+++	+++	✗	
Brightness		+++	+++	✗	
Texture		+	+++	✓	+
Colour (hue)			+++	✓	+++
Direction			++	✓	+
Symbolisation (shape)			+	✓	+++

Table 2.1: Properties for visual variables. Original table from NTNU TBA4240.

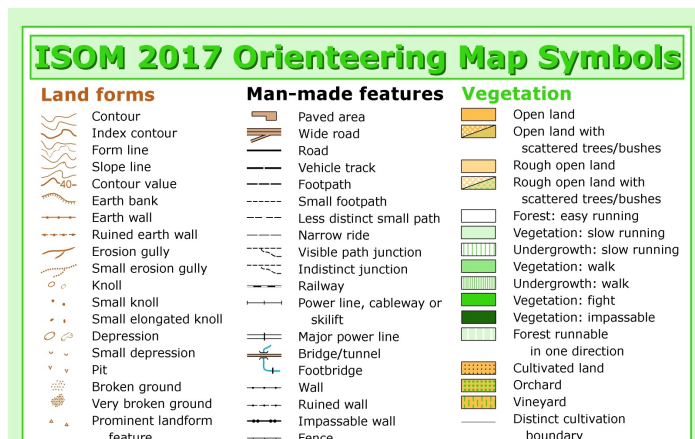


Figure 2.7: Cutout of the ISOM 2017 legend. Orienteering maps use texture to indicate terrain type, and many different symbols. Notably, green is associated with vegetation and used for no other purpose. Figure from [ISO17]

properties.) Congestion and cluttered visuals however can make direction and shape less useful overall, as can be evidenced by any basic, naive visualisation of huge amounts of map data; e.g. using a whole country’s worth of geometrical data is not going to produce anything readable..

Location is the essence of any map. Distances can be estimated on a map, points along a path can be ordered; obviously points are both selective and distinguishable.

While size is quantifiable and can be ordered, these can be rendered indistinguishable if the arrangement has a similar effect such as fig. 2.6.

Texture is commonly used to denote different land use or terrain. In orienteering maps especially, terrain type is particularly important for optimising one’s path. It is disputable whether these can be ordered, but visual density is something to go by. With finer details, differences may be hard to see at a glance, requiring a closer look instead.

Colour and brightness are typical traits to adjust in map design. Common design principles would have permeating objects (forestry, urban environments) be given background colours that do not demand attention; contrasting hues should be used to distinguish different objects, and strong/saturated colours in general are mostly reserved for foreground objects.

Additionally, colours are associated with values, feelings, objects in most cultures. Green is typically associated with nature and vegetation, blue with all watery things, and so these two colours are often reserved exclusively for representing such objects.

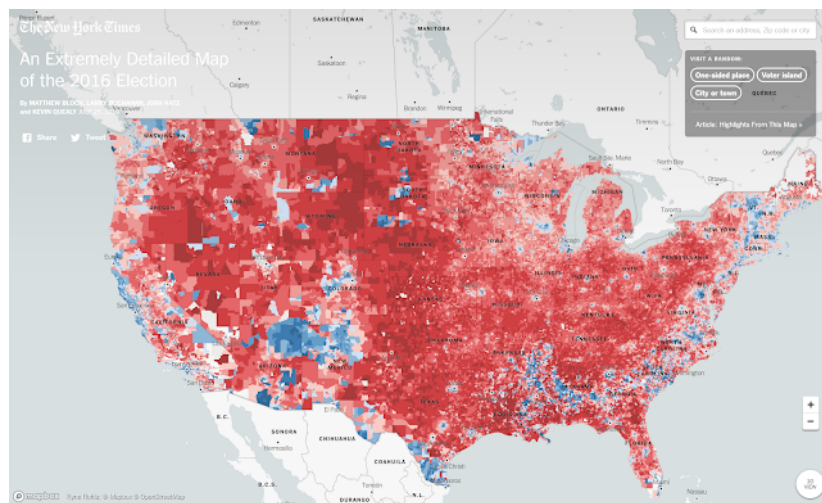


Figure 2.8: This choropleth map of the US 2016 presidential election, while detailed and accurate in its own respects, emphasises geography more so than the local results. It misleads viewers to assume a one-sided election. Figure from [Fie18]

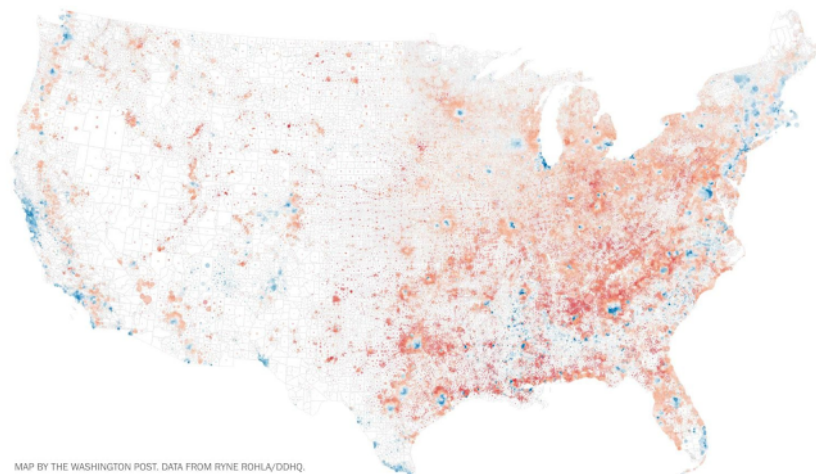


Figure 2.9: An adjusted version of the above map, accounting for population distribution, providing a more accurate idea of the US 2016 election outcome. Figure from [Fie18]

But even colours must be used carefully. In making a choropleth map, severity of some phenomenon is typically shown in ranked groups along a colour scale, often using dark shades to indicate severity, and with a small number of groupings, in the range of 5-7. Even these can be rather deceptive, however. Technical accuracy may deprive the map of usefulness, and the human mind observes things in context. An interpolated image of some phenomenon, may end up only showing exceptional changes and dominant areas, such as heavily concentrated population centres vs. mostly uninhabited areas.

While symbolisation may seem selective at first glance, it requires proper spacing. That said, appropriate usage is easily found on tourist maps that typically require proactive reading anyway, or on services such as Google Maps (fig. 2.1).

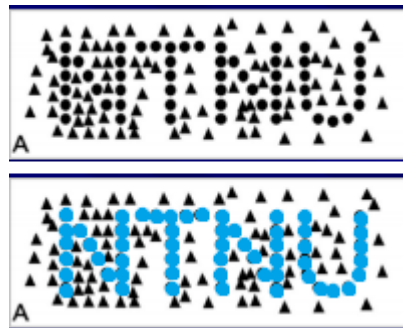


Figure 2.10: Symbolisation requires spacing or additional methods used. Original figure from NTNU TBA4240, coloured in addition.

2.1.3 Methods & Workflow

In generalisation, there are a number of processes used to modify a map, all of which can be well defined such that implementations can be made based on explicit algorithms. However, «as the generalisation of one element will affect the generalisation of others, it is necessary to follow the procedure, in the correct sequence» [All13]. Various sources and education material describe the following methods or similar things, and the correct sequence may vary.

Selection. This is in general the first step in making any map. There is an overwhelming abundance of data available these days, and only some is relevant; either to provide the immediate information of interest, or to assist as contextual information. E.g. a city map for tourist attractions should show the locations of interest, but it loses usefulness if it does not relate those locations to the street networks, landmarks, etc. For any navigation map for vehicles, buildings in particular may be redundant, especially when driving across larger regions, past cities and other inhabited areas. Closely related is **deselection**, namely hiding all data of a given category.

The following generalisation methods can be categorised as either semantic or geometric generalisation. Semantic usually comes first and is concerned with simplifying representation of data so information is unambiguously; in particular, classification, aggregation, exaggeration and symbolisation are related. Geometric generalisation is related to simplification, omission, and spatial reorientation.

Simplification. It is often enough not useful to provide data at the lowest level of detail, as this does not give a more informative picture of reality; even worse, this may defeat the purpose of a map, by making it impossible to read. Even in the most detailed data, geometric properties are typically simplified in some way w.r.t. reality. Common simplifications include point removal, e.g. reducing the complexity of a polygon or poly-line representing objects in space. Like most other processes, this is dependent on the map's purpose. The most extreme form of simplification can be found in metro maps such as fig. 2.4, where the main information of interest is connectivity.

Omission. The distinguishing characteristic of omission vs. selection, is that omission is performed on data that is still considered necessary, but by leaving out individual, excess elements. E.g. when zooming out on an interactive map, a cluster of buildings should be represented by yet another cluster but with fewer buildings, using the same symbolisation and retaining the spatial relations between them if possible, i.e. they should have about the same pattern; see fig. 2.11.

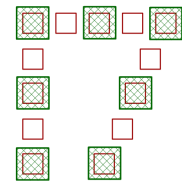


Figure 2.11: Omission: the green boxes would remain. Figure from [All13].

Exaggeration. If maps were to be truly representative of reality, a variety of narrow roads would be illustrated as very slim lines, such that they are nearly impossible to see on any map. If roads on maps today were taken at face value, using the scale for the map, one might be tempted to believe that any visible road on the map has several lanes, though the truth is obviously that they are exaggerated. As such, it is necessary to put emphasis on some objects by exaggerating them. The same can be said for streets and alleyways in city maps, otherwise they might be indistinguishable from lines meant to separate buildings. Small, significant objects generally may need to be enlarged.

Spatial reorientation. If elements overlap, especially after exaggeration, they must be moved away from each other. A loss of technical accuracy is generally an acceptable price for elevated usefulness. Some spatial properties however may be so small that there is no loss in usefulness in collapsing these. E.g. angles at 44° may as well be snapped to 45° since the differences are illegible, and if minimum distances are used in the generation of a generalised map, objects too close to each other may be connected rather than separated or distinguished, such as adjacent stores in the same building.



Figure 2.12: Roads are commonly exaggerated. Figure from norgeskart.no

Classification. The most typical example of classification is to colour water bodies blue and vegetation green, with or without different shades and textures to distinguish different types. In general there are two types of classification: qualitative and quantitative. The former typically identifies properties that are incomparable, such as land use, by using different hues. The latter is used to deal with phenomena that can be ordered and grouped,

such as average temperatures, and often for the purpose of making choropleth maps such as fig. 2.8, fig. 2.9, displaying the severity of some phenomenon w.r.t. predefined geographical boundaries. W.r.t. the HSV colour model, shading typically indicates severity, with hue denoting entirely different phenomena.

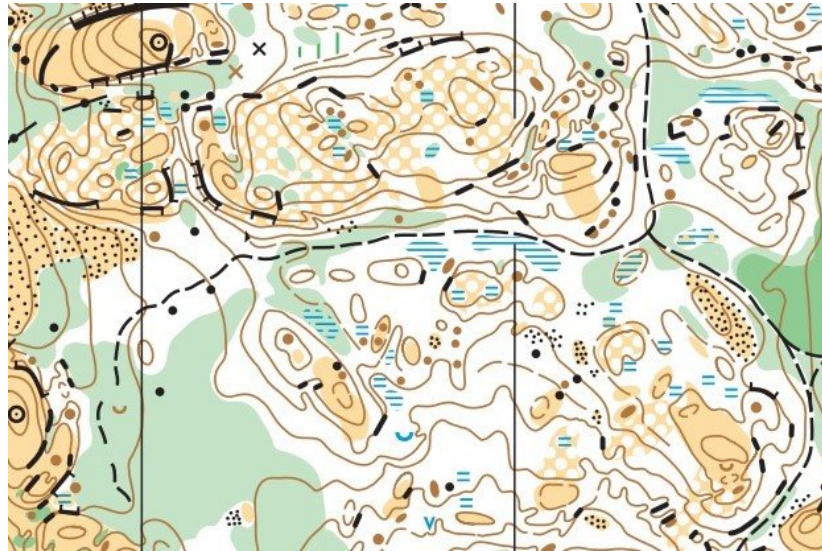


Figure 2.13: Provide both quantitative and qualitative information. Figure from [Ori18]

Aggregation. Though somewhat related to simplification, aggregation is used to replace several objects with the same representation but on a larger scale, such that a single symbol represents duplicates. E.g. a cluster of buildings may be replaced by a building that occupies their bounding box. A cluster of trees may be replaced by a singular tree. If this area is instead represented as a green area, it is classified, not aggregated. Similarly, there is also **typification**: representing many symbols with fewer.

Symbolisation. In reducing reality to a simpler overview, a map should make use of associations where possible. Some objects are associated with certain symbols, and therefore it makes sense to use said symbols. But pictorial symbols may also be used, i.e. miniature drawings or caricatures of their real-world counterparts. A Christian church may be represented by a Latin cross or a miniature drawing. Frequently, abstract symbols are used instead to highlight interesting locations, such as solid red dots marking tourist attractions on maps.

As mentioned already, these cannot be done arbitrarily without considering the order

of procedures. According to GITTA [All13], a typical order deals with the following objects in the given order: water bodies, contour lines and heights, specific locations and correcting their positions, objects related to populated areas, and lastly, land use and vegetation.

Additionally, generalisation is typically performed by a combination of 1) generalising map elements, and 2) reducing the scale of the map elements, in either order, according to one's priorities. For a complex map, it is better to generalise first and then reduce scale, whereas a reduction first may produce a better map *at that scale*. This may come at the price of correctly applied generalisation, however.

Lastly: generalisation can be performed on original data or pre-processed data, i.e. maps/data that have already been generalised to some scale. It is generally preferable to generalise on original data since pre-processed maps are subject to decisions that may be untraceable and impossible to recover from. There is also the notion of «garbage in, garbage out», w.r.t. any algorithm or procedure: functions of all kinds presume some kind of structure in what they are dealing with, and will not work for inappropriate inputs. Using an already generalised map can lead to subtle errors in the generalisation process.

2.2 Deep Learning

Within the field of Machine Learning, there is Deep Learning, specialising on solving visual tasks through artificial neural networks (NNs). The term was coined when NNs with many *layers* proved to be rather effective, giving rise to the notion of depth as a property.

2.2.1 Neural Networks basics

The inspiration for NNs is human neurology, or rather human nerve cells. Signals can be transmitted between them, manipulated, and passed along to others for further processing. This can be regarded as a computational process where inputs are passed to some function whose output becomes the input for various other functions, such as in fig. 2.14.

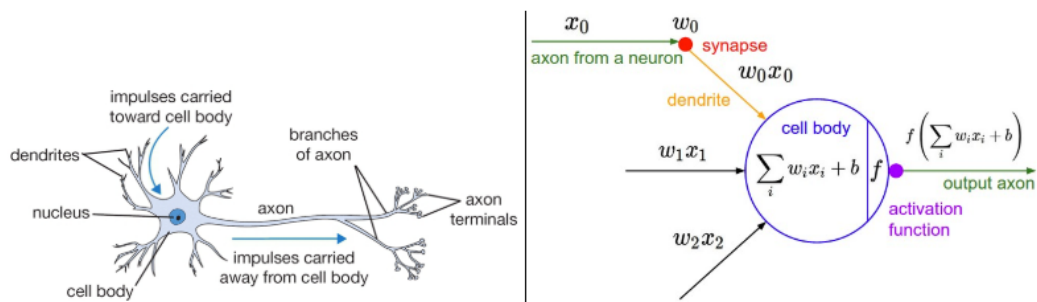


Figure 2.14: Human nerve cells can be formulated as computational graphs such that nodes take weighted inputs, apply some function and then output the results. Figure from Stanford CS231n [12320].

At first glance this may not seem like it amounts to anything particularly complex. But the key in NNs is not that these solve tasks instantly, but rather that these can simulate a computational method that *learns*, that becomes increasingly adapted to solving a given set of tasks; it is an entirely different paradigm of solving problems, where no methods are made explicit aside from general learning capability. To emphasise this even more, it can be shown that NNs can model arbitrary functions, given arbitrary computational power [Nie15]. And in recent years, the progress made has reached astounding levels such that problems considered solvable mostly by humans, can be solved without explicit instructions, unlike traditional, well-defined algorithms. For examples of state-of-the-art performance, see chapter 1.

Essentially, we want to adjust the parameters of some NN such that it maps every

input x to a desired output y , as though it were a long mathematical formula magically capable of fulfilling our intentions. By using a sufficient amount of **training examples**, each composed of corresponding input and output, a NN can «learn» to solve the set of problems given to it. If this set of problems can sufficiently cover the intended problem that we want the NN to solve, and the NN solves all these problems as intended, then the NN is said to be *generalised* well across the domain of its problem. At this point, the NN is a *model*, as it can translate input from one domain to a target codomain; e.g. perform classification.

Basic computational unit: the neuron

The basic computational unit of a classical NN is a **neuron**, shown on the right side of fig. 2.14 as a node in a graph; each node has a bias b , which is some real number. The node takes inputs x , weighs them according to values that make up w (represented by the connecting edges), and uses the sum $z = w \cdot x + b$ as the input for its function f ; w and x are (column) vectors, $w \cdot x$ denotes the dot product and is interchangeable with the matrix product $w^T x$. Lastly, the neuron outputs the **activation** $a = f(z)$; f is an **activation function** with a defined derivative, often squashing its input to a small range, such as $[-1, 1]$. The bias b shifts this function. These will be explained in further detail later.

Using the neuron in fig. 2.14 as an example: suppose it is fed one training example, it should only output the value of x_0 , and that $f(z) = z$. Repeated training examples would use input vectors x with random values, and the correct output value is the corresponding x_0 . In order to keep track of performance, (in)correctness is measured numerically across every training example. Functions such as the mean square error (2.1) can be used to measure that. Other functions can be used too, and these measurements are interchangeably referred to as *loss* or *cost* functions, penalising incorrect computations. Typically, 0 is the best possible loss, such that all wrongly computed answers increase the loss. So, let f be the linear function and:

$$\text{loss } L = \text{MSE} = \frac{1}{2N} \sum_i^N (\hat{y} - y)^2 \quad (2.1)$$

... where \hat{y} is the desired output and y is the computed output of the entire network, i.e. in this case we have $\hat{y} = x_0$ and $y = a = f(z) = w \cdot x + b$. This example uses $N = 1$, because only one training example is used at a time.

With only the above in mind, there is no guarantee that we have the correct solution

to this simple problem, since the initial weights w are not defined. With this example, it is easy enough to see how we can achieve a perfect loss of 0 with $w = [1, 0, 0 \dots 0]$ and $b = 0$, but for any non-trivial case, we do not know the optimal parameters in the beginning. This is where the mechanisms of NNs kick in and reach a solution automatically (hopefully).

Training a single neuron

The idea of training is: based on a loss function that measures performance, we can compute a partial derivative w.r.t. the parameters used. Using it, we can tell how much any given parameter contributes to the total loss, and make adjustments against that. This is known as **gradient descent**. To ensure stable convergence towards correctness, these adjustments are typically multiplied by a small factor η known as the **learning rate**, frequently in the range of 10^{-2} to 10^{-5} .

In implementations, the initial values for w can be distribution-based values in some given range, such as drawing random samples from the standard normal distribution. In any case, weights are typically small non-zero numbers.

It should be noted that a computational graph can be constructed, splitting up the computations needed for C and its derivative. This is because C is a function of y , which is a function of z , which is yet another function of w .

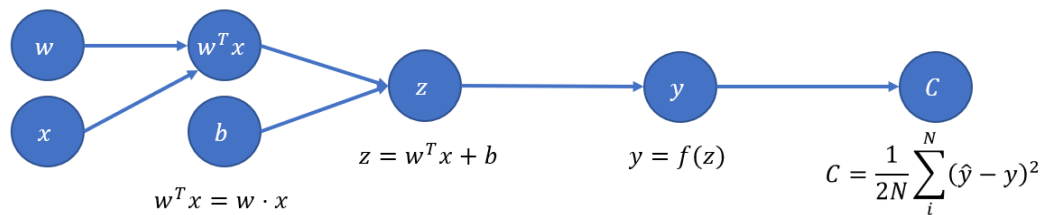


Figure 2.15: Computational graph for a neuron unit.

In adjusting any parameter of a NN, the partial derivative of the output w.r.t. said parameter is used to improve it by using the downwards slope, to reduce the loss L :

$$\theta := \theta - \eta \frac{\partial C}{\partial \theta} \quad (2.2)$$

Keeping in mind the computational graph, the partial derivative for C w.r.t. a given parameter θ can instead be written as chained derivatives. By using the chain rule for

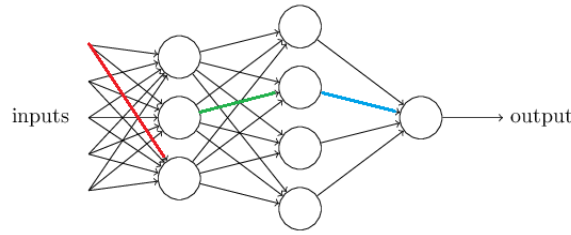


Figure 2.16: NN with 1 hidden layer, a basic perceptron. Original Figure from [Nie15].

differentiation and the definitions in fig. 2.15, we obtain:

$$w_i \leftarrow w_i - \eta \frac{\partial C}{\partial w_i} = w_i - \eta \frac{\partial C}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial w_i} = w_i - \eta(y - \hat{y})x_i \quad (2.3)$$

$$b \leftarrow b - \eta \frac{\partial C}{\partial b} = b - \eta \frac{\partial C}{\partial y} \frac{\partial y}{\partial z} \frac{\partial z}{\partial b} = b - \eta(y - \hat{y}) \quad (2.4)$$

... and given enough training, this neuron would eventually result in $w = [1, 0, \dots, 0]$ and $b = 0$, so that any output to is simply $y = a = f(z) = z = w \cdot x + b = [1, 0, \dots, 0] \cdot x + 0 = x_0$.

Perceptrons

It takes only a little amount of random experimentation to see that a singular neuron has limited usefulness. For example, one neuron is insufficient for computing Boolean logic with exactly 0 and 1 as output; e.g. AND, OR, NAND logical gates require multiple neurons. Neurons can be placed side-by-side and plugged into one another, so that outputs can be fed as inputs to other neurons. With these two architectural decisions, and different functions used, NNs can be trained to distinguish hand-drawn digits on greyscale images [Lec+98], with a decent level of correctness. This is an instance of *image classification*, one of various tasks under the category of object detection. (Object detection subtasks are distinguished by 1) visual precision, e.g. classify the whole image, regions, pixel-level precision a.k.a. masks; and 2) quantity of objects to detect, i.e. one or multiple objects in the image.)

Such NNs are known as *perceptrons*. With these changes, various new features and concepts should be outlined, with more to come later.

Loss functions. Mean squared error is also known as L2 loss, whereas L1 loss replaces the squared error with the absolute error $|\hat{y} - y|$. These are typically used for *regression* tasks, and MSE is unfit for data where anomalies result in huge variance. Various functions are better suited to select purposes. E.g. cross entropy (eq. 2.5) is suited for

binary classification, whereas the softmax function (eq. 2.6) is considered to convert NN outputs from a dense layer to multiple probabilities, beyond two classes.

$$\text{Cross entropy loss } L = -\frac{1}{N} \sum_i^N [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] \quad (2.5)$$

$$\text{Softmax loss } L(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \quad (2.6)$$

Activation functions. The output from individual neurons are called *activations*. There are many of these, with variations built upon them:

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \quad (2.7)$$

$$\text{ReLU}(x) = \max(0, x) \quad (2.8)$$

$$\text{Leaky ReLU}(x) = x \text{ if } x > 0, \text{ otherwise } 0.01x \quad (2.9)$$

... and there are various desired properties for such functions: non-linear, finite range, continuously differentiable, monotonic (also in the derivative) [DHS00].

Layers. In fig. 2.16, there are three layers. The input layer is the first column of nodes, and the output layer is the last node. «Hidden layer» denotes every layer between. Each layer has a set number of computational units, and for perceptrons, every node in one layer is connected to every node in the adjacent layers. Each layer can vary in the number of inputs and outputs.

Matrix notation. The typical input for this perceptron can be a vector. However, multiple inputs can be fed simultaneously if we instead use matrices. When computing the output y (which may be a vector too), matrices can be used instead, with special indexing such that $w_{j,i}^l$ denotes the weight (or the edge) in layer l , connecting node j in column $l + 1$ with node i in column l ; «to-from» indexing, so to speak, with the leftmost layer starting at index 1; now, $\mathbf{z} = w\mathbf{x} + \mathbf{b}$, where $\mathbf{z}, \mathbf{x}, \mathbf{b}$ are column vectors. E.g. $w_{3,1}^1$, $w_{2,2}^2$ and $w_{1,2}^3$ correspond to the red, green and blue edges in fig. 2.16. In the same vein, z_i^l denotes z for node i in layer l . Activation functions may now be *vectorised* such that a function $f(\mathbf{z}^l)$ simply outputs $\mathbf{a} = [f(z_1^l), f(z_2^l) \dots f(z_n^l)]$.

Forward pass, backpropagation, gradient descent. The cascading computations leading up to the calculation of loss/error, is known as a forward pass. The outputs in the first layer affect the output of the following layers, so the partial derivatives of earlier layers depend on the partial derivatives of the following layers. This means that some of the computation is already done by the time we update the parameters of the earliest layers. By using the chain rule and defining selected parts of the derivatives, the following formulas can be used in a manner likened to dynamic programming, to compute the parameter adjustment in the immediately preceding layers precisely once, rather than wasting time on repeated computations [Nie15]:

$$\delta^L = \nabla_a C \odot f'(\mathbf{z}^L) = \frac{\partial C}{\partial a^L} \odot f'(\mathbf{z}^L) \quad (2.10)$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot f'(\mathbf{z}^l) \quad (2.11)$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (2.12)$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad (2.13)$$

... where L is the index for the last layer, \odot denotes the Hadamard product; i.e. $[2, 3] \odot [1, 4] = [2 \times 1, 3 \times 4] = [2, 12]$. These formulas provide a notion that a parameter's update depends on its input, and the resulting error that is propagated thereafter.

This type of update method is **gradient descent**. In literature it is called *stochastic* when each training example individually updates parameters. In DL, batches of varying sizes are generally used, where simultaneously computed training examples have their parameter updates aggregated, further processed in some way (such as batch normalisation), and then applied. Other variants use the previously computed gradients in addition to a newly computed one; this is gradient descent **with momentum**, and such variations are commonly used to deal with the problems inherent to «standard» gradient descent [Rud16]. (Fundamentally different update methods, such as genetic algorithms, are used not in Deep Learning but in other types of Machine Learning.)

These layers of neurons are also known as *fully connected layers* or *dense layers*. Equations eq. (2.10) to eq. (2.13) apply to such layers, and are shown to illustrate that backpropagation is a general strategy for computing parameter updates.

2.2.2 Convolutional neural networks (CNNs)

The perceptron was introduced back in 1958, but the development of power NNs began with usage of **convolutional layers** [Lec+98], and increasing computational power. When feeding a NN with images as input, they can be flattened into vectors such that perceptrons such as fig. 2.16 can be used, with entire images being given as input to one layer of neurons. There is an inherent flaw in this however: spatial structures are not recognised, so perceptrons are lacking in capacity to capture complex patterns. But other operations can be performed, such as **convolution**, which does not use this flattening.

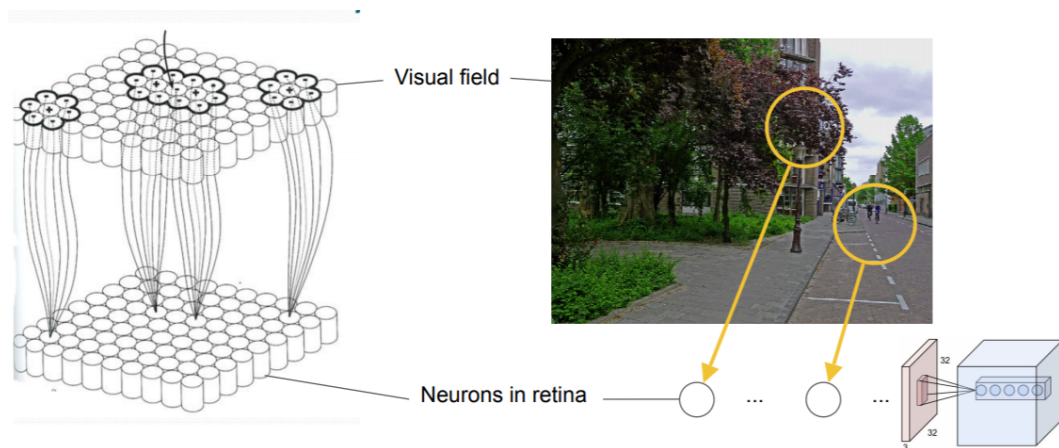


Figure 2.17: The receptive field in the human eye vs. convolution in NNs. Figure from TDT4265 Computer Vision & Deep Learning.

Convolution serves as a visual method to detect patterns, somewhat similar to the human visual system. **Convolutional filters** such as the 3×3 *kernel* in fig. 2.18 look for specific patterns, and output **activation maps** where patterns are detected. Visually speaking, these can look like heatmaps, highlighting something like an ear, or a nose. Like neurons, there can be multiple of these in the same layer, resulting in filters looking for various patterns. These kernels can converge such that they recognise certain patterns such as edges, corners, and so on. Given the right architecture, they can distinguish all kinds of features, ranging from basic elements like edges and corners to complex structures like human faces, where a visual hierarchy exists.

Afterwards it is common to use **pooling**; it makes more sense when the input image is notably large such as 28×28 , or greater. This *downsampling* operation is typically used to compress information and reduce NN/model complexity; it also has no parameters of its own, and therefore does not contribute to increased memory consumption when

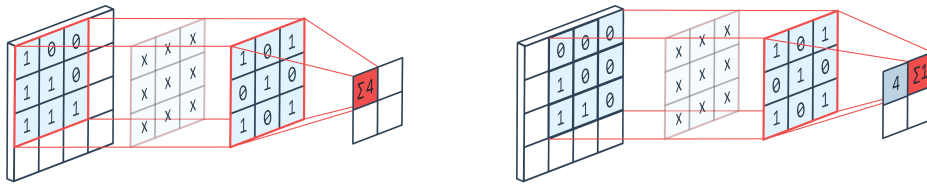


Figure 2.18: Convolution operation. Filter size = 3×3 , padding=0, stride=1. Figure from [Pel20]

training a NN.

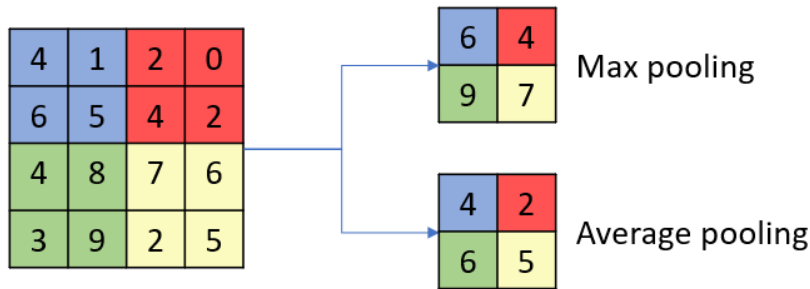


Figure 2.19: Max pooling is commonly used.

At this point, some more concepts and terms should be introduced.

Filter size. Convolutional filters are typically odd in size: 1×1 , 3×3 , 5×5 . A convolutional layer may contain multiple filters, usually of the same size. Each filter is constructed of parameters that can be updated. A filter has weights $N \times N + 1$ bias, so a full layer with K filters usually has $K \times C \times N^2 + K$ learnable parameters. C is the number of channels in the input image. E.g. colour images have channels red, green, and blue, so $C = 3$.

Padding. Before performing convolution, the original image may be appended on each side with a number of pixels, to account for the dimension reduction inherent to convolution as shown in fig. 2.18. E.g. to get the same output size when using a 3×3 filter, the input image must be padded with 1 pixel in all four directions. Zero-padding is the most commonly used version, i.e. the new pixels are given 0 in value, but this may vary.

Strides. When computing convolutions, the filter may skip possible locations, which also reduces the size of the output image. Stride = x means that every x locations are

used to compute an output. E.g. stride = 2 outputs an image that is approximately half the size of the input image.

Feature map. A conv. layer with K filters produces K feature maps, also known as *activation maps*. These correspond to the right-most layer in fig. 2.18. These feature maps feed may feed their output to yet another conv. layer, but this is typically done after a **pooling layer**, which serves to reduce the size of the outputted images and number of parameters.

Architecture and complexity. NNs have specific architectures, with details such as layers chained together in given manners, loss functions used, resolution specified. The number of parameters between fully connected layers of neurons, is very high: for a 28x28 image, there are 784 weights + 28 biases, which is about 800 parameters to update. Using 8 convolutional filters of size 3x3, the number of weights are reduced to $8 \times 3 \times 3 + 8 = 80$. Convolution is one of various techniques used to produce better NNs, with greater complexity for low(er) computational demands.

More about NNs

There are various problems, terms, notions, techniques, and additional details surrounding NNs.

Supervised vs. unsupervised learning. In the context of DL, supervised learning means that a desired output is defined, and used to enable learning; the method of learning is also explicitly defined. E.g. the input-output relation $y = F(x)$, should be learned by a NN by humans **explicitly** giving information about y , i.e. labelling data, and using an error function. However, there are ways to emulate an unsupervised setting, to be demonstrated later in section section 2.2.3.

Batch & epoch. One iteration through all the available training data, is called an **epoch**. NNs are typically trained on several thousands of examples, but even these are reused, by training NNs for several epochs. During training it is common to compute parameter updates resulting from multiple training examples at the same time; the number of such parallel computations is the batch size.

Regularisation. The loss function used at the output layer can vary, as they have different capabilities. A scaled, weight-dependent function is frequently appended to the loss function such that the total loss $L' = L + \lambda \sum_i |w|$ or $L' = L + \lambda \sum_i w^2$, where i iterates over a given batch, and λ is the regularisation rate. Using $|w|$ is known as L1 regularisation, while w^2 is called L2 regularisation.

Data for NNs. For a generic task, there is a training set, validation set, and a test set. The training set, as implied, is used to train a NN so it hopefully learns to solve a given task. The validation set is a fraction of the full training data, not used for training (i.e. parameter updates) but rather to check performance; it helps in detecting *overfit*. Loss is calculated when validation data is used to input, and used as a measure of performance on *unseen data*. After all the training is completed — depending on some user-defined criteria — there may be a test set. Test sets are typically intended to be representative of *future* data, i.e. benchmark testing the application of a NN. Sometimes there may not be test data available, however.

Additionally, the quality of a NN depends on the data it is trained on. If the training set does not adequately describe the problem to solve, the NN cannot be expected to generalise well on all possible future data. This illustrates the general principle of «garbage in, garbage out».

Data augmentation can also be used to increase the size of a dataset. This means to make small variations based on the original data; images can be mirrored, rotated, and so on. However, this is not appropriate when the problem to solve barely has any such cases; e.g. frontal images of human faces are generally aligned with eyes above the mouth and ears aligned at the same height, at the sides of the image. Forcing a NN to account for orientation in addition to other tasks, is pointless (especially if this can be done by other, computationally cheaper methods).

Optimisation. NNs use not only mathematically proven but statistically proven methods, in order to find the optimal parameters for solving a given problem. However, convergence towards the optimal solutions — i.e. parameters that give the minimal loss — are not always guaranteed to reach the global loss minimum. When a given NN is training and instead finds a local minimum, it may be stuck if the learning method is not fit to handle this. Variations of gradient descent using previous parameter updates, are common [Rud16]. Additionally, lower loss does *not* imply a better solution, because no dataset can fully describe a problem. Optimisation also requires that parameter

changes are meaningful, i.e. not zero over longer periods of training.

Gradient problems. The backpropagation method is based on computing chained derivatives. When these numbers are all smaller than 1 (in magnitude), a sufficient number of factors smaller than 1 will lead to a very small product. Parameter updates near 0 change nothing, leading to no learning. This is called the *vanishing gradient problem*, frequently leading to *underfit* problems and bad optimisation; «deeper» NNs with more layers tend to be more exposed to this problem, though there are solutions to this [He+15]. On the other end there is the problem of chained derivatives generally larger than 1. The product can become very large, and this is known as the *exploding gradient problem*, where the learning process has gone off the rails entirely, and the problem of *overfitting* occurs.

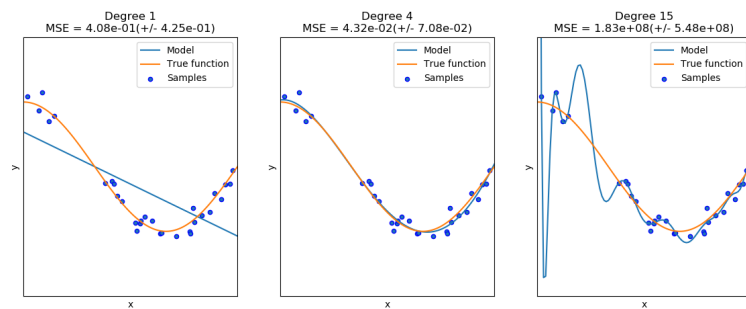


Figure 2.20: Left to right: underfit, proper generalisation, overfit. Figure from sci-kit learn [dev19].

Overfit and underfit. A good NN is said to be generalised. But a NN that is not, is typically suffering from either overfitting or underfitting. Overfitting means that the NN has grown too complex. Rather than learning to solve a given type of problem, it instead solves the exact problems given to it, like it has learned to memorise the given examples instead of generalise the intended problem; at this point, the NN is also said to have low bias and high variance.

Underfitting means that it never even came close to generalising, i.e. the NN's capacity for modelling the problem was not utilised or it is insufficient for modelling a solution to the given problem that is represented by the training data.

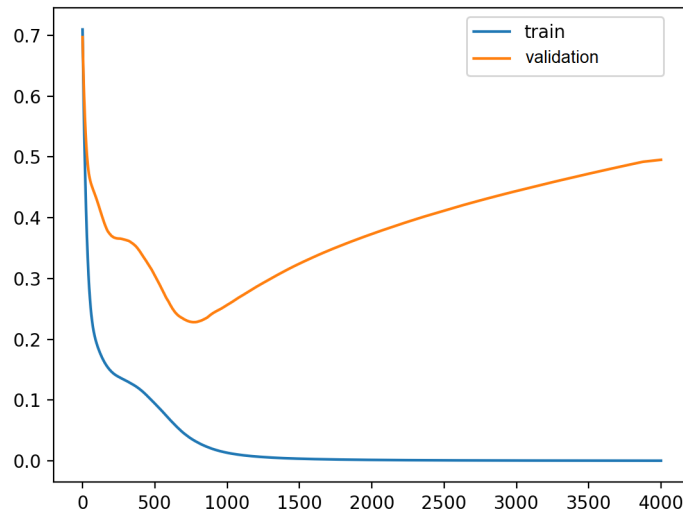


Figure 2.21: Loss changes during training. The test minimum marks the best state of the NN during training. Consistent increases in validation loss while training loss keeps decreasing, typically indicates overfitting. Original figure from [Bro18].

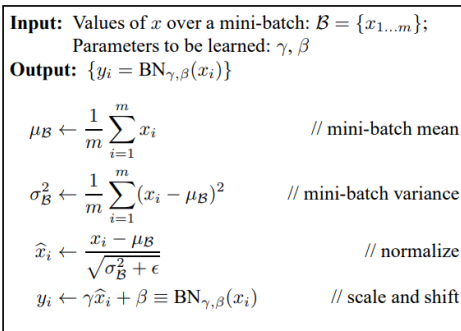


Figure 2.22: Batch normalisation algorithm. Figure from [IS15]

Normalisation. Between layers in a NN, various problems with inputs may occur. When inputs gain too much variation or large offsets, the **internal covariate shift problem** arises [IS15]. This in turn leads to saturation, which further results in the vanishing gradient problem. Batch normalisation is believed to handle this problem, and also often used. There are also other kinds of normalisation.

Upsampling. Some tasks in DL require that NNs not only analyse the input images, but return information about regions of interest; one such task is *image segmentation*; i.e. precisely mark and distinguish segments in an image, possibly per pixel. However, NNs typically condense information and reduce image size when using convolution and downsampling methods. Data reconstruction becomes necessary. This is typically performed with **transpose convolution**, which aims to perform the convolution process in reverse. This is a learned upsampling method, as opposed to naive methods such as attempting to do reversed max-pooling, that suffer from loss of information.

Hyperparameters. These are properties defined before any amount of training, typically kept constant throughout the training phase. Some of these are: batch size, input dimensions, output dimensions, learning rate, regularisation rate. For certain architectures and modes of training, such as ProGAN [Kar+17] or variations of gradient descent [Rud16], these can be adjusted during training.

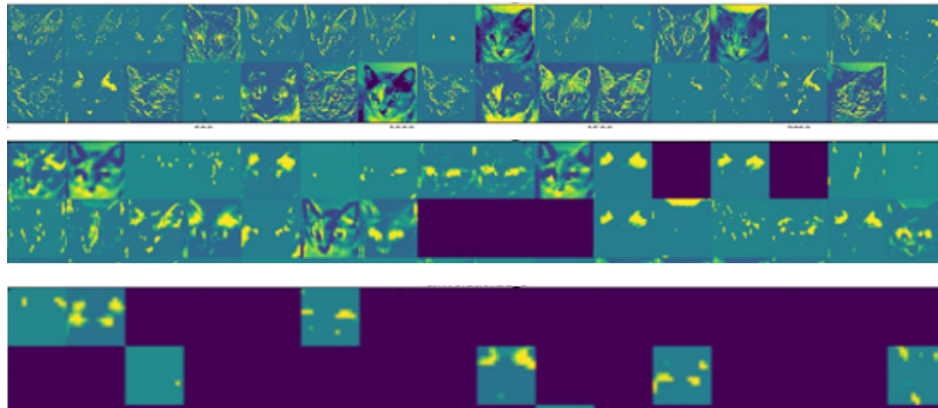


Figure 2.23: Conv. filter activations, when testing a NN tasked with detecting cats. Top to bottom: layers 5, 7, 8, with 32 filters shown from each. The deeper layers are increasingly abstracted away, with some filters entirely inactive.

Visualisation of NNs processing. An obvious question to ask is: what happens in the middle of input-output, during the feedforward process? What happens in the hidden layers? This is not necessarily easy to grasp, even if some meaningful visualisations can be made. E.g. activation maps in early or even middle layers of a deep NN designed to detect cats, may well highlight traits such as ears, eyes and cranial outline. The deeper layers become increasingly abstract however: they may well show a heatmap that highlights the cat only, but later, the abstractions become unintelligible.

Further reading for backpropagation through convolution and other functions such as transposed convolution are less consequential for understanding how NNs work, and is left to external resources [Jef16] [lab20] [ODO16].

2.2.3 Generative Adversarial Networks (GANs)

The original formulation for GANs by Goodfellow et al. [Goo+14] describes the following scenario, designed to make a NN that produces realistic images of some kind: consider a game where a generator \mathbf{G} forges some items, and a discriminator \mathbf{D} compares these to the «real» thing. \mathbf{D} is supposed to distinguish between the real and forged items,

and is told the correct answer after evaluation. Based on whatever judgements made by D, G adjusts its own methods, but D is also updating its evaluation method. Ideally, D and G learn together so that G eventually learns how to make the real thing, such that D may as well do a coinflip instead of making qualified judgements. In literature, discriminators are also called critics.

This can be formulated as a minimax game or zero-sum game, where G and D are working against each other. In this article, G and D were (multilayer) perceptrons, such that only D used a loss function, but also functioned as a loss function for G, since it made adjustments based on D and only indirectly used the loss function. It can therefore be said that GANs provide an indirect method of estimating a way to discern the real images from the fake ones; D acts as a learned loss function, to G. This also transforms the setting to an unsupervised one: G learns according to D, not any human-made labels; the labels are also trivial. G is also not meant to generalise well unto new data. (Using components from supervised learning is insufficient reason to call this supervised.)

In this paper, the following loss function, known as GAN loss or adversarial loss, is used:

$$\min_{\mathbf{G}} \max_{\mathbf{D}} V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \quad (2.14)$$

... but in practice, when training the generator, $\log D(G(z))$ is maximised rather than minimising $\log(1 - D(G(z)))$; i.e. gradient *ascent* is used on the generator. This is because early on, the generator output is easily distinguished from real data, such that $\log(1 - D)$ saturates and thus leads to bad learning.

The general idea with such NNs is that they take some input — a latent vector z , from an N-dimensional vector space — and output a «translation» of the input, $x = G(z)$, which should be virtually indistinguishable from «real» images. The generated output constitutes a density distribution, and the «real» data constitutes yet another distribution.

In the GAN paper, Goodfellow et al. [Goo+14] experimented with the **MNIST** dataset i.e. hand drawn digits, and successfully produced realistic fake samples. Another discovery was that an interpolated vector in the vector space of z (e.g. weighted averages), would present an interpolated result in the output space $G(z)$. In a later paper, this was done with human faces instead [RMC15], showing that such vector spaces are used in meaningful ways by generator NNs.

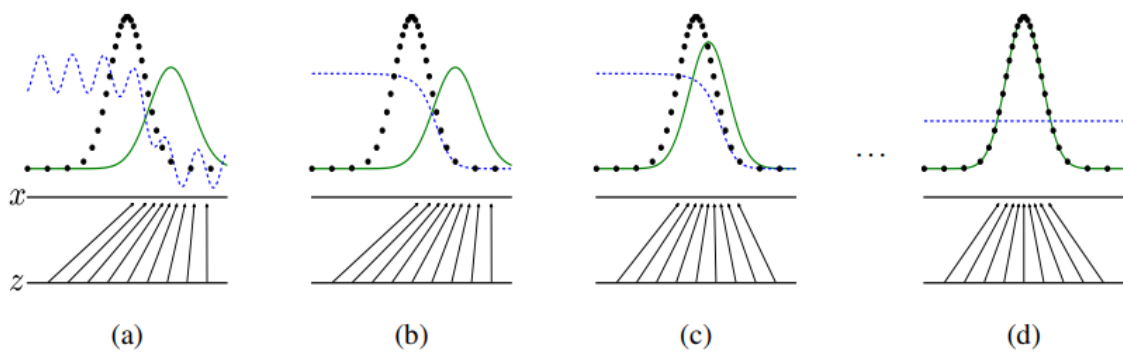


Figure 2.24: Green curve: density distribution of generator output $x = G(z)$. Blue curve: discriminative distribution. Black curve: density distribution of the «real» items, p_{data} . (a) Training scenario near convergence. (b) D converges to $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}$. (c) G converges such that p_g is closer to p_{data} , i.e. $x = G(z)$ is more likely to be classified as legitimate data. (D) G has successfully converged such that $p_g = p_{\text{data}}$, and at this point, $D(x) = 1/2$; i.e. it cannot tell which one is real. Figure from [Goo+14].



Figure 2.25: Goodfellow et al. found that interpolated images $G(z)$ could be generated by interpolating through the input space. Figure from [Goo+14].

This framework established a new paradigm in the problem of generating realistic images. GANs gained much popularity within just a few years, and it is one of various techniques that could be feasibly used for creating «deepfakes», e.g. fake images and «photoshopped videos», where faces are replaced seamlessly; the latter is created more so with other techniques [Ade20], due to limitations with GANs.

Various findings have followed. Some «newfound» applications for GANs are: super-resolution (upscaling), high-resolution generation [Kar+17], style transfer [Iso+17] [Zhu+17] (and by extension, reversing pixel-level classification of images); i.e. semantic image synthesis [Par+19], see fig. 2.26. All of these are variants of **image translation**, and have been achieved with contributions from research in «classical» object detection tasks such as image classification [He+15], i.e. say what is in the image but not where; and image segmentation [RFB15], i.e. classify every pixel as part of something.

Like other NNs, however, GANs also have problems. Goodfellow et al. discovered *mode collapse*: G starts to mislead D on what a real item really is. An indicator of this is when G produces approximately the same output for (many) different inputs. Another problem is **artifacts**, patterns that show up repeatedly in completely unwanted

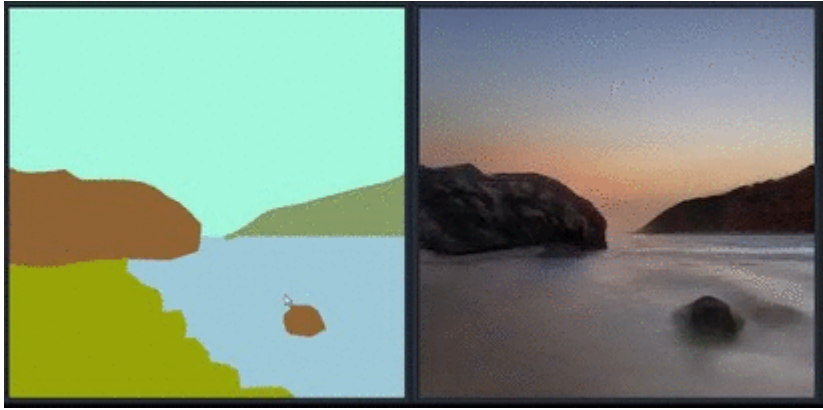


Figure 2.26: Image synthesis using a GAN produced by Nvidia. This GAN takes labelled pixels as input, «undoing» semantic labelling of images by returning what could have been the original image. Each colour corresponds to a given class; e.g. blue = water, cyan = sky, green = general terrain, brown = rocky surfaces. Figure from [Par+19].

manners, such as unintelligible symbols placed on maps for no apparent reason.

Validation sets are a somewhat invalid concept when it comes to GANs, as these are supposed to generate data instead of making some conclusion on given data. Also, the loss values observed during training, do not indicate anything beyond *relative* performance; this is can be made very apparent in training, especially if the learning efficiency between G and D is matching. In such a scenario, their losses would oscillate and be near the same average.

Key innovations (and implementations) for this thesis are the following GANs: CycleGAN, Pix2Pix, and WGANs, presented later in chapter 3.

3. Related Work

3.1 Map generalisation

Comparisons with methods in the following papers is not necessarily viable; some use algorithmic, well-defined methods. GANs learn such methods implicitly and define them internally without any clear translation to instructions for replication. NN mechanisms may be difficult to interpret at any given step. Additionally, «traditional» methods are typically based on vector data, whereas GANs use raster data. The results are more relevant than the methods themselves, as benchmarks for comparison.

3.1.1 Transferring multiscale map styles using GANs [KGR19]

The theme of this research article is the closest to this thesis' theme. The authors use two datasets with approximately 870 and 1850 training images (substantially fewer than this thesis), but these datasets are also not particularly diverse.



Figure 3.1: Map generalisation applied on *simple styled maps* using Pix2Pix, to the Google Maps equivalent, on zoom levels 15 (left) and 18 (right). Figure from the article.

Kang et al. use the GANs Pix2Pix [Iso+17] and CycleGAN [Zhu+17] to translate «simple styled maps» based on OpenStreetMap (OSM) data, to Google Maps. Papers on these two will be introduced later in chapter. In a sense, this is generalisation of raw data to maps.

Various problems are likely inherent to the authors' method: labels, symbols, text-like artifacts are generated in incoherent ways or simply absent, generally because the input data contains no such things, so there is no clear input-output relation. CycleGAN produced less consistent geometry, but areas such as lakes and grasslands were not reliably translated. The authors note that colour diversity may improve when expanding geographic extents and hence diversity of data.

Three generalisation methods are noted, and likely employed by the generators of both GANs: enhancement (or exaggeration), selection (or omission), typification (representing multiple points of interest with one).

3.1.2 Building generalization using Deep Learning [SFT18]

In this paper, the authors use a NN that is *very* similar to U-Net, to perform generalisation, choosing to stop down-sampling at a higher resolution. Worth mentioning is that this does not use a discriminator, and is therefore not a GAN. It is used to perform semantic segmentation, i.e. pixel-level classification.

The experiments of this paper are limited to **binary images** based on OSM data. The output is compared to generalisations made using a generalisation program called CHANGE. The target map scales are 1:10 000, 1: 15 000, and 1:25 000. The authors intend for their NN to successfully integrate the following operations in a holistic manner: selection, trimming/simplification, aggregation, exaggeration, displacement, and «typification» (change of representation).

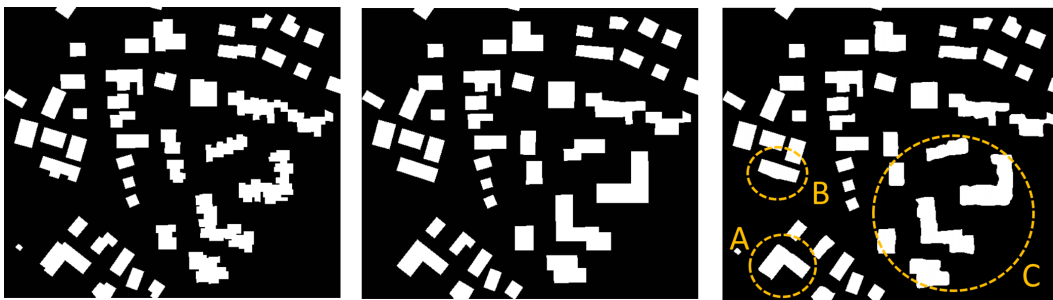


Figure 3.2: Left: original input. Centre: generalisation by CHANGE. Right: experiment result. Figures from the article.

The authors use output from CHANGE as a baseline for measuring performance quantitatively; pixel-level intersection-over-union ($\text{IoU} = \frac{A \cap B}{A \cup B}$) is used to capture meaningful accuracy values, as opposed to pixel-based accuracy (related to true/false positives/negatives), i.e. terms used in confusion matrices. This is because IoU ignores

areas without buildings; blank spaces should obviously not contribute to a generalisation metric.

The authors conclude that their NN fails only in producing outputs that are regularly rectangular or parallel; i.e. non-sharp edges are a consistent difference between CHANGE and their NN. While the authors suggest post-processing to remedy this issue, it defeats the purpose of using NNs to perform *all* generalisations. Still, the authors note the following observations in their results: simplification, omission, aggregation, trimming (of rough edges).

3.1.3 A heuristic approach to the generalization of complex building groups in urban villages [YZZ19]

The focus of this paper is aggregating tightly clustered buildings for multiscale generalisations. These clusters are referred to as «urban villages» by the authors. The heuristic method used relies on Delaunay triangulation so that it «aggregates and simplifies each building group separately». The heuristics used are «*polygonal area (PA)* and *length of face-to-face boundaries of adjacent buildings (LFBAB)*»; both are derived from Delaunay triangulation. This method deals with various superfluous details such as minor angular offsets, small polygons, trims jagged boundaries, and uses a gap space determined from the Delaunay triangulation method.



Figure 3.3: (a) Regular neighbourhood vs. (b) urban village. Figures from the article.

The authors work on only two specific regions, but note that the building-to-building spacing is usually less than 0.2 mm on map scales 1:10 000. Traditional distance indicators are fit for regular neighbourhoods, and would merge entire urban villages as a singular block.

In this procedure, buildings are joined together (aggregation), followed by some simplifications, and then trimming jagged boundaries. The last step is performed by

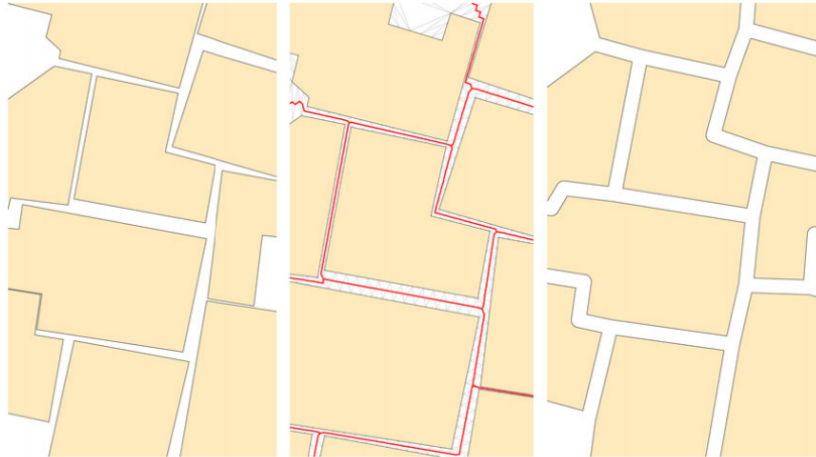


Figure 3.4: Left: input. Middle: buildings aggregated, with «skeleton lines» drawn on remaining streets. Right: trimming performed with a buffer around skeleton lines, resulting in exaggeration for streets. Figure from the article.

using a buffer for «skeleton lines»; these lines mark the middle of remaining streets. This therefore results in exaggeration too, as smaller streets are given more space in the map. Some small, individual buildings are removed entirely, so some level of omission has taken place.

3.1.4 Specifying Map Requirements for Automated Generalization of Topographic Data [Sto+09]

In this paper, Stoter et al. aim experiment with producing a generalisation process that does not require human interaction in any intermediate step, and begin by using available 1:10k data from the Dutch national mapping agency (NMA), the Kadaster. This data is object oriented, and does not distinguish database- vs. cartographic representation.

The vector datasets used already have some level of generalisation applied, such as exaggeration and displacement of motorways. The distinction between such operations as a matter of modelling or cartography is lacking, so the authors integrate requirements for model- *and* cartographic generalisation. Additional precision can be acquired by simply using more accurate data such as 1:10k datasets, referred to as TOP10NL. The maps produced by the authors are referred to as TOP50, which is namely at the scale 1:50k.

The experiments focused on the following topographic classes: buildings, roads, and land use, with the rationale that automating these types of generalisation would be

very efficient for all future production of maps. The authors also argue that quality of the final result of a generalisation process is also mostly based on these features, and that these must be generalised in a holistic manner because the processes interact with one another.

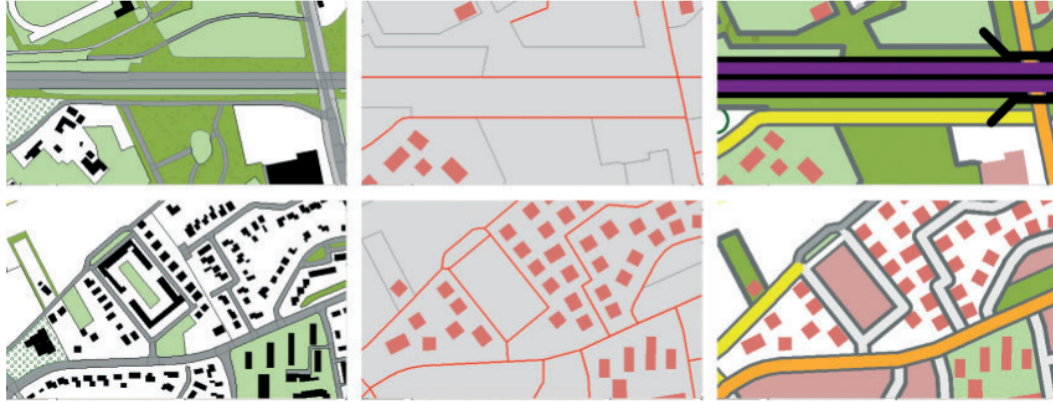


Figure 3.5: Generalisation example. Left: TOP10NL. Middle: TOP50 vector data. Right: TOP50 maps produced automatically. Figure from the article.

During production, the authors use various means to discover requirements and also find how data may be in an undesirable format: in the construction of TOP50vector data, roads are shown by centerlines, whereas in TOP10NL, each centerline is an individual lane. This was one of various data problems to deal with.

In the results, various types of generalisations are made: road polygons may be collapsed to polylines, connected polygons may be aggregated, some objects are exaggerated and others are moved to make space, and colouring varies depending on context (classification). In fig. 3.5, roads are classified with 5 unique colours.

3.1.5 Fully automated generalization of a 1:50k map from 1:10k data [Sto+14]

This paper focuses on presenting a workflow using more traditional methods to develop a fully automated process; additionally, the result was considered an acceptable replacement for existing 1:50k maps. It is arguably a continuation of the previous paper, with the same lead author but a different team. The tools included were ArcGIS Desktop 10.0, FME tools, and self-made tools in Python. Data is reclassified, road representation ignores lane count, (road/street) networks are pruned, «enclosing holes» are closed (e.g. gaps in roads), etc.



Figure 3.6: Generalisation example: 1:10k to 1:50k. Figure from article.

In total, the authors use **200 generalisation operations** in a «context-aware» model (e.g. urban vs. rural environments are treated differently), to generalise the entirety of the Netherlands. All the types of generalisation mentioned in chapter 2 are used.

The generalisation method in this paper is very comprehensive; processing all the TOP10NL data requires 50 hours for the Netherlands, which is about 2GB [Kad13] on unspecified hardware. The authors also note that user input was crucial for the development of particular methods used. User-centric requirements such as «currency and recency of geographical information», as opposed to fulfilling cartographic guidelines, were considered more valuable.

3.2 GANs

The following sections present the most significant contributions relevant for this thesis, i.e. methods to perform image translation and hopefully cartographic generalisation.

3.2.1 Conditional GANs (cGANs) [MO14]

Goodfellow et al. mention the possibility of producing *conditional generative models* in the original GAN paper [Goo+14]; various researchers have made such attempts since then.

In this paper, the authors use the desired output y to enable «conditioning» of the model (in the NN). The generator is given two inputs simultaneously: a random noise vector z and «real» image y , such that it outputs $x = G(z, y)$, where x should mimic «real» data. The discriminator is given both $G(z, y), y$. The authors tested this on the MNIST dataset, and produced promising results, while admitting that these results were «extremely preliminary».

The authors used this GAN also to annotate images with descriptive tags, e.g. a picture of a river being tagged with «river, waters, creek».

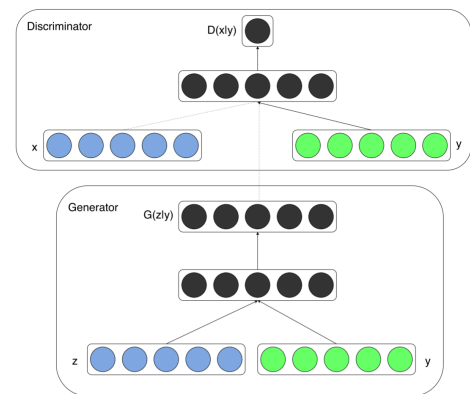


Figure 3.7: Diagram for conditional GAN. Figure from the article.

3.2.2 Pix2Pix [Iso+17]

Also building upon the idea of conditional GANs, Isola et al. compare an encoder-decoder architecture vs. U-Net, with and without a *conditional setting*; i.e. for every input, generated output should take the form of pre-determined output. The task is to convert images of one format to another.

Encoder-decoder architectures (not to be confused with auto-encoders) attempt to efficiently compress information to a more «compact» representation. This is then followed by upsampling to rebuild the original data, with perfect recreation being the desired goal.

U-net is in part based on the architecture of encoder-decoders. U-net has the benefit of using **skip connections** in the generator, presented in the ResNet paper [He+15]. When using skip connections, the outputs of one layer are passed along to the next layer of matching input dimensions. E.g. skipping one layer provides an input $x_{i+1} = F(x_i) + x_i$, where F can be some function like convolution.

Meanwhile, the discriminator D in this setting is a 70×70 *PatchGAN* classifier, penalising structure in patches sized at 70×70 pixels [LW16]. D can also be described

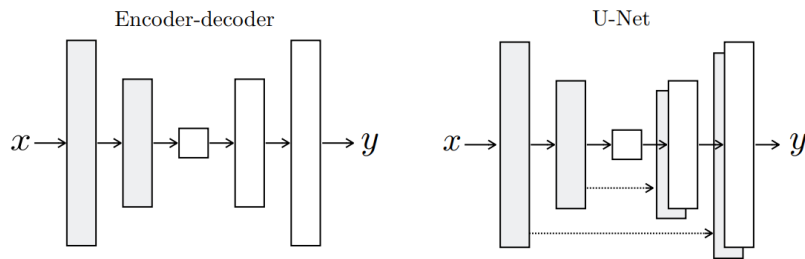


Figure 3.8: NN architectures tested by Isola et al: encoder-decoder vs. U-net. Figure from the article.

as a CNN, and the size of these patches is implicitly determined by the architecture [Iso17].

The authors did indeed show that putting GANs in a conditional setting improved image translation performance. However, they found that z was outright *ignored*, and promptly discarded it. Instead, noise was simulated through *dropout*; a mode of training where some neurons in fully connected layers, are turned on and off during training, enabling better learning at the price of requiring more training.

A problem discovered with the usage of transpose convolution in GANs, however, is grid-like artifacts. These may arise naturally due to the arithmetic of transpose convolution layers; output pixel cells may overlap, such that some pixel cells are given two values added together rather than just one. This is particularly visible on bright pixels. Such artifacts may be handled by well trained GANs to the point that these are impossible to see at first glance, but it is an issue inherent to «naive» transpose convolution, whereas alternative methods could help with this [ODO16].

3.2.3 CycleGAN [Zhu+17]

In the task of image translation, an analogy can be made regarding languages. Given sufficient capacity, objects from different sets can be translated back and forth fully intact. In that sense, bijective translations describe a scenario where two functions F , G and domains X , Y operate as follows: $G : X \rightarrow Y$ and $F : Y \rightarrow X$. In this paper, the goal is to produce such translations, indicated when $x \approx F(G(x))$ and $y \approx G(F(y))$.

In this scenario, two pairs of generator-discriminator are trained at the same time, using a least-squares GAN loss. To enforce the notion of information preservation after fulfilling a cycle, i.e. comparing x vs $G(F(x))$, a «cycle consistency loss» is appended to the loss function. The authors also find use for of an identity loss which enforces the notion that, if $G : X \rightarrow Y$ is given an input $y \in Y$, it should still output y .

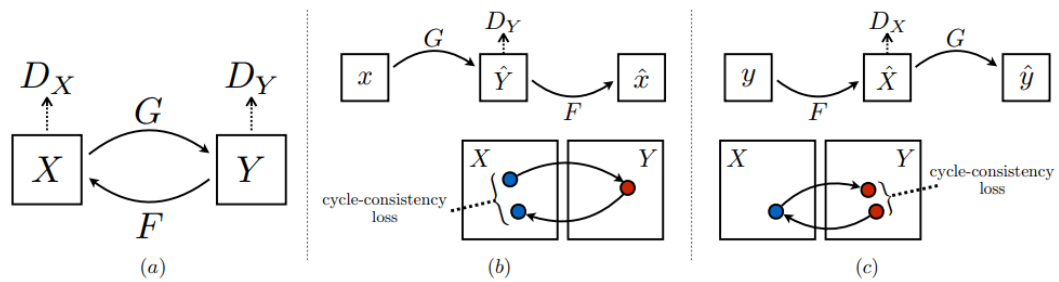


Figure 3.9: CycleGAN concept. Figure from the article.

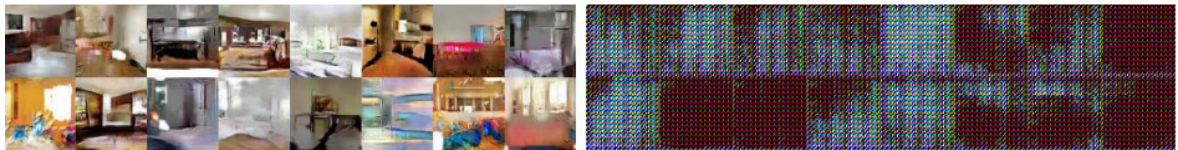


Figure 3.10: Left: WGAN with DCGAN architecture without batch normalisation. Right: standard GAN, which has failed completely to learn anything. Figure from the article.

3.2.4 Wasserstein GAN (WGAN) [ACB17]

In this paper, a different loss function is suggested, which is the defining trait of WGANs. The inspiration for this is a metric known as the *Earth Mover distance*, which can be described as: the minimum loss required to change one distribution to another, or by analogy, the minimum required to change one pile of dirt to another. The formulation for this distance is as follows:

$$W(\mathbb{P}_1, \mathbb{P}_2) = \inf_{\gamma \in \Pi(\mathbb{P}_1, \mathbb{P}_2)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|] \quad (3.1)$$

... where $\mathbb{P}_1, \mathbb{P}_2$ are density distributions of generated and desired output, and $\gamma(x, y)$ describes the amount of mass that must be transported from x to y .

The authors test this with the following architectures: DCGAN [RMC15], DCGAN without batch normalisation and using a constant number of filters across conv. layers, multi-layer perceptrons (MLPs, with ReLU). The experiments show that Wasserstein distance can outperform «adversarial loss» as defined by Goodfellow et al [Goo+14], on the LSUN dataset [Yu+15]. Using a DCGAN with batch normalisation, both WGAN and adversarial loss perform well, but the comparison changes with architecture, with WGAN beating adversarial loss.

The authors note that the Wasserstein distance can be approximated by using NNs in «compact spaces», i.e. the weights must be in some small range $[-c, c]$; this also implies that the full function that the NN represents, is K -Lipschitz, i.e. the derivative

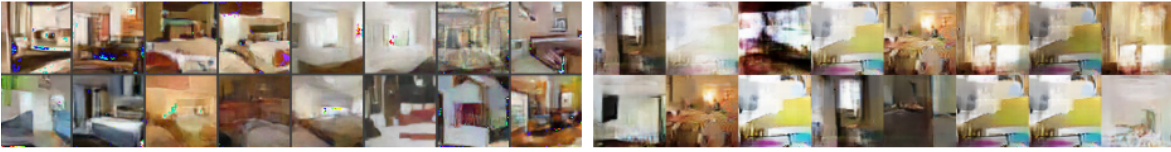


Figure 3.11: Left: WGAN on a MLP architecture. Right: standard GAN using a MLP architecture, demonstrably suffering from mode collapse, i.e. the same outputs are produced multiple times. Figure from the article.

of said function is, in magnitude, always smaller than K , which in turn depends on the compact space. The authors suggest limiting weights with $c = 0.01$ to enforce the constraint of compact space, though they admit that this is a terrible method. The next paper improves upon this issue.

A inherent benefit of using such a loss function, however, is that it is now meaningful beyond merely relative performance for generator and discriminator. The objective is fulfilled when the loss function is zero, i.e. when the generated output has a virtually indistinguishable distribution compared to the desired output. For «adversarial loss» eq. (2.14), the loss terms show only relative performance.

3.2.5 Improved Training of Wasserstein GANs [Gul+17]

Building upon the work of the previous paper, the authors introduce another method to enforce constraints inherent to Wasserstein distance: **gradient penalty** (GP). In this method, weight clipping is dismissed in favour of an additional loss term, that favours parameter updates where the Euclidean norm is equal to 1. It is formulated as:

$$C = \underbrace{\mathbb{E}_{\tilde{x} \sim \mathbb{P}_g}[D(\tilde{x})] - \mathbb{E}_{x \sim \mathbb{P}_d}[D(x)]}_{\text{Wasserstein distance}} + \underbrace{\lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\tilde{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{Gradient penalty}} \quad (3.2)$$

... where $\|x\|$ denotes Euclidean norm. The authors recommend using $\lambda = 10$.

The authors argue by presenting experiments on toy datasets, i.e. relatively simple datasets, used for visualisation and lighter experiments. Weight clipping is a serious case of unused modelling capacity in NNs, and makes NNs biased to learning «very simple approximations», thus failing to learn about «higher moments» in the given datasets. Additionally, weight clipping requires careful tuning to avoid gradient problems, whereas GP avoids this problem altogether, resulting in stable learning for complicated NNs.

A particular result is a comparison of 4 types of GANs, where WGAN-GP is the only loss function across 7 architectures that manages to learn well, with minimal hyperparameter tuning. The authors even show that in a conditional setting (such as

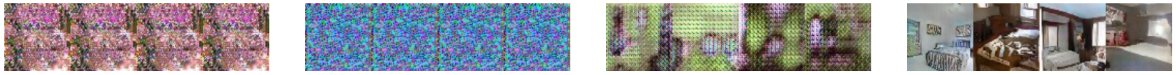


Figure 3.12: DCGAN vs. LSGAN vs. WGAN (clipping) vs. WGAN-GP. This is the largest architecture tested by the authors and only WGAN-GP manages to learn successfully.

Pix2Pix), WGAN-GP beats most competitors at the time of writing. The sole exception is SGAN, where a main pair of GANs are trained against an ensemble of GANs; each GAN in the ensemble learn in isolation and exist only to help the main GANs.

A peculiar benefit of WGAN-GP was found in the discriminator’s failure to learn from a 1000-image fragment of the MNIST dataset [Lec+98]: the training loss may increase while validation decreases, thus indicating overfitting (usually switched premises). When using weight clipping, both losses keep decreasing, which gives an illusion of learning in progress.

4. Methodology

As mentioned in the introduction, zoomout and content-wise generalisations are tested. Zoomout generalisation includes content-wise generalisation, but not the other way around.

Type 1: zoom-out generalisation. Using map tiles at zoom level z vs. level $z + 1$, the problem can be illustrated as in fig. 4.1. The «reconstructed» image on the left is made by using the 4 «underlying» tile images at zoom level $Z+1$, constituting a 512×512 image, and then using bilinear interpolation back to the original resolution 256×256 .

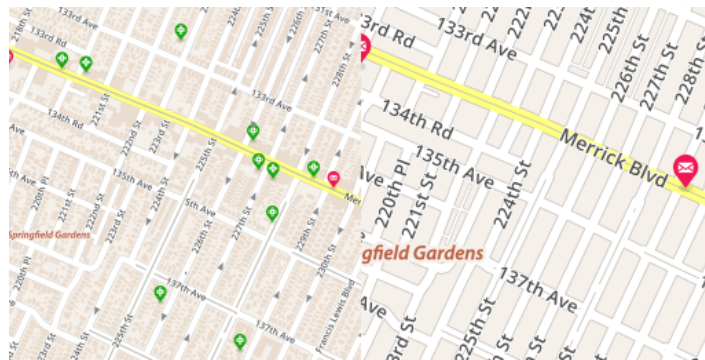


Figure 4.1: Image translation problem: going from details present in a larger scale (left) to a smaller scale (right). Michelin maps. The image contents respectively belong to zoom levels 17 and 16.

Type 2: content generalisation. Using tiles from different tilesets instead, with different levels of detail, the goal is that the more detailed maps are converted to simpler maps, one such example being OSM Mapnik into Michelin maps. See fig. 4.2. This is conceptually somewhat similar to style transfer [KLA18].

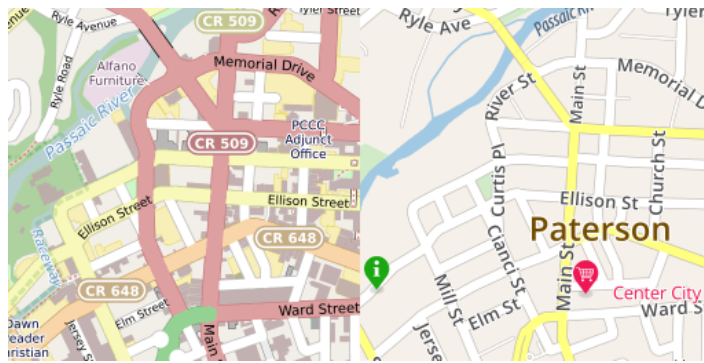


Figure 4.2: In this example, the goal is to generalise from the detailed OSM Mapnik format (l), to Michelin map format (r), using data from the same zoom level.

4.1 GANs & specifications

CycleGAN and Pix2Pix were used, using most of the original hyperparameters for training. The original architectures were used, and alternative versions with Wasserstein distance and gradient penalty were also tested. The implementations used are from code provided by the authors at [the official GitHub repository](#), based on PyTorch rather than the original frameworks. A spring 2019 version was used, and WGAN-GP variations were made for both, henceforth referred to as Pix2Pix-GP and CycleGP. The only architectural difference is use of a different cost function, nothing else.

There are multiple reasons for using resolution at 256x256 rather than 512x512: GANs are demonstrably capable of «fixing» resolution issues [Led+16] in the direction of blurry-to-sharp, so there is a possibility that unclear elements in general may be treated correctly. Computation costs and time may also become prohibitive, so a limit must be set. Other techniques of training may also be prohibitive, such as progressive growing of NNs [Kar+17].

4.2 Datasets

Datasets used are made of RGB images with a resolution of 256x256, that are predominantly static images from tilesets. Said tilesets use the same zoom levels. These were collected from the following sources: Mapbox, the Wikimedia Foundation, and Michelin maps. The tilesets all share the same projection, **EPSG:3857** or WGS 84/Pseudo Mercator, so individual tiles have equivalents in other styles with the exact same geographical extents. Pix2Pix in particular absolutely requires this.

The datasets used can be viewed on <https://mc.bbbike.org/> under the following

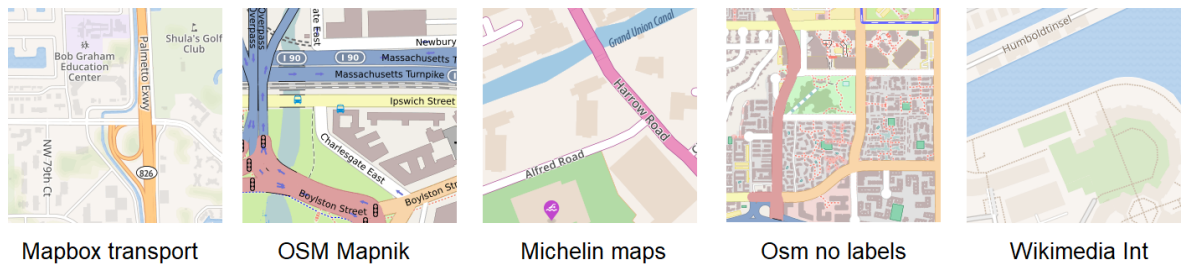


Figure 4.3: Tileset samples.

names and the given zoom levels. Specific urban locations were used such that each image contains plenty of information to generalise; otherwise there would be a risk of using predominantly homogeneous, low-entropy training samples such as images showing vegetation or waters only, where no generalisation is applied.

Tileset	Zoom levels used
Mapbox transport	15, 16
Michelin maps	15, 16, 17
OSM Mapnik	17
OSM no labels	15, 16
Wikimedia Int	17, 18

Table 4.1: Tilesets and zoom levels used.

Country	Locations
Germany	Berlin
Japan	Kyoto, Nagoya
United Kingdom	London
USA	Boston, Long Island, Los Angeles, Miami, New York City

Table 4.2: Tileset locations. Extents were selected to avoid spaces lacking visual content, i.e. blank or uniform areas.

Some tilesets are useless and were not used. E.g. Nagoya on Michelin maps is useless. Other tilesets were not used due to various issues such as lack of generalisation when zooming out, highly conflicting data (e.g. ESRI), different projections (e.g. Baidu tilesets), subscription requirement (Google maps), lack of data (national datasets), and so on.

The tilesets were used to construct the datasets using the combinations in table 4.3, with the given zoom levels. The goal is to enable a good translation from tileset A to B. For most *aligned* datasets, i.e. Pix2Pix datasets, this means that two images are put next to each other. Unaligned versions can be made, i.e. a CycleGAN dataset, where

images are split into groups/folders A and B. The other way around can be fulfilled in the context of this thesis.

During dataset construction, all unaligned datasets for CycleGAN were made from aligned datasets for Pix2Pix. This is not at all required for training unaligned datasets, but is done to ensure comparisons can be made when testing.

From ... to ...		Referred to as	Dataset size (imgs)
Mapbox transport-16	Mapbox transport-15	Mapbox	5 200
OSM Mapnik-17	Michelin-17	OSM Mapnik-Michelin	21 700
Michelin-16	Michelin-15	Michelin	2 300
OSM no labels-16	OSM no labels-15	OSM-no-labels	4 200
Wikimedia Int-18	Wikimedia Int -17	Wikimedia	3 900

Table 4.3: Datasets made and used for testing.

4.2.1 Dataset properties.

The input vs. desired output in these datasets have some relations that can be explained as follows, as though these methods were used to generalise the tilesets from one to the other. The datasets fit into either type 1 or type 2 generalisation.

Type 1: Mapbox. Buildings are unlikely to be generalised to any extent since they are only rendered differently between scales. Buildings are not generalised between the given zooms 16, 15, not even aggregated. Streets, paths, highways and such objects should be exaggerated further, as is the case in this data. Small symbols should be removed, including possible landmarks, but larger ones stay and be enlarged to maintain visibility. Waterbodies have horizontal wave-shaped textures. At a glance, this dataset does not appear to be too challenging.

Type 2: Mapnik-Michelin. Buildings *may* be omitted entirely from an image; or rather **deselected**. This dataset is inconsistent on whether to render buildings. If rendered, they should at least be aggregated across boundaries if not spatial gaps. Most text labels should generally be removed and replaced by whatever they are blocking. Other objects may also be moved. Various objects should be given different colours, i.e. classified differently, including roads and buildings. Because both input and target output is at the same scale, streets and such are not expected to be enlarged. Instead a relation should be determined, such that objects are rendered as the «corresponding» category, which may define properties such as colour and geometry.

Type 1: Michelin. Buildings will generally be deselected, but attractions may remain, particularly those marked with (Michelin?) stars on their labels. Streets should be enlarged, but small, green streets should be omitted. Symbols should be enlarged if they are not removed instead.

Type 1: OSM-no-labels. Adjacent buildings should be aggregated across boundaries, but not spatial gaps. Streets and such should be enlarged according to their relative significance and may in fact be reduced in size, to avoid distracting from the more important ones. Some types of streets are consistently removed. Textures such as trees regularly spaced on vegetation should be appropriately rescaled, whereas most symbols should be removed altogether. This dataset is somewhat distinct from all the others in that it has no text.

Type 1: Wikimedia. Adjacent buildings should be aggregated across boundaries but not spatial gaps. Those belonging to larger polygons denoting things like hospital areas, are turned to darker shades rather than having the standard building colour, so these are simultaneously reclassified. Streets and paths should generally be enlarged. Symbols should be enlarged and retain their labels. This dataset is also not one with significant demands, to the naked eye.

Ideally, these generalisations will be observed, but inconsistent patterns may result in inconsistent behaviour. There is always a problem in using not-self-made datasets, as treatments applied may be impossible to know. This is especially the case for Mapnik-Michelin, which is a particularly large dataset.

Datasets were not mixed with different tilesets in the case of type 1 generalisation, because the generalisations found in any given tileset can vary. Furthermore, it may be required to use *many* different tilesets rather than only five, to construct datasets representing the problem of *style-independent* generalisation. Presently, each dataset used for training represents generalisations for that particular dataset, such that datasets lacking aggregation are unlikely to produce a NN that performs aggregation.

No toy datasets have been used, such as datasets with significantly greater difference between input and output.

4.3 Hardware & software environment

Most of the training of NNs has been performed remotely on a Linux desktop PC with 2x Nvidia GeForce GTX 1080. Testing was done on a local Windows laptop with Nvidia GTX 1050 Ti, since testing has much lower requirements for swift performance.

The remote desktop used to train GANs for this project used Ubuntu 18.04 (Linux) as the OS. Python 3.7 from Anaconda 4.7.12 was the only programming language used, and all code is based on the PyTorch library, developed for computing with neural networks. Nvidia CUDA 10.1 was used, installed through Anaconda. PyTorch with GPU computation enabled is necessary for sufficient computation speed during training.

The laptop used an equivalent software environment.

4.4 Performance evaluation

Due to the inherent difficulty of evaluating GANs quantitatively, especially if errors arise, the results are evaluated by hands-on inspection, with criteria given in table 4.4.

While cost graphs for GANs may not necessarily indicate good performance, they may indicate bad performance *if* there is a trend of increasing cost. Note that whatever the discriminator does, that is not of interest. The purpose of GANs is to train a good generator, and the discriminator is only a method to do so.

Cartographic considerations	Observations
Buildings	Are these aggregated, simplified, etc? Rendered correctly, distinctly, where they should be, or even omitted/deselected appropriately?
Roads, paths, etc	Exaggerated, resized, or omitted? Are these still intelligible?
Geometric precision	Are edges straight where they should be? Do objects retain width, length and curvature correctly? Are polygons closed?
Symbols	Enlarged, omitted? E.g. hospital symbols are generally enlarged.
Land use	Water bodies, vegetation, landmarks, highlighted areas correctly rendered? Are some even ignored entirely i.e. rendered as background?
Miscolouring	Are some objects mistakenly given a wrong colour, and by implication a different classification? E.g. blue is reserved for water bodies usually.
Text labels	Though text is not expected to be intelligible due to results in [KGR19], can text be a problem from the input? How does it interact with other elements?
Technical aspects	Observations
Noise/blur	Does the output suffer from random graphical noise/jitter?
Artifacts	Are there certain patterns that make no sense but show up repeatedly in generated images?
Misinterpretations	Is anything rendered as something entirely different, indicating all-encompassing issues?
Mode collapse	Is the input consistently transformed to a mistaken type of output, partially or entirely? The above technical details are used to judge this.

Table 4.4: Performance metrics for NNs, based on properties from datasets used.

5. Results

Test samples for each dataset are generated and used by each NN specification. One test set with 50 input images is made per dataset, for a total of 250 test images, to compare NNs on the same images when possible. Images shown here may be cropped for presentation purposes.

During testing of Pix2Pix(-GP), images shown by the test code are, left to right: **input** image, **generated** image, and **target** output image. The second is naturally used to evaluate NN performance.

For CycleGAN there are three outputs that are of interest, when translating images from set A to set B : **real_A** (original image), **fake_B** = $G(\text{real_A})$, and **rec_A** = $F(\text{fake_B})$. Generally speaking, **fake_B** is used to evaluate CycleGAN(-GP). There is one exception to this for CycleGAN alone, for unknown reasons: the Michelin dataset seems to have no generalisations applied to **fake_B** but instead to **rec_A**; the same applies in the translation direction B to A . For pragmatic reasons, **rec_A** will be used in that specific case.

In the figures presenting cost graphs, note that a local minimum need not indicate good performance, and that some numerical values are severe indicators. This is especially the case if there is still a severe disparity between generator vs. discriminator performance.

For implementations using WGAN-GP, it is possible to converge at minimums because such a NN tries to minimize differences. However, minimums may be local rather than global, and rather bad too.

For Pix2Pix, the cost **G_GAN** (in **in blue**) is of interest, and if it is above 1, generated output is likely bad. If **G_GAN** is competitive with **D_real** and **D_fake**, performance could be good;

For CycleGAN, the cost **G_A** (in **orange**) is of interest, primarily. Costs **D_B**, **G_B**, **cycle_B**, **idt_B**, are left out for visibility and because only the generalisation process is of interest, not the other other way around. **cycle_A** relates to the error between the

original image x and its cycle-translated counterpart $\hat{x} = F(G(x))$.

5.1 Pix2Pix

Cartographic aspect	Observations
Buildings	Buildings can be aggregated but edges seem to be smoothed in general. Aggregation could be a byproduct rather than intentional.
Roads, paths, etc	Generally enlarged properly.
Geometric precision	Varies, but not impressive, partially due to noise. Street polygons are frequently not closed at dead ends.
Symbols	While these were enlarged, artifacts showed up on these.
Land use	Vegetation and waters generally suffer from grid artifacts.
Miscolouring	In the case of Mapnik-Michelin, coloured foreground such as water and vegetation objects are misclassified.
Text labels	Text is unintelligible as expected, but landmark labels are either replaced by blanks or text-like symbols with rather specific, repeated patterns. Text may even be outputted in senseless locations.
Technical aspect	Observations
Noise/blur	Lots of noise; details are unintelligible. There is enough noise that aggregation for buildings can be doubted.
Artifacts	Grids, blurred patches with dots; see fig. 5.3. The Mapbox dataset in particular has dots on the boundary of every image.
Misinterpretations	Fringe cases such as almost only vegetation, can be rendered entirely incorrectly. See fig. 5.4
Mode collapse	In a limited sense, yes. A large fraction of generated images contain artifacts, to the point of questionable usefulness. Certain elements cause artifacts more so than others.

Table 5.1: Performance for Pix2Pix.

Overall, Pix2Pix is neither adequate nor robust. At minimum there are issues like graphical noise. Imprecise geometry in the images, and artifacts show up across all test sets, and in some these are very predictable. Text labels may be drawn in some parts, despite none being present in the input image. The cost graph in fig. 5.2 tells a similar story, for all datasets used: generator cost diverges.

Good generalisations are largely limited to roads and streets being enlarged appropriately; Mapbox and Wikimedia have the best geometry, but other issues still ruin performance.

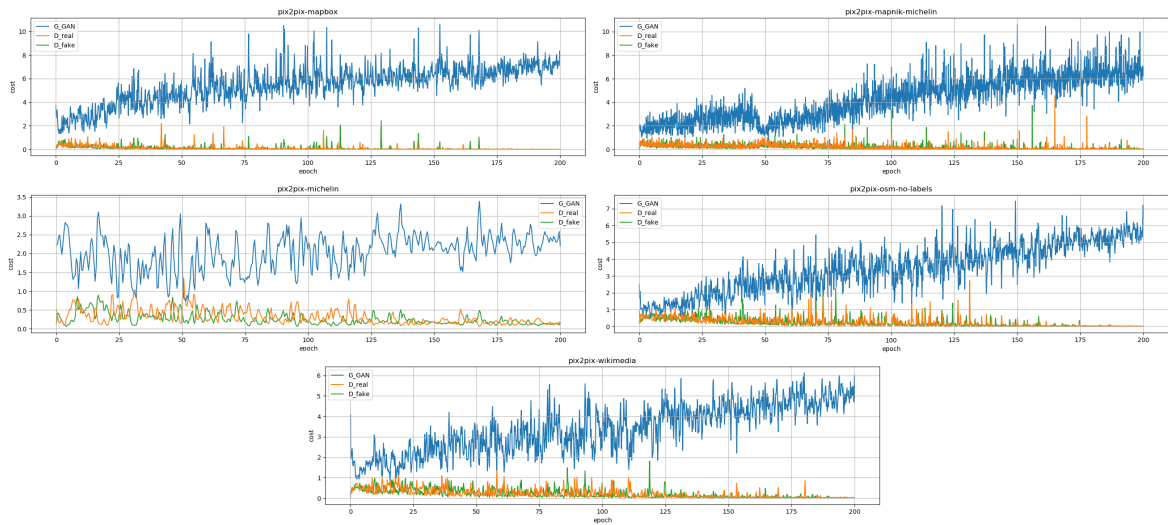


Figure 5.1: Pix2Pix cost graphs. All generators diverged.

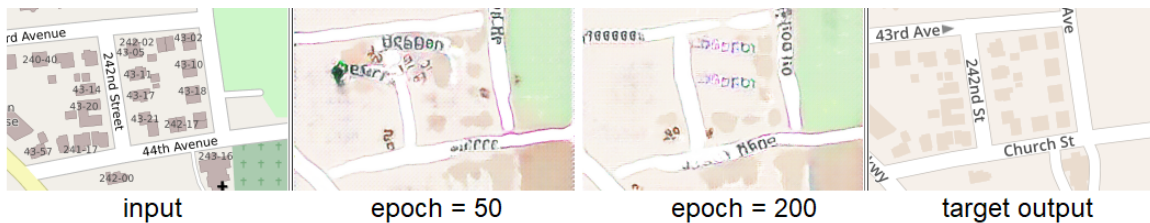


Figure 5.2: Pix2Pix on Mapnik-Michelin. Performance at epoch=200 was no better, if not worse than epoch=50. Note bad omission increasing along a given direct, rather than deselection, bad geometry and misclassification of bottom-right vegetation.



Figure 5.3: Left to right: input, generated, target image. Artifacts in OSM-no-labels can be seen in the middle image (zoom in). The generated images frequently have grid problems. The top box shows repeated artifacts, and the right box shows a blurred patch formed like an L rotated 180°, which is also an artifact.

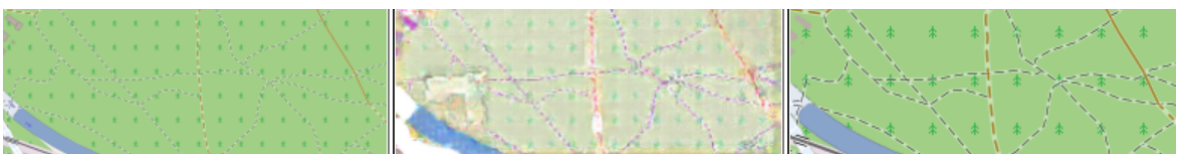


Figure 5.4: Also from OSM-no-labels. While structures like paths and the green texture are still recognised, it is exceptionally bad to interpret vegetation (and more) like this case.



Figure 5.5: Pix2Pix: misclassification.

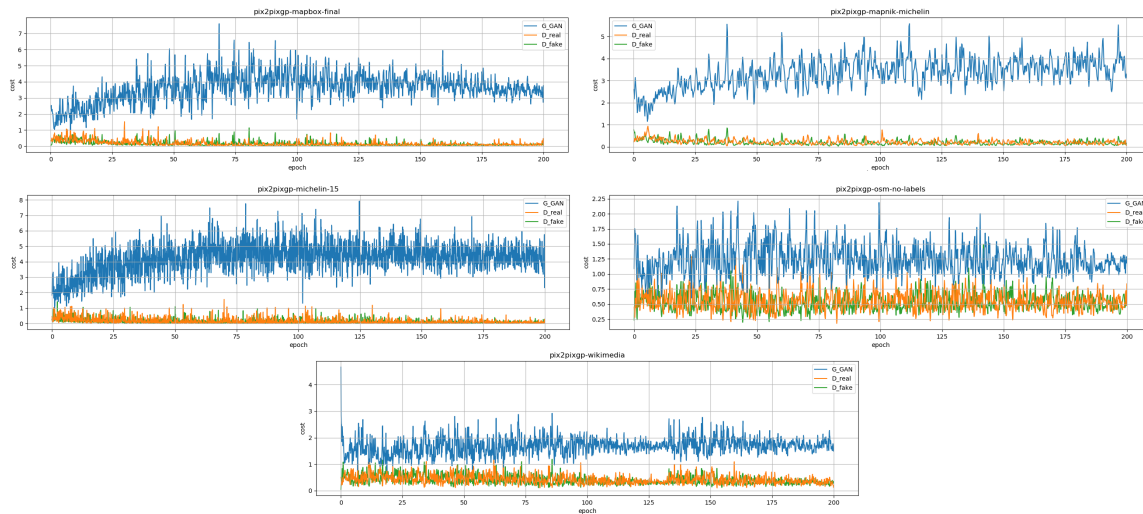


Figure 5.6: While the cost graphs seem to be upper-bounded, WGAN-GP does not seem to improve convergence so well.

A particular case of errors is Mapnik-Michelin. In this dataset, buildings should be deselected or kept. What Pix2Pix has done is likely a poor compromise to satisfy both outcomes at the same time, thus rendering some buildings but not others, while also losing geometric precision. Also, misclassification can be seen in fig. 5.5

5.2 Pix2pix-GP

Pix2Pix-GP provides only minimal improvements compared to Pix2Pix. Its best improvement is a reduction in graphical noise. While it may have *lesser* issues than Pix2Pix overall, it still fails to solve these problems to substantial degrees. The best cost graph oscillates but averages at around the same level throughout most of training, while the worst ones continue to diverge.

Other problems found include grid patterns on waters especially, but grids can be



Figure 5.7: Like Pix2Pix on Mapnik-Michelin (fig. 5.2), a compromise between absence and presence of buildings is attempted rather than one or the other. Note substantial loss of geometry and a blue artifact in the generated image; street polygons are frequently not closed at their ends.



Figure 5.8: OSM-no-labels. The blurred patch with various colours is found repeatedly in the same area, on most images from OSM-no-labels.



Figure 5.9: Mapnik-Michelin. Though buildings are deselected as intended, green roads are rendered with artifact-like elements, lacking in geometric precision too at intersections. The highlighted artifact is particularly bad and blocks other content. The H symbol is rendered badly too.

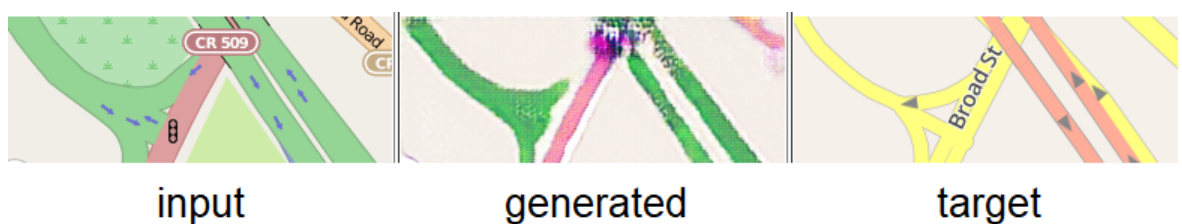


Figure 5.10: Pix2Pix-GP: misclassification.

Cartographic aspect	Observations
Buildings	Mixed results. Adjacent structures were aggregated but not across gaps. Noise has made this unintelligible at times.
Roads, paths, etc	Enlarged properly in spite of noise.
Geometric precision	Good for most objects. Street polygons are frequently not closed at dead ends, as with Pix2Pix.
Symbols	While treated correctly, they may be rendered as graphical artifacts. OSM-no-labels consistently deselected symbols as intended.
Land use	Vegetation, water bodies and highlighted land use may suffer severe grid patterns or text-like artifacts.
Miscolouring	
Text labels	Text can be misinterpreted to be hiding streets or blanks, as opposed to hiding whatever is nearby; street name-like symbols are placed sensibly but are unintelligible.
Technical aspect	Observations
Noise/blur	Graphical jitter is mostly found when artifacts arise but can be seen when zooming in closely.
Artifacts	Grids are still unavoidable, but to varying extents between the datasets. Severe artifacts may show up on labels such as signs showing «A 111», even on objects like actual roads.
Misinterpretations	Less serious than Pix2Pix, such as fig. 5.4.
Mode collapse	Largely similar problems as Pix2Pix.

Table 5.2: Performance for Pix2Pix-GP.

seen if looking for them, on most images. Mapbox in particular has grid problems on the edges of the image.

5.3 CycleGAN

Cartographic aspect	Observations
Buildings	Treated as expected as according to section 4.2.1, except for of OSM-no-labels: a handful of times, buildings are rendered despite the input having no buildings in these areas, which suggests some overfit.
Roads, paths, etc	Enlarged appropriately where applicable. Green streets are translated into «normal» streets, though they should have been omitted. A few times, street polygons are not closed.
Geometric precision	The best among NNs tested; almost as precise as target output.
Symbols	Generally enlarged and intelligible , and accompanying text was could be partially meaningful.
Land use	Texture patterns with a width of 1 pixel are treated almost correctly, blurring accounted for. Certain symbols are accompanied by text-like artifacts. Vegetation, water bodies and other highlighted areas are generally well rendered.
Miscolouring	Michelin dataset: green streets are rendered like normal streets.
Text labels	Text is sometimes intelligible, even if this is outside of the scope of this thesis. Labels could also interfere, being interpreted as though they were hiding background.
Technical aspect	Observations
Noise/blur	Negligible amounts of blur.
Artifacts	Only for some symbols: text-like artifacts.
Misinterpretations	In the Mapnik-Michelin dataset, highway labels/signs could be rendered as buildings.
Mode collapse	None at all.

Table 5.3: Performance for CycleGAN.

Graphical tokens similar to «straße» can be found, i.e. the German word for «street». Unlike the other NNs, CycleGAN can output labels with some level of correctness (even if this could be done in other ways, by using datasets without labels and adding labels as a final procedure). Textured surfaces such as tree symbols on vegetation are replaced with something similar and appropriate spacing, despite being a «high-resolution texture» with lots of space between these symbols. There are minimal problems with water bodies, vegetation or even highlighted areas; these are overwhelmingly dominated by correct treatment.

The most egregious error made by CycleGAN is to treat small, green streets (often with dead ends) as though they were any other «normal» road, rather than deselect

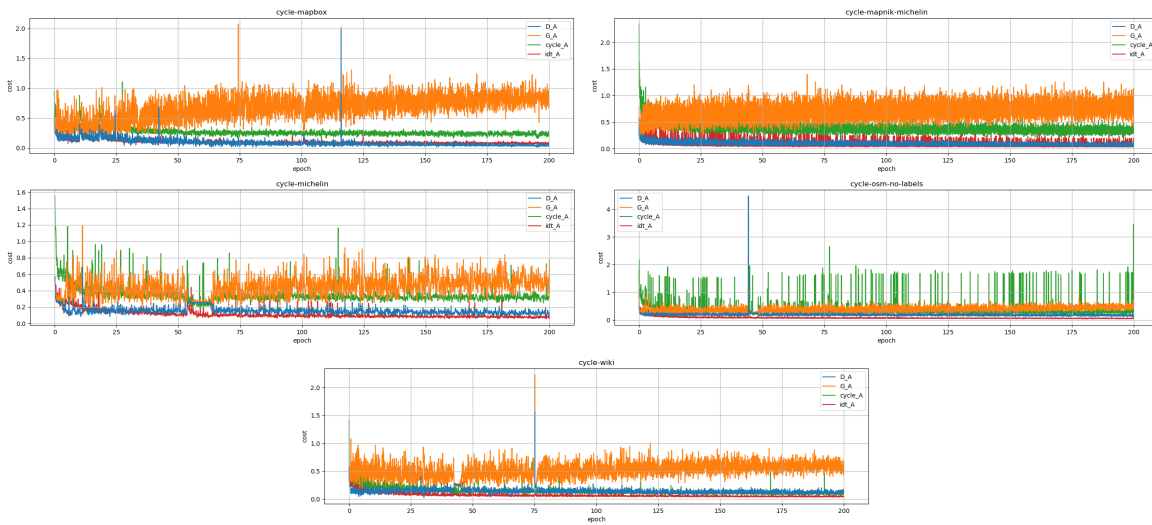


Figure 5.11: Though cost graphs were hardly improving during training, CycleGAN performed well on all datasets. One indicator that is much in favour of CycleGAN here, contrary to all other NNs tested, is that the generator has a consistent cost below 1.



Figure 5.12: CycleGAN does not compromise on buildings appearing or not, unlike Pix2Pix and Pix2Pix-GP. It can even output intelligible text, such as numbers. Labels on top of certain objects may be misunderstood to hide background colours or other objects such as buildings. Colours are correctly applied.

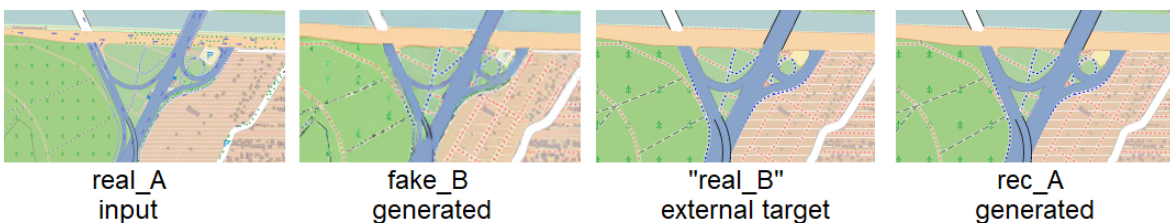


Figure 5.13: OSM-no-labels: CycleGAN avoids hiding the entire pattern of 1 pixel wide lines, and performs excellently in other regards.

them and close off the connecting street.

Though it is not entirely relevant for generalisation purposes, it seems the original information can be recovered after image translation. Omission of symbols and labels, aggregation, and deselection of buildings seen in fig. 5.12, are all essentially undone in `rec_A` despite their apparent absence in `fake_B`. It is not at all obvious how the original data is reproduced, however.

5.4 CycleGP

Nothing is generalised. Egregious grid artifacts are produced instead, overlaid on the original input. CycleGP produces nothing despite relatively good performance on cost graphs, though the repeated loss of performance likely indicates a systemic problem. Figure 5.14 and fig. 5.15 shows the general trend of using CycleGP; grid patterns were found in the overwhelming majority of test images, the other artifact being small, round blanks.

In spite of this, CycleGP learned to remove grid artifacts, evidenced by `rec_A` in fig. 5.16, though this is generally an uninteresting result.

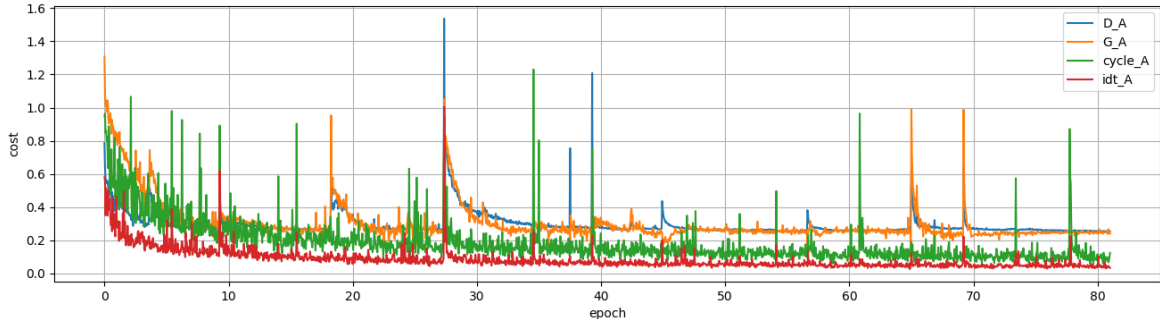


Figure 5.14: General trend of cost measurements while training CycleGP on the Michelin dataset. Convergence is usually followed by a spike in cost. Throughout training, no improvements are observed.



Figure 5.15: Severe failure on the Michelin dataset. A darker shade was applied. Grid artifacts showed up, despite the absence of transpose convolution in this architecture.

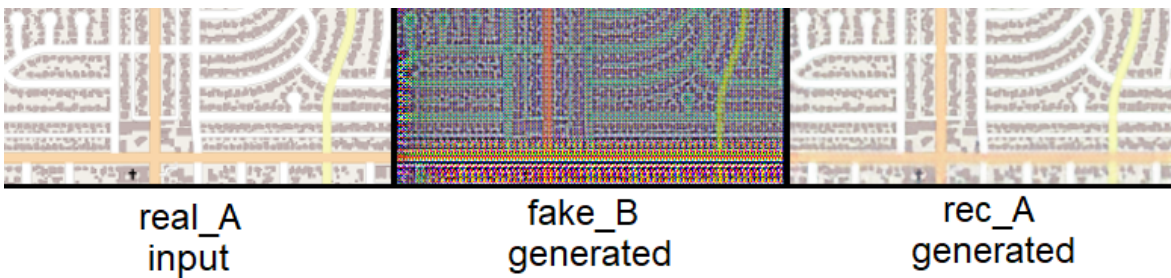


Figure 5.16: Severely pronounced grid artifact on OSM-no-labels, with no other effect. rec_A is slightly blurry, and yet mostly the same as the input.

6. Discussion

6.1 Generalisation capabilities

Pix2Pix and Pix2Pix-GP seem to have minor differences with lackluster results, that do not motivate usage of either. Ambiguity in generalisations are badly handled by way of compromise rather than decisiveness. Artifacts seem to be unavoidable problems, and w.r.t. visual sharpness, neither are particularly impressive, suffering from noise, blur or inability to process fine details. Both are capable of exaggeration of street elements, and to lesser extents, aggregation of buildings and omission of symbols. Classification on the Mapnik-Michelin dataset was simply bad, where the generated output often would use colours found in neither input nor target output.

CycleGAN outperforms the other NNs substantially, demonstrating that even pixel-level precision could be achieved while applying most forms of generalisation found in the datasets. It achieved good performance in most metrics most of the time, and most notably, it performs deselection in the Mapnik-Michelin dataset despite that it is ambiguous when to perform it, whereas Pix2Pix(-GP) does not even learn *how* to deselect/omit data. Its results also show aggregation, exaggeration, classification.

CycleGP is by all means hopeless, as it does not do anything.

In comparison to traditional methods using explicitly defined algorithms, the best method/results — i.e. CycleGAN — may provide some level of competition. Its best cases are near indistinguishable from the target output, but its worst cases fall short. The average case is absolutely usable, but also imperfect.

Compared to other work on generalisation through Deep Learning/NNs, the results for CycleGAN show not only quality performance but also robustness, adaptability. The research by Kang et al. [KGR19] which shares the theme of this thesis, uses CycleGAN too, but not for as many datasets, nor do they attempt generalisation retaining the same overall style. The findings here on the Mapnik-Michelin dataset, support their **quantitative** conclusion that CycleGAN performs better than Pix2Pix on datasets

that make great changes on the styling. One caveat worth noting, however: Pix2Pix in this thesis, is not performing *significantly* better than in their research. At no point has Pix2Pix(-GP) shown especially usable results, unlike CycleGAN (in this thesis).

6.2 Limitations

Some of the limitations here are mentioned to draw lines on what conclusions can be made, or answer questions pre-emptively. They may also outline possibilities for future work, which is explicitly mentioned in the next chapter.

Representation/data encoding. The tilesets in this thesis use 3 layers of colour information to encode various data, which must be interpreted by GANs. Raster data isolating each type of information in a separate layer could be preferable, though such datasets would likely have to be converted from vector data.

Labels, symbols. Text in particular has been a problem, due to various inconsistent behaviours. Text has been replaced by all kinds of elements across NN-dataset combinations. Datasets without labels produce no artifacts similar to text, and also avoid the issue of forcing a NN to guess what is hidden by a label, though it may still occur with symbols and other possible overlapping content instead. Alternatively, labels and symbols could be treated better by having them in the last steps of rendering processes, bypassing these problems entirely.

(Lack of) quantitative analysis. Admittedly, this thesis presents no numerical method to evaluate results, relying instead on visual comparison to input, generated output and known target output. Most of it is described rather than shown.

One possible method to deal with this is to use yet another NN, which is the method described in [KGR19]. Inception score may be considered [Sal+16], though it is designed to evaluate generation of *diverse objects*. Map generalisation of the types in this thesis, may not quite fulfil this trait as strongly as other problems would.

Datasets used. As mentioned already, the Mapnik-Michelin dataset shows inconsistent deselection on buildings. No dataset in this thesis is immune to inconsistency in the underlying data, even if they are based on the same data. The procured tilesets are files



Figure 6.1: CycleGAN trained on Wikimedia, used on Mapbox data instead. Lower saturation is one of various unintended effects.

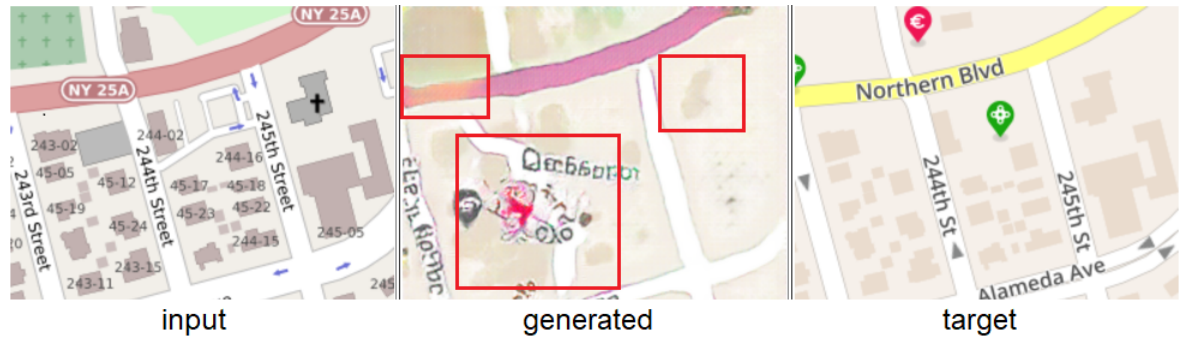


Figure 6.2: Pix2Pix arguably tried to compromise between deselection and retaining objects, resulting in incoherent omission with a substantial loss of geometrical precision. It also produced unintelligible artifacts such as the bottom left box.

saved on servers and may well be outdated and therefore missing data. Only selected areas were used.

The tilesets are also very different. In general, a NN trained to work with one particular dataset, cannot be expected to work with another with different characteristics. See fig. 6.1.

The producers of these tilesets may also simply have treated data differently, just as in [Sto+09]. E.g. Mapbox does not aggregate buildings whereas other datasets might do that, but such information cannot necessarily be recovered without external data.

These tilesets are already treated in some way that is intractable and thus irreversible. All at the same time, this can be a method to test robustness to anomalies, diversity, and to see if inconsistency can indeed be trained rather than landing at a poor middle point. E.g. in training a GAN to achieve inconsistent deselection *as opposed to omission*, a poorly trained GAN generator may try to satisfy the discriminator by presenting *smaller/a subset of* buildings. This was indeed found to be the case in Pix2Pix(-GP) with Mapnik-Michelin; even worse, buildings may be generated where there is only background, and quite badly too. See fig. 6.2.

Architecture, settings, hyperparameters. Throughout this thesis, the default properties have been preferred where no significant changes are motivated, such as learning

rate and network initialisation (setting initial values for parameters). Default architectures were used. Flipping images to artificially augment (size of) datasets was not used, because the datasets were already sufficiently large (compared to related work), and some elements could cause problems this way; it makes no sense to train NNs for unrealistic problems like upside-down labels or symbols.

Exploring alternatives and recommended methods would greatly increase the scope of this thesis. The only changed setting is batch size, for increased computation speed. For Pix2Pix(-GP) it was set to 16, and CycleGAN(-GP) used 4. Researchers have found different performance depending on this, such as BigGAN [BDS18], but this requires much resources.

The authors of the WGAN-GP paper use no normalisation layers in their experiments, and recommend layer normalisation to replace batch normalisation. Neither options were used in this thesis. Pix2Pix uses batch normalisation, which runs only when training. CycleGAN uses instance normalisation, which runs at all times.

7. Conclusion

7.1 Study objectives

The research questions in this thesis were:

- How do selected NNs perform, given the generalisations that are found in a given dataset?
- How robust are the specific methods used? I.e. to what extent is performance largely independent of properties for any given dataset, such that even diverse information can be handled without errors?

Compared to the generalisation-through-Deep-Learning papers presented in 3.1, the methods used here use more data and more complex input data to train NNs with. One of those papers uses the same NNs, and those are the most prominent and powerful among the presented.

Pix2Pix and Pix2Pix-GP *can* perform generalisation in some ways, but the methods used are not so precise that the end result can be expected to be viable or even useful. Considering how badly these two also performed on the Mapnik-Michelin dataset, where geometry was exceptionally bad, it seems futile to use these two in the tested implementations; so they are not at all robust, nor do they perform too well with the given datasets. Usage of CycleGP cannot be defended in any sense.

CycleGAN produces competitive output samples that could feasibly be mistaken for real output, especially its best results. Its performance across different datasets suggests that it is indeed robust; for the type 2 generalisation, where significant style change occurs, the results of Kang et al. [KGR19] are corroborated, showing that CycleGAN can perform rather well, and better than Pix2Pix. Even better performance may have been found, compared to their research, evidenced by possibly better performance but also in the face of more diverse data. This could also be argued against, because different generalisation methods are observed or tested for.

It appears that CycleGAN in particular is a viable technique for the given generali-

sations of the datasets used, especially for those with consistent patterns. The other NNs fall short of conventional demands. Any confident statement on *more* types of generalisation, however, in isolation or in combination, cannot be made based on these findings. A NN trained on a given task, should not be used for or evaluated based on other tasks.

7.2 Future work

There are many ways to expand upon the work done here, also mentioned in the previous chapter. An obvious idea is to use newer, more capable NNs suited for image translation.

Raster datasets. Traditional, well defined generalisation methods mostly use vector data. If these could be converted to raster datasets and laid on top of each other, this could provide datasets with much more specialised training, such that each layer in any given input image contains information about one type of object, such as buildings. E.g. if 10 types of data exist as binary or numerical data in separate layers, then perhaps this could be easier for a NN to interpret rather than arbitrarily defined colours. One of the more obvious benefits is the possibility of using data without labels, as these could always be added in post-processing (after using NNs). (Note that all such data must follow the same format, the layers cannot be switched around.)

Mixing datasets. Generalisation is not a process limited to any one map of a given style, but it is context dependent. To enable a generalisation method that largely takes whatever map is input but generalises it while retaining that same artistic style, this is a possibility. But, this may be unnecessary or already possible with NNs like [Liu+19]; in this paper, the authors generalise from one domain to another (e.g. switching between animal species) with very, very few samples of the codomain.

Change settings. This thesis has used mostly the default architectures of Pix2Pix and CycleGAN. There is always a possibility that alternative configurations would lead to noticeable change. Testing the «hyper-parameter space» is best done with *much* stronger hardware, however. Time spent training each combination of neural network-dataset, in this thesis, extended well beyond 24 hours.

8. Acknowledgements & Legalities

OSM-based tiles hosted by the Wikimedia Foundation was downloaded with explicit permission for educational purposes. The OSM-based data are made available under the Open Database License. Any rights in individual contents of the data used are licensed under the Database Contents License.

Data hosted by Mapbox was downloaded through an API and a personal user token, respecting the terms of service.

Michelin maps tilesets were downloaded in fulfilment of terms and conditions.

Bibliography

- [RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV]. (Visited on 04/2020) (pages 2, 33).
- [Tes20] Tesla. *Tesla Autopilot AI*. 2020. URL: <https://www.tesla.com/autopilotAI> (visited on 04/2020) (page 2).
- [Ele19] Electrek. *Tesla Autopilot safety numbers*. Oct. 2019. URL: <https://electrek.co/2019/10/23/tesla-autopilot-safety-9x-safer-than-average-driving/> (visited on 04/2020) (page 2).
- [Kar+17] Tero Karras et al. “Progressive Growing of GANs for Improved Quality, Stability, and Variation”. In: (2017). arXiv: 1710.10196 [cs.NE]. (Visited on 04/2020) (pages 3, 31, 33, 48).
- [Iso+17] Phillip Isola et al. “Image-to-Image Translation with Conditional Adversarial Networks”. In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. 2017. (Visited on 04/2020) (pages 3, 4, 33, 35, 41).
- [Int14] ICA International Cartographic Association. *ICA: Mission*. 2014. URL: <https://icaci.org/mission/> (visited on 02/2020) (pages 7, 8).
- [All13] Geographic Information Technology Training Alliance. *Generalisation of Map Data*. Ed. by Geographical Information Technology Training Alliance. 2013. URL: <http://www.gitta.info/Generalisati/en/html/index.html> (visited on 05/2020) (pages 8, 9, 15, 16, 18).
- [Twi17] TerribleMaps @ Twitter. *Map of all of the churches in Poland*. 2017. URL: <https://twitter.com/TerribleMaps/status/944362832627486720/photo/1> (visited on 03/2020) (page 8).
- [All10] Robert Allison. *US Population Density*. 2010. URL: http://robslink.com/SAS/democd59/us_population_density.htm (visited on 03/2020) (page 8).
- [map20] Berlin metro map. *Berlin U Bahn Map*. 2020. URL: <https://berlinmap360.com/berlin-metro-map> (visited on 03/2020) (page 10).
- [Kry18] Marisa Krystian. *Do This, Not That: Pie Charts*. 2018. URL: <https://infogram.com/blog/do-this-not-that-pie-charts/> (visited on 04/2020) (page 11).
- [Bjø05] Jan Terje Bjørke. *Kartografisk Kommunikasjon*. 2005 (page 11).

- [Wor11] The Design Work. *65 Amazing Optical Illusion Pictures*. 2011. URL: <https://www.thedesignwork.com/65-amazing-optical-illusion-pictures/> (visited on 04/2020) (page 11).
- [ISO17] ISOM. *ISOM2017 Orienteering Map Symbols*. 2017. URL: <https://www.quantockorienteers.co.uk/info/competitors-resources/control-descriptions> (visited on 04/2020) (page 12).
- [Fie18] Kenneth Field. *Cartographic hyperbole*. 2018. URL: <http://cartonerd.blogspot.com/2018/07/cartographic-hyperbole.html> (visited on 02/2020) (page 13).
- [Ori18] Norges Orienteringsforbund. *Meld deg på kurs i produksjon av orienteringskart!* 2018. URL: <http://orientering.no/nyheter/meld-deg-pa-kurs-i-produksjon-av-orienteringskart/> (visited on 03/2020) (page 17).
- [12320] 123. *C231n Convolutional Neural Networks for Visual Recognition*. 2020. URL: <http://cs231n.stanford.edu/> (visited on 04/2020) (page 19).
- [Nie15] Michael A. Nielsen. *Neural Networks and Deep Learning*. 2015. URL: <http://neuralnetworksanddeeplearning.com/> (visited on 05/2020) (pages 19, 22, 24).
- [Lec+98] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. (Visited on 05/2020) (pages 22, 25, 45).
- [DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. USA: Wiley-Interscience, 2000, p. 307. ISBN: 0471056693. (Visited on 05/2020) (page 23).
- [Rud16] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. URL: <https://ruder.io/optimizing-gradient-descent/index.html> (visited on 03/2020) (pages 24, 28, 31).
- [Pel20] Peltarion. *2D Convolution Block*. 2020. URL: <https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/blocks/2d-convolution-block> (visited on 04/2020) (page 26).
- [He+15] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV]. (Visited on 04/2020) (pages 29, 33, 41).
- [dev19] sci-kit learn developers. *Underfitting vs. Overfitting*. 2019. URL: https://scikit-learn.org/stable/auto_examples/model_selection/plot_underfitting_overfitting.html (visited on 04/2020) (page 29).
- [Bro18] Jason Brownlee. *Use Early Stopping to Halt the Training of Neural Networks At the Right Time*. 2018. URL: <https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/> (visited on 03/2020) (page 30).

- [IS15] Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015. arXiv: [1502.03167](https://arxiv.org/abs/1502.03167) [cs.LG]. (Visited on 05/2020) (page 30).
- [Jef16] Jefkine. *Backpropagation In Convolutional Neural Networks*. 2016. URL: <https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/> (visited on 03/2020) (page 31).
- [lab20] LISA lab. *Convolution arithmetic tutorial*. 2020. URL: http://deeplearning.net/software/theano_versions/dev/tutorial/conv_arithmetic.html (visited on 05/2020) (page 31).
- [ODO16] Augustus Odena, Vincent Dumoulin, and Chris Olah. “Deconvolution and Checkerboard Artifacts”. In: *Distill* (2016). DOI: [10.23915/distill.00003](https://doi.org/10.23915/distill.00003). URL: <http://distill.pub/2016/deconv-checkerboard> (visited on 03/2020) (pages 31, 42).
- [Goo+14] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: [1406.2661](https://arxiv.org/abs/1406.2661) [stat.ML]. (Visited on 04/2020) (pages 31–33, 41, 43).
- [RMC15] Alec Radford, Luke Metz, and Soumith Chintala. “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”. In: *CoRR* abs/1511.06434 (2015). (Visited on 06/2020) (pages 32, 43).
- [Ade20] Sally Adee. *What Are Deepfakes and How Are They Created?* 2020. URL: <https://spectrum.ieee.org/tech-talk/computing/software/what-are-deepfakes-how-are-they-created> (visited on 02/2020) (page 33).
- [Zhu+17] Jun-Yan Zhu et al. “Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks”. In: *Computer Vision (ICCV), 2017 IEEE International Conference on*. 2017. (Visited on 04/2020) (pages 33, 35, 42).
- [Par+19] Taesung Park et al. “Semantic Image Synthesis with Spatially-Adaptive Normalization”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019. (Visited on 04/2020) (pages 33, 34).
- [KGR19] Yuhao Kang, Song Gao, and Robert E. Roth. “Transferring multiscale map styles using generative adversarial networks”. In: *International Journal of Cartography* 5.2-3 (2019), pp. 115–141. DOI: [10.1080/23729333.2019.1615729](https://doi.org/10.1080/23729333.2019.1615729). eprint: <https://doi.org/10.1080/23729333.2019.1615729>. URL: <https://doi.org/10.1080/23729333.2019.1615729> (visited on 05/2020) (pages 35, 53, 65, 66, 69).
- [SFT18] M. Sester, Y. Feng, and F. Thiemann. “BUILDING GENERALIZATION USING DEEP LEARNING”. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLII-4* (2018), pp. 565–572. DOI: [10.5194/isprs-archives-XLII-4-565-2018](https://doi.org/10.5194/isprs-archives-XLII-4-565-2018). URL: <https://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XLII-4/565/2018/> (visited on 04/2020) (page 36).

- [YZZ19] Wenhao Yu, Qi Zhou, and Rong Zhao. “A heuristic approach to the generalization of complex building groups in urban villages”. In: *Geocarto International* 0.0 (2019), pp. 1–25. DOI: [10.1080/10106049.2019.1590463](https://doi.org/10.1080/10106049.2019.1590463). eprint: <https://doi.org/10.1080/10106049.2019.1590463>. URL: <https://doi.org/10.1080/10106049.2019.1590463> (visited on 04/2020) (page 37).
- [Sto+09] Jantien Stoter et al. “Specifying Map Requirements for Automated Generalization of Topographic Data”. In: *The Cartographic Journal* 46.3 (2009), pp. 214–227. DOI: [10.1179/174327709X446637](https://doi.org/10.1179/174327709X446637). eprint: <https://doi.org/10.1179/174327709X446637>. URL: <https://doi.org/10.1179/174327709X446637> (visited on 04/2020) (pages 38, 67).
- [Sto+14] Jantien Stoter et al. “Fully automated generalization of a 1:50k map from 1:10k data”. In: *Cartography and Geographic Information Science* 41.1 (2014), pp. 1–13. DOI: [10.1080/15230406.2013.824637](https://doi.org/10.1080/15230406.2013.824637). eprint: <https://doi.org/10.1080/15230406.2013.824637>. URL: <https://doi.org/10.1080/15230406.2013.824637> (visited on 04/2020) (page 40).
- [Kad13] The Kadaster. *Dataset: Basisregistratie Topografie (BRT) TOPNL*. 2013. URL: <https://www.pdok.nl/downloads/-/article/basisregistratie-topografie-brt-topnl> (visited on 06/2020) (page 40).
- [MO14] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: [1411.1784](https://arxiv.org/abs/1411.1784) [cs.LG]. (Visited on 04/2020) (page 41).
- [LW16] Chuan Li and Michael Wand. *Precomputed Real-Time Texture Synthesis with Markovian Generative Adversarial Networks*. 2016. arXiv: [1604.04382](https://arxiv.org/abs/1604.04382) [cs.CV]. (Visited on 04/2020) (page 41).
- [Iso17] Phillip Isola. *GitHub: issues*. 2017. URL: <https://github.com/junyanz/pytorch-CycleGAN-and-pix2pix/issues/39#issuecomment-305575964> (visited on 06/2020) (page 42).
- [ACB17] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: [1701.07875](https://arxiv.org/abs/1701.07875) [stat.ML]. (Visited on 05/2020) (page 43).
- [Yu+15] Fisher Yu et al. “LSUN: Construction of a Large-scale Image Dataset using Deep Learning with Humans in the Loop”. In: *CoRR* abs/1506.03365 (2015). arXiv: [1506.03365](https://arxiv.org/abs/1506.03365). URL: <http://arxiv.org/abs/1506.03365> (visited on 05/2020) (page 43).
- [Gul+17] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. arXiv: [1704.00028](https://arxiv.org/abs/1704.00028) [cs.LG]. (Visited on 05/2020) (page 44).
- [KLA18] Tero Karras, Samuli Laine, and Timo Aila. *A Style-Based Generator Architecture for Generative Adversarial Networks*. 2018. arXiv: [1812.04948](https://arxiv.org/abs/1812.04948) [cs.NE]. (Visited on 04/2020) (page 47).
- [Led+16] Christian Ledig et al. “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network”. In: *CoRR* abs/1609.04802 (2016). arXiv: [1609.04802](https://arxiv.org/abs/1609.04802). URL: <http://arxiv.org/abs/1609.04802> (visited on 05/2020) (page 48).

- [Sal+16] Tim Salimans et al. *Improved Techniques for Training GANs*. 2016. arXiv: [1606.03498](https://arxiv.org/abs/1606.03498) [cs.LG] (page 66).
- [BDS18] Andrew Brock, Jeff Donahue, and Karen Simonyan. “Large Scale GAN Training for High Fidelity Natural Image Synthesis”. In: *CoRR* abs/1809.11096 (2018). arXiv: [1809.11096](https://arxiv.org/abs/1809.11096). URL: <http://arxiv.org/abs/1809.11096> (visited on 06/2020) (page 68).
- [Liu+19] Ming-Yu Liu et al. *Few-Shot Unsupervised Image-to-Image Translation*. 2019. arXiv: [1905.01723](https://arxiv.org/abs/1905.01723) [cs.CV]. (Visited on 05/2020) (page 70).

