Robert Harb, BSc.

Unsupervised Semantic Image Segmentation with Mutual Information Maximization

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme
Information and Computer Engineering

submitted to

Graz University of Technology

Supervisor

Univ.-Prof. Dipl.-Ing. Dr.techn. Thomas Pock
Institute for Computer Graphics and Vision

Advisor

Dipl.-Ing. Patrick Knöbelreiter
Institute for Computer Graphics and Vision

Graz, Austria, Feb. 2019

TO MY PARENTS

Abstract

Semantic image segmentation is a key topic in computer vision, and a crucial part in many applications such as medical imaging, autonomous driving or augmented reality. Consequently a large portion of research has been devoted to semantic image segmentation in the past, with recent work mostly focusing on supervised methods utilizing Convolutional Neural Networks (CNNs). However these approaches perform only well if there is sufficient labeled training data available. This is rather unfavourable since collecting pixel-wise semantic annotations of images is particularly expensive.

In this work we tackle the task of unsupervised semantic image segmentation. And propose a novel approach that is based on recent work in the area of unsupervised image representation learning and mutual information estimation. We argue that semantically similar image areas have high mutual information. Therefore we first calculate local patch-wise image features, and then assign image areas covered by local features with high mutual information to the same semantic segments.

Our proposed method operates entirely unsupervised end-to-end, and scales to an arbitrary number of semantic classes. We quantitatively and qualitatively evaluate our results on two subsets of the COCO dataset: COCO-Stuff and COCO-Persons. The former has already been previously introduced as a benchmark for unsupervised semantic segmentation methods, and the latter is introduced by us in this work. On both datasets we outperform all compared methods, and achieve a relative increase of 31% in pixel-wise accuracy on COCO-Stuff compared to the current state-of-the-art IIC.

Keywords. Semantic Image Segmentation, Deep Neural Networks, Mutual Information, Unsupervised

Kurzfassung

Die semantische Bildsegmentierung ist ein Schlüsselthema im Gebiet des Maschinellen Sehens, und ein entscheidender Bestandteil vieler Anwendungen wie der medizinischen Bildgebung, dem autonomen Fahren oder Augmented Reality. Ein großer Teil der Forschung im Bereich der semantischen Bildsegmentierung konzentrierte sich in der Vergangenheit auf Methoden des überwachten Lernens mittels Convolutional Neural Networks (CNNs). Diese Ansätze funktionieren jedoch nur hinreichend wenn genügend entsprechend gekennzeichnete Trainingsdaten verfügbar sind. Dies ist jedoch ungünstig, da die Erstellung von Datensätzen mit pixelweisen semantischen Annotierungen besonders aufwändig ist.

In dieser Arbeit widmen wir uns dem Problem der unüberwachten semantischen Bildsegmentierung. Und stellen einen neuartigen Ansatz vor, der sich auf die jüngsten Ergebnisse im Bereich des unüberwachten Lernens und der Abschätzung der Transinformation stützt. Wir argumentieren, dass semantisch ähnliche Bildbereiche eine hohe Transinformation besitzen. Daher berechnen wir zunächst lokale Patch-weise Bildmerkmale, und weisen dann Bildbereiche den gleichen semantischen Segmenten zu welche eine hohe Transinformation besitzen.

Die von uns vorgeschlagene Methode arbeitet völlig unüberwacht end-to-end und skaliert zu einer beliebigen Anzahl von semantischen Klassen. Wir bewerten quantitativ und qualitativ unsere Ergebnisse mittels zweier Teilmengen des COCO-Datensatzes: COCO-Stuff und COCO-Persons. Ersterer wurde bereits früher als Benchmark für unüberwachte semantische Segmentierungsmethoden vorgeschlagen. Und letzterer wird von uns in dieser Arbeit eingeführt. Im Vergleich mit beiden Datensätzen übertreffen wir alle verglichenen Methoden und erreichen eine relative Steigerung von 31% in der pixelweisen Genauigkeit bei COCO-Stuff im Vergleich zum aktuellen Stand der Technik: IIC.

Affidavit

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used.

The text document uploaded to TUGRAZonline is identical to the present master's thesis dissertation.

Date	Signature

Acknowledgments

First of all, I would like to thank my advisor Patrick for providing me with an outstanding supervision throughout this thesis. I sincerely appreciate not only his extensive expertise but also his excellent ability to share it in a clear and understandable way. Furthermore I also want to thank Prof. Thomas Pock for his support throughout this thesis, and letting me be part of his flourishing research group.

I also want to thank my parents for their unconditional support over all the years of my education. And last but not least, I also want express my gratitude to all of my friends for making life outside of university enjoyable.

Contents

List of Figures

List of Tables

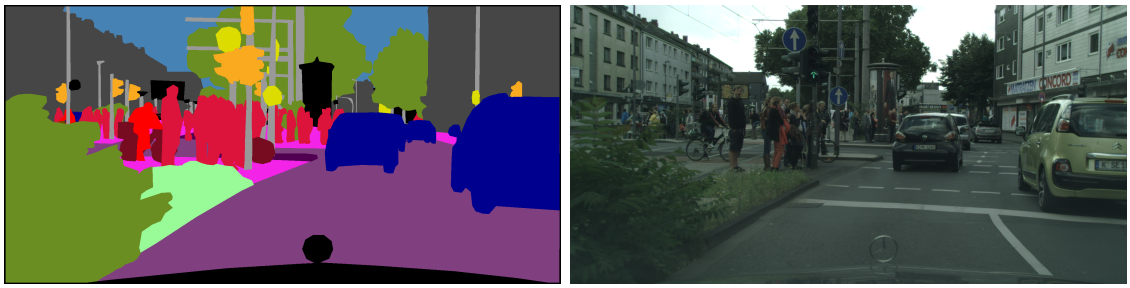
1 Introduction

Contents

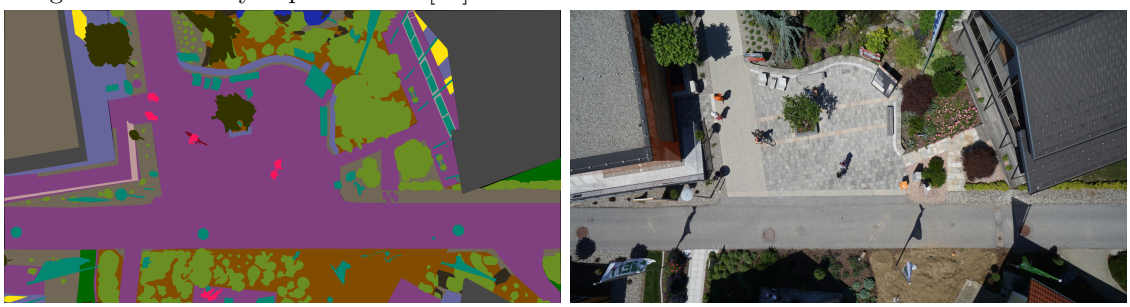
1.1 Motivation

Computer Vision has become an important part of our daily life. Only recently applications increased remarkably due to advances in the development of image acquisition devices and the rising availability of computational resources. This includes autonomous cars that use cameras to navigate streets, image-based diagnostic that has become an inherent tool for many medical professionals, and augmented reality that extends the real-world with digital information. All of these applications rely on systems that have to automatically answer complex questions about the content of images, e.g.: What objects are visible? Where are they located at? What is their scale? What is the distance of a specific point from the camera? Many of these questions can be naturally answered by humans, but it is inherently difficult to come up with automated procedures that provide answers in an efficient and reliable way. While digital images numerically describe the visual appearance of a real-world scene, they contain no high-level information about its content. And extracting useful information from images is the main challenge of Computer Vision. While the field is covering a wide range of tasks such as object detection, 3D reconstruction or the extraction of depth information from images. We will focus in particular on semantic image segmentation in this work.

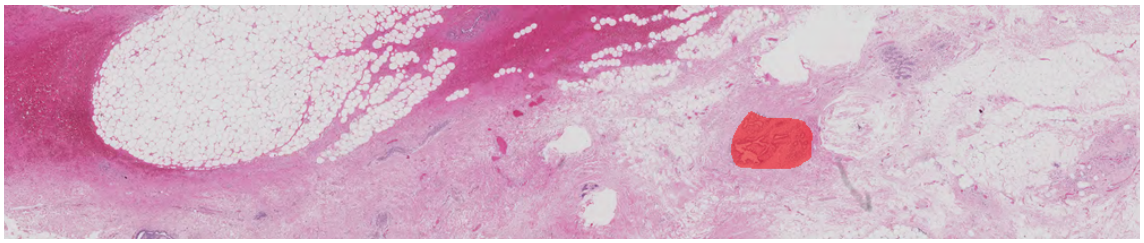
Semantic image segmentation aims to jointly solve the task of detection and localization of semantic entities in images. Thus, every pixel of an input image is assigned to one



(a) **Autonomous driving**, the left image shows pixel-wise class assignments of the right input image. Classes are encoded by different colors e.g. persons in red, cars in blue and plants in green. Images from the Cityscapes dataset [22].



(b) **Aerial imaging**, the left image shows pixel-wise class assignments of the right input image. A similar color encoding is used as above. Images from the DroneDataset [1].



(c) **Medical imaging**, an area depicting an invasive carcioma is detected in a microscopy image of breast tissue. The respective area is highlighted with a red overlay. Image from the BACH's dataset [5].

Figure 1.1: Applications of **Semantic Image Segmentation**.

out of multiple semantic classes. Figure 1.1 shows several examples how semantic image segmentation is used in practical applications. The raw color information of images is converted into a high-level representation of the depicted scenes in all examples. Such representations are essential for many applications. Just consider autonomous driving as an example, it is crucial for safe operation to reliable detect persons and their position. Consequently an extensive amount of research has been devoted to semantic segmentation in the past. With the vast majority of recently proposed work mainly focusing on methods that utilize supervised learning together with Artificial Neural Networks (ANNs).

Supervised learning is a well-known paradigm in machine learning. The parameters of an algorithm are learned by exposing it to input data together with the desired out-

put data. Thus, an algorithm is trained to perform the desired operation by providing a large amount of exemplary calculations. Considering supervised semantic image segmentation, state-of-the-art methods require large datasets of images together with pixel-wise semantic annotations. Unfortunately, collecting such pixel-level annotations is particularly expensive. For example, annotating a single image of Cityscapes [22] – a commonly used dataset for semantic segmentation that contains driving scenes – took 90 minutes on average. Another drawback of supervised learning is that the performance often significantly decreases when the domain gap between training and input data increases. Consequently an algorithm for semantic segmentation trained with images recorded on european streets might perform significantly worse when used in an asian country. Clearly this dependence on expensive large-scale human-labeled data sets considerably limits the applicability of semantic segmentation throughout a variety of practical applications.

Various approaches have been proposed that try to alleviate annotation cost in semantic segmentation. Transfer learning aims to reduce the decrease of performance due to a domain gap between training and input data [25]. Weakly-supervised approaches require no costly pixel-level annotations but instead only annotations at a coarser level [26, 60, 97, 103, 115]. And semi-supervised approaches incorporate unlabeled data together with annotated data during training [46, 116]. However all of these approaches still require human-annotated data at some point.

In this work we aim to tackle the task of semantic image segmentation without any human-annotated data, i.e. unsupervised semantic segmentation. This poses an exciting topic of research. Not merely because many practical applications would benefit tremendously from eliminating expensive data annotation. But also because it still remains an open question how close the performance of supervised approaches can be matched by unsupervised ones. In the past significantly less methods have tackled unsupervised than supervised semantic segmentation. And there is still a significant performance gap between supervised and unsupervised methods.

1.2 Challenges and Approach

The main challenge of this work is posed by the extraction of semantic information from images without using labeled training data, i.e. unsupervised. And furthermore incorporate this information into a unified architecture that enables the semantic segmentation of images. In this section we first describe the main differences between supervised and unsupervised learning, to outline the challenges of unsupervised approaches. Followed by the introduction of recent results in the area of unsupervised image representation learning. And finally we give overview on how we aim to tackle unsupervised semantic image segmentation.

Supervised methods are provided with labeled training images, i.e. a vast amount of examples for each semantic class. And they can use these examples to infer characteristic features about the visual appearance of each class. Thus, class labels are given and for

each class common features have to be found that are shared among all examples labeled with the same class. If these features are not only present in the training set, but also in unseen images having the same class. They can be used to effectively classify novel images. Since low-level appearance often varies vastly among images depicting the same semantic classes, e.g. cars might be shown in a variety of different colors and shapes. Such features rather need to correspond to high-level properties of semantic entities.

In unsupervised approaches only unlabeled images are given. And conclusions about their semantic content has to be done solely from their visual appearance. This is generally considered as an inherently difficult task. Since the visual appearance of image areas can differ vastly even if semantically similar objects are shown. Just consider the semantic class of persons as an example, persons can have different skin color, texture of clothing and shape of hair. Therefore images of persons might cover a vast range of differences in color, texture or shape. And consequently simply observing such low-level features is not sufficient for effective semantic interpretation, rather a high-level understanding of images is required.

Only recently, promising progress has been made in the area of unsupervised image representation learning. Hjelm et al. [42] proposed a method for extracting high-level information from images without any human supervision. Their Local Deep InfoMax architecture incorporates an image encoder that extracts multiple local and a single global feature of an image. The local features encode individual image patches, and the global feature the entire image. Provided that the features of both types, i.e. the local and global, are of limited capacity. Hjelm et al. [42] showed that the global feature contains a high-level representation of the image when its average mutual information with all local features is maximized. Their approach operates entirely unsupervised. And mutual information maximization is done by using a similar approach as proposed by Belghazi et al. [8]. That showed how mutual information between high-dimensional continuous random variables can be estimated using variational lower bounds parametrized by an *ANN*.

While Hjelm et al. [42] proposed how to effectively extract high-level information from images with an entirely unsupervised approach. They did not further investigate how to incorporate their approach in a way to solve practical tasks, e.g. image classification or segmentation, in an unsupervised manner. To validate that their representations contain in fact high-level semantic image information they simply used them as input for shallow supervised classifiers. While their results are highly relevant, the gap to methods that perform practical tasks entirely unsupervised has yet to be closed. And this is exactly what we aim to do in this work. Even though our model is vastly different to Local DeepInfo Max, some major considerations in our work are still based on the results from Hjelm et al. [42].

The main steps of our method on unsupervised semantic image segmentation can be summarized as following. For a given input image we calculate local features, each covering overlapping image patches. As well as multiple high-level features, each covering

the entire image. The number of high-level features is determined by the number of given semantic classes, and each high-level feature should encode features of one semantic class. To segment images, we then assign image areas that are covered by local features with high mutual information to the same semantic segments. And to learn the features, we maximize during training the mutual information between each local feature and the high-level feature it has the largest mutual information with.

1.3 Contribution and Outline

In this work we propose a novel framework for unsupervised semantic image segmentation based on mutual information maximization across different image representations. Our architecture uses solely unlabeled images, and scales arbitrarily to any number of semantic classes. Training starts from randomly initialized weights, i.e. we do not use any pre-trained backbone networks. And we demonstrate both quantitatively and qualitatively that our proposed method is able to extract semantic segmentations for given input images without requiring any annotations.

For evaluation we use two datasets, whereby both are subsets of the well-known COCO [76] dataset. The first is COCO-Stuff [12], which has previously been proposed as a benchmark for unsupervised semantic image segmentation. And the second is COCO-Persons, which we propose as an additional benchmark. Compared to the current state-of-the-art: Invariant Information Clustering (IIC) [51] we achieve a relative increase in pixelwise-accuracy of about 31% in COCO-Stuff. Furthermore we also illustrate and discuss several problems that occur at the quantitative evaluation of entirely unsupervised image segmentation methods.

This work is structured as following. In Chapter 2 we first introduce the concept of mutual information together with all prerequisites from information and probability theory, followed by a thoroughly discussion of several recently proposed approaches on mutual information estimation. Chapter 3 contains an introduction about *ANNs*, including an overview of Convolutional Neural Network (CNN) based methods that tackle semantic image segmentation. Then we thoroughly introduce our approach on unsupervised semantic image segmentation in Chapter 4. Followed by an extensive quantitative and qualitative evaluation of our results in Chapter 5. Finally, in Chapter 6 we will give a conclusion of our work, and propose several suggestions for future improvements.

2 Mutual Information

Contents

2.1 Introduction

Mutual information models dependencies between random variables, and has been successfully used in a wide range of domains such as quantum physics [134], analysing financial markets [33], describing the behaviour of chaotic systems [34] or predicting the progression of alzheimer's disease [31]. It also plays a crucial role in the theory behind modern Deep Neural Networks (DNNs) [108, 125], as well as in many practical *DNN* architectures [42, 51, 127]. Unfortunately, the estimation of mutual information is a complex task, especially for high-dimensional continuous random variables. Only recently, promising progress has been made in this area by utilizing variational bounds on mutual information parametrized by *DNNs* [8], these approaches are essential for this work and thoroughly discussed in this section.

We start this section by introducing preliminary concepts of probability and information theory in Section 2.2 respectively Section 2.3. Followed by a review of several approaches on mutual information estimation in Section 2.4. Finally in Section 2.5 we will discuss recent work incorporating mutual information for unsupervised image segmentation [51] and unsupervised image representation learning [42].

2.2 Probability Theory

2.2.1 Introduction

We start with the definition of the *sample space* Ω that is a set containing all possible outcomes of a random experiment, and denote a single element of the sample space as *sample point* $\omega \in \Omega$. A *random variable* X is then defined as a function from the sample space Ω to the set of real numbers $\mathbb{R}$:

$$X : \Omega \to \mathbb{R}. \tag{2.1}$$

We call $X(\omega)$ a *realization* of the random variable X, and the set $\mathcal{X}$ containing all realizations the *support set* $\mathcal{X}$ of X. Furthermore an *event* E is defined as a subset of the sample space i.e. $E \subseteq \Omega$.

If the support $\mathcal{X}$ of a random variable X is a countable set, X is denoted as a *discrete random variable*. In this case we use uppercase $\mathrm{P}(x)$ to denote the probability that X takes the value x, whereby:

$$P(x) \in [0,1] \quad \forall x \in \mathcal{X} \quad \text{and} \quad \sum_{x \in \mathcal{X}} P(x) = 1, \tag{2.2}$$

and $P(X)$ is called the *probability mass function*. In the other case, when the support $\mathcal{X}$ of a random variable X is not a countable set, X is denoted as a *continuous random variable*. The probability that X takes the value x is then denoted by lowercase $\mathrm{p}(x)$, whereby:

$$p(x) \in [0,1] \quad \forall x \in \mathcal{X} \quad \text{and} \quad \int_{\mathcal{X}} P(x)dx = 1, \tag{2.3}$$

and $p(X)$ is called the *probability density function*.

2.2.2 Expectation

If a discrete random variable X is drawn from $P(x)$, the expected value of X is defined as:

$$\mathbb{E}_{x \sim P(x)}[X] = \sum_{x \in \mathcal{X}} x \cdot P(x). \tag{2.4}$$

The expectation is the average value of all possible values of X weighted by their respective probabilities. Note that we might also write $\mathbb{E}_{P(x)}[X]$ instead of $\mathbb{E}_{x \sim P(x)}[X]$ if it is clear from the context which variable is drawn from the distribution. Furthermore we can also calculate the expected value of a function $f(X)$ of the random variable X:

$$\mathbb{E}_{P(x)}[f(X)] = \sum_{x \in \mathcal{X}} f(x)P(x). \tag{2.5}$$

And if X is a continuous random variable the expectation can be computed as follows:

$$\mathbb{E}_{p(x)}[f(x)] = \int_{\mathcal{X}} f(x)p(x)dx. \tag{2.6}$$

Given a constant k the following properties of the expectation operator hold:

$$\mathbb{E}[kX] = k\mathbb{E}[X] \qquad \mathbb{E}[X \pm k] = \mathbb{E}[X] \pm k. \tag{2.7}$$

Furthermore, if two random variables X and Y are independent, the following factorization holds:

$$\mathbb{E}[XY] = \mathbb{E}[X]\mathbb{E}[Y]. \tag{2.8}$$

An important theorem when working with expectations of random variables is *Jensen's Inequality*. Given a random variable X and a convex function f then:

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]. \tag{2.9}$$

And furthermore if f is a concave function, then:

$$f(\mathbb{E}[X]) \geq \mathbb{E}[f(X)]. \tag{2.10}$$

2.3 Information Theory

The field of Information Theory was established in 1948 when Claude Shannon published his article: *A Mathematical Theory of Communication* [111]. Shannon for the first time proposed quantitative models that allow to describe communication as a statistical process. While his concepts were initially proposed to be used for applications in communication systems, like compression or determining the error rate of noisy channels, the fundamental concepts of Information Theory are nowadays applied in a wide range of fields like Physics, Mathematics or Medicine and, most importantly for this work, also in Machine Learning.

In the following sections we will introduce the most important quantities of Information Theory including mutual information in Section 2.3.7.

2.3.1 Self-information

The first quantity of Information Theory we introduce is the self-information $I(x)$ of an event x, which is given as:

$$I(x) = -\log P(x). \tag{2.11}$$

If the logarithm is calculated with base 2 the self-information is measured in bits, and if the logarithm has base e it is measured in nats.

Self-information is used to quantify how much information we gain from observing an event x, and it is inversely proportional to the probability of observing that event.

Therefore if we observe an event with low probability we gain a high amount of information, and contrarily if we observe a more likely event we gain a lower amount of information. This already represents one of the most important concepts of Information Theory: *The observation of an unlikely event is more informative than the observation of a likely event.*

Often self-information is also referred to as a measure of uncertainty of an event i.e. the uncertainty about a likely event is low and the uncertainty about an unlikely event is high.

2.3.2 Entropy

While self-information considers the outcome of a *single* event x, entropy measures the amount of uncertainty in an entire probability distribution. It is defined as the expected value of the self-information over a probability distribution P, therefore it is for a discrete random variable X calculated as following:

$$\begin{aligned} H(X) &= \mathbb{E}_{P(x)}[I(X)] \\ &= \mathbb{E}_{P(x)}[-\log P(x)] \\ &= -\sum_{x \in \mathcal{X}} P(x) \log P(x). \end{aligned} \tag{2.12}$$

The entropy is always positive and it is upper bounded by the cardinality of $\mathcal{X}$ i.e. $0 \leq H(x) \leq |\mathcal{X}|$. Similarly we can also calculate entropy for a continuous random variable X, in this case it is also referred to as *differential entropy* $h(X)$ and can be calculated as following:

$$h(X) = -\int_{\mathcal{X}} p(x) \log p(x). \tag{2.13}$$

A simple example about entropy can be made by modeling the outcome of a coin toss. Considering a biased coin modeled by the discrete random variable X that takes values out of the set $\mathcal{X} = \{\text{Tails}, \text{Heads}\}$, and the respective probabilities $P(X = \text{Heads}) = q$ and $P(X = \text{Heads}) = 1 - q$. We can calculate the entropy of X as following:

$$H(X) = -q \log q - (1 - q) \log(1 - q). \tag{2.14}$$

Figure 2.1 shows the value of $H(X)$ as a function of q, it can be seen that the maximum amount of information is present when when observing a fair coin toss i.e. $P(X = \text{Heads}) = P(X = \text{Tails}) = 0.5$. Contrarily observing a biased coin that always shows heads or tails provides no new information, i.e. there is no uncertainty about the outcome, and therefore the entropy becomes zero.

In general there are multiple definitions of entropy and we introduced in particular the Shannon entropy, other definitions include for example the Guessing entropy or the Rényi entropy. Therefore we note that when we use the term entropy in this work we always refer to the Shannon entropy.

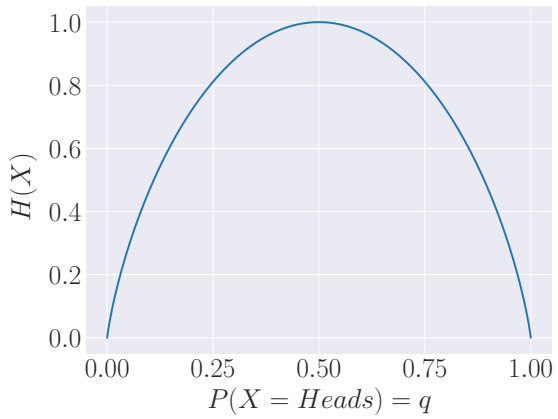


Figure 2.1: Entropy $H(X)$ of a coin toss modelled by a random variable X as a function of $P(X = Heads)$. The entropy is given in bits.

2.3.3 Conditional Entropy

The conditional entropy $H(Y|X)$ measures the amount of additional information we need to describe the outcome of a random variable Y under the assumption that we already know the outcome of another random variable X. Given that X and Y have the support $\mathcal{X}$ respectively $\mathcal{Y}$, the conditional entropy is defined as follows:

$$H(Y|X) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P(x,y)\log P(y|x). \tag{2.15}$$

An important property of conditioning is that it can only reduce the entropy, therefore:

$$H(Y|X) \leq H(Y), \tag{2.16}$$

with equality iff X and Y are independent. Intuitively this means knowing the content of a random variable can only decrease the uncertainty about another random variable, but never increase it.

As for the entropy, there also exists a counterpart of the conditional entropy for continuous random variables. Given the continuous random variables X and Y with the respective supports $\mathcal{X}$ and $\mathcal{Y}$ the *conditional differential entropy* $h(Y|X)$ is defined as:

$$h(Y|X) = -\int_{\mathcal{X}}\int_{\mathcal{Y}} p(x,y)\log p(y|x)dxdy. \tag{2.17}$$

2.3.4 Cross-entropy

Given two distribution P and Q the cross-entropy of P relative to Q is defined as:

$$\begin{aligned} H(P,Q) &= \mathbb{E}_{P(x)}[-\log Q(X)] \\ &= -\sum_{x\in\mathcal{X}} P(x)\log Q(x). \end{aligned} \tag{2.18}$$

Apparently the definition of cross-entropy is similar to that of entropy, with the difference that the expected value is calculated with respect to an additional distribution. We can interpret cross-entropy as the number of bits required to explain the difference of two probability distributions, and consequently it is commonly used to measure the difference between two probability distributions. Furthermore the cross-entropy is always larger than entropy, except when $P = Q$ then the cross-entropy is equal to the entropy.

2.3.5 Kullback-Leibler Divergence

Similar as the cross-entropy the Kullback-Leibler (KL) divergence is a similarity measure between two probability distributions. It is defined as the difference between cross-entropy and entropy:

$$D_{\mathrm{KL}}(P\|Q) = H(P,Q) - H(P). \tag{2.19}$$

And as shown in B.1 we can reformulate equation 2.19 as following:

$$\begin{aligned} D_{KL}(P\|Q) &= \sum_{x\in\mathcal{X}} P(x)\log\frac{P(x)}{Q(x)} \\ &= \mathbb{E}_{P(x)}\left[\log\frac{P(x)}{Q(x)}\right]. \end{aligned} \tag{2.20}$$

Similarly for continuous random variables the *KL* divergence can then be calculated as following:

$$\begin{aligned} D_{KL}(P\|Q) &= \int_{x\in\mathcal{X}} p(x)\log\frac{p(x)}{q(x)} \\ &= \mathbb{E}_{p(x)}\left[\log\frac{p(x)}{q(x)}\right]. \end{aligned} \tag{2.21}$$

A typical application of the *KL* divergence is that a target distribution P and an approximating distribution Q are given, and we want to determine how well Q approximates P.

Important properties of the *KL* divergence include that if $P = Q$ it follows that $D_{KL}(P\|Q) = 0$ since $\log P(x) - \log P(x) = 0$. And if $P \neq Q$ the $D_{KL}(P\|Q)$ is always positive, as shown in B.1.2. Furthermore it should be noted that the *KL* divergence is not symmetric i.e. $D_{KL}(P\|Q) \neq D_{KL}(Q\|P)$, and therefore not a distance measure.

Donsker & Varadhan [28] showed that the following dual representation of the *KL* divergence holds:

$$D_{KL}(P\|Q) = \sup_{f\in\mathcal{F}}\{\mathbb{E}_{p(x)}\left[f(x)\right] - \log\left(\mathbb{E}_{q(x)}\left[\exp(f(x))\right]\right)\}, \tag{2.22}$$

where $\mathcal{F}$ includes all functions that give finite values of the two expectations. By considering Equation 2.26 that shows the relation between *KL* divergence and mutual information, Equation 2.22 can be used to lower bound mutual information. This was first done by Belghazi et al. [8], and we show in Section 2.4.2.1 how the bound of Belghazi et al. can be derived.

2.3.6 Jensen-Shannon Divergence

Another similarity measure between two probability distributions is the Jensen-Shannon (JS) divergence:

$$D_{JS}(P\|Q) = \frac{1}{2}\left(D_{KL}(P\|M) + D_{KL}(Q\|M)\right), \tag{2.23}$$

whereby M is a mixture density of the two probability distributions P and Q:

$$M = \frac{P+Q}{2}. \tag{2.24}$$

Major differences between the *KL* divergence and the *JS* divergence include that the *JS* divergence is bounded i.e. $0 \leq \mathrm{JS}(P\|Q) \leq 1$, and furthermore that it is symmetric i.e. $D_{JS}(P\|Q) = D_{JS}(Q\|P)$. The *JS* divergence is therefore also often referred to as a bounded symmetrization of the unbounded *KL* divergence.

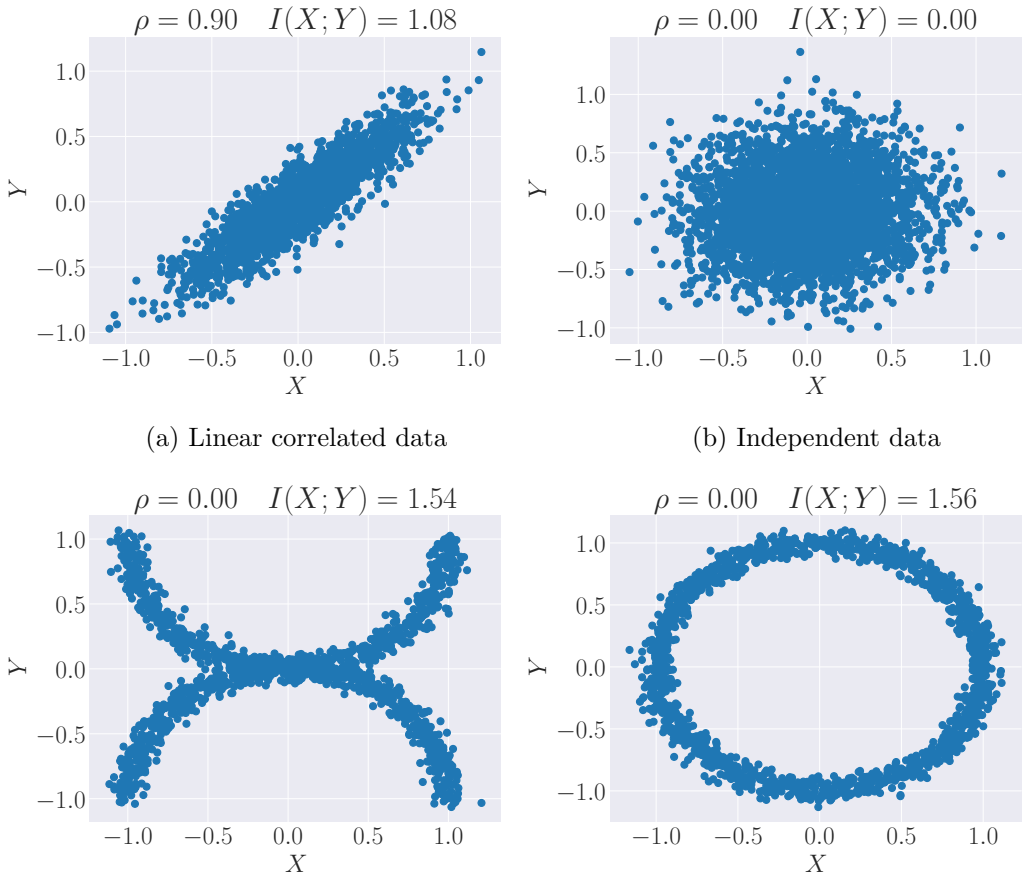
Nowozin et al. [95] proposed the following variational lower-bound on the *JS* divergence:

$$D_{JS}(P\|Q) \geq \sup_{T\in\mathcal{T}}\left\{\mathbb{E}_P\left[-\operatorname{sp}\left(-T(x)\right)\right] - \mathbb{E}_Q\left[\operatorname{sp}\left(T(x)\right)\right]\right\}, \tag{2.25}$$

where $\operatorname{sp}(x) = \log\left(1+e^x\right)$ and $\mathcal{T}$ is the class of functions under that the expectations are finite. We show how to derive this bound in Section 2.4.2.2. Furthermore Hjelm et al. [42] showed that there is a strong monotonically increasing relationship between $D_{JS}(p(x,y)||p(x)p(y))$ and $I(X;Y)$ i.e. the mutual information between two random variables X and Y. Therefore maximization of the bound given in Equation 2.25 can be used for mutual information maximization, which has been done successfully in several works [42, 101, 127].

2.3.7 Mutual Information

Mutual information quantifies the amount of information we gain about one random variable if we observe the outcome of another random variable. For two continuos random variables X and Y it is defined as the *KL* divergence between the joint $p(x,y)$ and product



(a) Linear correlated data

(b) Independent data

(c) Nonlinear correlated data

(d) Nonlinear correlated data

Figure 2.2: **Mutual information** $I(X;Y)$ and **pearson correlation** ρ of two-dimensional example datasets. Both quantities indicate a dependency between random variables with a value unequal zero. In Figure 2.2a and 2.2b both $I(X;Y)$ and ρ capture correctly that there is a dependency respectively no dependency between X and Y. However for the nonlinear correlated data in Figure 2.2c and 2.2d ρ fails to detect any dependency contrarily to $I(X;Y)$. *Mutual information was estimated using Histograms.*

of marginal distributions $p(x)p(y)$:

$$I(X;Y) = D_{KL}[p(x,y)||p(x)p(y)]. \tag{2.26}$$

Considering the definition of the *KL* divergence we can express Equation 2.26 equivalently as:

$$I(X;Y) = \mathbb{E}_{p(x,y)}\left[\log \frac{p(x,y)}{p(x)p(y)}\right]. \tag{2.27}$$

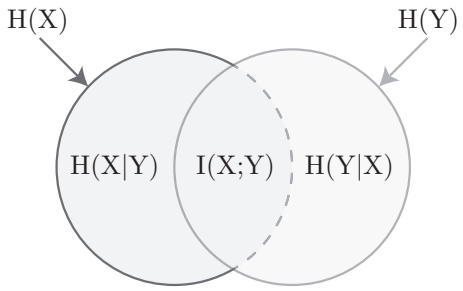


Figure 2.3: Venn diagram showing relationships between various information theoretical measures.

And furthermore, Equation 2.26 can also be rewritten as the difference between the entropy of X and the conditional entropy of X given Y:

$$\begin{aligned} I(X;Y) &= h(X) - h(X|Y) \\ &= h(X) + h(Y) - h(X,Y). \end{aligned} \tag{2.28}$$

A visualization of this relationship is shown in Figure 2.3. Note that all of the above definitions also hold for discrete random variables by replacing the respective quantities.

Mutual information is commonly used to measure dependency between random variables, and contrarily to the pearson correlation coefficient ρ that only captures linear dependencies, mutual information also captures nonlinear dependencies. Figure 2.2 shows the mutual information and pearson correlation for several two-dimensional example datasets.

The following properties hold for mutual information $I(X;Y)$ of two random variables X and Y. At first, if X and Y are independent then their mutual information is zero i.e. $I(X;Y) = 0$. This can be shown by considering that for two independent random variables the joint distribution factors into the marginal distributions $p(x,y) = p(x)p(y)$, and consequently the *KL* divergence becomes zero. Secondly, for non-independent random variables X and Y mutual information is always strictly positive, i.e. $I(X;Y) > 0$. Since knowing the value of one random variable cannot increase the uncertainty about another random variable. And thirdly, mutual information is symmetric i.e. $I(X;Y) = I(Y;X)$.

2.4 Mutual Information estimation

The mutual information of two random variables X and Y can be calculated using Equation 2.27 if the marginal and joint distributions of X and Y are known. Unfortunately in a lot of practical applications we have no knowledge of the underlying distributions but only access to data samples. Consequently a lot of methods have been proposed on estimation of mutual information using solely data samples [34, 45, 90, 120, 121]. However mutual information estimation is nontrivial, and especially when working with high-dimensional

continuous random variables it poses an intricate problem.

In the following we start with an overview on mutual information estimation methods based on density estimation in Section 2.4.1. While these methods perform well with low-dimensional data, they all fail at scaling to high-dimensional data. In this area only recently promising progress has been made by utilizing variational bounds parametrized by *DNNs* [8]. Most importantly the Donkser & Varadhan bound on the *KL* divergence [28], and the bound proposed by Nowozin et al. [95] on the *JS* divergence. We introduced both bounds briefly in the previous section and will show how they can be derived in Section 2.4.2.1 respectively Section 2.4.2.2. Afterwards we will discuss the Mutual Information Neural Estimator (MINE) proposed by Belghazi et al. [8] in section 2.4.3. The *MINE* is based on the Donkser & Varadhan bound on *KL* divergence, utilizes an end-to-end trainable *DNN* architecture and was proposed to be linearly scalable in dimensionality as well as in sample size. While Belghazi et al. primarily used toy problems to demonstrate the effectiveness of *MINE*, local Deep InfoMax a recent follow-up by Hjelm et al. [42] showed how to do unsupervised image representation learning based on ideas introduced in *MINE*. We discuss local Deep InfoMax together with Invariant Information Clustering (IIC) [51] in Section 2.5.2 respectively 2.5.1. *IIC* is a recently proposed clustering algorithm that relies on exact mutual information computation of discrete low dimensional random variables, which can be used for unsupervised semantic image segmentation.

Throughout this section we will consider the following setting. Given N *pairs* of samples of the joint distribution $p(x, y)$ from two random variables X and Y as $\mathcal{S} = \{(\mathbf{x}_i, \mathbf{y}_i) | \mathbf{x}_i \in \mathbb{R}^D, \mathbf{y}_i \in \mathbb{R}^D)\}_{i=1}^N$, we want to calculate an estimate of the mutual information between X and Y i.e. $\hat{I}(X; Y)$. Note that even though that $\mathcal{S}$ has been sampled from the joint distribution $p(x, y)$, we can also create samples from the product of marginal distribution $p(x)p(y)$ by simply pairing a $\boldsymbol{x}_i$ with another randomly selected $\boldsymbol{y}_j$ from $\mathcal{S}$.

2.4.1 Density based mutual information estimation

Since knowledge of the underlying densities allows exact computation it seems natural to tackle mutual information estimation by computing estimates of the underlying densities, i.e. $\hat{p}(\mathbf{x}, \mathbf{y})$, $\hat{p}(\mathbf{x})$ and $\hat{p}(\mathbf{y})$. Density estimation is a well studied subject in statistics and consequently various approaches have been made to apply it for mutual information estimation. In particular we will discuss Histograms in Section 2.4.1.1, Kernel Density Estimation (KDE) in Section 2.4.1.2, Density Ratio Estimation in Section 2.4.1.3 and the Edgeworth expansion in Section 2.4.1.4.

2.4.1.1 Histograms

One of the simplest density estimation methods are Histograms, and Fraser et al. [34] have been the first who proposed to use it for Mutual Information (MI) estimation. To create the respective density estimates we first partition the data samples into bins, then count the number of samples that fall within each bin, and finally turn the bin counts into

probabilities by dividing them through the total number of samples. Note that while in principle this approach can be used for any arbitrary dimension D, we will focus on $D = 1$ in the following for the ease of notation.

We first define the boundaries of K bins by their positions:

$$t_k = o + (k-1)h \quad \forall k = 1, \dots (K+1), \tag{2.29}$$

where h is the width of each bin, and o the origin. Then the number of samples from X respectively Y that fall in the k-th bin are given as a_k respectively b_k:

$$a_k = |\{x_i \colon t_{k-1} \geq x_i > t_k\}_{i=1}^N|, \qquad b_k = |\{y_i \colon t_{k-1} \geq y_i > t_k\}_{i=1}^N|. \tag{2.30}$$

And furthermore we also count the number of joint occurrences where X lies in the j-th and Y in the k-th bin:

$$c_{jk} = |\{(x_i, y_i) \colon t_{j-1} \geq x_i > t_j \text{ and } t_{k-1} \geq y_i > t_k\}_{i=1}^N|. \tag{2.31}$$

Finally we normalize all counts by N and insert the resulting density estimates into Equation 2.40, a mutual information estimate can then be calculated by:

$$\hat{I}(X;Y) = \log N + \frac{1}{N} \sum_k^K \sum_j^K c_{jk} \log \frac{c_{jk}}{a_k b_j}. \tag{2.32}$$

A major drawback of histogram density estimation is that the choice of the bin width h and the origin o have a significant impact on the estimate [113]. Even though several approaches have been made on how to automatically select those parameters, e.g. by Legga et al. [72], up to now there exists no universally applicable approach.

Histogram based *MI* estimation is commonly used in multimodal image registration, for example in medical image analysis where images from one modality like Magnetic Resonance Imaging (MRI) need to be combined with images from another modality like Computed Tomography (CT). In general these images are recorded in different coordinate systems and therefore need to be aligned properly. A comprehensive review of mutual information in image registration can be found in [98].

2.4.1.2 Kernel Density Estimation

Another well studied approach for density estimation is *KDE*, and it was first proposed by Moon et al. [90] to use it for mutual information estimation.

The resulting mutual information estimator is simply given by replacing all densities in Equation 2.27 with respective estimates:

$$\hat{I}(\boldsymbol{X};\boldsymbol{Y}) = \frac{1}{N} \sum_{i=1}^N \log \frac{\hat{p}\,(\mathbf{x}_i, \mathbf{y}_i)}{\hat{p}\,(\mathbf{x}_i)\,\hat{p}\,(\mathbf{y}_i)}, \tag{2.33}$$

where each of these estimates is created using *KDE*. Simply put, in *KDE* a kernel is placed on each sample and the density estimate is then given by the sum of these kernels. For example if a Gaussian kernel is used, an estimate for the marginal density of X is given as:

$$\hat{p}(\boldsymbol{x}) = \frac{1}{Nh^D\sqrt{(2\pi)^D|\boldsymbol{\Sigma}|}} \sum_{i=1}^{N} \exp\left(-\frac{(\mathbf{x}-\mathbf{x}_i)^T\,\boldsymbol{\Sigma}^{-1}\,(\mathbf{x}-\mathbf{x}_i)}{h^2}\right), \tag{2.34}$$

where h controls the kernel width and Σ is the covariance matrix. Both $\hat{p}(\boldsymbol{y})$ and $\hat{p}(\boldsymbol{x},\boldsymbol{y})$ can be calculated similarly, and approaches on selecting h and Σ appropriately have been discussed by Moon et al. [90].

A general problem when using *KDE* is the curse of dimensionality. The number of required samples N to estimate a density up to a certain accuracy grows exponentially with the dimension D. Considering we can estimate a density up to a certain accuracy when $N = 50$ and $D = 1$, more than $N = 10^6$ observations are required if $D = 10$ to achieve the same accuracy [92]. Even though several methods have been proposed that try to overcome the curse of dimensionality, all of them come with additional assumptions or restrictions. For example [78] depends on a sparsity condition of the unknown density function, or [92] assumes a simplified vine copula model for the dependence between variables.

2.4.1.3 Density Ratio Estimation

Up to now all approaches we discussed relied on individual estimation of the densities $p(\mathbf{x})$, $p(\mathbf{y})$ and $p(\mathbf{x},\mathbf{y})$. However if we consider Equation 2.27 we can see that mutual information involves evaluation of the following density ratio:

$$w(\boldsymbol{x},\boldsymbol{y}) = \frac{p(\boldsymbol{x},\boldsymbol{y})}{p(\boldsymbol{x})p(\boldsymbol{y})}. \tag{2.35}$$

For a lot of problems it has been shown that the estimation of a density ratio can be less difficult compared to separate density estimation [118], this is utilized by Maximum Likelihood Mutual Information (MLMI) [120] and Least-Squares Mutual Information (LSMI) [121]. Both calculate an estimate of the density ration $\hat{w}(\boldsymbol{x},\boldsymbol{y})$ and then estimate mutual information as following:

$$\widehat{I}(X,Y) = \frac{1}{N}\sum_{i=1}^{N} \log \widehat{w}\,(\boldsymbol{x}_i,\boldsymbol{y}_i)\,. \tag{2.36}$$

In particular, *MLMI* estimates the density ratio $w(\boldsymbol{x},\boldsymbol{y})$ with a linear combination of B basis functions:

$$\widehat{w}(\boldsymbol{x},\boldsymbol{y}) = \boldsymbol{\alpha}^{\top}\boldsymbol{\varphi}(\boldsymbol{x},\boldsymbol{y}), \tag{2.37}$$

where the basis functions are defined as:

$$\varphi(\boldsymbol{x},\boldsymbol{y}) = (\varphi_1(\boldsymbol{x},\boldsymbol{y}),\varphi_2(\boldsymbol{x},\boldsymbol{y}),\ldots,\varphi_B(\boldsymbol{x},\boldsymbol{y}))^{\top}\,. \tag{2.38}$$

Furthermore the respective weights of each basis function are given as $\boldsymbol{\alpha} = (\alpha_1, \alpha_2, \ldots, \alpha_B)^\top$, and log-likelihood maximization is used to learn the weights.

2.4.1.4 Edgeworth Expansion

Despite knowing the densities, mutual information can also be calculated when the entropies $h(X)$, $h(Y)$ and $h(X, Y)$ are known, as shown in equation 2.28. This was utilized by Hulle et al. [45] to propose mutual information estimation by approximating these entropies using the the Edgeworth expansion. Unfavorably the Edgeworth expansion assumes that the target density is Gaussian, and therefore this approach is only suitable if all densities follow a Gaussian distribution. Furthermore the computation of the Edgeworth expansion is only tractable for very low-dimensional data.

2.4.2 Variational Bounds

In the following we will show how the Donsker & Varadhan bound on the Kullback Leibler divergence [28] and the Nowozin bound on the Jensen Shannon divergence [95] can be derived. A comprehensive review of variational bounds of mutual information can be found in [99]. Furthermore we give in Section 2.4.2.3 a short review about the results from McAllester and Statos [87] that derived some theoretical limitations regarding any lower bound on mutual information.

2.4.2.1 Donsker & Varadhan Bound on the Kullback-Leibler Divergence

The Donsker & Varadhan bound on the Kullback Leibler divergence [28] was used as following by Belghazi et al. [8] for the *MINE* to lower bound mutual information:

$$I(X;Y) \geq \sup_{f:\mathcal{X}\times\mathcal{Y}\to\mathbb{R}} \left\{ \mathbb{E}_{p(x,y)}[f(x,y)] - \log\left(\mathbb{E}_{p(x)p(y)}\left[e^{f(x,y)}\right]\right)\right\}. \tag{2.39}$$

To derive this bound we start with the definition of mutual information given in Equation 2.27 and apply the chain rule of probability:

$$I(X;Y) = \mathbb{E}_{p(x,y)}\left[\log \frac{p(x|y)}{p(x)}\right]. \tag{2.40}$$

By introducing an arbitrary variational distribution $q(x|y)$ we get:

$$I(X;Y) = \mathbb{E}_{p(x,y)}\left[\log \frac{q(x|y)}{p(x)}\frac{p(x|y)}{q(x|y)}\right]. \tag{2.41}$$

Applying the law of total expectation gives:

$$= \mathbb{E}_{p(x,y)}\left[\log \frac{q(x|y)}{p(x)}\right] + \mathbb{E}_{p(y)}\left[\mathbb{E}_{p(x|y)}\left[\log \frac{p(x|y)}{q(x|y)}\right]\right] \tag{2.42}$$

$$= \mathbb{E}_{p(x,y)}\left[\log \frac{q(x|y)}{p(x)}\right] + \mathbb{E}_{p(y)}\left[D_{KL}(p(x|y)\|q(x|y))\right]. \tag{2.43}$$

And due to the non-negativity of the *KL* divergence the following inequality holds:

$$I(X;Y) \geq \mathbb{E}_{p(x,y)}\left[\log \frac{q(x|y)}{p(x)}\right] \tag{2.44}$$

$$= \mathbb{E}_{p(x,y)}\left[\log q(x|y)\right] + \mathbb{E}_{p(x,y)}\left[-\log p(x)\right] \tag{2.45}$$

$$= \mathbb{E}_{p(x,y)}\left[\log q(x|y)\right] + h(X), \tag{2.46}$$

whereby we note that equality is achieved when $q(x|y) = p(x|y)$. This lower bound on mutual information has already been introduced by Barber & Agakov [7], and is for example used in InfoGan [17]. Unfortunately still evaluation of $q(x|y)$ and $h(X)$ is required, which is in general often untractable. However we can get rid of this dependency by using the following energy based distribution for $q(x|y)$:

$$q(x|y) = \frac{p(x)}{Z(y)} e^{f(x,y)} \qquad Z(y) = \mathbb{E}_{p(x)}\left[e^{f(x,y)}\right], \tag{2.47}$$

where $f(x, y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$. Inserting this distribution into equation 2.46 gives:

$$I(X;Y) \geq \mathbb{E}_{p(x,y)}\left[\log \frac{p(x)}{Z(y)} e^{f(x,y)}\right] + h(X) \tag{2.48}$$

$$= \mathbb{E}_{p(x,y)}\left[\log \frac{e^{f(x,y)}}{Z(y)}\right] + \mathbb{E}_{p(x)}\left[\log p(x)\right] - \mathbb{E}_{p(x)}\left[\log(p(x))\right] \tag{2.49}$$

$$= \mathbb{E}_{p(x,y)}[f(x,y)] - \mathbb{E}_{p(y)}[\log Z(y)]. \tag{2.50}$$

Using Jensen's inequality to state $\mathbb{E}_{p(y)}\left[\log(Z(y))\right] \leq \log(\mathbb{E}_{p(y)}[Z(y)])$ gives:

$$I(X;Y) \geq \mathbb{E}_{p(x,y)}[f(x,y)] - \log(\mathbb{E}_{p(y)}[Z(y)]) \tag{2.51}$$

$$= \mathbb{E}_{p(x,y)}[f(x,y)] - \log\left(\mathbb{E}_{p(x)p(y)}\left[e^{f(x,y)}\right]\right). \tag{2.52}$$

And since the inequality holds for any function f we can write:

$$I(X;Y) \geq \sup_{f:\mathcal{X}\times\mathcal{Y}\to\mathbb{R}} \left\{\mathbb{E}_{p(x,y)}[f(x,y)] - \log\left(\mathbb{E}_{p(x)p(y)}\left[e^{f(x,y)}\right]\right)\right\}, \tag{2.53}$$

which is again Equation 2.39.

2.4.2.2 Nowozin Bound on the Jensen Shannon Divergence

Hjelm et al. [42] used the following bound on the *JS* divergence in DeepInfomax.

$$D_{JS}(p(x,y)\|p(x)p(y)) \geq \mathbb{E}_{p(x,y)}[-\operatorname{sp}(-T(x,y))] - \mathbb{E}_{p(x)p(y)}\left[\operatorname{sp}\left(T\left(x,y\right)\right)\right], \tag{2.54}$$

where $\operatorname{sp} = \log\left(1+e^{x}\right)$. Note that we already discussed the relationship between the *JS* divergence and mutual information in Section 2.3.6. A general formulation, valid for any two arbitrary distributions P and Q, of this bound was first proposed by Nowozin et al. [95] and in the following we show how to derive it. We start with the definition of the f-divergence, which is a measure of difference between two probability distributions P and Q:

$$D_f(P\|Q) = \int_{\mathcal{X}} q(x) f\left(\frac{p(x)}{q(x)}\right) dx, \tag{2.55}$$

where $f : \mathbb{R}_+ \to \mathbb{R}$ is a convex lower-semicontinuous function such that $f(1) = 0$. Note that many common divergences used to measure differences between probability distributions are just special cases of the f-divergence. For example the *KL* divergence as given in Equation 2.21 can be derived by letting:

$$f(u) = u\log(u), \tag{2.56}$$

and the *JS* divergence as given in Equation 2.25 can be derived by letting:

$$f(u) = -(u+1)\log\left(\frac{1+u}{2}\right) + u\log\left(u\right). \tag{2.57}$$

At next we note that the convex conjugate f^* for a lower-semicontinuos function f is given as following [41]:

$$f^*(t) = \sup_{u\in\operatorname{dom(f)}} \{ut - f(u)\}. \tag{2.58}$$

Considering that f^* is also convex and lower-continuous, and that $f^{**} = f$ we can also represent f as following [95]:

$$f(u) = \sup_{t\in\operatorname{dom}(f^*)} \left\{tu - f^*(t)\right\}. \tag{2.59}$$

Inserting this into Equation 2.62 gives:

$$D_f(P\|Q) = \int_{\mathcal{X}} q(x) \sup_{t\in\operatorname{dom}(f^*)} \left\{\frac{p(x)}{q(x)}\cdot t - f^*(t)\right\} dx \tag{2.60}$$

If we let $\mathcal{T}$ be an arbitrary class of functions $V : \mathcal{X} \to \mathbb{R}$ and use Jensen's inequality, the following lower bound holds:

$$D_f(P\|Q) \geq \sup_{V \in \mathcal{T}} \left\{ \int_{\mathcal{X}} q(x) \left(\frac{p(x)}{q(x)} \cdot V(x) - f^*(V(x)) \right) dx \right\} \tag{2.61}$$

$$= \sup_{V \in \mathcal{T}} \left\{ \mathbb{E}_P[V(x)] - \mathbb{E}_Q\left[f^*(V(x))\right] \right\}. \tag{2.62}$$

This lower bound on the f-divergence was first proposed by Nguyen et al. [94], and we followed for derivation closely the work of Nowozin et al. [95]. Furthermore Nowozin et al. proposed that the *JS* divergence can be restored from the bound by using $f^* = -\log\left(2 - e^t\right)$ and $V(x) = \log 2 - \log\left(1 + \exp(-T(x)\right)$, inserting these identities into Equation 2.62 gives:

$$D_{JS}(P\|Q) \geq \sup_{T \in \mathcal{T}} \left\{ \mathbb{E}_P\left[\log 2 - \log\left(1 + e^{-T(x)}\right)\right] - \mathbb{E}_Q\left[-\log\left(2 - e^{\log 2 - \log\left(1 + e^{-T(x)}\right)}\right)\right] \right\} \tag{2.63}$$

$$= \sup_{T \in \mathcal{T}} \left\{ \mathbb{E}_P\left[\log 2 - \log\left(1 + e^{-T(x)}\right)\right] - \mathbb{E}_Q\left[-\log\left(2 - \frac{2}{1 + e^{-T(x)}}\right)\right] \right\} \tag{2.64}$$

$$= \sup_{T \in \mathcal{T}} \left\{ \log 2 + \mathbb{E}_P\left[-\log\left(1 + e^{-T(x)}\right)\right] - \mathbb{E}_Q\left[\log\left(1 + e^{T(x)}\right)\right] + \log 2 \right\} \tag{2.65}$$

And using the definition of the softplus function $\mathrm{sp}(x) = \log\left(1 + e^x\right)$ gives:

$$D_{JS}(P\|Q) \geq \sup_{T \in \mathcal{T}} \left\{ 2\log 2 + \mathbb{E}_P\left[-\mathrm{sp}\left(-T(x)\right)\right] - \mathbb{E}_Q\left[\mathrm{sp}\left(T(x)\right)\right] \right\} \tag{2.66}$$

$$\geq \sup_{T \in \mathcal{T}} \left\{ \mathbb{E}_P\left[-\mathrm{sp}\left(-T(x)\right)\right] - \mathbb{E}_Q\left[\mathrm{sp}\left(T(x)\right)\right] \right\}. \tag{2.67}$$

Equation 2.54 can then simply be restored by letting P and Q be the joint respectively product of marginal distributions of two random variables X and Y.

2.4.2.3 Formal Limitations

McAllester and Statos [87] pointed out some general limitations of any lower bound on *KL* divergence. Since according to Equation 2.26 mutual information is equal to the *KL* divergence between the joint $p(x, y)$ and product of marginal distributions $p(x)p(y)$ of two random variables X and Y, this limitations also apply to any bound on mutual information.

While in their results McAllester & Statos [87] assumed that the marginal distribution $p(x)$ of X is fully known, resulting in an even stricter limitation. We will show in the following a slight reformulation provided by Ozair et al. [96] of the original result from McAllester & Statos, where $p(x)$ is assumed to be unkown:

Theorem 1. *Let $p(x)$ and $q(x)$ be two distributions, and $R = \{x_i \sim p(x)\}_{i=1}^{N}$ and $S = \{x_i \sim q(x)\}_{i=1}^{N}$ be two sets of N samples from $p(x)$ and $q(x)$ respectively. Let δ be a confidence parameter, and let $B(R, S, \delta)$ be a real-valued function of the two samples S and R and the confidence parameter δ. We have that, if with probability at least $1 - \delta$,*

$$B(R, S, \delta) \leq KL(p(x) \| q(x))$$

then with probability at least $1 - 4\delta$ we have

$$B(R, S, \delta) \leq \log n.$$

This means at least $N = \exp(I(X;Y))$ samples are required for any lower-bound on mutual information to achieve equality with high confidence. In other words, the number of required samples grows exponentially with the value of mutual information.

Even though the results from McAllester & Statos pose a strong theoretical limitation of lower bounding mutual information, it is not clear how strongly these limitations affect performance in practical applications. Despite these limitations, several works have been published recently that achieved promising performance incorporating mutual information estimation similar as proposed by *MINE* [31, 42, 101, 127]. We attribute this to the fact that none of these methods relies on exact numerical estimates of mutual information, but rather all pursue the objective of maximizing mutual information of different data representations. In this setting also less accurate estimates might be sufficient.

2.4.3 Mutual Information Neural Estimation - MINE

Belghazi et al. [8] proposed to lower-bound mutual information using the Donsker & Varadhan bound on *KL* divergence parametrized by a *DNN*. Their *MINE* can be obtained if we let f in Equation 2.53 be a *DNN* parametrized by $\boldsymbol{\theta} \in \boldsymbol{\Theta}$ denoted as $f_{\boldsymbol{\theta}}(x, y) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$:

$$\hat{I}(X;Y) = \sup_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \mathbb{E}_{p(x,y)} \left[f_{\boldsymbol{\theta}}(x, y) \right] - \log \left(\mathbb{E}_{p(x)p(y)} \left[e^{f_{\boldsymbol{\theta}}(x,y)} \right] \right). \tag{2.68}$$

In the following we will denote the lower-bound on mutual information resulting from Equation 2.68 as:

$$\mathcal{V}(\boldsymbol{\theta}) = \mathbb{E}_{p(x,y)} \left[f_{\boldsymbol{\theta}}(x, y) \right] - \log \left(\mathbb{E}_{p(x)p(y)} \left[e^{f_{\boldsymbol{\theta}}(x,y)} \right] \right). \tag{2.69}$$

To estimate mutual information at first a set of parameters $\boldsymbol{\theta}^*$ that maximizes $\mathcal{V}(\boldsymbol{\theta})$ is obtained, and then an estimate is given by evaluating $\mathcal{V}(\boldsymbol{\theta}^*)$. Note that $\mathcal{V}(\boldsymbol{\theta})$ is maximized if the network $f_{\boldsymbol{\theta}}$ assigns high values to sample pairs drawn from the joint distribution $p(x, y)$, and low values to sample pairs drawn from the product of marginal distributions $p(x)p(y)$. Therefore we can consider $f_{\boldsymbol{\theta}}$ as a discriminator.

When training the *MINE*, one should consider that the gradient of $\mathcal{V}(\boldsymbol{\theta})$ is:

$$\nabla_{\boldsymbol{\theta}}\mathcal{V}(\boldsymbol{\theta}) = \mathbb{E}_{p(x,y)}\left[\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(x,y)\right] - \frac{\mathbb{E}_{p(x)p(y)}\left[\nabla_{\boldsymbol{\theta}} f_{\boldsymbol{\theta}}(x,y) e^{f_{\boldsymbol{\theta}}(x,y)}\right]}{\mathbb{E}_{p(x)p(y)}\left[e^{f_{\boldsymbol{\theta}}(x,y)}\right]}. \tag{2.70}$$

And if mini-batch based approaches like Stochastic Gradient Descent (SGD) are used to maximize $\mathcal{V}(\boldsymbol{\theta})$, the resulting gradient estimates of the second term are biased. Therefore Belghazi et al. [8] suggested to replace the denominator of the second term with an exponential moving average to reduce the bias. Furthermore they also proposed a second bound on mutual information denoted as MINE-f that does not suffer from a biased gradient. However, the bound is less tight than *MINE*.

2.5 Applications in Computer Vision

While in the previous sections we have introduced several approaches on mutual information calculation and estimation, in this section we focus on practical applications. In particular we discuss *IIC* [51] in Section 2.5.1, and local Deep InfoMax [42] in Section 2.5.2.

IIC is a recently introduced method for unsupervised semantic segmentation. Similar to our approach, *IIC* is also based on the maximization of mutual information. However, we pursue a fundamentally different concept. While *IIC* maximizes the mutual information between discrete random variables modeling the class assignments, we maximize the mutual information between high-dimensional continuous random variables that model the semantic content visible in the image. While in *IIC* the mutual information is simply calculated exactly. Our approach requires a complex estimator based on *MINE*. We will show in our experiments that our approach performs significantly better than *IIC*.

Local Deep InfoMax is a recently introduced method for unsupervised image representation learning, and our approach to learn high-level representations of image segments builds up on it. It is important to note that in Deep InfoMax the downstream task image classification is *not* done unsupervised, only image representations, i.e. high-dimensional embeddings, are learned unsupervised. This is a strong restriction regarding practical usability of Deep InfoMax, since high-dimensional representations of images yet alone are of limited usefulness. In contrast, in our work we perform unsupervised semantic image segmentation entirely end-to-end.

2.5.1 Invariant Information Clustering

In general *IIC* [51] is a generic clustering algorithm based on mutual information maximization. And while it also has been shown that *IIC* can be used for image clustering, we will focus on unsupervised semantic image segmentation in the following, since this application is the most relevant to our work.

Given N RGB images $\{\boldsymbol{x}^{(i)} \in \mathbb{R}^{3\times H\times W}\}_{i=1}^{N}$ and image patch centers $u \in \Omega = \{1,\dots,H\} \times \{1,\dots,W\}$. A Convolutional Neural Network (CNN) $\Phi_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$ takes an image $\boldsymbol{x}$ as input, and predicts for each image patch $\boldsymbol{x}_u$ centered at u a class probability vector $\boldsymbol{z}_u \in [0,1]^K$, where K denotes the number of classes. Furthermore we use Z to denote a discrete random variable taking values over $\mathcal{K} = \{k_i\}_{i=1}^{K}$ and let $P(Z = k_i|\boldsymbol{x}_u) = z_{u,i}$ where $z_{u,i}$ is the i-th entry of $\boldsymbol{z}_u$. Note that a semantic segmentation of an input image $\boldsymbol{x}$ can be obtained by letting $P(Z = k_i|\boldsymbol{x}_u) = z_{u,i}$ model the class probabilities for the corresponding pixels at the position u.

To train the network, randomly perturbed versions $\boldsymbol{x}' = g\boldsymbol{x}$ of training images are created, and the respective predictions are denoted as $\boldsymbol{z}'_u$. The perturbations g used by *IIC* include geometric transformations such as image rotation, scaling, skewing or flipping or photometric transformations such as random jittering of image contrast, saturation or brightness. The training objective is to maximize the mutual information between the class predictions of the perturbed and of the unperturbed patches for all images and patch centers:

$$\max_{\boldsymbol{\theta}} \frac{1}{N|\Omega|} \sum_{i=1}^{N} \sum_{u\in\Omega} I(\boldsymbol{z}_u^{(i)}; g^{-1}\boldsymbol{z}_u'^{(i)}), \tag{2.71}$$

where g^{-1} reverses geometric transformations. Additionally, *IIC* also enforces local spatial invariance of the pixel-wise predictions by maximizing the mutual information not solely between predictions of perturbed and unperturbed patches, but also between predictions of adjacent image patches. For clarity we omitted this in equation 2.71.

Using Equation 2.28 we can see that the training objective of *IIC* given in Equation 2.71 is equal to the following:

$$\max_{\boldsymbol{\theta}} \frac{1}{N|\Omega|} \sum_{i=1}^{N} \sum_{u\in\Omega} H(\boldsymbol{z}_u^{(i)}) - H(\boldsymbol{z}_u^{(i)}|g^{-1}\boldsymbol{z}_u'^{(i)}). \tag{2.72}$$

Therefore for each image, *IIC* maximizes the entropy $H(\boldsymbol{z}_u^{(i)})$ while the conditional entropy $H(\boldsymbol{z}_u^{(i)}|g^{-1}\boldsymbol{z}_u'^{(i)})$ is minimized. Minimizing $H(\boldsymbol{z}_u^{(i)}|\boldsymbol{z}_u'^{(i)})$ causes the pixel-wise predictions of the perturbed images to contain as much information as possible about the predictions of the unperturbed images. Since even though we have no knowledge about the ground-truth segmentations, we know that the segmentations of an unperturbed image $\boldsymbol{x}^{(i)}$ and the perturbed version $\boldsymbol{x}'^{(i)}$ should be similar. This holds under the condition that the perturbation g does not change the semantic content of the image. Note that $H(\boldsymbol{z}_u^{(i)}|\boldsymbol{z}_u'^{(i)})$ can simply be minimized if all pixels of all images are assigned to the same segment. However such degenerate solutions are avoided by the additional maximization of $H(\boldsymbol{z}_u^{(i)})$, which is maximized if all classes are assigned with equal probability.

2.5.2 Local Deep InfoMax

Local Deep InfoMax [42] performs unsupervised image representation learning by simultaneous estimation and maximization of high-dimensional continuous local and global image features. Mutual information estimation and maximization is done similar as proposed by *MINE* [8].

Given N RGB images $\{\boldsymbol{x}^{(i)} \in \mathbb{R}^{3\times H\times W}\}_{i=1}^{N}$, for each image local and global features are computed using a *CNN* parametrized by $\boldsymbol{\theta}$. At first local features covering overlapping image patches are computed by $C_{\boldsymbol{\theta}}(\boldsymbol{x}) = \{C_{\boldsymbol{\theta}}^{(i)}\}_{i=1}^{M\times M}$, where M^2 is the number of total local features. Then all local features are summarized to a global feature covering the entire image by computing $E_{\boldsymbol{\theta}}(\boldsymbol{x}) = F_{\boldsymbol{\theta}} \circ C_{\boldsymbol{\theta}}(\boldsymbol{x})$, where $F_{\boldsymbol{\theta}}$ denotes a convolutional block and $\circ$ the function composition operator. The objective of Local Deep InfoMax is to find a configuration of the parameters $\boldsymbol{\theta}$ such that the global feature calculated by $E_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)})$ contains a high-level representation of the input image $\boldsymbol{x}^{(i)}$. To achieve this, Local Deep InfoMax maximizes the average mutual information between the local and global features:

$$\max_{\boldsymbol{\theta}} \frac{1}{M^2} \sum_{i=1}^{M^2} \hat{\mathcal{I}}_{\boldsymbol{\theta}} \left(C_{\boldsymbol{\theta}}^{(i)}(\boldsymbol{X}); E_{\boldsymbol{\theta}}(\boldsymbol{X}) \right), \tag{2.73}$$

where $\hat{\mathcal{I}}_{\boldsymbol{\theta}}$ denotes a mutual information estimator between local and global features.

Intuitively we can explain the training objective given in Equation 2.73 as follows. Assuming that the global feature is of limited capacity, the encoder can not simply copy the entire content from all local features into the global feature, but rather needs to decide which information is passed forward. Furthermore to maximize mutual information with *all* local patches, the global feature needs to contain information that is shared across as many patches as possible to minimize the loss. This encourages the encoder to extract high-level information into the global feature, and discard semantically uninformative information like noise that is only locally present.

Mutual information estimation is done using the Nowozin lower-bound on the *JS* divergence, which we already introduced in Section 2.4.2.2. Hjelm et al. [42] mainly motivated their choice of using a *JS* divergence bound instead of the Donsker & Varadhan *KL* divergence bound as proposed by *MINE* [8], by stating that it led to generally more stable performance during training. The resulting estimator is given as follows:

$$\begin{aligned} \hat{I}_{\boldsymbol{\theta}}(C_{\boldsymbol{\theta}}(\boldsymbol{X}); E_{\boldsymbol{\theta}}(\boldsymbol{X})) =& \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x})} \left[-\operatorname{sp}\left(-T_{\boldsymbol{\theta}}\left(C_{\boldsymbol{\theta}}(\boldsymbol{x}), E_{\boldsymbol{\theta}}(\boldsymbol{x}) \right) \right) \right] \\ & - \mathbb{E}_{\boldsymbol{x}\sim p(\boldsymbol{x}), \boldsymbol{x}'\sim p(\boldsymbol{x})} \left[\operatorname{sp}\left(T_{\boldsymbol{\theta}}\left(C_{\boldsymbol{\theta}}(\boldsymbol{x}'), E_{\boldsymbol{\theta}}(\boldsymbol{x}) \right) \right) \right], \end{aligned} \tag{2.74}$$

where $T_{\boldsymbol{\theta}}$ is a discriminator that takes a pair of local and global features as input and returns a scalar. Note that while in the first expectation term the local and global features are both calculated from the same image $\boldsymbol{x}$, in the second expectation term two independently sampled images are used i.e. $\boldsymbol{x}'$ and $\boldsymbol{x}$. This means Equation 2.73 is maximized if the discriminator $T_{\boldsymbol{\theta}}$ assigns high values to a pair of local and global features

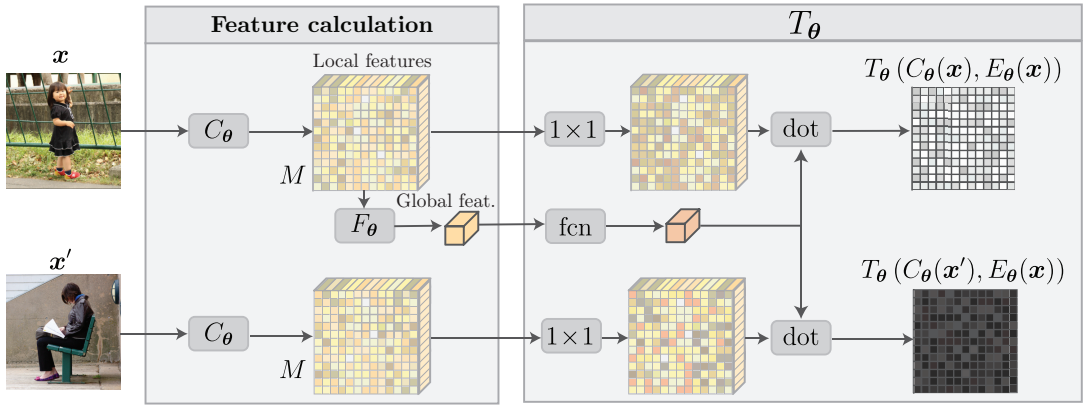


Figure 2.4: **Local Deep InfoMax**. **Feature calculation**: Each image is passed through C_θ to calculate a $M \times M$ map of local features. The global feature of $\mathbf{x}$ is calculated by F_θ. **Discriminator** T_θ: The discriminator score between $\boldsymbol{x}$ and $\boldsymbol{x}'$ is calculated by a dot product of the respective embedded local and global features. Bright colors indicate a large dot-product and dark colors a small dot-product. *dot denotes a pointwise dot-product, fcn denotes a fully-connected layer and* 1×1 *a* 1×1 *convolutional layer.*

both calculated from the same image, and low values if they are calculated from different images.

In general the discriminator T_θ can be any arbitrary function, e.g. a *CNN*, that takes a pair of features as input and outputs a real-valued score. However it should be considered that the function needs to be calculated separately for every feature pair to evaluate. Hjelm et al. proposed to calculate the score of two given features by first encoding them through some additional layers, and then calculating the dot-product of the respective feature encodings. Therefore, only a single forward pass through the network has to be done, and then the score of any arbitrary feature combination can be efficiently calculated by the respective dot-product.

Hjelm et al. [42] used the following procedure to evaluate the suitability of the high-level features learned by local Deep InfoMax for image classification. At first, they trained the network as described previously using a set of unlabeled images. Then they trained supervised image classifiers taking the high-level features as input, in particular a Support Vector Machine (SVM) and a Multi Layer Perceptron (MLP) with a single hidden layer of 200 neurons. This was repeated with representations of other unsupervised representation learning approaches and the final classification performance of local Deep InfoMax was superior compared to all other tested methods. For this evaluation the CIFAR-10 [65], CIFAR-100 [65], STL-10 [20] and Tiny ImageNet [23] dataset were used.

3 Artificial Neural Networks

Contents

3.1 Introduction

Artificial Neural Networks (ANNs) are systems containing a set of artificial neurons that are roughly inspired by the functionality of biological neurons. While the computations done by each individual neuron are very simple, it has been shown that *ANNs* can perform complex tasks by interconnecting a vast amount of neurons. Today applications where *ANNs* are used cover a wide range of areas, including a lot of problems where other available methods obtain only inferior performance, for example the translation of text between different languages [56], teaching computers to play the game of Go at a human level [112], creating complex language models that are able to generate naturally appearing text [100] or protein structure analysis [110]. Also a lot of areas in computer vision are currently dominated by *ANN* based approaches such as semantic image segmentation [16, 105, 122, 138], object detection [36, 114], stereo matching [15, 58, 64] or image denoising [77].

ANNs can be seen as function approximators that calculate an output by mapping an input through a set of interconnected artificial neurons. Each individual neuron has a set of parameters that control what computations it carries out. Consequently if we want a network to perform a certain task a respective configuration of parameters has to be found, this process is referred to as training.

How individual neurons are interconnected in an *ANN* is mainly dependent on the respective application, and a variety of different architectures have been proposed. Most of these architectures follow the basic concept of merging multiple neurons into layers, and use different approaches to connect individual layers. The number of layers is then often referred to as the depth of the network, and networks with a large depth are called Deep Neural Networks (DNNs). In this chapter we will describe two common neural network architectures: Feed Forward Neural Networks (FFNNs) where the input of each layer is given as the output of its preceding layer, and Convolutional Neural Networks (CNNs) that can be seen as a variation of *FFNNs*, which is well suited to process image data. Besides those two other commonly used architectures are for example: Recurrent Neural Networks (RNNs) where connections between layers form a directed cycle, or Deep Belief Networks (DBNs) where connections between layers are undirected.

We will begin this chapter with a brief overview of the history on *ANNs* in Section 3.2. Then we will introduce artificial neurons and activation functions used by them in Section 3.3 respectively Section 3.4. Followed by the introduction of *FFNN* in Section 3.5 and *CNNs* in Section 3.6. The training of *ANN* is discussed in section 3.7, and at the end of this chapter we will give an overview on recent approaches based on *CNNs* for semantic image segmentation.

3.2 History

A first mathematical description of an artificial neuron was already introduced in 1943 by Warren S. McCulloch and Walter H. Pitts [88]. The McCulloch-Pitts (MCP) neuron was designed to recognize if an input $\boldsymbol{x}$ belongs to one of two different classes. McCulloch and Pitts proposed to resemble biological neurons by the following simple decision rule:

$$z(\boldsymbol{x}) = \begin{cases} 1 \text{ if } \sum_i x_i > \Theta \\ 0 \text{ otherwise} \end{cases} . \tag{3.1}$$

Hence, if the sum of the components of an input $\boldsymbol{x}$ is larger than a threshold Θ the neuron returns one and zero otherwise. The threshold Θ needs to be set manually by a human operator. Using the proper thresholds a single *MCP* neuron can be used to represent simple boolean functions such as OR, AND or NOT.

Following the *MCP* neuron, in 1958 the Rosenblatt's single layer perceptron [106] and in 1960 the Adaptive Linear Element (ADALINE) [133] were introduced. Both models proposed methods to learn automatically the parameters of an artificial neuron by pro-

cessing given examples from each class. This largely resembles the concept of supervised learning as it is commonly used today. Furthermore *ADALINE* already used a variant of Stochastic Gradient Descent (SGD) for training, like a lot of modern *DNN* models.

After these early successes in the 1950s and 1960s a major backlash against research in *ANNs* followed, initiated when Marvin Minsky published his book *Perceptrons: An Introduction to Computational Geometry* [89] in 1969. In this work Minsky drawed attention to some major flaws concerning all *ANN* models introduced until then, most famously that a single-layer perceptron is not able to model a simple XOR function.

While the work of Minsky led to a decline on *ANNs* research during the 1970s, it gained again traction in the early 1980s when the connectionism movement emerged. Connectionism evolves around the idea that intelligent behaviour can be achieved when a large number of simple computational units are networked together. During this time a lot of concepts have been introduced that are still commonly used today, like an early version of the back-propagation algorithm introduced by Yann LeCun et al. [70], which allowed to train an *ANN* with multiple hidden units to perform handwriting recognition in 1989. Furthermore also the Long Short-Term Memory (LSTM) cell [43] was introduced in 1997, which today is used in a wide variety of tasks that require sequence modeling such as machine translation and speech recognition. However besides these advancements it also became apparent that *ANNs* are difficult to train, and the computational resources necessary for many practical applications were simply not available at that time. Consequently most of the research in machine learning headed towards other fields such as Graphical Models [53] or Kernel Machines [11, 109], and as a result *ANN* research declined again.

This second decline in *ANN* research ended with the introduction of novel methods and the rising availability of the computational resources required to train deeper *ANNs*, respectively *DNNs*. Already in 2006, G. Hinton et al. [40] introduced a method that enabled the training of *ANNs* that were significantly deeper than those used by previous approaches. It still lasted until 2012 for the widespread adoption of *DNNs* to start, when Alex Krizhevsky et al. [66] introduced AlexNet. AlexNet is a *CNN* based architecture for image classification that was submitted to the yearly held ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where it beat all other submissions in 2012 by a large margin. This was the first approach showing that *DNNs* perform well at large-scale image classification tasks. Since then various *DNN* based methods have been introduced tackling a huge variety of problems, by far not only limited to image classification.

3.3 Artificial Neurons

The scalar output z of a single artificial neuron is calculated by first doing an affine transformation of the input $\boldsymbol{x} \in \mathbb{R}^D$, followed by a nonlinear transformation with an

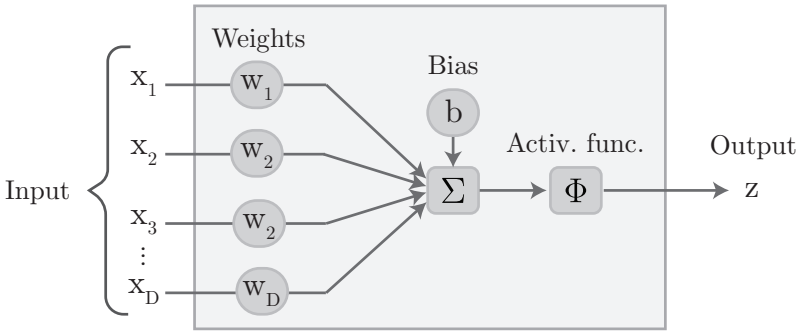


Figure 3.1: Visualization of an **artificial neuron**. Each individual component x_i of the input $\mathbf{x}$ is multiplied by a respective weight w_i, the weighted components are summed up, a bias b is added, and the result is passed through an activation function $\Phi(.)$ to calculate the output z.

activation function $\Phi : \mathbb{R} \mapsto \mathbb{R}$:

$$z = \Phi\Big(\sum_{i=1}^{D} w_i x_i + b\Big). \tag{3.2}$$

Consequently the output of an artificial neuron depends on weights $\boldsymbol{w} \in \mathbb{R}^D$ and bias $b \in \mathbb{R}$ of the affine transformation, and the activation function Φ. During training usually the parameters $\boldsymbol{w}$ and b are changed, while the activation function Φ is invariant. Figure 3.1 shows the structure of a single artificial neuron.

3.4 Activation functions

In this section we will describe the most commonly used activation functions in *DNNs* and their respective properties. Figure 3.2 shows plots of the activation functions we discuss in this section.

We begin with the **sigmoid** activation function which is defined as:

$$\sigma(x) = \frac{1}{1 + e^{-x}}. \tag{3.3}$$

Historically the sigmoid activation function was commonly used. However in more recent work it has fallen out of favor, mainly because of the following two disadvantages.

At first the vanishing gradient problem. For either large positive or large negative input values the function saturates. Therefore even a relatively large change of the input leads only to a small change of the output, and consequently in those regions the gradient becomes very small. This effect can also be seen in Figure 3.2. Too small gradients are rather unfavourable during the training of *DNNs* with backpropagation. Mainly because the gradient of each neuron is proportional to the gradient of its previous neuron, and

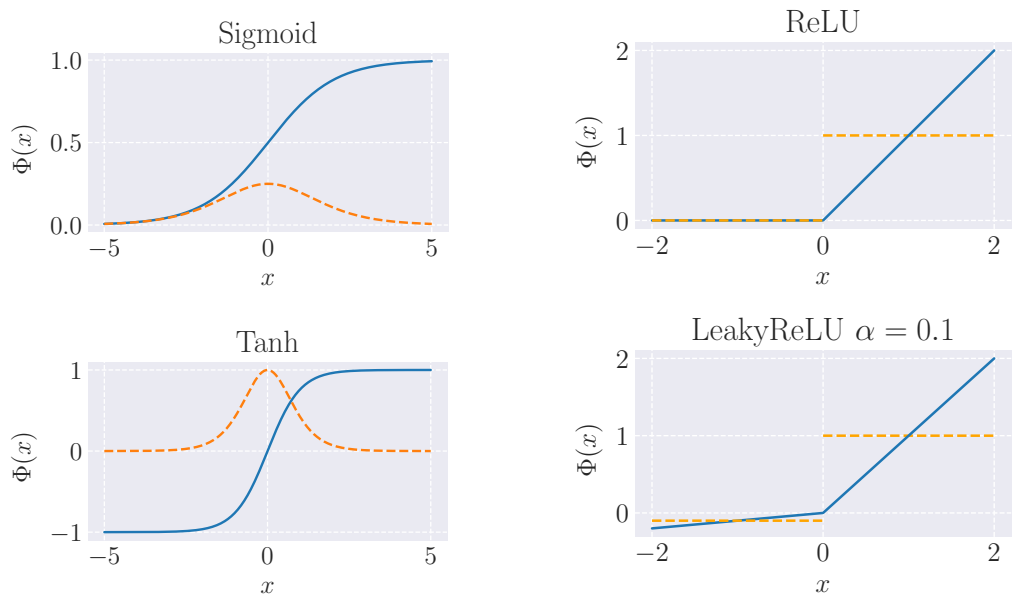


Figure 3.2: Common **activation functions** in *DNNs* and their derivatives. The functions are depicted as solid blue lines, and their derivatives as dashed orange lines. Note that the derivative for ReLU and LeakyReLU is not defined at $x = 0$.

therefore a neuron with a small gradient also shrinks the gradients of all following neurons. Consequently this can lead to exponential shrinking of gradients, which as a result slows down learning.

The second property that is often claimed as a disadvantage of the sigmoid activation function is that its output value is not zero-centered. More specifically, the output of the sigmoid is always positive. Consequently, the input components x_i of a neuron that receives the output of a sigmoid activated neuron as input are always positive. And therefore the gradients of the weight components w_i are either all positive or negative. This can be unfavourable during training because an optimal step might require increasing the value of one weight component, while decreasing the value of another. However, it should be noted that when gradients are accumulated across multiple input samples, like in mini-batch based training methods, this problem is reduced. Since adding up individual gradients enables the resulting updates on each w_i to have variable signs.

While also being susceptible to the vanishing gradient problem, the **hyperbolic tangent** activation function - also called tanh - fixes the problem of sigmoid having a non zero-centered output. The function is defined as:

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \tag{3.4}$$

and its output ranges from -1 to 1. Furthermore given the definition of the sigmoid

activation function in equation 3.3 it can be seen that $\tanh(x)$ is equal to a scaled and shifted version of the sigmoid:

$$\tanh(x) = 2\sigma(2x) - 1. \tag{3.5}$$

A commonly used activation function in recent *DNNs* is the **Rectified Linear Unit (ReLU)**, which is defined as:

$$\text{ReLU}(x) = \max(0, x). \tag{3.6}$$

Since the gradient of a *ReLU* is either zero or one, it does not suffer from the vanishing gradient problem as the sigmoid and the tanh. This is a major advantage of using *ReLUs*, and often leads to superior performance compared to using sigmoid or tanh. For example Krizhevsky et al. have shown that their network which won the 2012 *ILSVRC* convergences six times faster when using *ReLUs* compared to using tanh [66].

One major issue still present when using *ReLUs* is the dying ReLU problem, where neurons using *ReLUs* become inactive by outputting zero for any given input. Once a neuron reaches this state of inactivity it is not able to escape it. Since always outputting zero also leads to the gradient to become always zero, and therefore no further updates are applied on the weights of the neuron during training.

To overcome the issue of dying *ReLUs* the **LeakyReLU** activation function has been introduced. It does not output zero for negative inputs but has rather a slight slope in the negative range, preventing the gradient to become zero. It is defined as:

$$\text{LeakyReLU}(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}, \tag{3.7}$$

where the parameter α controls the negative slope of the activation function. In general α is set to a small constant e.g. $\alpha = 0.01$. However there also exists a variation of the LeakyReLU activation function where α is adaptively learned with the rest of the *ANNs* parameters. This variation was introduced as Parametric Rectified Linear Unit (PReLU) and it was shown that it can lead to improved performance compared to the *ReLU*. Especially at relatively deep or wide network architectures [39].

3.5 Feed Forward Neural Networks

As already described in Section 3.2 single artificial neurons are rather limited in their expressibility. Most famously they fail at modeling a simple XOR function, which has already been shown in 1969 [89]. More generally speaking, single artificial neurons are only able to detect linearly separable patterns. This limitation can be overcome by interconnecting multiple neurons to larger networks. And one common approach for doing this are *FFNNs*, where neurons are organized in consecutive layers. Each layer contains

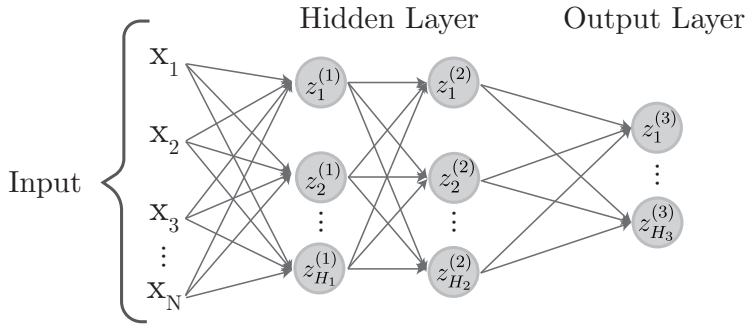


Figure 3.3: Structure of a **Multi Layer Perceptron**. The input is propagated through two hidden layers of neurons before the output is calculated. This structure trivially extends to *MLPs* having arbitrary many hidden layers, each with a varying number of neurons.

multiple neurons, and the input of each layer is given as the output of its preceeding layer. Furthermore we refer to a *FFNNs* as a Multi Layer Perceptron (MLP) if it contains at least three layers, and if it is fully connected i.e. all neurons of a layer are connected to all neurons of the consecutive layer. For *MLPs* it has been shown that they can approximate any continuous function with arbitrary accuracy if they contain a sufficient number of neurons [44]. Figure 3.3 shows the structure of a simple *MLP* with three layers.

To describe how the activations of individual neurons in a *MLP* are calculated we will use the following notation. Considering a *MLP* with K consecutive layers, each layer with index $i \in \{1, \dots, K\}$ contains H_i neurons with their activations given as $\boldsymbol{z}^{(i)} \in \mathbb{R}^{H_i}$, where $z_j^{(i)}$ is the activation of the j-th neuron in layer i. For two consecutive layers indexed by (i) and $(i-1)$ the weights between their neurons are determined by the matrix $\boldsymbol{W}^{(i)} \in \mathbb{R}^{H_i \times H_{i-1}}$, where the entry $w_{jl}^{(i)}$ is the weight between $z_j^{(i)}$ and $z_l^{(i-1)}$. And the biases for each layer (i) are determined by the vector $\mathbf{b}^{(i)} \in \mathbb{R}^{H_i}$, where the component $b_j^{(i)}$ contains the bias for the neuron $z_j^{(i)}$. Furthermore we consider the input values $\boldsymbol{x} \in \mathbb{R}^N$ as activations of an layer with index $i = 0$, therefore we set $\boldsymbol{z}^{(0)} = \boldsymbol{x}$ and $H_0 = N$.

Using the above notations we can now simply calculate the output of the j-th neuron in the i-th layer as follows:

$$z_j^{(i)} = \Phi^{(i)} \left(\sum_{l=1}^{H_{i-1}} w_{jl}^{(i)} z_l^{(i-1)} + b_j^{(i)} \right), \tag{3.8}$$

where $\Phi^{(i)}$ denotes the activation function used in the i-th layer. Equivalently we can write this using matrix multiplications as:

$$\boldsymbol{z}^{(i)} = \Phi^{(i)} \left(\mathbf{W}^{(i)} \boldsymbol{z}^{(i-1)} + \boldsymbol{b}^{(i)} \right), \tag{3.9}$$

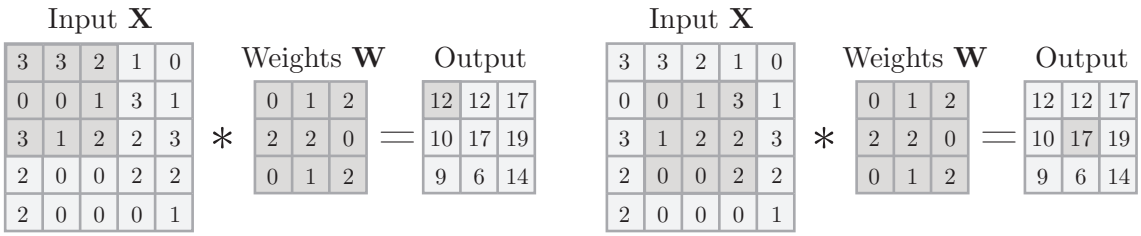


Figure 3.4: Exemplary calculation of a **two-dimensional discrete convolution**. The shaded areas in X and W indicate the values that are used to calculate the shaded value in the output.

where $\Phi^{(i)}$ is applied element-wise.

3.6 Convolutional Neural Networks

While *FFNNs* are a simple way of connecting multiple neurons, in the area of image processing it is much more common to use *CNNs*. This is motivated by the fact that *FFNNs* with fully-connected neurons as described in section 3.5 have several properties that are rather unfavorable when processing image data.

At first the number of parameters rapidly grows with the dimensionality of the data that is processed. This is rather unfavorable because increasing the number of parameters also increases the memory and runtime requirements, and can also cause overfitting as described in Section 3.7.2. Secondly the spatial structure of data is not taken into account. For example when image data is processed, it might be favorable to use an architecture that treats spatially close pixels differently than distant ones. And thirdly fully connected layers are not spatially equivariant. This means if we process two images containing the same object at two different positions, the respective outputs might be vastly different, which is undesirable in a lot of applications.

CNNs try to tackle all of the above mentioned problems. They use one or multiple kernels to parametrize the mapping between two subsequent layers, whereby these kernels have the same number of spatial dimensions as the input, but generally a much smaller size i.e. number of parameters. To calculate the output of a layer these kernels are then convolved with the input. Unlike a matrix multiplication as used in fully-connected layers, the convolution operation explicitly considers the spatial relations between input points and is spatially equivariant. In principle *CNNs* can be used to process data having any arbitrary number of spatial dimensions e.g. one-dimensional for processing time series, two-dimensional for images, or three-dimensional for processing volumetric data. We will focus on the processing of two-dimensional images in this work.

In this section we will discuss the most important concepts of *CNNs* regarding image processing. In particular we will introduce Convolutional Layers in Section 3.6.1, Pooling Layers in Section 3.6.2 and Normalizaton Layers in Section 3.6.3.

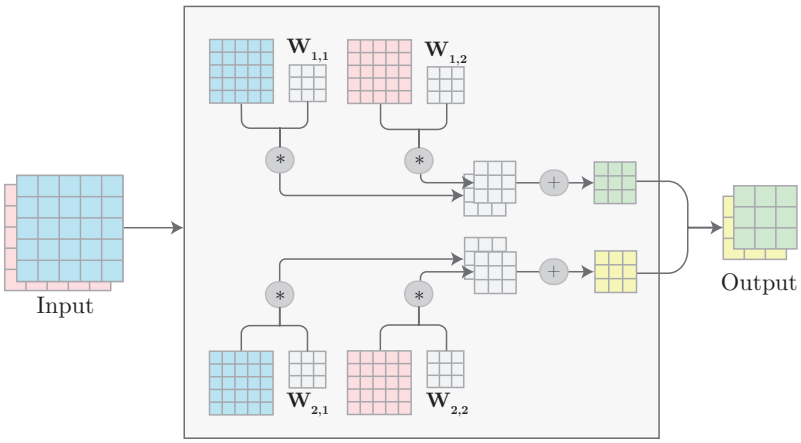


Figure 3.5: **Output calculation in a convolutional layer**. Example shown for $C_{in} = 2$ and $C_{out} = 2$. At first each of the two input channels is convolved with a separate kernel for each of the two output channels, then the respective results are merged together with an elementwise addition, and finally the output is given by stacking the two merged results together. *Note that we omit the application of the bias and activation function for clarity.*

3.6.1 Convolutional Layer

A convolutional layer takes an input $\boldsymbol{x} \in \mathbb{R}^{M \times N \times C_{in}}$ and maps it to an output $\boldsymbol{z} \in \mathbb{R}^{M' \times N' \times C_{out}}$, where $(M \times N)$ and $(M' \times N')$ denote the spatial dimension of the input and output and C_{in} and C_{out} denote the number of input respectively output channels. Having multiple input channels enables a single layer to processes multiple inputs of the same spatial resolution together, for example the color channels of an RGB image. Furthermore having multiple output channels enables a layer to compute different output representations of the input. This means a single convolutional layer might for example extract horizontal and vertical edges of an input image stored in separate output channels. In the following we will first show how the output can be calculated when both input and output have one channel, and then generalize this concept to multiple channels.

Given an input image $\boldsymbol{X} \in \mathbb{R}^{M \times N}$, where $x_{m,n}$ denotes the image pixel at the spatial position m, n, the weights of the kernel denoted by $\boldsymbol{W} \in \mathbb{R}^{A \times A}$, where $w_{a,a'}$ denotes the weight at the spatial position a, a', and the scalar bias b of the kernel. The output $\boldsymbol{Z}$ of the layer is calculated by first convolving $\boldsymbol{X}$ and $\boldsymbol{W}$, then adding the bias b, and finally an elementwise application of the activation function $\Phi(.)$:

$$\boldsymbol{Z} = \Phi\big((\boldsymbol{X} * \boldsymbol{W}) + b\big). \tag{3.10}$$

An elementwise representation of the output $\boldsymbol{Z}$ can simply be obtained as following:

$$z_{i,j} = \Phi\Big(\sum_{a}^{A}\sum_{a'}^{A} x_{i-a,j-a'} w_{a,a'} + b\Big). \tag{3.11}$$

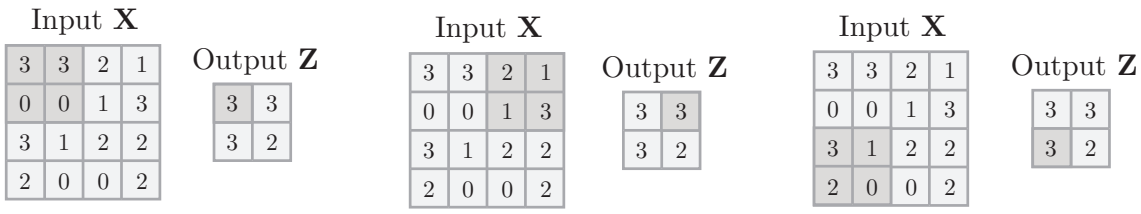


Figure 3.6: Exemplary calculation of a **max pooling** operation with a 2×2 window size and a stride of 2. At each step the shaded area in $\boldsymbol{Z}$ is calculated by the maximum value in the shaded area of $\boldsymbol{X}$. The resulting output resolution of $\boldsymbol{Z}$ corresponds to the resolution of $\boldsymbol{X}$ downsampled by a factor of 2.

Figure 3.4 shows an example how a two-dimensional discrete convolution is computed.

We now consider the general case of a convolutional layer with C_{in} input channels and C_{out} output channels. Instead of a single kernel we now have $C_{in} \cdot C_{out}$ kernels, therefore for each individual input output channel combination a separate kernel is used. Consequently the weight matrix of the layer is given as $\boldsymbol{W} \in \mathbb{R}^{C_{out} \times C_{in} \times A \times A}$, where we denote the kernel that maps from the input channel i to the output channel j as $\boldsymbol{W}_{i,j} \in \mathbb{R}^{A \times A}$. Furthermore a different bias b_j taken from $\boldsymbol{b} \in \mathbb{R}^{C_{out}}$ is added for each output channel. The j-th output channel can then be calculated as follows:

$$\boldsymbol{Z}_j = \Phi \left(\sum_{i=1}^{C_{in}} \boldsymbol{X}_i * \boldsymbol{W}_{ij} + b_j \right). \tag{3.12}$$

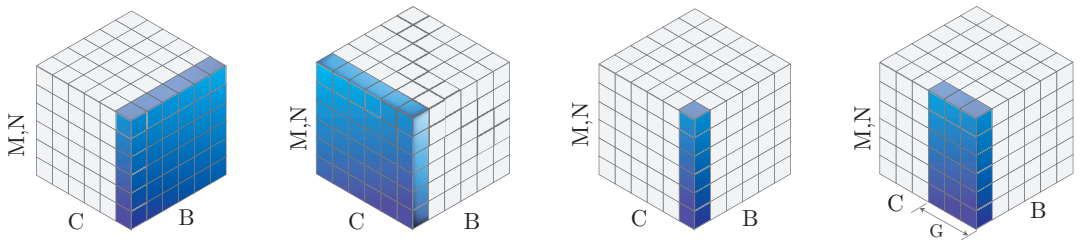
We show an example of this calculation in Figure 3.5.

It also should be noted that there exist alternative ways how multiple channels are treated in in convolutional layers, whereby most of them have been introduced with the purpose of increasing computational efficiency. This includes for example depthwise separable convolutions [18] or grouped convolutions [55].

3.6.2 Pooling Layer

In a lot of applications additional pooling layers are added between certain convolutional layers to progressively reduce the spatial dimension of the data. Consequently, the first layers of a network might process data at the full input resolution, while subsequent layers process data at a downsampled resolution. This is mainly motivated by the following advantages: At first computational complexity is reduced if data with a smaller resolution has to be processed. Secondly, it increases translational invariance by merging multiple neighbouring values in a single value. And thirdly, the effective receptive field is increased, whereby the effective receptive field refers to the area of the input that can possibly influence the activations of a layer.

In general a pooling layer is determined by a respective pooling operation, a window size and the stride. To calculate the output of a pooling layer a window is slided over the input,



(a) Batch Norm. (b) Layer Norm. (c) Instance Norm. (d) Group Norm.

Figure 3.7: Different **normalization techniques** used in *CNNs*. B denotes the batch dimension, C the channel dimension and M,N denote the spatial dimensions of an input image. The colored blocks indicate over which regions the normalization is calculated. In Group Normalization the number of groups G used for normalization is variable.

and the output at each spatial position is determined by applying the pooling operation to all pixels inside the window. The stride determines how many pixels the window is moved in each subsequent sliding step, and consequently also controls the downsampling rate of the layer. Commonly used pooling operations are for example taking the maximum, the average or the minimum. An example for max pooling is shown in Figure 3.6, whereby the same concept trivially adapts to pooling with different operations.

3.6.3 Normalization Layer

In a lot of *CNNs* architectures normalization layers are used. Since in a variety of cases it has been shown that they can help to increase convergence speed and generalization in *CNNs*. Whereby Batch Normalization [48] is currently the most commonly used normalization layer, and we also utilize it in our experiments. We will describe it in the following.

Given the set of activations from a convolutional layer in a single mini-batch as $\mathcal{B} = \{z_{b,c,m,n}\}$, where b indexes the batch dimension (see Section 3.7 for a description about what is referred to as mini-batches in *ANN* training), c the channel, and m, n the spatial indexes. In Batch Normalization we first calculate the mean over the batch dimension of channel c:

$$\mu_c = \frac{1}{B} \sum_{b,m,n} z_{b,c,m,n}, \tag{3.13}$$

where B denotes the mini-batch size. And the variance over the batch dimension of channel c:

$$\sigma_c = \frac{1}{B} \sum_{b,m,n} (z_{b,c,m,n} - \mu_c)^2. \tag{3.14}$$

Then the output of a Batch Normalization layer is calculated as follows:

$$\hat{z}_{b,c,m,n} = \gamma_c \frac{z_{b,c,m,n} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}} + \beta_c, \tag{3.15}$$

where ϵ is a small constant for numerical stability. This first normalizes the input activations such that they have zero mean and unit variance, followed by a channel-wise affine transformation parametrized by γ_c and β_c. Whereby γ_c and β_c are learned during the training of the network. The affine transformation after the normalization is done to restore the representational power of the network [48]. Furthermore usually during training a moving average of μ_c and σ_c is stored which is then used for normalization during inference. This is done since during inference one might want to pass only a single sample through the network from which μ_c and σ_c can not be directly estimated.

It should be noted that it has been shown empirically that Batch Normalization can significantly improve training speed and regularization [48] in certain cases. There is still an ongoing debate about the theory behind its effectiveness. While Ioffe et al. [48] initially proposed that Batch Normalization works so well because it reduces the internal covariate shift, which they describe as the "change in the distribution of network activations due to the change in network parameters during training". More recent work has actually disproven that the reduction of internal covariate shift is the major reason of Batch Normalizations effectiveness [107]. And several studies have been conducted trying to find alternative explanations [10, 80, 107]. Where up to now no commonly accepted explanation has been found.

Besides Batch Normalization recently also a variety of other normalization approaches have been introduced. Where the main difference between them is given by which activations they use to calculate the mean and variance for normalization. Layer Normalization normalizes the activations of all channels in a layer [6], Group Normalization normalizes groups of channels separately [135], and Instance Normalization [126] normalizes the activations of each channel separately. Figure 3.7 illustrates different normalization techniques as they are used in *CNNs* processing two-dimensional data.

3.7 Training Artificial Neural Networks

Until now we have primarily focused on how to calculate the activations of a *ANN* using a given input, and we assumed that the parameters of the network are given. In this section we will now describe how the parameters of a *ANN* can be learned, this process is commonly referred to as training.

Note that throughout this section we will denote the set containing all learnable parameters of an *ANN* as $\boldsymbol{\theta}$, the input of the network as $\boldsymbol{x} \in \mathbb{R}^D$ and the output of the network as y. Furthermore the output of the network is calculated by $y = f(\boldsymbol{x}; \boldsymbol{\theta}) : \mathbb{R}^D \mapsto \mathbb{R}$.

3.7.1 Loss function

During training a scalar $J(\boldsymbol{\theta})$ referred to as cost or loss function is calculated. This cost depends on the current parameters $\boldsymbol{\theta}$ of the network, and the objective of training is to find a new set of parameters that minimize the value of this cost function. A generic formulation of $J(\boldsymbol{\theta})$ as it is commonly used in *supervised* learning is given as follows:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},y)\sim p_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y), \tag{3.16}$$

where the expectation is evaluated using data samples $(\boldsymbol{x}, y)$ drawn from a distribution p_{data}. Furthermore $L(.)$ is a per-example loss function that compares the output of the network $f(\boldsymbol{x};\theta)$ with a given label y, returning low values if they are similar and large values conversely. Therefore during supervised training we want to find a configuration of the weights $\boldsymbol{\theta}$ such that the network outputs for each given sample $\boldsymbol{x}$ the respective given target y. Contrarily during *unsupervised* training solely input data $\boldsymbol{x}$ is given, therefore equation 3.16 reduces to:

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta})). \tag{3.17}$$

The function $L(.)$ now needs to be chosen such that it is minimized by a parametrization $\boldsymbol{\theta}$ for which the network calculates the desired output, without using any exemplary output y. In unsupervised learning it is often a major challenge to find an appropriate function $L(.)$ for the desired target task. We thoroughly describe how we derive a loss suitable for unsupervised semantic segmentation in Section 4.

3.7.2 Overfitting

In general when training an *ANN* we do not have access to the true underlying distribution of our data p_{data}. Instead, we have only access to a limited set of samples given by the empirical distribution $\hat{p}_{data}$, whereby these samples are referred to as *training data*. Consequently when minimizing equation 3.16 respectively 3.17 we can only estimate the expected value of $J(\boldsymbol{\theta})$ using the available training data. Given the set $\mathcal{X} = \{\boldsymbol{x}_i\}_{i=1}^{N}$ of N unlabeled training samples drawn from $\hat{p}_{data}$, $J(\boldsymbol{\theta})$ is therefore calculated as follows:

$$J(\boldsymbol{\theta}) = \frac{1}{N}\sum_{i=1}^{N} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta})). \tag{3.18}$$

However while $J(\boldsymbol{\theta})$ is now minimized using samples drawn from $\hat{p}_{data}$, it still should be small for data drawn from p_{data} which is unseen during training. If there is a huge deviation between the loss evaluated on our training data $\hat{p}_{data}$ and the loss evaluated on p_{data} we speak of *overfitting*.

Overfitting is a major problem in a lot of applications using *DNNs*. It occurs if the true underlying data distribution is only insufficiently approximated by the empirical data distribution used for training. In theory this can be counteracted by increasing the

number of samples in $\hat{p}_{data}$ and making sure they are drawn independently from p_{data}. However in practice this is often difficult, because data is expensive to acquire, especially in the supervised setting where it also needs to be labeled. This is a major advantage of unsupervised learning. Since the amount of available training data substantially increases for a lot of tasks by eliminating the necessity of expensive data labeling.

Another approach to overcome overfitting are regularization methods. Generally speaking they introduce modifications during training that try to increase the generalization ability of the network to unseen data. This includes for example weight decay [67], dropout [117] or dataset augmentation [130].

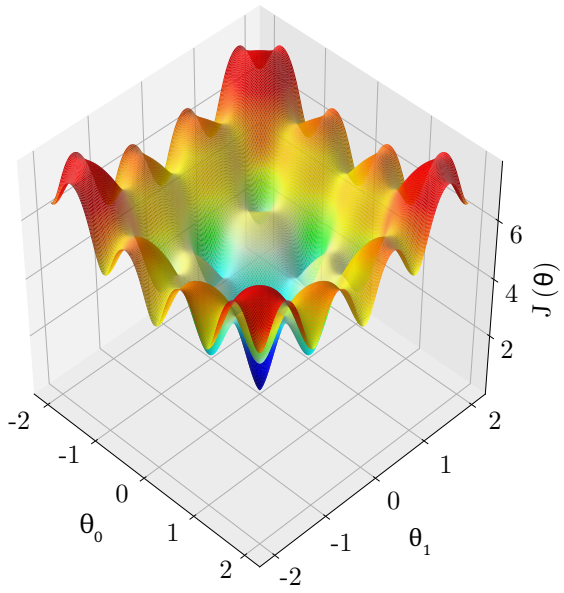
3.7.3 Optimization

To minimize the loss $J(\boldsymbol{\theta})$ most current *DNN* approaches rely on first-order iterative optimization algorithms. This includes for example Stochastic Gradient Descent [104], Nesterov Momentum [119], RMSProb, Adam [63] and AdaGrad [29]. We will start this section by describing Gradient Descent and Stochastic Gradient Descent. Then we will discuss problems these approaches face at the optimization of functions with pathological curvature, and how Momentum can be used to solve them. Finally we will describe Adam, which is the optimizer we use in our experiments.

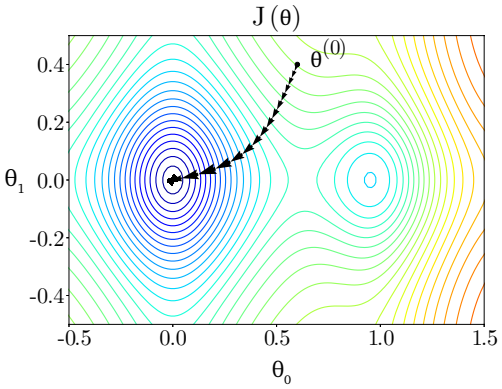
3.7.3.1 Gradient Descent

Gradient Descent [14] is a simple and well known method for finding a local minimum of a function. It is based on iteratively first determinig the direction that decreases the cost function the fastest, and then making a small change of the parameters $\boldsymbol{\theta}$ in that direction. Assuming that our cost function $J(\boldsymbol{\theta})$ is differentiable, its gradient is defined as $\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ and points in the direction that increases $J(\boldsymbol{\theta})$ the fastest. This means if we make a small step of $\boldsymbol{\theta}$ in the opposite direction of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, the value of the cost function decreases. Whereby the size of this step is defined by the hyperparameter η which is also referred to as the learning rate.

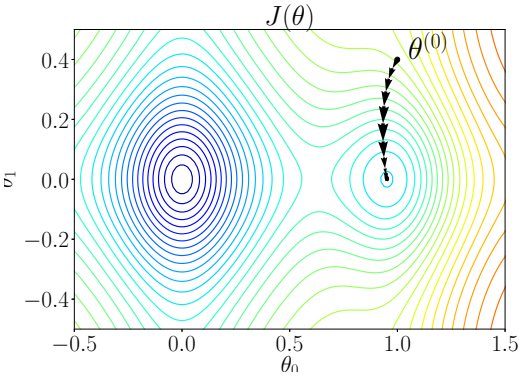
An often discussed property of Gradient Descent and related methods like *SGD* and Adam is that they have no guarantee of converging to a global minimum, but rather might get stuck at a local minimum. Figure 3.8 illustrates this problem for a simple two-dimensional function by showing how the algorithm converges to a stationary point, depending on the point from which the Gradient Descent algorithm is started $\boldsymbol{\theta}^{(0)}$. It is however an ongoing topic of debate how this affects training of *DNN* architectures in practice. Loss functions of common *DNN* architectures are very high dimensional and typically have 10^5 up to 10^8 parameters, and the structure of such high-dimensional loss surfaces is yet poorly understood. Earlier work stated that training typically converges to saddle points rather than local or global minima [69]. However more recent work claims that local minima might in fact be more frequent than previously believed [38]. Furthermore it should be noted that finding the global minimum of the loss evaluated on



(a) 3D plot of the function $J(\boldsymbol{\theta})$ for which we want to find a miniumum. The Ackley Function is used in this example.



(b) Contour plot of the Gradient Descent iterations for $\boldsymbol{\theta}^{(0)} = [0.6, 0.4]$



(c) Contour plot of the Gradient Descent iterations for $\boldsymbol{\theta}^{(0)} = [1.0, 0.4]$

Figure 3.8: Example for minimizing a function with **Gradient Descent**. In 3.8a one can see that the given function has several local minima and a single global minima at $[0, 0]$. We show the trajectory of the algorithm for two different initializations $\boldsymbol{\theta}^{(0)}$ in 3.8b respectively 3.8c. While both initializations lie relatively close to each other, the global minima is only reached in one case, in the other case the algorithm converges to a local minima.

a training set might not even be desirable in practice, since there is no guarantee that this global minimum also generalizes well to data unseen during training [19].

3.7.3.2 Stochastic Gradient Descent

Gradient Descent as discussed in the previous section uses the complete set of training examples $\mathcal{X}$ to calculate the gradient of the loss function in each iteration. However for the training of *DNNs* it is common to use mini-batch based methods. These methods use in each iteration a small subset of samples from the complete training set $\mathcal{X}$ to calculate an estimate $\hat{\boldsymbol{g}}$ of the gradient. This subset is commonly referred to as mini-batch, randomly

Algorithm 1 Stochastic Gradient Descent [104]

Require: Parameter initialization $\boldsymbol{\theta}$

1: Initialize iteration counter: $k \leftarrow 1$
2: **while** $k \neq$ maxiter **do**
3: Sample a mini-batch of B samples $\mathcal{B} = \{\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(B)}\}$
4: Compute a gradient estimate for current mini-batch: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{B} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)\right)$
5: Perform step in opposite direction of gradient: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \hat{\boldsymbol{g}}$
6: Increase iteration counter: $k \leftarrow k + 1$
7: **end while**

sampled from $\mathcal{X}$, and its size is denoted as batch size B.

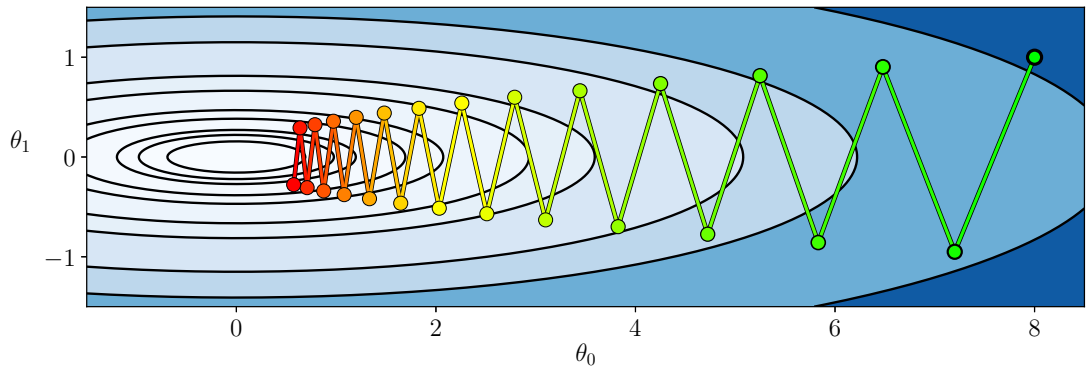
One reason to use mini-batches is that computing the exact gradient using the complete dataset $\mathcal{X}$ is often computationally expensive. Instead, it is often sufficient to use an estimate calculated with a mini-batch. Furthermore, it has been shown that the mini-batch based methods can help to improve generalization [86] and escape local minimas and saddlepoints.

A commonly used mini-batch based variant of Gradient Descent is *SGD* [104], which is summarized in Algorithm 1.

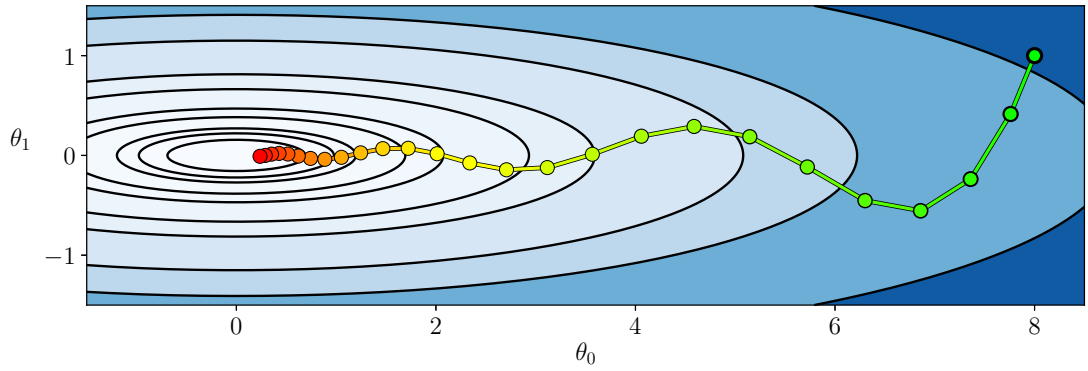
3.7.3.3 Momentum

As described in the previous section Gradient Descent respectively *SGD* rely solely on the first order derivative of the cost function to determine a descent direction at each iteration. While this is simple, there are also certain cases where plain first-order methods have problems. For example when optimizing loss functions with pathological curvature, i.e. functions with a long narrow valley where the base of the valley is a direction of low reduction and low curvature that needs to be followed to decrease the cost [83]. For such functions Gradient Descent is prone to produce iterations following a zig-zag path instead of going straight in the direction of a minima, this significantly slows down convergence. Another problem present in Gradient Descent optimization is given by functions with flat regions. In flat regions the respective gradients are small, and consequently also the speed at which the function is traversed by Gradient Descent.

One approach to alleviate the above mentioned problems is to use second-order optimization algorithms e.g. Newton's method [83], which additionally incorporate the functions curvature. However second-order methods require the calculation of the Hessian, which is intractable for large *DNNs* because the computational complexity of calculating the Hessian grows quadratically with the number of parameters. Even though several approaches have been made to decrease computational complexity of second-order methods by approximating the Hessian, e.g. inexact-Newton methods [84, 129] and quasi-Newton methods [9, 131]. The currently most dominant approach to *DNN* training is to use first-order methods in combination with Momentum. Momentum can alleviate problems



(a) Gradient Descent



(b) Gradient Descent with Momentum $\alpha = 0.7$

Figure 3.9: Visualization of Gradient Descent iterations. Figure 3.9a shows how Gradient Descent iterates the cost function along a zig-zag path. Figure 3.9a shows how adding **Momentum** helps to attenuate this problem resulting in a smoother trajectory.

attributed to the unawareness of function curvature [119], without actually requiring the expensive calculation of second-order derivatives. Figure 3.9 shows an example of how Gradient Descent iterates a function with pathological curvature in a zig-zag path, and how this problem can be attenuated by using Momentum.

To add Momentum to *SGD*, Algorithm 1 can simply be modified by first storing an exponentially decaying moving average of gradients at each iteration in $\boldsymbol{v}$:

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \eta \hat{\boldsymbol{g}}, \tag{3.19}$$

where $\alpha \in [0, 1)$ is a hyperparameter controlling how quickly the influence of past gradients decays. And then applying updates to $\boldsymbol{\theta}$ at each iteration using $\boldsymbol{v}$ instead of the gradient estimate $\hat{\boldsymbol{g}}$:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}. \tag{3.20}$$

Algorithm 2 Adam [63]

Require: Parameter initialization $\boldsymbol{\theta}$, small constant ϵ for numerical stability
Require: Exponential decay rates for moment estimates β_1 and β_2 both in $[0, 1)$
Require: Step size η
Initialize iteration counter: $k \leftarrow 1$
Initialize 1st and 2nd moment variables: $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
while $k \neq$ maxiter **do**
  Sample a minibatch of B samples $\mathcal{B} = \{\boldsymbol{x}^{(0)}, \boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(B)}\}$
  Compute gradient estimate for current minibatch: $\hat{\boldsymbol{g}} \leftarrow \frac{1}{B} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)\right)$
  Update biased first moment estimate: $\boldsymbol{s} \leftarrow \beta_1 \boldsymbol{s} + (1 - \beta_1)\, \hat{\boldsymbol{g}}$
  Update biased second moment estimate: $\boldsymbol{r} \leftarrow \beta_2 \boldsymbol{r} + (1 - \beta_2)\, \hat{\boldsymbol{g}} \odot \hat{\boldsymbol{g}}$
  Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \beta_1^k}$
  Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \beta_2^k}$
  Compute update: $\Delta\boldsymbol{\theta} = -\eta \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \epsilon}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  Increase iteration counter: $k \leftarrow k + 1$
end while

As a theoretical explanation how Momentum works often an analogy to physics is used. The consecutive iterations downwards the slope of the cost function can be seen as a ball rolling down a hill, and the velocity of the ball is modeled by $\boldsymbol{v}$. Consecutive gradients pointing in the same direction are leading to an increase of velocity in that respective direction, while gradients in opposite directions are cancelled out. This can also be seen in the example shown in Figure 3.9, where the zig-zag path is smoothed by decreasing the strength of updates in the direction of θ_0 while the strength of updates for θ_1 is increased. A more comprehensive study about why Momentum works can be found in [37].

3.7.3.4 Adam

Another commonly used optimization algorithm for *DNN* training is Adam [63]. Together with others like RMSProb and AdaGrad [29] Adam belongs to the group of algorithms that use adaptive learning rates. Simply put, they use an individual step size for each parameter, contrarily to *SGD* where a global step size is used jointly for all parameters. This is motivated by the fact that the value of the cost function is often more sensitive to the change of some parameters, while being less sensitive to the change of other parameters. In the following we will give an introduction about the intuition behind Adam, and refer the interested reader to the work of Kingma and Ba [63] for a more comprehensive review.

At each iteration Adam performs the parameter update step $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$ whereby

$$\Delta\boldsymbol{\theta} = -\eta \cdot \underbrace{\frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}}}}_{SNR}, \tag{3.21}$$

and $\hat{\boldsymbol{s}}$ are $\hat{\boldsymbol{r}}$ are exponential decaying averages of the first respectively second order non-central moment of the gradient estimates $\hat{\boldsymbol{g}}$. Therefore the size of the step for each parameter is scaled by the ratio between the estimated mean and square root of uncentered variance of $\boldsymbol{g}$. Kingma and Ba [63] denote this ratio as Signal-to-Noise Ratio (SNR), and argue that a larger *SNR* indicates the gradient estimate $\hat{\boldsymbol{g}}$ to be closer to the true gradient. Therefore larger steps are made for parameters with a large *SNR*, respectively smaller steps for parameters with a smaller *SNR*.

Algorithm 2 shows pseudo-code for Adam. The exponential decaying averages of the first and second order non-centered moments are stored in $\boldsymbol{s}$ respectively $\boldsymbol{v}$, and the decay rates of these moving averages are set by the parameters β_1 and β_2, typical choices are for example $\beta_1 = 0.9$ and $\beta_2 = 0.999$. At the beginning of training both decaying averages are initialized as $\boldsymbol{s} = \boldsymbol{r} = \mathbf{0}$. However this introduces a bias on the estimate towards smaller values, especially during the first iterations. Therefore a bias correction has to be applied on both estimates, and the bias corrected values are stored in $\hat{\boldsymbol{s}}$ and $\hat{\boldsymbol{r}}$.

In general Adam is less sensitive to choice of η compared to non-adaptive learning rate algorithms like *SGD*. This is a significant advantage in practice, since tuning the learning rate can often be a computationally expensive process. However this does not come without drawbacks. It has been reported that while Adam performs well in the initial portion of training, it is often outperformed by SGD at later stages of training, and furthermore that Adam fails to converge to an optimal solution in certain cases [102]. Consequently several works have been published trying to establish modified versions of Adam improving performance e.g. [59, 79, 81]. Nevertheless Adam as it was proposed by Kingma and Ba [63] is still the most dominant adaptive learning rate optimizer used for *DNN* training, and we also use it in our experiments.

3.8 Convolutional Neural Networks for Semantic Image Segmentation

The first end-to-end trainable approach on semantic image segmentation using *CNNs* were Fully Convolutional Networks (FCN) introduced in 2015 by Long et al. Since then a variety of further approaches on semantic segmentation with *CNNs* have been introduced, including DeepLab [16], Pyramid Scene Parsing Network [139], U-Net [105] and Gated Shape *CNNs* [122]. However all of these previously mentioned methods come at a cost: they are trained supervised, and therefore require human-created, pixel-level annotations of semantically consistent areas in images. Consequently various approaches have

been introduced that try to alleviate annotation cost in semantic segmentation. Weakly-supervised approaches require only coarse annotations such as bounding boxes i.e. location and scale of segments [26, 60, 97, 103, 115], scribbles i.e. partially pixel-wise annotation of segments [74, 82, 123, 124] or image-level annotations [2, 62, 71, 132]. Semi-supervised approaches require a small dataset with pixel-wise annotations and a large unlabeled dataset [46, 116]. However all approaches mentioned above still require human-annotated data at some point, and in this work we want to concentrate on entirely unsupervised approaches.

Until now a remarkably small amount of work on unsupervised semantic segmentation has been published, including Invariant Information Clustering (IIC) [51], which we will thoroughly describe in Section 5.2. Another recently introduced approach that states to perform unsupervised semantic segmentation is SegSort [47]. SegSort uses an Expectation-Maximization framework to map each pixel of an input image into a feature space, and pixels that are close in this space are assigned to the same segment. While they achieve good performance on the Pascal VOC dataset [30], their approach still relies on annotations at multiple levels. To be more specific, they use a backbone network that is pre-trained on ImageNet using image-level class annotations. During training they use pseudo ground truth masks generated by a HED contour detector [137]. Whereby the HED contour detector is trained with the annotated images of the BSDS500 [4] dataset. Therefore SegSort can not be considered as unsupervised, and we omit it for further comparisons.

Besides the above mentioned methods there are several other unsupervised *CNN* based approaches for segmentation, which are however not designed to create semantic segmentation but only visually consistent segmentations. This includes the approach of [61] that incorporates the Mumford-Shah [91] functional with *CNNs*. W-Net [136], that uses an Encoder-Decoder architecture together with a normalised cut loss for regularization. And the approach of Kanezaki et. al. where a *CNN* is trained such that spatially close pixels with similar representations in the feature space are assigned to the same segments [57].

4 Method

4.1 Introduction

We start this section by briefly motivating the usage of mutual information for semantic image segmentation in Section 4.1.1, followed by a high-level overview of our approach on unsupervised semantic image segmentation in Section 4.1.2. We formally define the problem we want to tackle in Section 4.2, followed by a thorough description of our approach on unsupervised semantic image segmentation in the following sections.

4.1.1 Motivation

Traditional image segmentation methods rely on low-level features like color, texture or edges to assign pixels to segments [3, 13, 21, 32, 52, 73, 128]. While those methods perform well at creating *visually* consistent segmentations, they often fail to be *semantically* consistent. Primarily because heterogenous appearance of low-level features does not imply semantic similarity [85].

Figure 4.1 illustrates this problem. Consider the sub-areas of the depicted person in Figure 4.1a: face, hair and clothing are of completely different color, texture and shape. By only observing low-level features of these areas, it is eminently difficult to detect that they depict a single semantic entity: a person. The segmentation shown in Figure 4.1b was created by assigning pixels with similar color to the same image segments. And it can be seen how areas from the persons face and clothing were assigned to different segments due to their vast difference in color intensity. Contrarily, the segmentation shown in Figure 4.1c is semantically consistent, whereby all pixels depicting the person are assigned to one segment and all pixels depicting background segments are assigned to another segment.

We argue that mutual information is effective at determining if two image areas are semantically similar. For example consider two sets $\mathcal{P}_1$ and $\mathcal{P}_2$ both containing pixels depicting a person. Since the visual appearance of $\mathcal{P}_1$ and $\mathcal{P}_2$ might be vastly different, purely pixel-based measures will most likely fail to detect that both sets are semantically

(a) Input (b) Seg. 1 (c) Seg. 2

Figure 4.1: Two possible segmentations for the input image shown in **(a)**, **(b)** segments the image into dark and and bright areas, and **(c)** segments the image into areas depicting a person and a background area. While both segmentations are visually consistent, only the second segmentation is also semantically consistent.

very similar. Contrarily mutual information, as discussed in Section 2.3.7, quantifies the decrease of uncertainty in one variable if we know the value of another. In other words, any information in $\mathcal{P}_1$ that tells us something about the content of $\mathcal{P}_2$ increases their mutual information. Therefore in general we can assume that $\mathcal{P}_1$ and $\mathcal{P}_2$ have larger mutual information if they depict semantically similar areas, as if they depict semantically completely different areas.

4.1.2 Overview

Deep InfoMax [42], which we discussed in Section 2.5.2, has already shown that mutual information can effectively be utilized to learn high-level features of images. Their approach works entirely unsupervised, and they showed that these high-level features can be used to determine the semantic similarity of entire images. Our approach builds up on this concept, however we propose to learn high-level features not representing entire images, but rather multiple high-level features, each encoding image areas depicting one semantic class.

Similar as Deep InfoMax, we learn these high-level features by maximizing their mutual information with low-level features covering overlapping image patches. However we do not maximize mutual information with all low-level features of an image, but only with those that cover image areas depicting the class of the respective high-level feature. Since we do not use to ground truth annotations during training, we use mutual information estimates to determine for each high-level feature the respective low-level features to maximize mutual information with.

4.2 Problem Definition

Given a set of N unlabeled images $\mathcal{I} = \{\mathbf{x}^{(u)} \in \mathcal{X}\}_{u=1}^{N}$ with $\mathcal{X} = \mathbb{R}^{C \times M \times N}$, drawn from the empirical distribution $\hat{p}_{\text{data}}$. For each image $\mathbf{x}^{(u)}$, every single pixel should be assigned to one class out of the set of K classes $\mathcal{Z} = \{0, \ldots, K-1\}$. Therefore, each image in $\mathcal{I}$ is segmented into a *maximum* number of K different segments. The corresponding semantic image segmentation can be defined as the matrix $\mathbf{Z}$ with the entries:

$$Z_{i,j} \in \mathcal{Z}, \quad i = (1, \ldots, M), j = (1, \ldots, N). \tag{4.1}$$

Furthermore all pixels that are assigned to the same segment in $\mathcal{Z}$ should depict semantically similar areas, whereby this should not only hold for each separate image $\mathbf{x}^{(u)}$ but also across all images in $\mathcal{I}$. Consequently pixels from any two images $\mathbf{x}^{(u)}$ and $\mathbf{x}^{(v)}$ that are assigned to the same segment in $\mathcal{Z}$ should depict semantically similar areas.

4.3 Our Approach

As described in Section 4.1, for each image $\mathbf{x}^{(u)}$ we extract local features covering overlapping image-patches and multiple high-level features each covering the entire input image. The local features are defined as $\mathbf{L} \in \mathbb{R}^{U \times V \times P}$, where $U = (M/d)$, $V = (N/d)$ and $d \in \mathbb{N}_+$ is a downsampling rate from the resolution of $\mathbf{x}^{(u)}$. Furthermore, we use $\mathbf{L}_{i,j} \in \mathbb{R}^P$ to denote the P-dimensional local feature at the spatial position (i, j). The high-level features are defined as $\mathbf{H} \in \mathbb{R}^{K \times P}$, and we use $\mathbf{H}_k \in \mathbb{R}^P$ to denote the high-level feature that encodes the content of the k-th segment. Furthermore, we define the mutual information volume $\mathbf{V} \in \mathbb{R}^{U \times V \times K}$, where each element is given by the mutual information between a local and a high-level feature i.e. $V_{i,j,k} = I(\mathbf{L}_{i,j}; \mathbf{H}_k)$.

Considering that a local feature $\mathbf{L}_{i,j}$ that encodes image areas depicting pixels of the k-th segment has a large mutual information with the k-th high-level feature $\mathbf{H}_k$. A segmentation $\mathbf{Z}'$ can be calculated by determining at each spatial position of $\mathbf{V}$ the index k of the high-level feature with the largest mutual information. Consequently each element of $\mathbf{Z}'$ is given by:

$$Z'_{i,j} = \arg\max_{k \in \mathcal{Z}} V_{i,j,k}, \quad i = (1, \ldots, U), j = (1, \ldots, V). \tag{4.2}$$

We obtain the final image segmentation $\mathbf{Z}$ by upsampling $\mathbf{Z}'$ to the resolution of the input image $\mathbf{x}^{(u)}$, with:

$$\mathbf{Z} = \text{up}(\mathbf{Z}'), \tag{4.3}$$

where up(.) is a bilinear upsampling operation.

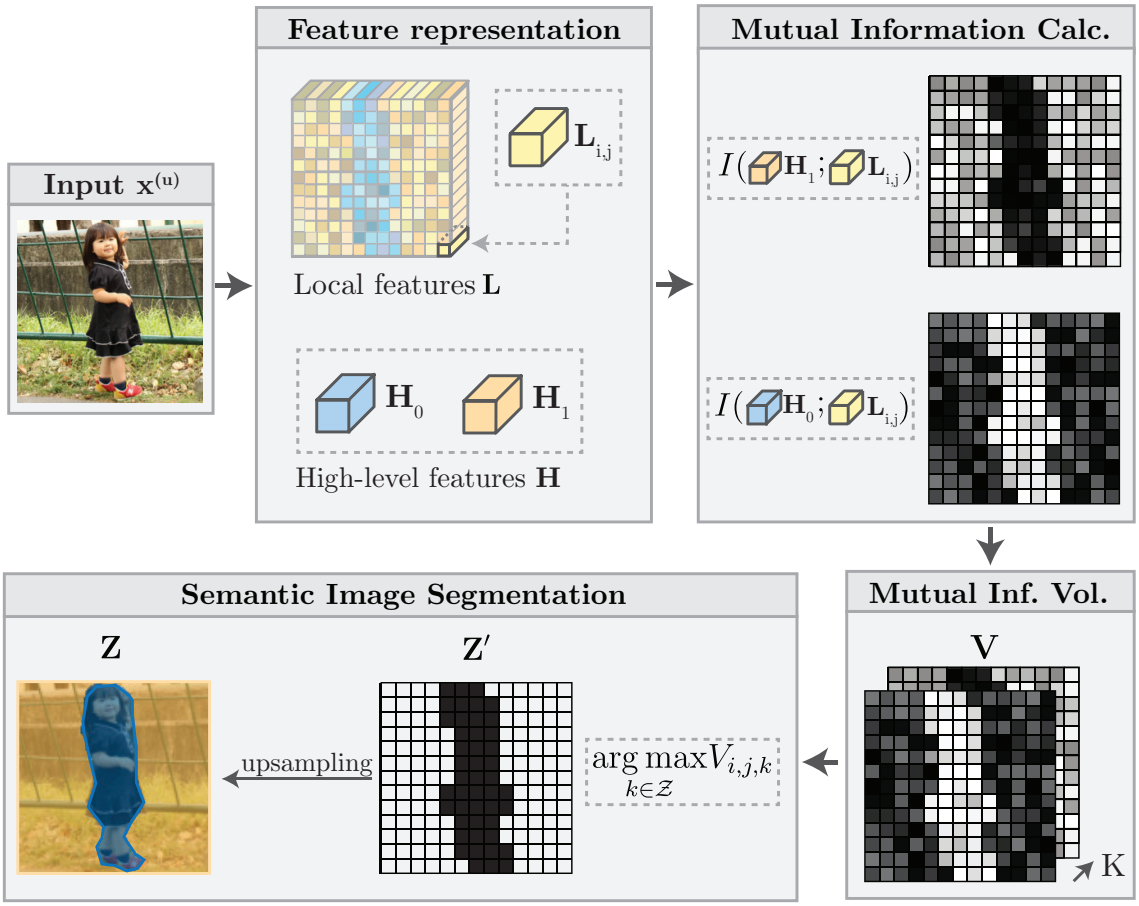
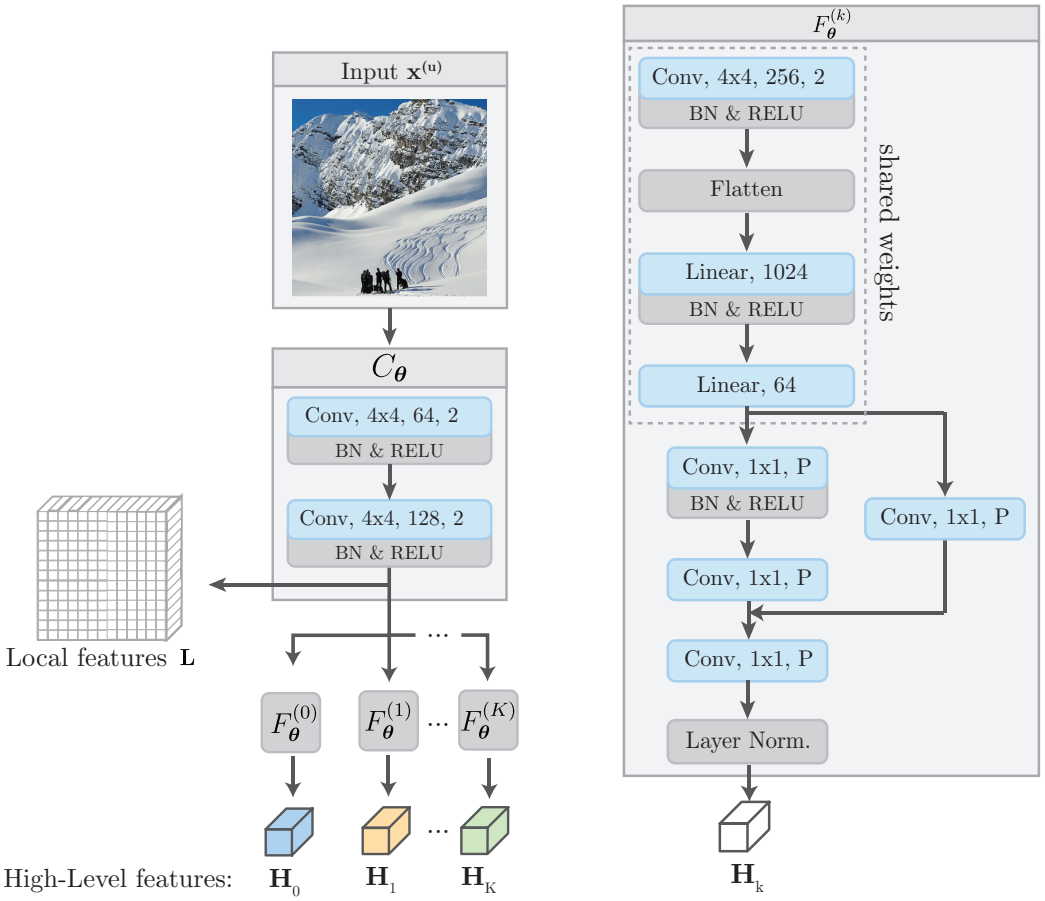


Figure 4.2: Overview of our approach for two segments, i.e. $K = 2$. **Feature representation**: We calculate for an input image $\mathbf{x}^{(u)}$ *local* features $\mathbf{L}$ that cover overlapping patches and the *high-level* features $\boldsymbol{H}$ that cover the entire image. **Mutual Information Calculation**: We then calculate the mutual information between each local feature $\mathbf{L}_{i,j}$ and each high-level feature in $\boldsymbol{H}_k$, where light colors indicate high and dark colors low mutual information. **Mutual Information Volume**: The mutual information volume $\mathbf{V}$ is created by stacking the mutual information values of all K high-level features. **Semantic Image Segmentation**: We calculate $\mathbf{Z}'$ by $\arg\max_{k \in \mathcal{Z}} v_{i,j,k}$ at each spatial position of $\mathbf{V}$, and then upsample $\mathbf{Z}'$ to the resolution of the input image to get the final image segmentation $\mathbf{Z}$. *Note that by increasing K our approach naturally extends to more classes.*

4.3.1 Feature Computation

To compute the local features $\boldsymbol{L}$ and high-level features $\boldsymbol{H}$ we use a Deep Neural Network (DNN) parametrized by $\boldsymbol{\theta}$. In the following we introduce the composition of our network, and Figure 4.3 depicts the structure in more detail.

Given an input image $\boldsymbol{x} \in \mathcal{X}$, the local feature at the spatial position (i, j) is computed by $C_{\boldsymbol{\theta}}^{(i,j)} : \mathcal{X} \to \mathcal{Y}$, whereby $\mathcal{Y} = \mathbb{R}^P$. We denote the local feature extractor $C_{\boldsymbol{\theta}}(\boldsymbol{x})$



(a) Main computation path (b) High-level feature extractor $F_{\boldsymbol{\theta}}$

Figure 4.3: Feature computation. **(a) Main computation path**: An input image $\boldsymbol{x}^{(u)}$ is first passed through $C_{\boldsymbol{\theta}}$ to compute the local features $\boldsymbol{L}$, and then a separate high-level feature extractor $F_{\boldsymbol{\theta}}^{(k)}$ is used to compute each of the K high-level features. **(b) High-level feature extractor** $F_{\boldsymbol{\theta}}$: takes the local features $\boldsymbol{L}$ as input and computes the high-level feature $\mathbf{H}_k$. The weights of the respectively annotated blocks are shared over all K high-level feature extractors. Table 4.1 lists the notations we used in the illustration.

Block	Description
Conv, W×W, C, d	Convolution with a filter size of W×W, C channels and a stride of d
BN & RELU	Batch Normalization layer followed by a ReLu activation function
Linear, C	Fully-connected layer with C channels
Flatten	Reshapes a 2D feature map to a 1D vector
Layer Norm.	Layer Normalization

Table 4.1: Notation used for the computational blocks in Figure 4.3.

computing all $U \cdot V$ local features of an input image $\boldsymbol{x}$ as following:

$$C_{\boldsymbol{\theta}}(\boldsymbol{x}) = \{C_{\boldsymbol{\theta}}^{(i,j)}\}_{i,j=1}^{U \times V}. \tag{4.4}$$

The k-th high-level feature $\boldsymbol{H}_k$ is calculated by $G_{\boldsymbol{\theta}}^{(k)}\colon \mathcal{X} \to \mathcal{Y}$, whereby each high-level feature is computed by further processing the low-level features $\boldsymbol{L}$, i.e. the output of $C_{\boldsymbol{\theta}}(\boldsymbol{x})$. Therefore we use for each of the K high-level features a separate high-level feature extractor $F_{\boldsymbol{\theta}}^{(k)} : \mathcal{Y}^{U \times V} \to \mathcal{Y}$ taking all local features computed by $C_{\boldsymbol{\theta}}(\boldsymbol{x})$ as input, $G_{\boldsymbol{\theta}}^{(k)}$ is then given as:

$$G_{\boldsymbol{\theta}}(\boldsymbol{x})^{(k)} = F_{\boldsymbol{\theta}}^{(k)} \circ C_{\boldsymbol{\theta}}(\boldsymbol{x}) \quad \forall k \in \{1, \ldots, K\}, \tag{4.5}$$

where $\circ$ denotes the function composition operator.

4.3.2 Training Objective

As already described in Section 4.3, we want to maximize mutual information between each local feature $\mathbf{L}_{i,j}$ and the respective high-level feature $\mathbf{H}_k$ it has the largest mutual information with. This could be done by first computing the values of the mutual information volume $\mathbf{V}$, to determine the indices $k'_{i,j}$ that give for each spatial position (i, j) the respective high-level feature with the largest mutual information:

$$k'_{i,j} = \arg\max_{k \in \mathcal{Z}} V_{i,j,k} \quad i = (1, \ldots, U), j = (1, \ldots, V). \tag{4.6}$$

Followed by maximization of the mutual information between each $\mathbf{L}_{i,j}$ and the respective high-level feature with maximum mutual information according to $k'_{i,j}$:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{X \sim \hat{p}_{\text{data}}} \left[\frac{1}{UV} \sum_{i,j}^{U,V} \widehat{I}\left(L_{\boldsymbol{\theta}}^{(i,j)}(X); G_{\boldsymbol{\theta}}^{(k'_{i,j})}(X) \right) \right], \tag{4.7}$$

where $\hat{I}$ is a mutual information estimator, which we describe in Section 4.3.4.

However this approach comes with several challenges. One should consider that we do not know the exact mutual information values, but rather compute only estimates. In particular at the beginning of training these estimates are entirely arbitrary, since they depend on the randomly initialized network parameters. Relying on those estimates to maximize the mutual information between a $\mathbf{L}_{i,j}$ and an incorrectly chosen $\boldsymbol{H}_k$ could easily lead to degenerate solutions, since incorrect estimates would be reinforced. Furthermore determining the indices $k'_{i,j}$ requires the usage of a non-differentiable argmax function.

To alleviate those problems, we do not make hard assignments by simply choosing for each $\mathbf{L}_{i,j}$ the corresponding $\boldsymbol{H}_k$ with maximum mutual information, but rather make soft assignments based on an assignment probability proportional to the mutual information estimates. Therefore we transform the mutual information volume $\mathbf{V}$ into a probability volume $\mathbf{P}' \in \mathbb{R}^{U \times V \times K}$ with each element $p'_{i,j,k}$ given by the application of the following

element-wise mapping:

$$p'_{i,j,k} = \frac{\exp\left(\tau \cdot v_{i,j,k}\right)}{\sum_{k'} \exp\left(\tau \cdot v_{i,j,k'}\right)}, \tag{4.8}$$

where τ is a hyperparameter that controls the smoothness of the resulting distribution. For $\tau = 0$ probability is distributed uniformly across high-level features; for $\tau = 1$ the distribution becomes a softmax; and for $\tau \to \infty$ all probability mass is concentrated at the pair of local and high-level features with maximum mutual information, i.e. mutual information estimates are treated as ground truth.

Using the probability volume $\boldsymbol{P'}$ we can now define the function $S_{\boldsymbol{\theta}}^{(i,j)}\colon \mathcal{X} \to \mathcal{Y}$ that computes for each spatial position (i, j) a soft global feature assignment by:

$$S_{\boldsymbol{\theta}}^{(i,j)}(X) = \sum_k p'_{i,j,k} \cdot G_{\boldsymbol{\theta}}^{(k)}(X). \tag{4.9}$$

And rewrite Equation 4.7 as following to get our training objective:

$$\max_{\boldsymbol{\theta}} \mathbb{E}_{X \sim \hat{p}_{\text{data}}} \left[\frac{1}{UV} \sum_{i,j} \widehat{I}\left(L_{\boldsymbol{\theta}}^{(i,j)}(X); S_{\boldsymbol{\theta}}^{(i,j)}(X) \right) \right], \tag{4.10}$$

4.3.3 Class Assignment Probabilities

We can use the probability volume $\boldsymbol{P'}$ to model the class assignment probabilities for each pixel of an input image $\boldsymbol{x}$. We let $p_{i,j,k}$ the entries of a probability volume $\mathbf{P} \in \mathbb{R}^{M \times N}$ that is given by upsampling $\mathbf{P'}$ to the resolution of the input image $\boldsymbol{x}$, i.e. $\mathbf{P} = \text{up}(\mathbf{P'})$. And furthermore let $Z_{i,j}$ be a discrete random variable taking values from our discrete labeling set $\mathcal{Z}$, modeling the probability that the pixel at the spatial position (i, j) should be assigned to the class z_k. We can then simply interpret the entries of $\mathbf{P}$ as the respective class assignment probabilities by:

$$P(Z_{i,j} = z_k) = p_{i,j,k}. \tag{4.11}$$

4.3.4 Mutual Information Estimation

While in the previous sections we simply used $\hat{I}$ to denote a mutual information estimator, we will now describe in this section how we tackle mutual information estimation in our approach. We already thoroughly discussed several approaches on variational mutual information estimation in Section 2.4.

At first we want to note that our approach does not rely on the exact value of mutual information, but we rather assign pixels to segments by comparing relative mutual information estimates. Therefore, using mutual information estimators is suitable even if they are not of high confidence. This is also confirmed by our empirical results. For further discussion about the formal limitations of variational mutual information estimators we

refer to Section 2.4.2.3

Similar as proposed in Deep InfoMax [42] we will use the Nowozin lower bound on the Jensen-Shannon (JS) divergence, which we introduced in Section 2.4.2.2, as a lower bound on mutual information. Therefore an estimator for the mutual information present in Equation 4.10 is given as following:

$$\widehat{I}\left(L_{\boldsymbol{\theta}}(X); S_{\boldsymbol{\theta}}(X)\right) = \mathbb{E}_{\mathbb{J}}\left[-\operatorname{sp}\left(-T\left(L_{\boldsymbol{\theta}}(X), S_{\boldsymbol{\theta}}(X)\right)\right)\right] - \mathbb{E}_{\mathbb{M}}\left[\operatorname{sp}\left(L_{\boldsymbol{\theta}}(X), S_{\boldsymbol{\theta}}(X)\right)\right], \quad (4.12)$$

whereby we omit the spatial indices (i, j) for clarity and use $sp(z) = \log(1 + e^z)$ to denote the softplus function. Furthermore $T\colon \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ is a discriminator function that should assign high scores to features coming from the joint distribution $\mathbb{J}$ and low scores to features coming from the product of marginal distribution $\mathbb{M}$. Samples from $\mathbb{J}$ are drawn by computing $L_{\boldsymbol{\theta}}(\mathbf{x}^{(u)})$ and $S_{\boldsymbol{\theta}}(\mathbf{x}^{(u)})$ both from the same image $\mathbf{x}^{(u)}$, and samples from $\mathbb{M}$ are drawn by computing $L_{\boldsymbol{\theta}}(\mathbf{x}^{(u)})$ and $S_{\boldsymbol{\theta}}(\mathbf{x}^{(v)})$ from two different images $\mathbf{x}^{(u)}$ and $\mathbf{x}^{(v)}$. In other words, the input of the discriminator is a pair of features and the discriminator T needs to determine if they have been computed from the same image or not.

In principle the discriminator function T can be any arbitrary function that takes a pair of features as input and outputs a real-valued score. However this function needs to be evaluated separately for every feature pair to evaluate. An efficient approach to implement T was proposed by the encode-and-dot-product architecture of Local DeepInfo Max [42], where the score of two given features is their dot product. Therefore, for each image in a batch we only need to perform a single forward pass through $L_{\boldsymbol{\theta}}$ and $S_{\boldsymbol{\theta}}$ to compute all local and global features. The score of any arbitrary feature combination can then efficiently be calculated by the dot-product between the two respective features:

$$T\left(\boldsymbol{H}_k, \boldsymbol{L}_{i,j}\right) = \langle \boldsymbol{H}_k, \boldsymbol{L}_{i,j} \rangle. \quad (4.13)$$

A further advantage of using the dot-product as a discriminator function T is that it also allows to interpret the dot-product between two features as a similarity measure proportional to their mutual information, therefore we can compute the values of $\boldsymbol{P}'$ as follows:

$$p'_{i,j,k} = \frac{\exp\left(\tau \cdot \langle \mathbf{L}_{i,j}, \mathbf{H}_k \rangle\right)}{\sum_{k'} \exp\left(\tau \cdot \langle \mathbf{L}_{i,j}, \mathbf{H}_k \rangle\right)}. \quad (4.14)$$

4.4 Matching Segments

After training we have given image segmentations that assign each pixel to one class in $\mathcal{Z} = \{z_1, \ldots, z_K\}$. To quantitatively compare these segmentations with the ground truth annotations of a dataset that assign each pixel to one class in $\mathcal{Z}' = \{z'_1, \ldots, z'_K\}$ we need to map each of the classes in $\mathcal{Z}$ to one class in $\mathcal{Z}'$.

To calculate these mappings, we first define a complete weighted bipartite graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the vertices $\mathcal{V} = \mathcal{Z} \cup \mathcal{Z}'$, and edges $\mathcal{E} = \{(z_i, z'_i)\}_{z_i \in \mathcal{Z}, z'_i \in \mathcal{Z}}$. Then we set each

entry W_{ij} of the weight matrix $\boldsymbol{W} \in \mathbb{R}^{K \times K}$ to the amount of pixels that are predicted as z_i and labeled with z'_j. And finally we solve the maximum weight matching problem using the hungarian matching algorithm [68].

When we evaluate a given dataset, we use all images in the dataset to calculate $\boldsymbol{W}$, solve the matching problem, and then use the *same* mapping for *each* image.

5 Experimental Evaluation

Contents

5.1 Introduction

In this Section we provide a quantative and qualitative evaluation of our proposed model. At first we introduce in Section 5.2 all of the datasets we use for evaluation. Then we define in Section 5.3 all of the quantitative metrics we will use throughout this section. In Section 5.4 an overview is given on implementation specific details. Followed by baseline results based on K-Means clustering in Section 5.5. And finally we show and discuss the results of our proposed model in Section 5.6.

5.2 Data

Unfortunately until now there is no public dataset available that was specifically designed for evaluation of unsupervised semantic image segmentation algorithms. While it is in general possible to use datasets that have been proposed for supervised algorithms, e.g. Common Objects in COntext (COCO) [12] or Cityscapes [22]. One has to consider that most of these datasets come with some properties that might be problematic for the evaluation of unsupervised methods. This includes for example huge imbalances between the

COCO-stuff class	*COCO* class
Building	Bridge, Building-other, House, Roof, Skyscraper, Tent
Ceiling	Ceiling-other, Ceiling-tile
Floor	Carpet, Floor-marble, Floor-other, Floor-stone, Floor-tile, Floor-wood food, Food-other, Fruit, Salad, Vegetable
Furniture	Cabinet, Counter, Cupboard, Desk-stuff, Door-stuff, Furniture-other, Light, Mirror-stuff, Shelf, Stairs, Table
Ground	Dirt, Gravel, Ground-other, Mud, Pavement, Platform, Playingfield, Railroad, Road, Sand, Snow
Plant	Branch, Bush, Flower, Grass, Leaves, Moss, Plant-other, Straw, Tree
Rawmaterial	Cardboard, Metal, Paper, Plastic
Sky	Clouds, Sky-other
Solid	Hill, Mountain, Rock, Solid-other, Stone, Wood
Structural	Cage, Fence, Net, Railing, Structural-other
Textile	Banner, Blanket, Cloth, Clothes, Curtain, Mat, Napkin, Pillow, Rug, Textile-other, Towel
Wall	Wall-brick, Wall-concrete, Wall-other, Wall-panel, Wall-stone, Wall-tile, Wall-wood
Water	Fog, River, Sea, Water-other, Waterdrops
Window	Window-blind, Window-other

Table 5.1: List of the corresponding *COCO* [76] classes for each *COCO*-stuff [12] class.

number of available samples for each respective class, and semantic classes in the ground-truth annotations that are semantically very similar. We will discuss these problems throughout this section. And furthermore note that we compute all of our quantitative results by matching the class predictions of our method with the provided ground-truth annotations of a dataset as described in Section 4.4.

We use two datasets for evaluation, *COCO*-Stuff [12] and *COCO*-Persons. Both are subsets of the *COCO* [76] dataset, whereby it should be noted that evaluation on the full *COCO* dataset is not feasible for our approach. And has also not been done by any other currently available unsupervised semantic segmentation algorithm. Since both the number of images: 330k, and the number of different classes: 91, are relatively high and therefore difficult to handle. Using *COCO*-Stuff for evaluation of an unsupervised semantic image segmentation approach was proposed by Invariant Information Clustering (IIC), and we will describe the dataset in Section 5.2.1. The second dataset we use is COCO-Persons, which is a novel subset of *COCO* proposed by us in Section 5.2.2.

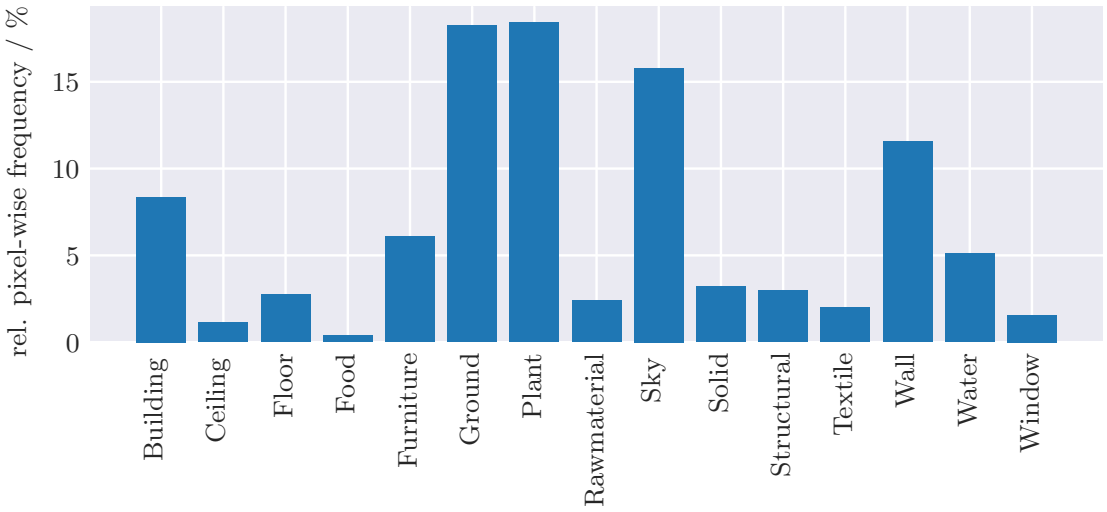


Figure 5.1: Relative pixel-wise class frequency in the *COCO*-Stuff [12] dataset. The classes are vastly imbalanced.

5.2.1 COCO-Stuff

The *COCO*-Stuff [12] dataset divides the classes of the *COCO* dataset [75] into a thing and a stuff category. The stuff classes have been used by the authors of *IIC* [51] to evaluate their unsupervised semantic segmentation approach. And Table 5.1 shows how the respective classes of *COCO* have been mapped to the *COCO*-stuff classes. The resulting dataset contains $K = 15$ classes. While we want to note that *COCO*-Stuff has some properties that make it a suboptimal benchmark for an entirely unsupervised algorithm, we still use it to to get a fair comparison of our approach with *IIC*.

At first the classes in *COCO*-Stuff are chosen rather arbitrary in some cases. For example there are separate classes for windows and for buildings, but pixels that depict doors are also assigned to the buildings class. There is no general reason why doors should be considered as part of a building class while windows are treated separately. This is rather an arbitrary choice from the dataset, and expecting any unsupervised algorithm to come up with this exact solution is unfeasable.

And secondly there is a vast imbalance in the number of provided pixels per class in the dataset. For example only 0.4% of all pixels depict areas labeled with the class food, while more than 15% depict plants. Figure 5.1 shows a complete enumeration of the relative occurence probability for all classes.

To provide a fair point of comparison we use the exact same data setup as in *IIC*. Consequently when we refer to *COCO*-Stuff, we mean the data as it was used by *IIC* for evaluation. And not the dataset as it was initially proposed by [12]. In *IIC* the initial *COCO*-Stuff dataset is reduced from 164k images to 52k images, each resized to

128×128 pixels. And the 15 stuff super-categories are used as classes e.g. water, plant, food, textile. Pixels that are labeled with thing classes are ignored during evaluation and training. Therefore, while we pass all pixels as input to the network, we only propagate gradients into regions labeled with stuff classes. Furthermore following *IIC* we also apply a Sobel filter on the images, and use the RGB channels of each image stacked with the filtering result as input.

5.2.2 COCO-Persons

The second dataset we use for evaluation is also a subset of the *COCO* [76] dataset. However to show the versatility of our approach, we deliberately chose images with quite different characteristics than those of the *COCO*-Stuff dataset. The appearance of the most prevalent classes in the *COCO*-Stuff dataset is relatively homogeneous across instances. For example areas labeled as sky are mostly large textureless regions or areas labeled as plants often have a characteristic green color. In contrast, we purposely let the classes of the *COCO*-Persons dataset have a vastly different appearance across images.

Each image shows one or multiple persons in a large variety of environments. And pixels depicting persons respectively the surroundings are labeled with two separate classes, i.e. $K = 2$. Whereby the persons are shown in different poses, scales and orientations; they have different age, gender, skin-tone and size; and the areas surrounding them are complex and range from indoor to outdoor scenes. Therefore, we argue that creating semantically consistent segmentations for the *COCO*-Persons dataset is much more challenging than for *COCO*-Stuff. In total the dataset consists of 15.399 images, each resized to 128×128 pixels.

5.3 Metrics

While a variety of metrics exist to evaluate semantic segmentation algorithms e.g. Pixel Accuracy (PA), mean *PA*, Frequency Weighted Intersection-Over-Union or mean Intersection-Over-Union (mIoU). We evaluate our results using the *mIoU* and the *PA*, since they are one of the most commonly used metrics in semantic image segmentation [35].

In the following we show how to derive the *mIoU* and the *PA*, following the notation of [24]. Considering a semantic image segmentation with K different classes we let $U \in \mathbb{N}_{+}^{K \times K}$ be a pixel-level confusion matrix with each entry u_{ij} given as following:

$$u_{ij} = \sum_{I \in \mathcal{D}} \left| \left\{ z \in I \text{ such that } S_{gt}^{I}(z) = i \text{ and } S_{ps}^{I}(z) = j \right\} \right|, \tag{5.1}$$

where the set $\mathcal{D}$ contains all images to evaluate, $S_{gt}^{I}(z) = i$ is the ground-truth label of pixel z in image I and $S_{ps}^{I}(z) = j$ is the prediction of pixel z in image I. Consequently we can calculate the total number of pixels *labelled* with i as $g_i = \sum_j u_{ij}$, the total number

of pixels *predicted* with j as $p_j = \sum_i u_{ij}$ and the number of correctly classified pixels as $\sum_i u_{ii}$.

Using the previously defined quantities, the *PA* can be calculated as the ratio between the correct classified pixels and the total amount of pixels:

$$\text{PA} = \frac{\sum_i u_{ii}}{\sum_i g_i}. \tag{5.2}$$

Despite its simplicity, the *PA* is biased in favour of overrepresented classes when evaluating imbalanced datasets. One metric that tries to tackle this problem is the *mIoU*, which can be calculated as following:

$$m_{IoU} = \frac{1}{K} \sum_{i=1}^{K} \frac{u_{ii}}{g_i + p_i - u_{ii}}. \tag{5.3}$$

The *mIoU* calculates the ratio between the intersection and the union of two sets, whereby for segmentation these are the sets of predicted segmentation and the ground truth. It lies in the interval $[0, 1]$ whereby higher values indicate better segmentation performance. Furthermore the *mIoU* can also be expressed as:

$$m_{IoU} = \frac{T_P}{T_P + F_N + F_P}, \tag{5.4}$$

whereby T_p, F_p, and F_n are the numbers of true positive, false positive, and false negative classified pixels for all images $I \in \mathcal{D}$ and all classes K. Therefore not only the number of correct classified pixels is considered but also the amount of false alarms and missed values.

5.4 Implementation Details

All of our experiments were done on a single GPU with 11GB of memory, and a batch size of 64. For training we used the Adaptive Moment Estimation (ADAM) optimizer [63], which we introduced in Section 3.7.3.4, with a learning rate of 10^{-4}. All experiments were stopped after 300 epochs. And the hyperparameters λ and P were set to $1/\sqrt{2048}$ respectively 2048. Additionally during training we did data augmentation by jittering brightness, contrast, saturation and hue and randomly flipped and cropped the input images.

To compute the results for *IIC* [51] we used the codebase provided by the authors, and let all parameters unchanged. Even though they were deliberately chosen for the *COCO*-Stuff dataset, we argue that they should also be reasonable for the *COCO*-Person dataset. Since both are subsets of the same dataset: *COCO*.

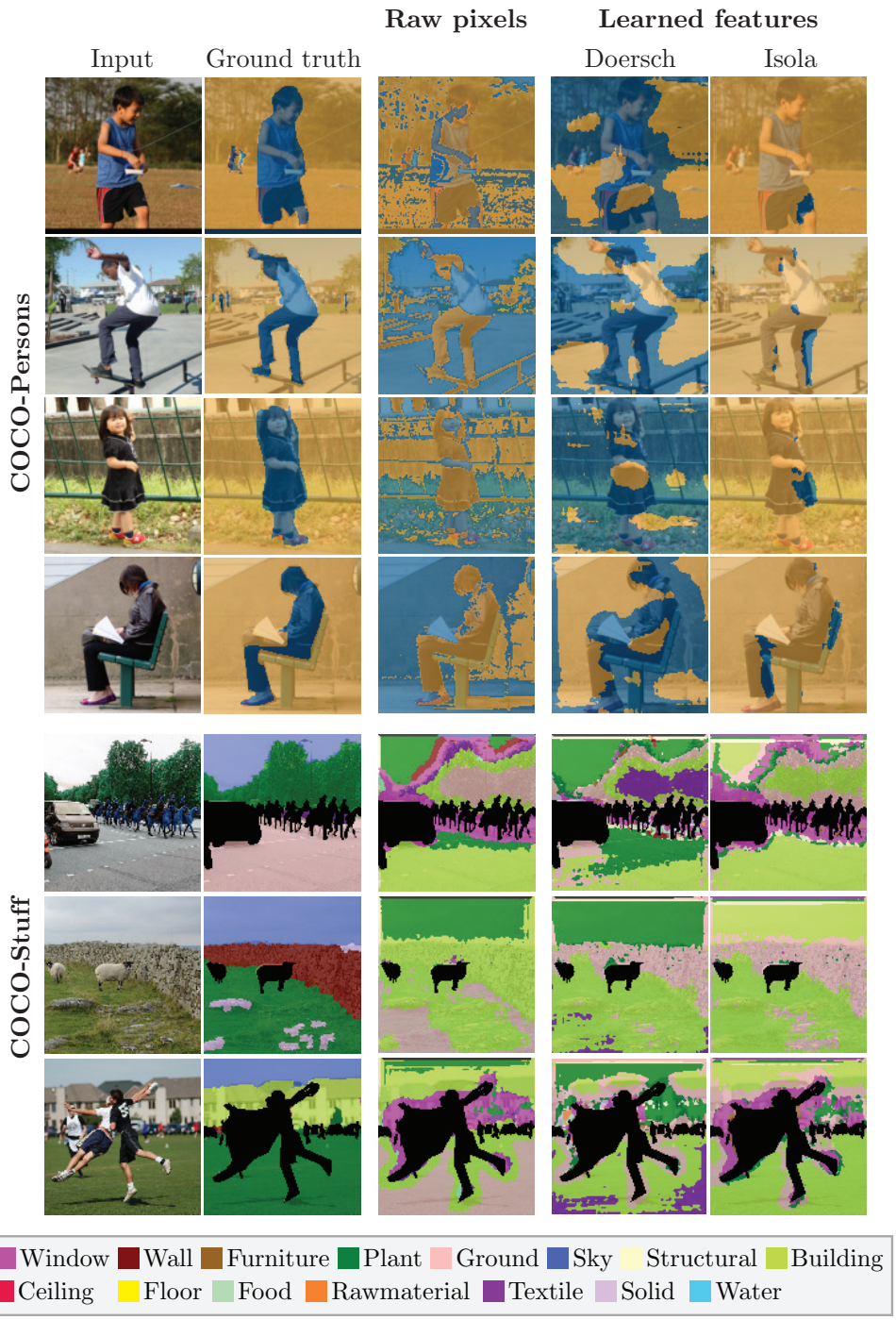


Figure 5.2: Quantitative comparison of the **K-Means segmentation** for the *COCO*-Persons and *COCO*-Stuff dataset. **Raw pixels:** the clustering is applied on the RGB values of the images. **Learned features**: the clustering is applied on pixel-wise features learned by unsupervised representation learning approaches proposed by Doersch et al. [27] respectively Isola et al. [49]. Note how all three approaches fail to resemble the ground truth annotations. Black areas indicate non-stuff pixels that are ignored during evaluation.

5.5 K-Means Segmentation

To show that simple clustering of raw color values is not sufficient to create semantic segmentations. We also compare our results to segmentations created by K-Means clustering. K-Means is a simple generic clustering algorithm. And while being applicable in a wide range of applications, K-Means is also commonly used for unsupervised image segmentation [54, 93]. Note however that K-Means segmentation of raw pixels, contrarily to our work, is not explicitly designed to create *semantic* image segmentation.

In the following we will describe how K-Means can be used for image segmentation. Given N d-dimensional vectors $\{\boldsymbol{x}_p \in \mathbb{R}^d\}_{p=1}^N$, whereby each $\boldsymbol{x}_p$ corresponds to an image pixel and its value is given by the color information of that pixel. The objective is to segment the image into K segments, i.e. assign each pixel to one out of K sets $S = \{\mathcal{S}_i\}_{i=1}^K$, which is done by minimizing the Within-Cluster Sum of Squares (WCSS):

$$\arg\min_{\mathcal{S}} \sum_{i=1}^{K} \sum_{\mathbf{x}_p \in \mathcal{S}_i} \|\mathbf{x}_p - \boldsymbol{\mu}_i\|^2 , \tag{5.5}$$

where $\boldsymbol{\mu}_i$ is the mean of all points in S_i. One approach to solve the optimization problem given in Equation 5.5 is Lloyd's algorithm. At first all cluster centers $\boldsymbol{\mu}_i$ are initialized randomly. Secondly each pixel in $\mathcal{X}$ is assigned to the segment $\mathcal{S}_i$ to which it has the smallest euclidean distance, resulting in the following new segment assignments:

$$S_i = \left\{ \boldsymbol{x}_p : \|\boldsymbol{x}_p - \boldsymbol{\mu}_i\|^2 \leq \left\|\boldsymbol{x}_p - \boldsymbol{\mu}_j\right\|^2 \quad \forall j, 1 \leq j \leq k \right\}_{p=1}^{N} . \tag{5.6}$$

And thirdly the cluster centers are updated using the new assignments:

$$\boldsymbol{\mu}_i = \frac{1}{|\mathcal{S}_i|} \sum_{\boldsymbol{x}_p \in \mathcal{S}_i} \boldsymbol{x}_p . \tag{5.7}$$

The second and third step are then repeated until convergence, which can for example be determined if the change at each iteration of the *WCSS* according to Equation 5.5 falls below a certain threshold. Note that we always apply K-Means clustering on all images of a dataset at once, i.e. one common set of cluster centers is found for all images. This is necessary since we want that image areas from multiple images that are assigned to the same class should be semantically similar. By applying K-Means on each image separately, there would be clearly no dependence between the cluster centers.

In addition to applying K-Means on raw pixel values, we follow *IIC* and also create segmentations by applying K-Means clustering on image features learned by unsupervised image representation learning approaches. In particular the approaches of Doersch et al. [27] and Isola et al. [49], whereby we use the implementations provided by *IIC* [51]. While none of these approaches was created for semantic segmentation in particular, they still have been created with the purpose of extracting high-level information from images. And

Method	PA	mIoU
K-Means †	14.1%	0.076
Doersch [27] †	23.1%	0.131
Isola [49] †	24.3%	0.139
IIC [50]	27.7%	0.163
Ours	**36**.4%	**0.223**

Table 5.2: Quantitative results for the **COCO-Stuff** dataset. † Methods based on K-Means clustering that have not been specifically designed for *semantic* image segmentation.

it therefore might be feasable to incorporate them for image segmentation.

We provide a qualitative evaluation of the resulting image segmentations using K-Means clustering in 5.2, and present the quantitative results together with our results on *COCO*-Stuff and *COCO*-Persons in Section 5.6.1 respectively Section 5.6.2. In general it can be said that all of the K-Means based approaches we use for comparison fail to create semantic segmentations of comparable quality than our results.

5.6 Results

In the following we will discuss the results of the *COCO*-Stuff and *COCO*-Persons dataset in Section 5.6.1 respectively 5.6.2.

5.6.1 COCO-Stuff

Figure 5.3 shows a qualitative evaluation of our predictions for the *COCO*-Stuff dataset. We trained our model to create a total number of $K = 15$ different segments. In the first four shown examples our predictions are significantly better than those of *IIC* and closely resemble the ground truth. In particular we want to highlight how the image areas depicting the semantic class 'plant' were correctly identified in all four examples. Especially the visual appearance of the bleak trees in the third example differs vastly from the depicted plant areas in the other three images. Detecting that they are semantically similar to the grass in the second image and to the other trees in the first and fourth image is challenging in an unsupervised setting.

The predictions of the examples in the last three rows all have some significant differences compared to the provided ground truth annotations. However, even though they do not match the ground-truth annotations of the *COCO*-Stuff dataset. They still provide predictions that might be considered as semantically correct. In the fifth and sixth example in Figure 5.3 pixels that are labeled with the class 'building' in the *COCO*-Stuff dataset are predicted as 'wall' by our approach. However both 'building' and 'wall' are semantically relatively close, and it might not be reasonable to expect an unsupervised approach to come up with a solution that differentiates between those two concepts. Fur-

Figure 5.3: Qualitative evaluation for the **COCO-Stuff** dataset. Black areas indicate non-stuff pixels that are ignored during evaluation. The first four examples resemble closely the ground-truth. The following three examples contain some errors that we discuss in Section 5.6.1.

Method	PA	mIoU
K-Means †	54.3%	0.374
Doersch [27] †	55.6%	0.381
Isola [49]†	57.5%	0.403
IIC [50]	57.1%	0.394
Ours	**68.3**%	0.521

Table 5.3: Quantitative results for the **COCO-Persons** dataset. † Methods based on K-Means clustering that have not been specifically designed for *semantic* image segmentation.

thermore it should be noted that our approach was still able to correctly identify that the areas labeled as 'building' are both semantically similar, since both were assigned to the same class.

A similar effect can be observed in the example shown in the last row. A large portion of the image is labeled with the class 'food' and predicted with the class 'plant' by our approach. Arguably 'food' and 'plant' are semantically very similar, and it is therefore reasonable to predict areas depicting food as 'plant'. Especially under the consideration that the *COCO*-Stuff dataset contains only a very small amount of image areas depicting food, as shown in Table 5.1.

Table 5.2 shows a quantitative evaluation. With a relative increase in pixelwise-accuracy of about 31%, our approach performs significantly better than the current state-of-the-art method: *IIC*.

5.6.2 COCO-Persons

Figure 5.4 shows a qualitative evaluation for the COCO-Persons dataset. We trained our model to create a total number of $K = 2$ different segments. In the first four rows our predictions closely resemble the ground truth. And we want to emphasize that the shown examples are particularly difficult. Both, the image areas depicting the persons as well as the surrounding areas have vastly different visual appearance. And neither *IIC* nor the K-Means segmentations have been able to create any reasonable segmentations for those examples. We argue that correctly identifying that the four persons shown in the examples depict in fact semantically similar concepts requires a high-level understanding of the scenes. Note also how even the small persons depicted in the background of the first image were correctly assigned to the same segment as the person in the foreground.

In the next two rows large portions of our predictions might still be considered as correct while they do not match the dataset annotation. According to the dataset annotations the depicted motorbikes are not belonging to the same class as the depicted persons. This illustrates a fundamental challenge in unsupervised semantic image segmentation. It is not clear if one of the two solutions is more correct than the other, or if both are equally valid. Determining this would require to answer the question: Are motorbikes semantically more

similar with persons or with streets? Which can not be definitely answered.

In the the last row an example is shown where our approach failed to create a reasonable segmentation. Note however that in this case the segmentation probability also indicates an uncertain prediction.

Table 5.3 shows a quantitative evaluation. While our approach reached a pixel-wise accuracy of 68.3%, all other compared approaches perform significantly worse. Furthermore *IIC* fails to perform significantly better than the K-Means results, contrarily to the results of the *COCO*-Stuff dataset. We attribute this to the fact that the images in the *COCO*-Persons dataset contain more complex scenes than those in the *COCO*-Stuff dataset, and are therefore significantly more difficult to segment.

Figure 5.4: Qualitative evaluation for the **COCO-Persons** dataset. The last column shows the class assignment probabilities according to Equation 4.11, white areas indicate an uncertain prediction and saturated red respective blue colors a more confident prediction. The first four rows show successful predictions, in the next two rows the segmentations are still reasonable but differ from the ground truth, and the last row shows a failure case.

6 Conclusion and Future Work

We have proposed a novel approach on unsupervised semantic image segmentation, i.e. the semantic segmentation of images without any annotated data. Our work is based on recently proposed methods in the field of unsupervised image representation learning [42] and mutual information estimation [8]. And we provided both quantitative and qualitative evaluation of our results. We achieved a relative increase in pixelwise-accuracy of about 31% on the COCO-Stuff dataset, compared to the current state-of-the-art in unsupervised semantic image segmentation: Invariant Information Clustering (IIC) [51]. Furthermore we illustrated and discussed some common challenges at the evaluation of unsupervised semantic segmentation methods. In particular, class imbalances in evaluation datasets and the ambiguity of semantic classes. We also gave a comprehensive review on recently proposed methods for mutual information estimation based on the maximization of variational bounds parametrized by Deep Neural Networks (DNNs) in Chapter 2.

We can think of several promising directions of research for future work. At first one could try to adapt our approach into a semi-supervised setting. And therefore incorporate for training a large dataset of unlabeled images together with a small labeled dataset. Even though this deviates from our entirely unsupervised setting, it still might be an interesting approach. Since for many applications it is still feasible to manually annotate a small amount of images.

It also should be considered that the approaches on variational mutual information estimation using *DNNs* we build up on, i.e. the Mutual Information Neural Estimator (MINE) [8], have only been proposed recently. Even though several works have been published since then that showed their effectivity among various applications [42, 101, 127, 140]. They are still far from being thoroughly studied, and future work might come up with improved methods. Since mutual information estimation is one crucial part of our method, improvements in this direction might help to increase performance.

A further improvement that might benefit the field of unsupervised semantic segmentation in general, would be the introduction of a novel dataset that is specifically designed for unsupervised approaches. We have shown in our experiments that the currently

used datasets, which all have been designed for supervised approaches, come with some problems at evaluation. Especially the quantitative evaluation of end-to-end unsupervised methods is particularly difficult. This is often caused by huge class imbalances or semantic classes in the ground-truth annotations that are semantically very similar, e.g. 'building' and 'wall'. A dataset that lacks these properties would make quantitative comparison of different methods easier.

B Derivations

B.1 Kullback-Leibler Divergence

B.1.1 Reformulation

In the following we show how to derive the reformulation of the Kullback-Leibler (KL) divergence used in Section 2.3.5:

$$\begin{aligned} D_{KL}(P\|Q) &= H(P,Q) - H(P) \\ &= \mathbb{E}_{x\sim P}[-\log Q(x)] - \mathbb{E}_{x\sim P}[-\log P(x)] \\ &= \mathbb{E}_{x\sim P}[-\log Q(x) - (-\log P(x))] \\ &= \mathbb{E}_{x\sim P}[-\log Q(x) + \log P(x)] \\ &= \mathbb{E}_{x\sim P}[\log P(x) - \log Q(x)] \\ &= \mathbb{E}_{x\sim P}\left[\log \frac{P(x)}{Q(x)}\right] \\ &= \sum_i P(i) \log \frac{P(i)}{Q(i)} \end{aligned} \tag{B.1}$$

B.1.2 Nonnegativity

In the following we use the fact that the log is a concave function and Jensen's inequality to show that *KL* divergence is non-negative for any two distributions P and Q. According

to the definition of the *KL* divergence we have:

$$\begin{aligned} -D_{KL}(P\|Q) &= -\sum_i P(i)\log\frac{P(i)}{Q(i)} \\ &= \sum_i P(i)\log\frac{Q(i)}{P(i)}. \end{aligned}$$

And applying Jensen's inequality gives:

$$\begin{aligned} &\leq \log\sum_i P(i)\frac{Q(i)}{P(i)} \\ &= \log\sum_i Q(i) \\ &= 0. \end{aligned} \tag{B.2}$$

Bibliography

[1] http://dronedataset.icg.tugraz.at. Accessed: 2020-02-10. (page 2)

[2] J. Ahn and S. Kwak. Learning pixel-level semantic affinity with image-level supervision for weakly supervised semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4981–4990, 2018. (page 48)

[3] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):898–916, 2010. (page 49)

[4] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 33(5):898–916, May 2011, https://doi.org/10.1109/TPAMI.2010.161. (page 48)

[5] G. Aresta, T. Araújo, S. Kwok, S. S. Chennamsetty, M. Safwan, V. Alex, B. Marami, M. Prastawa, M. Chan, M. Donovan, et al. Bach: Grand challenge on breast cancer histology images. *Medical image analysis*, 56:122–139, 2019. (page 2)

[6] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *stat*, 1050:21, 2016. (page 40)

[7] D. Barber and F. V. Agakov. The im algorithm: a variational approach to information maximization. In *Advances in neural information processing systems*, page None, 2003. (page 20)

[8] M. I. Belghazi, A. Baratin, S. Rajeshwar, S. Ozair, Y. Bengio, D. Hjelm, and A. Courville. Mutual information neural estimation. In *International Conference on Machine Learning*, pages 530–539, 2018. (page 4, 7, 13, 16, 19, 23, 24, 26, 71)

[9] A. S. Berahas, M. Jahani, and M. Takáč. Quasi-newton methods for deep learning: Forget the past, just sample. *arXiv preprint arXiv:1901.09997*, 2019. (page 44)

[10] N. Bjorck, C. P. Gomes, B. Selman, and K. Q. Weinberger. Understanding batch normalization. In *Advances in Neural Information Processing Systems*, pages 7694–7705, 2018. (page 40)

[11] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992. (page 31)

[12] H. Caesar, J. Uijlings, and V. Ferrari. Coco-stuff: Thing and stuff classes in context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1209–1218, 2018. (page 5, 59, 60, 61)

[13] V. Caselles, F. Catté, T. Coll, and F. Dibos. A geometric model for active contours in image processing. *Numerische mathematik*, 66(1):1–31, 1993. (page 49)

[14] A. Cauchy. Methode generale pour la resolution des systemes d'equations simultanees. *Comp. Rend. Sci. Paris*, 25(1847):536–538, 1847. (page 42)

[15] J.-R. Chang and Y.-S. Chen. Pyramid stereo matching network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5410–5418, 2018. (page 29)

[16] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018. (page 29, 47)

[17] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016. (page 20)

[18] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1251–1258, 2017. (page 38)

[19] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun. The loss surfaces of multilayer networks. In *Artificial Intelligence and Statistics*, pages 192–204, 2015. (page 43)

[20] A. Coates, H. Lee, and A. Y. Ng. Stanford stl-10 image dataset. `https://cs.stanford.edu/~acoates/stl10/`. (page 27)

[21] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (5):603–619, 2002. (page 49)

[22] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, B. Schiele, D. A. R&d, and T. U. Darmstadt. The Cityscapes Dataset for Semantic Urban Scene Understanding. Technical report, `www.cityscapes-dataset.net`. (page 2, 3, 59)

[23] S. CS231N. Tiny imagenet, 2017. (page 27)

[24] G. Csurka, D. Larlus, F. Perronnin, and F. Meylan. What is a good evaluation measure for semantic segmentation?. In *BMVC*, volume 27, page 2013. Citeseer, 2013. (page 62)

[25] D. Dai and L. Van Gool. Dark model adaptation: Semantic image segmentation from daytime to nighttime. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*, pages 3819–3824. IEEE, 2018. (page 3)

[26] J. Dai, K. He, and J. Sun. Boxsup: Exploiting bounding boxes to supervise convolutional networks for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1635–1643, 2015. (page 3, 48)

[27] C. Doersch, A. Gupta, and A. A. Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015. (page 64, 65, 66, 68)

[28] M. D. Donsker and S. S. Varadhan. Asymptotic evaluation of certain markov process expectations for large time, i. *Communications on Pure and Applied Mathematics*, 28(1):1–47, 1975. (page 13, 16, 19)

[29] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. (page 42, 46)

[30] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, June 2010. (page 48)

[31] A. Fedorov, R. D. Hjelm, A. Abrol, Z. Fu, Y. Du, S. Plis, and V. D. Calhoun. Prediction of progression to alzheimer's disease with deep infomax. In *2019 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI)*, pages 1–5. IEEE, 2019. (page 7, 23)

[32] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International journal of computer vision*, 59(2):167–181, 2004. (page 49)

[33] P. Fiedor. Networks in financial markets based on the mutual information rate. *Physical Review E*, 89(5):052801, 2014. (page 7)

[34] A. M. Fraser and H. L. Swinney. Independent coordinates for strange attractors from mutual information. *Physical review A*, 33(2):1134, 1986. (page 7, 15, 16)

[35] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez. A review on deep learning techniques applied to semantic segmentation. *arXiv preprint arXiv:1704.06857*, 2017. (page 62)

[36] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015. (page 29)

[37] G. Goh. Why momentum really works. *Distill*, 2017, `http://distill.pub/2017/momentum`. (page 46)

[38] M. Goldblum, J. Geiping, A. Schwarzschild, M. Moeller, and T. Goldstein. Truth or backpropaganda? an empirical investigation of deep learning theory. In *International Conference on Learning Representations*, 2020, `https://openreview.net/forum?id=HyxyIgHFvr`. (page 42)

[39] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. (page 34)

[40] G. E. Hinton, S. Osindero, and Y.-W. Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006. (page 31)

[41] J.-B. Hiriart-Urruty and C. Lemaréchal. *Fundamentals of convex analysis*. Springer Science & Business Media, 2012. (page 21)

[42] R. D. Hjelm, A. Fedorov, S. Lavoie-Marchildon, K. Grewal, P. Bachman, A. Trischler, and Y. Bengio. Learning deep representations by mutual information estimation and maximization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. (page 4, 7, 13, 16, 21, 23, 24, 26, 27, 50, 56, 71)

[43] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. (page 31)

[44] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989. (page 35)

[45] M. M. V. Hulle. Edgeworth approximation of multivariate differential entropy. *Neural computation*, 17(9):1903–1910, 2005. (page 15, 19)

[46] W.-C. Hung, Y.-H. Tsai, Y.-T. Liou, Y.-Y. Lin, and M.-H. Yang. Adversarial learning for semi-supervised semantic segmentation. In *BMVC*, 2018. (page 3, 48)

[47] J.-J. Hwang, S. X. Yu, J. Shi, M. D. Collins, T.-J. Yang, X. Zhang, and L.-C. Chen. Segsort: Segmentation by discriminative sorting of segments. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. (page 48)

[48] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. (page 39, 40)

[49] P. Isola, D. Zoran, D. Krishnan, and E. H. Adelson. Learning visual groups from co-occurrences in space and time. *arXiv preprint arXiv:1511.06811*, 2015. (page 64, 65, 66, 68)

[50] X. Ji, J. F. Henriques, and A. Vedaldi. Invariant Information Clustering for Unsupervised Image Classification and Segmentation. Technical report, `https://arxiv.org/pdf/1807.06653.pdf`. (page 66, 68)

[51] X. Ji, J. F. Henriques, and A. Vedaldi. Invariant information clustering for unsupervised image classification and segmentation. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019. (page 5, 7, 16, 24, 48, 61, 63, 65, 71)

[52] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, Aug 2000. (page 49)

[53] M. I. Jordan. *Learning in graphical models*, volume 89. Springer Science & Business Media, 1998. (page 31)

[54] L.-H. Juang and M.-N. Wu. Mri brain lesion image detection based on color-converted k-means clustering segmentation. *Measurement*, 43(7):941–949, 2010. (page 65)

[55] L. Kaiser, A. N. Gomez, and F. Chollet. Depthwise separable convolutions for neural machine translation. *arXiv preprint arXiv:1706.03059*, 2017. (page 38)

[56] N. Kalchbrenner and P. Blunsom. Recurrent continuous translation models. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, 2013. (page 29)

[57] A. Kanezaki. Unsupervised image segmentation by backpropagation. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1543–1547. IEEE, 2018. (page 48)

[58] A. Kendall, H. Martirosyan, S. Dasgupta, P. Henry, R. Kennedy, A. Bachrach, and A. Bry. End-to-end learning of geometry and context for deep stereo regression. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 66–75, 2017. (page 29)

[59] N. S. Keskar and R. Socher. Improving generalization performance by switching from adam to sgd. *arXiv preprint arXiv:1712.07628*, 2017. (page 47)

[60] A. Khoreva, R. Benenson, J. Hosang, M. Hein, and B. Schiele. Simple does it: Weakly supervised instance and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 876–885, 2017. (page 3, 48)

[61] B. Kim and J. Ye. Mumford-shah loss functional for image segmentation with deep learning. *IEEE transactions on image processing: a publication of the IEEE Signal Processing Society*, 2019. (page 48)

[62] D. Kim, D. Cho, D. Yoo, and I. So Kweon. Two-phase learning for weakly supervised object localization. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3534–3543, 2017. (page 48)

[63] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015, `http://arxiv.org/abs/1412.6980`. (page 42, 46, 47, 63)

[64] P. Knobelreiter, C. Reinbacher, A. Shekhovtsov, and T. Pock. End-to-end training of hybrid cnn-crf models for stereo. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017. (page 29)

[65] A. Krizhevsky, V. Nair, and G. Hinton. Cifar-10 and cifar-100 datasets. *URl: https://www. cs. toronto. edu/kriz/cifar. html*, 6, 2009. (page 27)

[66] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. (page 31, 34)

[67] A. Krogh and J. A. Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, 1992. (page 42)

[68] H. W. Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955. (page 57)

[69] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *nature*, 521(7553):436–444, 2015. (page 42)

[70] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. (page 31)

[71] J. Lee, E. Kim, S. Lee, J. Lee, and S. Yoon. Ficklenet: Weakly and semi-supervised semantic image segmentation using stochastic inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5267–5276, 2019. (page 48)

[72] P. A. Legga, P. L. Rosina, D. Marshalla, and J. E. Morganb. Improving accuracy and efficiency of registration by mutual information using sturges histogram rule. *Medical Image Understanding and Analysis 2007*, page 26, 2007. (page 17)

[73] C. Li, C. Xu, C. Gui, and M. D. Fox. Distance regularized level set evolution and its application to image segmentation. *IEEE transactions on image processing*, 19(12):3243–3254, 2010. (page 49)

[74] D. Lin, J. Dai, J. Jia, K. He, and J. Sun. Scribblesup: Scribble-supervised convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3159–3167, 2016. (page 48)

[75] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014. (page 61)

[76] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014, `http://arxiv.org/abs/1405.0312`. (page 5, 60, 62)

[77] D. Liu, B. Wen, Y. Fan, C. C. Loy, and T. S. Huang. Non-local recurrent network for image restoration. In *Advances in Neural Information Processing Systems*, pages 1673–1682, 2018. (page 29)

[78] H. Liu, J. Lafferty, and L. Wasserman. Sparse nonparametric density estimation in high dimensions using the rodeo. In *Artificial Intelligence and Statistics*, pages 283–290, 2007. (page 18)

[79] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han. On the variance of the adaptive learning rate and beyond. In *International Conference on Learning Representations*, 2020, `https://openreview.net/forum?id=rkgz2aEKDr`. (page 47)

[80] P. Luo, X. Wang, W. Shao, and Z. Peng. Towards understanding regularization in batch normalization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019. (page 40)

[81] J. Ma and D. Yarats. Quasi-hyperbolic momentum and adam for deep learning. In *International Conference on Learning Representations*, 2019, `https://openreview.net/forum?id=S1fUpoR5FQ`. (page 47)

[82] D. Marin, M. Tang, I. B. Ayed, and Y. Boykov. Beyond gradient descent for regularized segmentation losses. In *IEEE conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. (page 48)

[83] J. Martens. Deep learning via hessian-free optimization. In *ICML*, volume 27, pages 735–742, 2010. (page 44)

[84] J. Martens and I. Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040. Citeseer, 2011. (page 44)

[85] D. Martin, C. Fowlkes, D. Tal, J. Malik, et al. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. Iccv Vancouver:, 2001. (page 49)

[86] D. Masters and C. Luschi. Revisiting small batch training for deep neural networks. *CoRR*, abs/1804.07612, 2018, `http://arxiv.org/abs/1804.07612`. (page 44)

[87] D. McAllester and K. Statos. Formal limitations on the measurement of mutual information. *arXiv preprint arXiv:1811.04251*, 2018. (page 19, 22)

[88] W. S. McCulloch and W. Pitts. Neurocomputing: Foundations of research. pages 15–27, 1988, `http://dl.acm.org/citation.cfm?id=65669.104377`. (page 30)

[89] M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry*. MIT Press, Cambridge, MA, USA, 1969. (page 31, 34)

[90] Y.-I. Moon, B. Rajagopalan, and U. Lall. Estimation of mutual information using kernel density estimators. *Physical Review E*, 52(3):2318, 1995. (page 15, 17, 18)

[91] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on pure and applied mathematics*, 42(5):577–685, 1989. (page 48)

[92] T. Nagler and C. Czado. Evading the curse of dimensionality in nonparametric density estimation with simplified vine copulas. *Journal of Multivariate Analysis*, 151:69–89, 2016. (page 18)

[93] H. Ng, S. Ong, K. Foong, P. Goh, and W. Nowinski. Medical image segmentation using k-means clustering and improved watershed algorithm. In *2006 IEEE southwest symposium on image analysis and interpretation*, pages 61–65. IEEE, 2006. (page 65)

[94] X. Nguyen, M. J. Wainwright, and M. I. Jordan. Estimating divergence functionals and the likelihood ratio by convex risk minimization. *IEEE Transactions on Information Theory*, 56(11):5847–5861, 2010. (page 22)

[95] S. Nowozin, B. Cseke, and R. Tomioka. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in neural information processing systems*, pages 271–279, 2016. (page 13, 16, 19, 21, 22)

[96] S. Ozair, C. Lynch, Y. Bengio, A. Van den Oord, S. Levine, and P. Sermanet. Wasserstein dependency measure for representation learning. In *Advances in Neural Information Processing Systems*, pages 15578–15588, 2019. (page 22)

[97] G. Papandreou, L.-C. Chen, K. P. Murphy, and A. L. Yuille. Weakly-and semi-supervised learning of a deep convolutional network for semantic image segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1742–1750, 2015. (page 3, 48)

[98] J. P. Pluim, J. A. Maintz, and M. A. Viergever. Mutual-information-based registration of medical images: a survey. *IEEE transactions on medical imaging*, 22(8):986–1004, 2003. (page 17)

[99] B. Poole, S. Ozair, A. Van Den Oord, A. Alemi, and G. Tucker. On variational bounds of mutual information. In *International Conference on Machine Learning*, pages 5171–5180, 2019. (page 19)

[100] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8), 2019. (page 29)

[101] M. Ravanelli and Y. Bengio. Learning speaker representations with mutual information. *Proc. Interspeech 2019*, pages 1153–1157, 2019. (page 13, 23, 71)

[102] S. J. Reddi, S. Kale, and S. Kumar. On the convergence of adam and beyond. In *International Conference on Learning Representations*, 2018, `https://openreview.net/forum?id=ryQu7f-RZ`. (page 47)

[103] C. Redondo-Cabrera, M. Baptista-Ríos, and R. J. López-Sastre. Learning to exploit the prior network knowledge for weakly supervised semantic segmentation. *IEEE Transactions on Image Processing*, 28(7):3649–3661, 2019. (page 3, 48)

[104] H. Robbins and S. Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951. (page 42, 44)

[105] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. (page 29, 47)

[106] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. (page 30)

[107] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry. How does batch normalization help optimization? In *Advances in Neural Information Processing Systems*, pages 2483–2493, 2018. (page 40)

[108] A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox. On the information bottleneck theory of deep learning. In *International Conference on Learning Representations*, 2018, `https://openreview.net/forum?id=ry_WPG-A-`. (page 7)

[109] B. Schölkopf, C. J. Burges, A. J. Smola, et al. *Advances in kernel methods: support vector learning*. 1999. (page 31)

[110] A. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Zidek, A. Nelson, A. Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 2020. (page 29)

[111] C. E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27(3):379–423, 7 1948, `https://ieeexplore.ieee.org/document/6773024/`. (page 9)

[112] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016. (page 29)

[113] B. W. Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018. (page 17)

[114] B. Singh, M. Najibi, and L. S. Davis. Sniper: Efficient multi-scale training. In *Advances in neural information processing systems*, pages 9310–9320, 2018. (page 29)

[115] C. Song, Y. Huang, W. Ouyang, and L. Wang. Box-driven class-wise region masking and filling rate guided loss for weakly supervised semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3136–3145, 2019. (page 3, 48)

[116] N. Souly, C. Spampinato, and M. Shah. Semi supervised semantic segmentation using generative adversarial network. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5688–5696, 2017. (page 3, 48)

[117] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014, `http://jmlr.org/papers/v15/srivastava14a.html`. (page 42)

[118] M. Sugiyama, T. Suzuki, and T. Kanamori. Density ratio estimation: A comprehensive review (statistical experiment and its related topics). 2010. (page 18)

[119] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International conference on machine learning*, pages 1139–1147, 2013. (page 42, 45)

[120] T. Suzuki, M. Sugiyama, J. Sese, and T. Kanamori. Approximating mutual information by maximum likelihood density ratio estimation. In *New challenges for feature selection in data mining and knowledge discovery*, pages 5–20, 2008. (page 15, 18)

[121] T. Suzuki, M. Sugiyama, J. Sese, and T. Kanamori. A least-squares approach to mutual information estimation with application in variable selection. In *Proceedings of the 3rd workshop on new challenges for feature selection in data mining and knowledge discovery (FSDM 2008), Antwerp, Belgium*, 2008. (page 15, 18)

[122] T. Takikawa, D. Acuna, V. Jampani, and S. Fidler. Gated-scnn: Gated shape cnns for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5229–5238, 2019. (page 29, 47)

[123] M. Tang, A. Djelouah, F. Perazzi, Y. Boykov, and C. Schroers. Normalized cut loss for weakly-supervised cnn segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1818–1827, 2018. (page 48)

[124] M. Tang, F. Perazzi, A. Djelouah, I. Ben Ayed, C. Schroers, and Y. Boykov. On regularized losses for weakly-supervised cnn segmentation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 507–522, 2018. (page 48)

[125] N. Tishby and N. Zaslavsky. Deep learning and the information bottleneck principle. In *2015 IEEE Information Theory Workshop (ITW)*, pages 1–5. IEEE, 2015. (page 7)

[126] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016. (page 40)

[127] P. Velickovic, W. Fedus, W. L. Hamilton, P. Lio, Y. Bengio, and R. D. Hjelm. Deep graph infomax. In *International Conference on Learning Representations*, 2019, `https://openreview.net/forum?id=rklz9iAcKQ`. (page 7, 13, 23, 71)

[128] L. A. Vese and T. F. Chan. A multiphase level set framework for image segmentation using the mumford and shah model. *International journal of computer vision*, 50(3):271–293, 2002. (page 49)

[129] C.-C. Wang, C.-H. Huang, and C.-J. Lin. Subsampled hessian newton methods for supervised learning. *Neural computation*, 27(8):1766–1795, 2015. (page 44)

[130] J. Wang and L. Perez. The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Vis. Recognit*, page 11, 2017. (page 42)

[131] X. Wang, S. Ma, D. Goldfarb, and W. Liu. Stochastic quasi-newton methods for nonconvex stochastic optimization. *SIAM Journal on Optimization*, 27(2):927–956, 2017. (page 44)

[132] Y. Wei, H. Xiao, H. Shi, Z. Jie, J. Feng, and T. S. Huang. Revisiting dilated convolution: A simple approach for weakly-and semi-supervised semantic segmentation. In

Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 7268–7277, 2018. (page 48)

[133] B. WIDROW. Adaptive switching circuits. In *IRE WESCON Convention Record*, volume 4, pages 96–104, 1960. (page 30)

[134] M. M. Wolf, F. Verstraete, M. B. Hastings, and J. I. Cirac. Area laws in quantum systems: mutual information and correlations. *Physical review letters*, 100(7):070502, 2008. (page 7)

[135] Y. Wu and K. He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018. (page 40)

[136] X. Xia and B. Kulis. W-net: A deep model for fully unsupervised image segmentation. *arXiv preprint arXiv:1711.08506*, 2017. (page 48)

[137] S. Xie and Z. Tu. Holistically-nested edge detection. *Int. J. Comput. Vision*, 125(1-3):3–18, December 2017, `https://doi.org/10.1007/s11263-017-1004-z`. (page 48)

[138] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017-Janua:6230–6239, 2017. (page 29)

[139] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017. (page 47)

[140] H. Zhu, A. Zheng, and Huang. High-Resolution Talking Face Generation via Mutual Information Approximation. Technical report, `https://arxiv.org/pdf/1812.06589.pdf`. (page 71)