

# **Learning Deep Visual Features for Minimum Cost Multicut Problem**

**Kalun Ho**

**Dissertation**

zur Erlangung des Doktorgrades  
der Naturwissenschaft

Naturwissenschaftlich-Technischen  
Fakultät der Universität Siegen

March 2023

Supervisor and first appraiser  
Prof. Dr.-Ing. Margret Keuper  
University of Siegen

Second appraiser  
Prof. Dr. rer. nat. Michael Möller  
University of Siegen

I would like to dedicate this thesis to my loving parents



## **Declaration**

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne unzulässige Hilfe von Dritter und ohne Benutzung anderer, nicht angegebener Hilfsmittel angefertigt habe. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepten sind unter Angaben der Quellen gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder ähnlicher Form anderen Prüfungsbehörde vorgelegt. Es wurde keine Dienste eines Promotionsvermittlungsinstituts oder ähnlichen Organisation in Anspruch genommen.

Kalun Ho  
March 2023



## **Declaration**

Zu den vorgeschlagenen Promotionskommission besteht keine verwandschaftlichen Beziehungen, keine Verwandtschaft ersten Grades, Ehe, Lebenspartnerschaft oder eheähnliche Gemeinschaft

Kalun Ho  
March 2023





## Acknowledgements

The decision to pursue research goals has been my personal choice and the successful completion of this thesis was only possible with the support from many people. I would like to therefore thank everyone who had a positive impact on this very personal project.

First and foremost, I want to thank my supervisor Prof. Dr. Margret Keuper for her support and commitment. More importantly, she gave me the chance to pursue my research goals and believed in my work. Her guidance and support kept me motivated to pursue my own interest in research. Margret, thank you especially for your patience. I will be forever grateful for all the countless nights before a conference deadline. I also want to thank Prof. Dr. Janis Keuper for the supervision and valuable discussions and Prof. Dr. Michael Moeller for agreeing to be my co-examiner of this thesis.

I would also like to thank everyone from the HPC department at Fraunhofer ITWM, especially Dominik Strassel and Janis Keuper for providing me all the compute power I needed for my work and keeping the GPU-cluster alive all the time. Furthermore, I want to thank two special persons: Avraam and Ricard. Thank you both for the countless discussions and inspirations. No doubt, one of the best moments at Fraunhofer ITWM were our *scientific discussions* in the basement.

Last but not least, I want to thank my family for all the patience, love and support. My parents never pressured me throughout my childhood and taught me to pursue meaningful work and pursue the things, that makes me happy. This thesis would not have been possible without them.



## Abstract

Image clustering is one of the most important tasks of unsupervised learning in the area of computer vision. Deep learning approaches allow models to be trained on large datasets. In this thesis, image clustering objectives in the context of Triplet Loss induced embedding space are evaluated. Specifically, a simplification of the well-known Triplet Loss is proposed for learning an embedding space from data. This proposed loss function is designed for the *Minimum Cost Multicut Problem*. Furthermore, we highlight one key aspect of the *Minimum Cost Multicut Problem* in terms of scalability and propose a novel approach to overcome this issue. We show empirically, that the proposed algorithm achieves a significant speedup while preserving the clustering accuracy at the same time. The algorithm is able to cluster a dataset with approximately 100,000 images in under one minute using 40 computing threads, where the embedding space is trained with the simplified Triplet Loss. We then apply our proposed loss function on multiple person tracking problems. This problem is treated as a clustering problem on an imbalanced dataset, where each individual, unique person from the scene is considered as one cluster. We compare the tracking performance from two different approaches: the proposed Triplet Loss and an AutoEncoder architecture with reconstruction loss. Experiments show the effectiveness of the clustering task on a tracking application. Finally, we provide an empirical study on embedding space, trained on classification models. Various state-of-the-art models are evaluated against image corruptions. Our key finding suggests to utilize clustering as a predictor for model robustness.

Das Clustering von Bildern ist eine der wichtigsten Aufgaben des unüberwachten Lernens im Bereich der Computer Vision. Deep-Learning-Ansätze ermöglichen das Trainieren von Modellen auf großen Datensätzen. In dieser Arbeit werden die Ziele der Bildclustering im Kontext des Triplet Loss induzierten Einbettungsraums bewertet. Insbesondere wird eine Vereinfachung des bekannten Triplet Loss für das Lernen eines Einbettungsraums aus Daten vorgeschlagen. Diese vorgeschlagene Verlustfunktion ist für das *Minimum Cost Multicut Problem* konzipiert. Darüber hinaus heben wir einen Schlüsselaspekt des *Minimum Cost Multicut Problems* in Bezug auf die Skalierbarkeit hervor und schlagen einen neuen Ansatz vor, um dieses Problem zu überwinden. Wir zeigen empirisch, dass der vorgeschlagene Algorithmus eine signifikante Beschleunigung bei gleichzeitiger Beibehaltung der Clustering-Genauigkeit erreicht. Der Algorithmus ist in der Lage, einen Datensatz mit ca. 100.000 Bildern in weniger als einer Minute zu clustern, wobei 40 Threads zum Einsatz kommen und der Einbettungsraum mit dem vereinfachten Triplet Loss trainiert wird. Anschließend wenden wir die von uns vorgeschlagene Verlustfunktion auf das Problem der Verfolgung mehrerer Personen an. Dieses Problem wird als ein Clustering-Problem auf einem unausgewogenen Datensatz behandelt, wobei jede einzelne, einzigartige Person aus der Szene als ein Cluster betrachtet wird. Wir vergleichen die Verfolgungsleistung von zwei verschiedenen Ansätzen: den vorgeschlagenen Triplet Loss und eine AutoEncoder-Architektur mit Rekonstruktionsverlust. Experimente zeigen die Effektivität der Clustering-Aufgabe in einer Tracking-Anwendung. Schließlich bieten wir eine empirische Studie zum Einbettungsraum, die auf Klassifizierungsmodellen trainiert wurde. Verschiedene Modelle auf dem neuesten Stand der Technik werden anhand von Bildverfälschungen bewertet. Unsere wichtigste Erkenntnis ist, dass Clustering als Prädiktor für die Robustheit des Modells verwendet werden sollte.

# Table of contents

<b>List of figures</b>	<b>xvii</b>
<b>List of tables</b>	<b>xix</b>
<b>Nomenclature</b>	<b>xxi</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Motivation . . . . .	4
1.2 Fundamentals . . . . .	5
1.2.1 Deep Neural Network . . . . .	5
1.2.2 Convolutional Neural Network . . . . .	10
1.3 Theoretical Background . . . . .	14
1.3.1 Similarity Metric . . . . .	14
1.3.2 K-Means Clustering . . . . .	16
1.3.3 Minimum Cost Multicut Problem . . . . .	18
1.3.4 Minimum Cost Lifted Multicut Problem . . . . .	20
1.4 Contribution . . . . .	22
1.5 Publications . . . . .	23
<b>2 Feature Embedding of Image Data</b>	<b>25</b>
2.1 Introduction . . . . .	26
2.2 Related Work . . . . .	29
2.2.1 Clustering . . . . .	29
2.2.2 Correlation Clustering . . . . .	29
2.2.3 Deep Embedding Learning . . . . .	30
2.3 Contribution . . . . .	31
2.3.1 CNN-Architecture . . . . .	31
2.3.2 Loss Function . . . . .	32
2.3.3 Pairwise Cut Probabilities for Multicuts . . . . .	35

2.4	Experiments and Results on CIFAR-10 . . . . .	36
2.4.1	Evaluation of Cluster Accuracy . . . . .	36
2.4.2	Inter- and Intra-cluster distances . . . . .	38
2.4.3	Triplet Loss with Label Noise . . . . .	40
2.4.4	Qualitative Results . . . . .	44
2.5	Conclusion . . . . .	45
2.6	Limitation . . . . .	45
<b>3</b>	<b>Scalable Multicuts</b>	<b>47</b>
3.1	Introduction . . . . .	47
3.2	Related Work . . . . .	49
3.3	MSM: Multi-Stage Multicuts . . . . .	50
3.3.1	Notation and Methods . . . . .	51
3.3.2	Image Clustering with Multicuts . . . . .	52
3.3.3	MSM: Multi-Stage Multicuts . . . . .	54
3.3.4	Theoretical Proof . . . . .	56
3.4	Experiments . . . . .	62
3.4.1	Two-Stage Approach on CIFAR-10 Dataset . . . . .	63
3.4.2	Sparsity is Influencing the Runtime on Merge . . . . .	65
3.4.3	MSM with 3-Stages on CelebA . . . . .	67
3.4.4	Qualitative Results . . . . .	67
3.4.5	MSM with 4-Stages on CIFAR100 . . . . .	69
3.5	Conclusion & Future Work . . . . .	72
<b>4</b>	<b>Application on Multiple Object Tracking</b>	<b>73</b>
4.1	Introduction . . . . .	73
4.2	Motivation . . . . .	74
4.3	Related Work . . . . .	76
4.3.1	Multiple Object Tracking . . . . .	76
4.3.2	Self-supervised learning . . . . .	77
4.4	Method . . . . .	78
4.4.1	Multicut Formulation . . . . .	80
4.4.2	Deep Convolutional AutoEncoder . . . . .	81
4.4.3	AutoEncoder combined with Proposed Triplet Loss . . . . .	83
4.4.4	AutoEncoder-based Similarity Measure . . . . .	85
4.5	Experiments and Results . . . . .	86
4.5.1	Ablation Study . . . . .	88

---

4.5.2	Results . . . . .	90
4.6	Conclusion . . . . .	91
<b>5</b>	<b>Clustering as Robustness Predictor</b>	<b>93</b>
5.1	Introduction . . . . .	93
5.2	Related Work . . . . .	96
5.3	Feature Space Analysis . . . . .	98
5.3.1	Extracting Features from Classification Models . . . . .	99
5.3.2	Latent Space Clustering . . . . .	100
5.3.3	Cluster Quality Measures . . . . .	101
5.3.4	Performance Measure . . . . .	101
5.4	Experiments . . . . .	103
5.4.1	Setup . . . . .	103
5.4.2	Classification vs. Clustering . . . . .	106
5.4.3	Baseline Indicators: Intra- and Inter class-distances . . . . .	112
5.4.4	Robustness Indicators: Clustering Measures . . . . .	112
5.4.5	Adversarial Robustness . . . . .	117
5.5	Conclusion . . . . .	118
<b>6</b>	<b>Conclusion</b>	<b>119</b>
6.1	Summary of Contribution . . . . .	119
6.2	Future Work . . . . .	121
	<b>References</b>	<b>123</b>





# List of figures

1.1	Representation of an image in a computer. . . . .	2
1.2	Four Tasks of Computer Vision. . . . .	3
1.3	Neural Network inspired by Biological Systems. . . . .	6
1.4	Non-Linearity in Data. . . . .	7
1.5	Activation Functions. . . . .	8
1.6	Matrix multiplication for parallelization. . . . .	9
1.7	Convolution Operation. . . . .	10
1.8	Multi-layer Perceptrons vs. Convolutional Neural Network. . . . .	11
1.9	Pooling Layer. . . . .	12
1.10	AlexNet Architecture. . . . .	13
1.11	Distance Metrics. . . . .	15
1.12	Elbow technique for <i>K-Means</i> . . . . .	17
1.13	Visualization of Cycle Constraint. . . . .	19
1.14	Comparison Regular vs. Lifted Multicuts. . . . .	21
2.1	Embedding Space: Assumption vs. Reality. . . . .	27
2.2	Summary of Experiment Setup: Investigation of three Triplet Losses. . . . .	28
2.3	AlexNet Architecture: Proposed vs. Original. . . . .	31
2.4	Visualization of Triplet Loss. . . . .	32
2.5	Decision Boundary in Embedding Space. . . . .	34
2.6	Clustering Performance of three different Triplet Losses. . . . .	37
2.7	Inter- and Intra Cluster Distance trained with different Triplet Losses. . . . .	39
2.8	Label Noise for Triplet Loss. . . . .	40
2.9	Cluster Accuracy vs. Sample Noise. . . . .	43
2.10	TSNE-Visualization CIFAR-10 with a Multicuts. . . . .	44
3.1	Overview of MSM: Multi-Stage Multicuts. . . . .	50
3.2	Multicuts on Single CPU Thread. . . . .	53

3.3	MSM with Two-Stages ( $L = 2$ ). . . . .	55
3.4	Visualization of the function $h(p_{L-1}, L)$ for different stages $L$ . . . . .	62
3.5	MSM on CIFAR-10: Runtime and Cluster Accuracy. . . . .	64
3.6	MSM: Evaluation on CelebA Dataset. . . . .	66
3.7	MSM on CelebA dataset: Discovery of new Cluster. . . . .	68
3.8	More Examples of new Clusters. . . . .	68
3.9	Use ground truth in order to find the optimal $k$ on CelebA dataset. . . . .	69
3.10	MSM on CIFAR100 Dataset with $L = 3$ vs. $L = 4$ . . . . .	70
3.11	MSM on CIFAR100 Dataset: example cluster <i>tree</i> . . . . .	71
3.12	MSM on CIFAR100 Dataset: Cluster Histogram. . . . .	71
4.1	Multiple Object Tracking in Two Steps. . . . .	79
4.2	AutoEncoder Architecture. . . . .	82
4.3	Apply proposed Triplet Loss 2.3 on multiple person tracking. . . . .	83
4.4	Comparison with and without Clustering Loss. . . . .	84
4.5	TSNE Visualization of AutoEncoder Latent Space. . . . .	89
5.1	Clustering as a Predictor. . . . .	94
5.2	Robustness Predictor: Experiment Setup. . . . .	98
5.3	Evaluation Metric for Clusters. . . . .	101
5.4	Find Threshold on Pre-Trained models. . . . .	104
5.5	ImageNet-C Examples. . . . .	105
5.6	<i>K-Means</i> : cluster accuracy on ImageNet-C, grouped by severity levels . . . .	108
5.7	<i>K-Means</i> : cluster purity on ImageNet-C, grouped by severity levels . . . .	109
5.8	<i>Multicuts</i> : cluster accuracy on ImageNet-C, grouped by severity levels . . . .	110
5.9	<i>Multicuts</i> : cluster purity on ImageNet-C, grouped by severity levels . . . .	111
5.10	Robustness Ranking. . . . .	114
5.11	UMap Visualization: CNN vs. Transformer. . . . .	116
5.12	Correlation on Adversarial Robustness. . . . .	117

# List of tables

2.1	Clustering Performance of three Triplet Losses. . . . .	41
3.1	MSM: Datasets used for image clustering tasks. . . . .	63
4.1	Relative Tracking Performance on Transfer Task. . . . .	75
4.2	Multiple Object Tracking: Ablation Study on Performance . . . . .	86
4.3	Multiple Object Tracking Comparison. . . . .	87
5.1	List of Classification Models. . . . .	99
5.2	Evaluation of Robustness. . . . .	107
5.3	Baseline Indicator for Model Robustness. . . . .	112
5.4	Correlation by Severity Levels. . . . .	113
5.5	Rank Correlation by Severity. . . . .	115



# Nomenclature

## Roman Symbols

$CORR$  Set of Image Corruptions

$C_{ACC}$  Clustering Accuracy

$C_{Purity}$  Clustering Purity

$d_{i,j}$  Distance between Vector  $i$  and  $j$

$f$  Non-Linear Function

$L$  Loss Function

$LR$  Learning Rate

$p$  Probability

$P_{ACC*Purity}$  Relative Clustering Performance

$R^2$  Coefficient of Determination

$S_k$  Stage  $k$  of Multi-Stage Multicuts algorithm

$x_i$  Image  $i$

$CL_k$  Cluster  $k$

## Greek Symbols

$\alpha$  Margin Inter Class Distance

$\beta$  Margin Intra Class Distance

$\Delta$  Class Overlap

$\lambda$	Balance Between Reconstruction and Clustering
$\mu$	Centroid
$\mu_{inter}$	Mean Inter-Class Distance
$\mu_{intra}$	Mean Intra-Class Distance
$\phi$	Trainable Parameters
$\sigma$	Standard Deviation
$\sigma_{inter}$	Standard Deviation Inter-Class Distance
$\sigma_{intra}$	Standard Deviation Intra-Class Distance
$\tau$	Threshold
$\theta$	Trainable Parameters
$\tilde{\tau}$	Kendall Rank Coefficient

**Acronyms / Abbreviations**

CNN	Convolutional Neural Network
DNN	Deep Neural Network
MAE	Mean Absolute Error
MLP	Multi-layer perceptron
MOT	Multiple Object Tracking
MSE	Mean Squared Error
ReLU	Rectified Linear Unit
SOTA	State-of-the-Art

# Chapter 1

## Introduction

Computer vision allows the computer to understand visual representation of the real-world. While we humans have an incredible skill to detect and recognize known objects, a computer uses a set of algorithms in an attempt to solve the similar task. Our retina works similar to an active pixel sensor: it receives light signals and processes it. However, seeing an object through stimuli on our retina is one thing. Understanding the scenery is yet another challenge, particularly for a computer. The main difference between computers and humans in terms of image processing layers is our brain: while scientists have some hints regarding the functionality of the different areas of our brain, the exact way how it works still remains unanswered. On the other hand, computers formulate vision tasks as mathematical functions in order to represent real world problems and solving these equations enables the computer to *perceive* the real world through sensors. The objective of computer scientists is to develop a set of algorithms in order to solve these vision tasks. Image data are considered as unstructured data, where each pixel are represented as numerical values. When pixels are aligned in a certain order, it becomes visual features. Humans can recognize such features effortlessly while computers have very hard time to do the same task. Figure 1.1 illustrates the challenge for computers vision to detect known objects. In a), the array of numbers represents the pixel values, which represents a *heart* shape (b). The size of the image is 7x7 pixels in width and height and the *heart* covers the whole image. A high resolution image is displayed in c). Consider the following visual task: detection of the *heart*-shape as shown in c) on the high resolution image c). Humans can immediately recognize the visual feature (e.g. edges around *heart*-shape or color) while computer scientists have to design a way to find the target object. A very primitive way would be to scan through the whole image to find the pattern by matching the arrays. Despite the simplicity of such algorithm, it is inefficient and most likely ineffective due to transformations such as rotation (as shown in c) red).

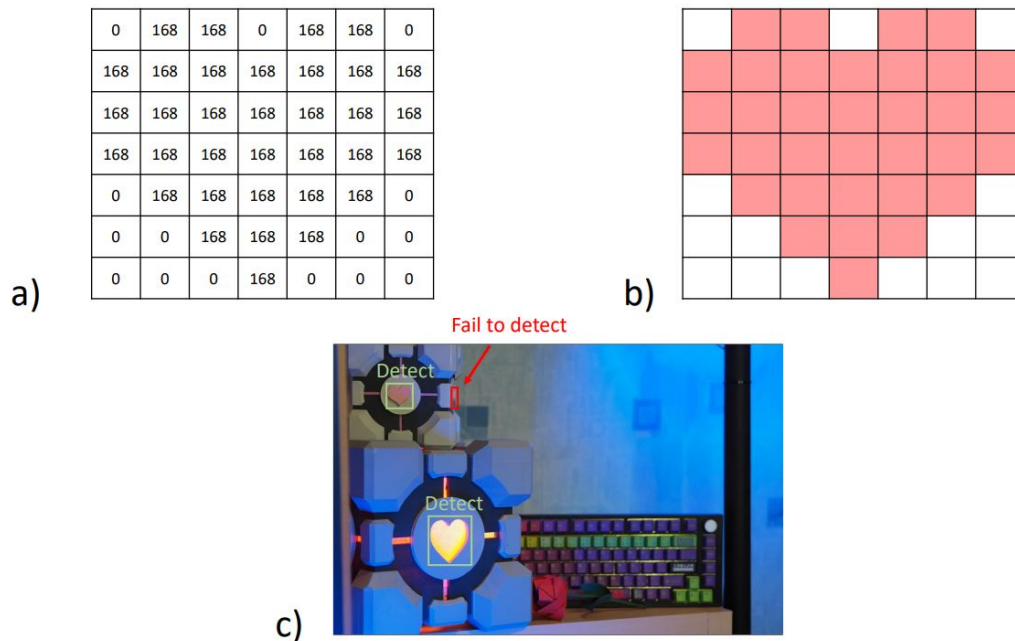


Fig. 1.1 Representation of an image in a computer.

An image is represented as an array of numbers (a). Each value correspond to a pixel, which forms the content (b) through the correct alignment. Humans have incredible capability to detect and recognize known objects on images (c) despite the rotation or translation of the target object. A simple pixel-by-pixel matching approach may fail to detect the *heart*-shapes in c).

With the introduction of convolutional neural network (CNN) [100] in the late 20th century, many solutions for computer vision tasks today are fundamentally based on the same principle: the visual features are extracted by applying a large number of filters on images. Such process is called convolution and the learned visual features can be invariant to any sort of rotation or translation of objects on an image. The main advantage of such algorithms are the fact that they follow a *black-box* strategy, where the applied filters are learned automatically based on the specific training dataset and target function (details are provided in the in section 1.2.1). Another big advantage is the fact that the process of convolutions can be highly parallelized on a modern graphical processing unit (GPU), which makes the learning and inference highly efficient. As the hardware improves over the last few years, algorithms are proposed by the vision community and their claimed results consistently beat the state-of-the-art (SOTA) performance on visual tasks <sup>1</sup>.

<sup>1</sup><https://paperswithcode.com/sota/image-classification-on-imagenet>



The key challenge is to define the clear (mathematical) objective for the vision task and gather the required data for the given problem. Often, data for approximating real-world problems through such mathematical functions are insufficient, which leads to poor ability to generalize. For instance the company Tesla<sup>2</sup> utilizes real time data from more than one million vehicles to develop a full self-driving algorithm based on computer vision only. Yet, full self-driving task still remains subject to future research.

In general, computer vision task can be categorized in four major fields: (1) image classification, (2) object detection, (3) image segmentation and (4) image generation as shown in Figure 1.2. These problems are mostly solved by supervised training and has been studied over last few decades. Various benchmark datasets [34, 25, 134, 99] have been proposed for specific tasks in order to evaluate the performance of the models. Recently, additional benchmark datasets [52, 54] have been proposed to specific target robustness of image classification models. Despite the recent success of transformer models [123, 32] on computer vision tasks, *CNNs* are still more suitable for real-world applications due to its efficiency in learning.

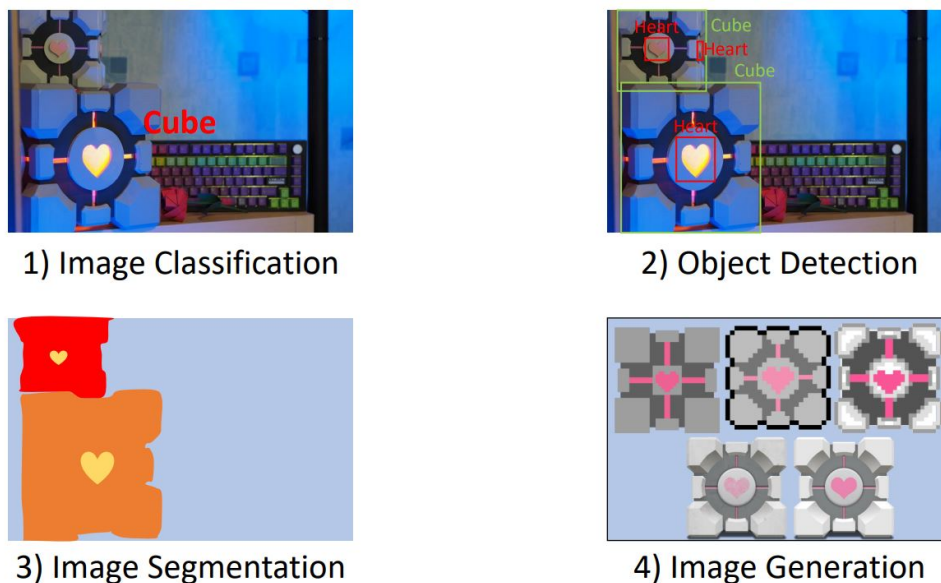


Fig. 1.2 Four Tasks of Computer Vision.

<sup>2</sup><https://www.tesla.com/AI>

## 1.1 Motivation

The recent success of deep learning methods (e.g. based on CNNs) are mostly driven by supervised learning methods: large number of label data are utilized in order to train the models for the specific tasks such as image classification or segmentation. However, benchmark datasets such as ImageNet [134] or MOT16 [115] often do not reflect the actual claimed results. This mismatch is caused by a mix of hypothesis bias and overfit in test dataset. For instance authors in [130] showed a significant generalization gap in classification performance when re-evaluation existing models on a new dataset. Apart from the benchmark datasets, real-world tasks often require additional new sets of data with labels for the specific target scenario in order to perform well for the specific task. In practices, one would first utilize a pre-trained model based on a large dataset, such as ImageNet, and then fine-tune the task to the specific scenario with a smaller dataset. Nevertheless, depending on the complexity of the task, the need for labeled data is inevitable and collecting such data can be time-consuming.

In contrast, unsupervised methods do not require label data. Such approach automatically finds pattern or visual features that are important for further processing, such as classification. One of the most important task of unsupervised learning is clustering of data: without any given label data, it finds structure or natural groups in the given dataset. For instance, *K-Means* iteratively clusters data based on some distance measure, which can be based on some features. It assumes data points to be around its center. Many deep learning approaches have been proposed in the past [164, 41, 118, 47] that is based on CNNs. Often, traditional clustering methods such as *K-Means* are incorporated [35, 14] as well.

On the other hand, the graph-based approach *Minimum Cost Multicut Problem* [85, 149, 78] has also shown promising results for clustering problems. In this particular approach, data are treated as nodes in a graph and distance measures between two nodes are required. Such metric can be for instance the Euclidean distance. The key challenge here is to find the right distance or similarity measure between two data points. Weak supervision can then be utilized in this approach, which is a good compromise between fully supervised and unsupervised learning, especially when there are very few labels available. A classic approach of such weak-supervised learning is the *Triplet Loss* function, where label information about the data are only available on a higher level (such as triplets). It is also called *Metric Learning*: an embedding space (or feature vector) is learned as opposed to [164, 14], where a specific size of output is defined and a softmax layer is required to categorize the output. Such embedding space can either be trained from scratch or fine-tuned via pre-trained models. Nevertheless, the distance metric is constructed task-specific and can be used for various tasks such clustering or nearest-neighbour classification [136].

## 1.2 Fundamentals

This section provides an introduction to deep neural network as well as an overview of the deep learning and how a model is trained by optimizing mathematical functions.

### 1.2.1 Deep Neural Network

Deep Neural Network (DNN) describes a set of algorithms that utilizes large amount of unstructured data in order to learn a target function by optimizing a loss function. Unlike traditional machine learning methods such as *linear regression* or *K-Means*, it often does not require sophisticated pre-processing of data. This is especially interesting in the area of computer vision, where input data are directly treated as input signals for the model. The structure of neural networks is inspired by our biological system, where nodes are interconnected and electronic signals are sent between nodes, which makes large and distributed communication possible. From the mathematical point of view, the input signals are mapped to another signal (e.g. label data) via a non-linear function  $f$ . Figure 1.3 depicts a biological neuron and mathematical model with  $f$ . The biological neuron (a) consists of dendrites, cell body and axon. Similarly, a neuron from a deep network (b) also consists of input  $x_n$ , the weight  $w_n$  and output  $y$ . Information is *feedforwarded* via the dot product:

$$y = \sum_i^n x_i \cdot w_i \quad (1.1)$$

In practice, an additional real value  $b$  is added to  $y$  before it is passed to a non-linear function  $f$ :

$$y = f\left(\sum_i^n x_i \cdot w_i + b\right) \quad (1.2)$$

The additional parameter  $b$  is called *bias* and  $f$  the *activation* function, respectively. By passing the weighed sum  $y$  into a non-linear activation function  $f$ , the model is able to map data into a complex, non-linear separable space through training. Figure 1.4 shows a dataset, that is non-linear separable. The color circles are data from different classes in a two-dimensional space. A logistic regression model fails to separate the data (a), while a neural network successfully perform the separation of the data (b).

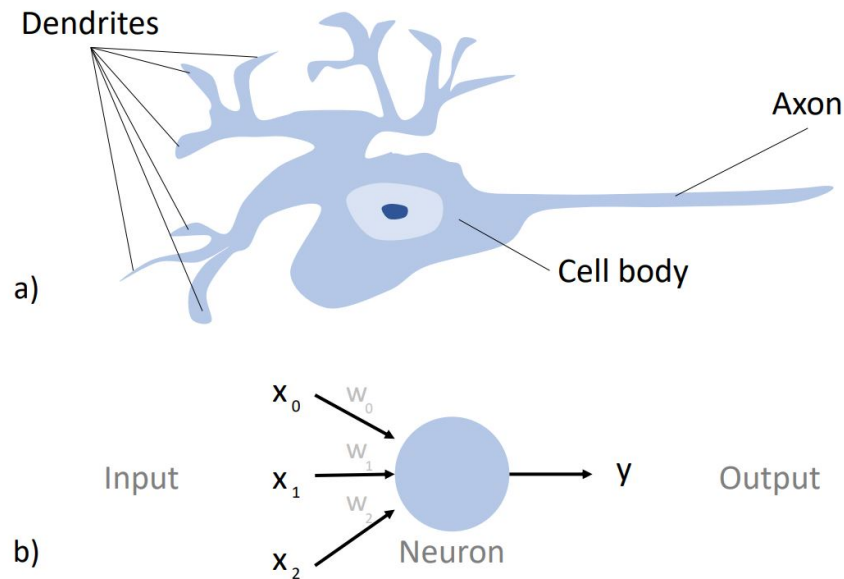


Fig. 1.3 Neural Network inspired by Biological Systems.

Deep Neural Network (b) is inspired by our biological system (a). Dendrites receive message and information are processed in the cell body. Electric signals are then transmitted via the axons.

**Activation Functions** Three most popular non-linear activation functions are *Sigmoid*-, *Tanh*- or *Rectified Linear Unit*-function (*ReLU*). While the activation function converts the weighed sum of the input (Equation 1.2), the choice of function determines the range of the output values:

- *Sigmoid*-function:  $y \in [0, 1]$
- *Tanh*-function:  $y \in [-1, 1]$
- *ReLU*-function:  $y \in [0, \infty]$

Consequently, the choice of activation functions also affects the trainability (e.g. time, until it reaches convergence) of the neural network: in order to actually learn from data, the weights  $w_n$  inside a neural network have to be adjusted iteratively in a way, that the model outputs the desired value given the input data. Figure 1.5 illustrates the three activation functions.

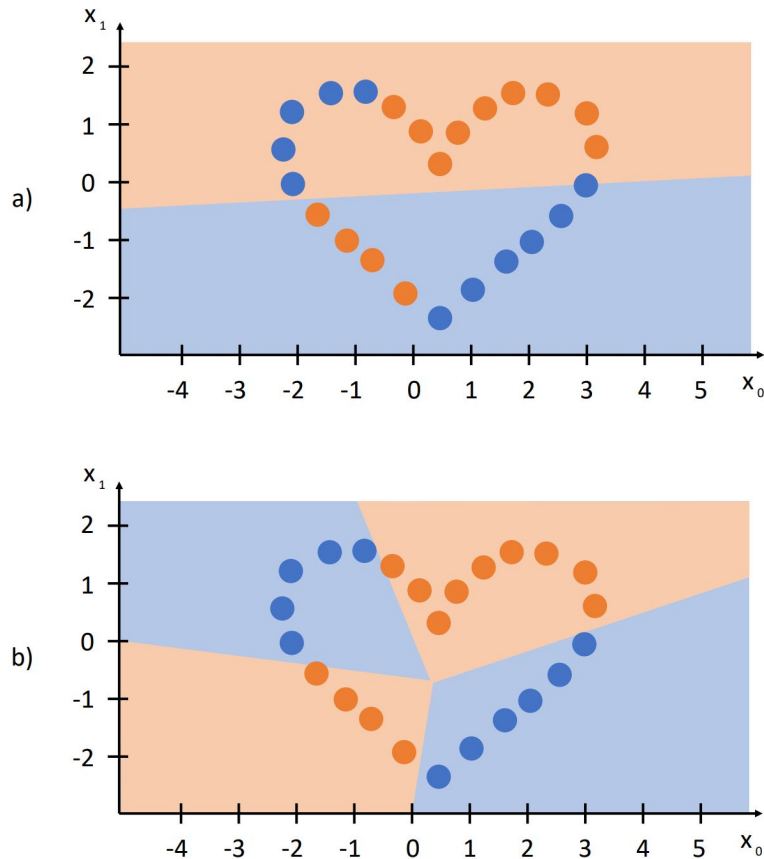


Fig. 1.4 Non-Linearity in Data.

An attempt to separate data of a non-linear separable dataset with two dimensions ( $x_0$  and  $x_1$ ) via logistic regression (a) and neural network (b).

**Loss Functions** In order to be able to adjust the weights, data are first feedforwarded across multiple layers. At the last layer, the expected value is compared with the actual output of the network and the degree of *correctness* is defined by the *loss-function*. For instance, if the loss value of the neural network outputs zero, it is performing the task perfectly based on the given data. In practice, a perfectly trained model (e.g.  $loss = 0$ ) is never achieved and the learning is often stopped at a certain iteration or if the loss value converges to a value (*minima*). Choosing the right loss function is crucial and is often depending on the specific objective. For instance, *Cross Entropy Loss* is most often chosen in a classification task, where the number of predicted class is known. Regression tasks often utilizes *Mean Squared Error Loss* (MSE) or *Mean Absolute Error Loss* (MAE). On the other hand, metric learning (details in Chapter 2.1) optimizes the distance between data points in a high dimensional space (called *embedding space* or *latent space*): instead of comparing the output label, as

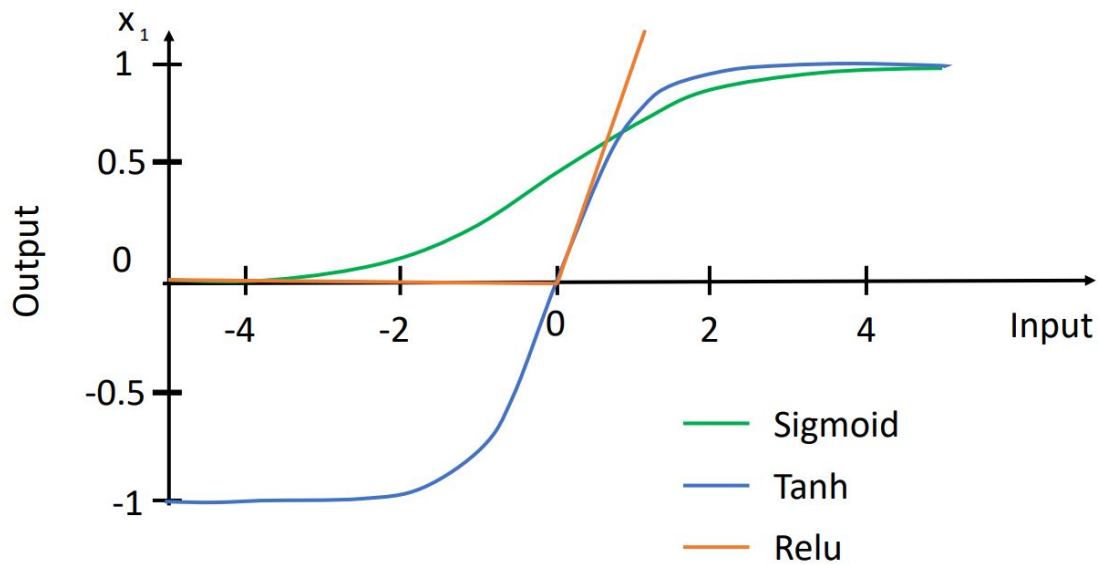


Fig. 1.5 Activation Functions.

Three most popular activation functions used in deep neural network, enabling non-linearity.

done for instance in *Cross Entropy Loss*, multi-dimensional vectors are being compared. Popular loss functions are *Contrastive Loss* or *Triplet Loss*.

**Backpropagation** Data inside the hidden layers have no chance to know, if the current representation is fitting the data or not until it reaches the last output layer. Each hidden layer therefore rely on the previous layer. Based on the loss function as described previously, the amount of adjustment is done successively layer by layer. The derivative of the activation value  $y$  points the direction of the adjustments of  $w_n$  during backpropagation. This explains, why activation functions are typically differentiable. Finding a good minima in context of deep neural network has been subject to ongoing research [103, 81, 82].

**Hardware acceleration** Figure 1.6 shows an example of a neural network with one hidden layer. All layers and weights are represented as matrices. When feedforwarding input data  $x_i$ , the matrices are multiplied by taking the dot product (shown in red). Therefore, each layer at each point in time only depends on its adjacent matrices (e.g. weights  $w_i$ ) and thus the dot product for each layer can be computed parallel on different compute threads. Modern graphical processing units (GPUs) have significantly more threads as opposed to the

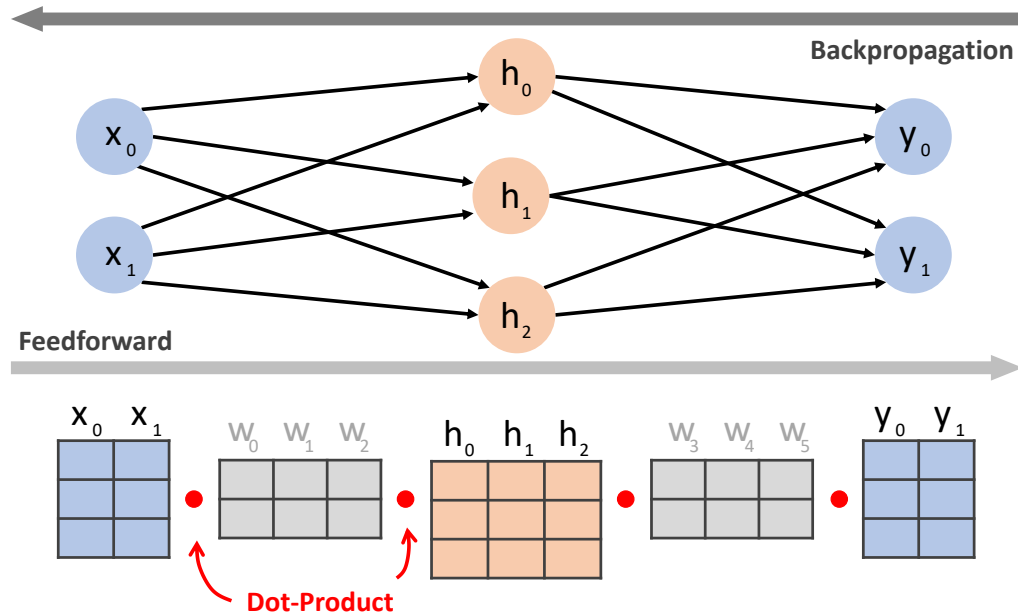


Fig. 1.6 Matrix multiplication for parallelization.

Matrix multiplication allows fast and parallel processing due to the large number of compute threads.

central processing unit (CPU), which consequently accelerates the training and inference of neural network models. In practice, GPUs process data several orders of magnitude faster than CPUs due to highly-parallel computation. Modern deep learning frameworks such as Tensorflow<sup>3</sup> or Pytorch<sup>4</sup> simplify the process of development of models by integrating the interface of the hardware (e.g. CUDA) into their framework. Thus, the computationally expensive operations are not only accelerated by modern GPUs, open-source frameworks also allow scientists to easily build, train and deploy their models.

<sup>3</sup><https://www.tensorflow.org/>

<sup>4</sup><https://pytorch.org/>

## 1.2.2 Convolutional Neural Network

Deep neural network with simple matrices as shown in Figure 1.6 are also called Multi-layer perceptrons (MLP). In computer vision, the input signals are images, which consists of pixels, for example as shown in Figure 1.1. Yet, representing each pixel value as input on a MLP is infeasible due to the large number of weights. The number of parameters grows significantly since each node within the hidden layers are interconnected with another, which are also called *fully connected layers*. The large amount of parameters not only makes the training inefficient, but it is also impossible to fit a large MLP into a modern GPU. Furthermore, arguably the biggest drawback using fully connected layers only for image processing is the inability to preserve spatial information, such as rotation or translation.

Convolutional Neural Network (CNN) solves exactly these two problems: 1) reducing unnecessary training parameters and 2) preserving spatial information. Input data in vision tasks are 2-dimensional. Hence, features, that are stored as simple numerical values have in fact dependencies. Unlike MLP, where the pixel values are individually mapped to the adjacent layers, a CNN uses the convolution operation with *kernels*. This is shown in Figure 1.7. Hence, the mapping of pixels are done by considering the surrounding pixels. Furthermore, the convolution operation is repeated for all pixels using the same kernel.

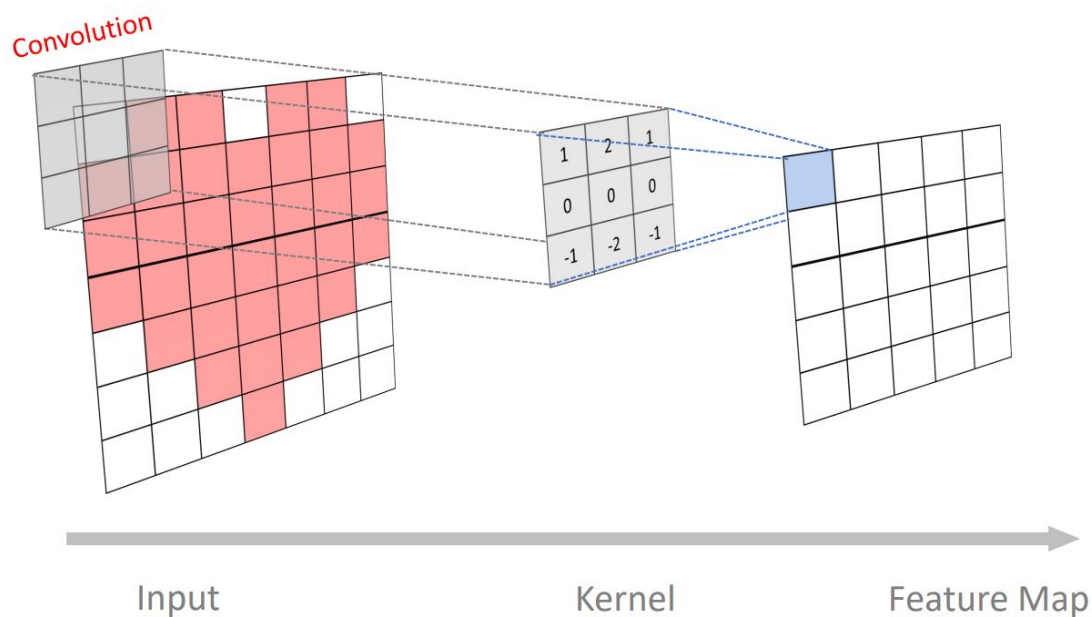


Fig. 1.7 Convolution Operation.

Apply convolution operation on 2-dimensional input data using a 3x3 kernel in order to obtain the feature map.



This way, the spatial dependency is preserved and features that are detected by the kernel at a specific location within the image can be detected on another location as well since the weights (e.g. kernel) are shared. The size of the kernel as well as the padding or stride are parameters, that are defined by the architecture of the model. By stacking multiple kernels with different settings on different color channels of an image, the convolutional neural network is able to learn efficiently and effectively visual features. Convolutional layers, that are close to input layers learn low level features such lines or edges. Increasing the depth allows the model to learn high level representation of objects such as round shapes or eyes of humans. The total number of parameters ultimately depends on the kernel size and the number of different kernels, which allows flexibility in designing the model's architecture. In contrast, the size of a fully connected layer is always fixed to the number of inputs as well as the hidden layers' size. Due to the lack of spatial preservation, different weights are often used for representing the same image feature, which leads to inefficiency. This is shown in Figure 1.8: the object *heart* appears on different locations in the image. While the CNN uses the same filter on both images, the MLP activates different units inside the hidden layers due to change in input values (marked as red).

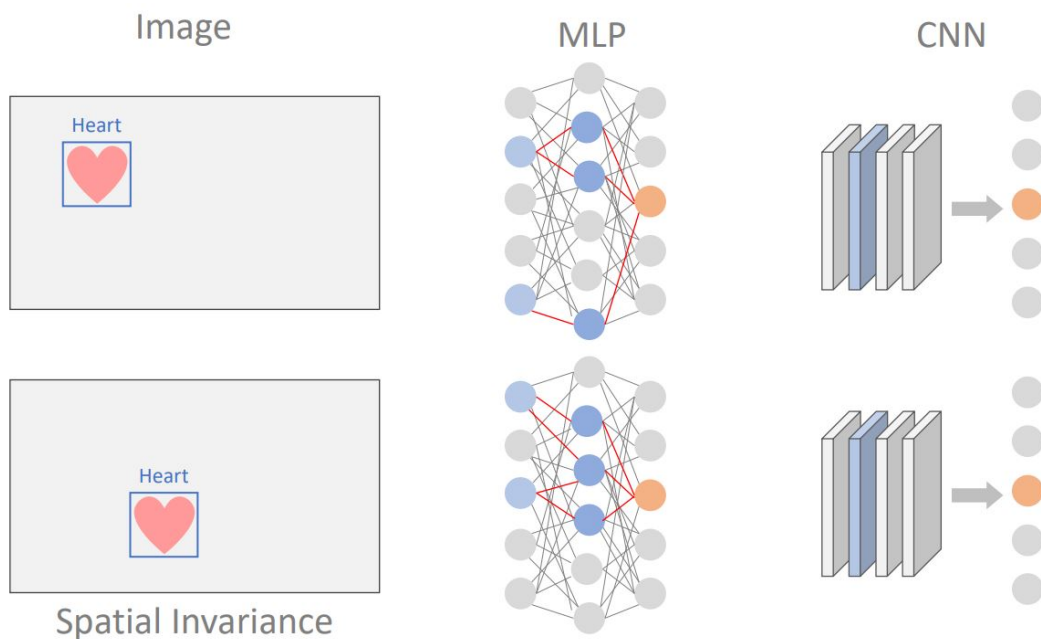


Fig. 1.8 Multi-layer Perceptrons vs. Convolutional Neural Network.

Convolutional Neural Network (CNN) preserves spatial information due to weight sharing. Multi-layer perceptron on the other hand uses same different activations for the same feature, which is inefficient.

**Pooling** Non-linearity is achieved by using non-linear activation. On one hand, the model is able to map various input data into the function, on the other hand, they are differentiable. Another important aspect of convolution is the pooling operation. It down samples the data and provide yet another form of non-linearity at the same time. By down sampling the feature map within the convolutional neural network, the number of spatial size is decreased. A low resolution version of the feature map still contains important information. Pooling increases the location of important information and discard those, that are not relevant. In fact, pooling allows the learned feature to be invariant to translations. Furthermore, it also leads to reduction of parameters, which increases the efficiency of the learning process. Figure 1.9 shows two popular pooling methods used in deep convolutional neural networks: 1) *Max-pooling* and *Average-pooling*. The pool size is  $2 \times 2$ , which the output size is reduced by a factor of 2 in width and height. The different colors illustrate the area, which will be aggregated. While *Max-pooling* chooses the largest value of the colored area, *Average-pooling* in contrast computes the mean. In practice, pooling operation is performed directly on the output feature map of each convolution layer. Similar to the convolution operation, the size of the pooling is a parameter, that is defined by the architecture of the model. After pooling, the activation is performed, for instance using *ReLU*.

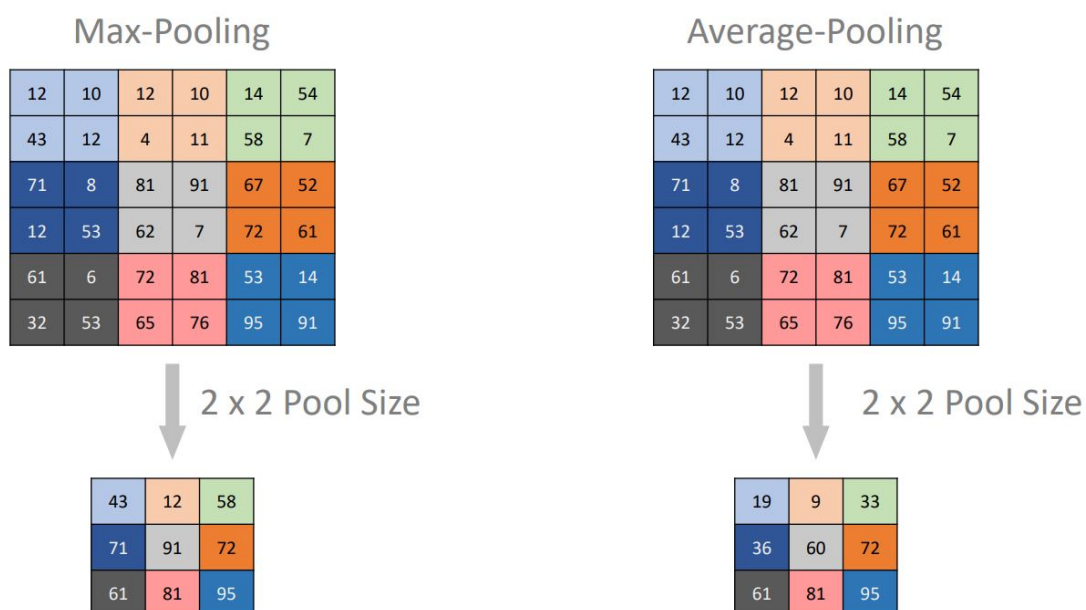


Fig. 1.9 Pooling Layer.

Two popular pooling operations: Max- and average Pooling. The example shows the operation on a  $6 \times 6$  input image with a pooling size of  $2 \times 2$ . The output dimension is halved.

**CNN-Architecture** Most CNN models from the literature [94, 174, 70, 51, 145, 172, 73, 153] follows the same structure: several convolutional- and pooling layers with a fixed number of kernels are stacked together in series. The size of the feature map is successively reduced, allowing the model learn low-level features at the beginning of the model first. As the depth of the model increases, high level features are learned from the input data. The last layer contains a fully connected layer and the features are represented as a vector. Depending on the vision task, another fully connected layer is added, for instance in *classification*. In contrast, metric learning for instance directly uses the feature vector to create a high dimensional feature space (Chapter 2). Figure 1.10 shows the AlexNet [92] architecture, which contains convolutional layers, two full connected layer and a classification head. The model was introduced to solve the ImageNet [134] classification task, which contains 1000 unique classes. The input layer takes an image with a resolution of 224x224 with three channels for colors. It has in total five convolutional layers and two fully connected layers as well as the classification head, which outputs the prediction of the class. CNN has also successfully been applied to many other vision tasks, e.g. image generation [43, 22], clustering [15], or segmentation [133]. Recently, the focus has been shifted to the search for the best architectures based on the specific vision task. Different methods has been proposed [107, 80, 147]. For instance [174] was discovered through neural architecture search [33].

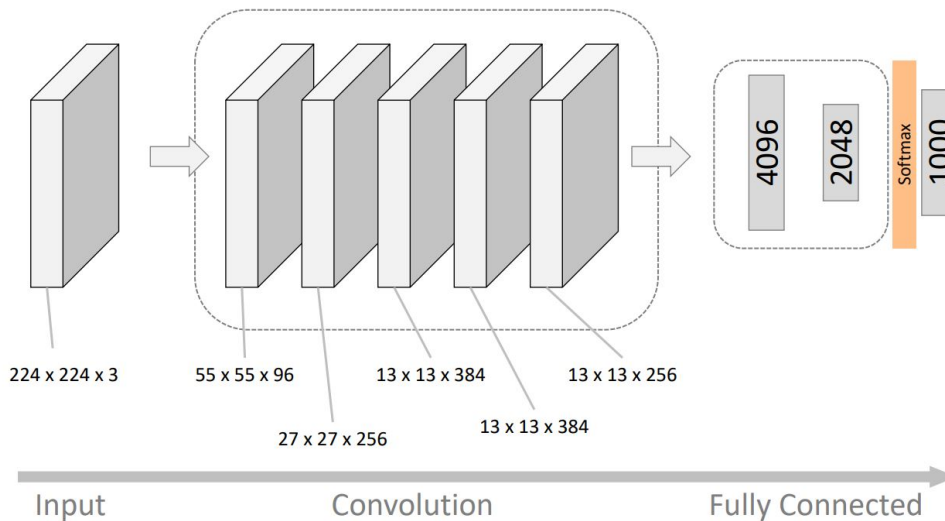


Fig. 1.10 AlexNet Architecture.

AlexNet [92] architecture for solving image classification task on ImageNet [134]. It consists of five convolutional layers and two fully connected layers. The dimension is successively reduced while the number of kernels are increased.

## 1.3 Theoretical Background

This section presents some theoretical background of several algorithms, that are related to this thesis. Specifically, an overview of different clustering techniques is presented. These approaches work well on traditional, structured data as well as on deep visual features.

### 1.3.1 Similarity Metric

Deep visual features that are learned from data are often represented as a multi-dimensional vector. Extracting these features requires the images to be passed through the trained model in order to obtain the vectors. Given an input image  $x_i$  and a trained model with parameters  $\theta$ , the deep visual feature  $z_i$  of  $x_i$  is obtained by passing  $x_i$  through the function  $f_\theta : X \rightarrow Z$ , with  $f_\theta(x_i) \in \mathbb{R}^d$  and  $d$  represents the dimension of the feature space. Standard clustering methods from statistics and data mining can then be utilized to generate clusters. Methods such as *Nearest Neighbour* classifier or *K-Means* rely on proximity of the data in the high dimensional (latent) space. Most distance metrics compare pairs of variables (e.g. vectors). Thus the distance between two points in the deep feature space represent the similarity or dissimilarity of the images. There are various metrics to measure similarities for clustering methods.

Euclidean Distance:

$$\|f_\theta(x_i) - f_\theta(x_j)\|_2 = \sqrt{\sum_{k=1}^d ([f_\theta(x_i)]_k - [f_\theta(x_j)]_k)^2}, \quad (1.3)$$

Squared Distance:

$$\|f_\theta(x_i) - f_\theta(x_j)\|_2^2 = \sum_{k=1}^d ([f_\theta(x_i)]_k - [f_\theta(x_j)]_k)^2, \quad (1.4)$$

Manhattan Distance:

$$\|f_\theta(x_i) - f_\theta(x_j)\|_1 = \sum_{k=1}^d |[f_\theta(x_i)]_k - [f_\theta(x_j)]_k|, \quad (1.5)$$

where  $[f_\theta(x_i)]_k$  is the  $k$ -th element of  $d$ -dimensional point  $f_\theta(x_i)$ .

Choosing the right similarity will affect the clustering result. For instance in metric learning, the distance is used directly in the optimization function, e.g. with *Triplet Loss* (Chapter 2). Figure 1.11 illustrates the difference between Euclidean and Manhattan Distance

in the two dimensional feature space. The distance between the coordinate (0, 0) and (2, 2) is greater when measuring with Manhattan distance.

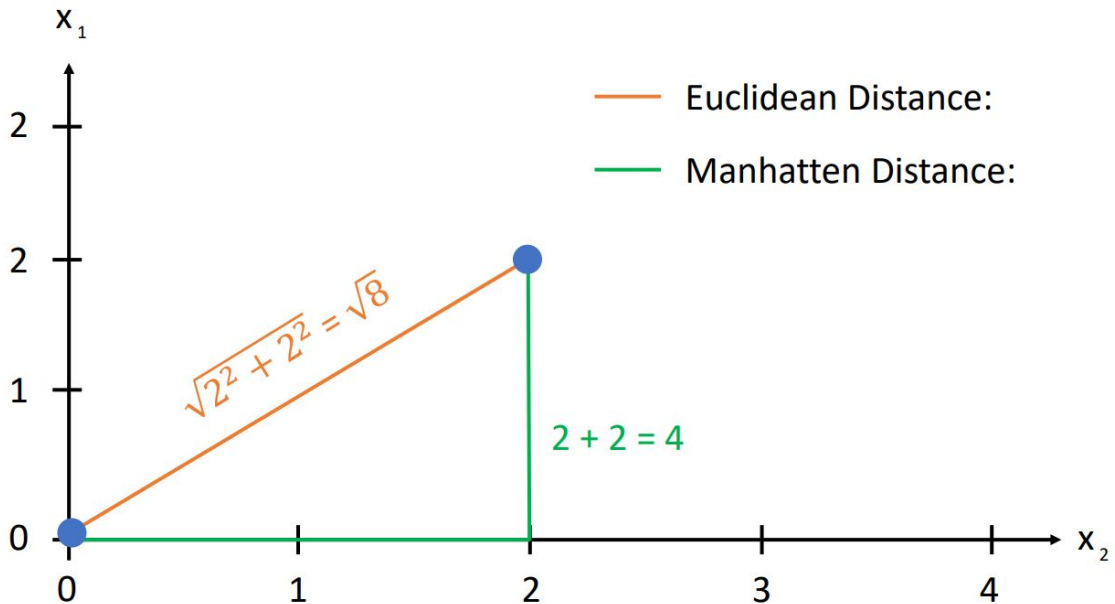


Fig. 1.11 Distance Metrics.

Choosing the right distance metric is essential as it affects the shape of clusters in the feature space. The example shows the difference between Euclidean Distance and Manhattan Distance between point (0, 0) and (2, 2) in a two dimensional space. The latter has a larger distance.

The usage of distance metric is essential in clustering. Data, that are closed to each other are more similar than when they are far apart. If the distance is zero, then the two data is considered as equivalent. In computer vision, deep visual features, that are extracted from some convolutional neural network often have very high dimension with size  $d$ . Furthermore, the features are represented as numerical floating point due to the non-linear activation functions, as shown previously. The authors in [1] show that Manhattan distance (L1 norm) is preferred over Euclidean distance (L2 norm) for distance based algorithms (such as *K-Means*) for data with very high dimensions. This is due to the *curse of dimensionality*. Thus they in fact propose a  $L_k$  norm with  $k < 1$ . However, this for utilizing fractional norm is later falsified by the authors in [116]. Euclidean distance still remains a popular choice for similarity measure when comparing deep visual features, for instance it is used in *Triplet Loss* for training facial recognition systems [136] or *Reconstruction Loss* in *AutoEncoders* [156].

### 1.3.2 K-Means Clustering

*K-Means* Clustering aims to partition a given dataset (e.g. images) into  $K$  clusters. This parameter  $K$  is essential and has to be specified in advance. In other words, the assumed number of clusters is known before the actual clustering. *K-Means* requires no label and thus belongs to the family of unsupervised learning methods. The goal of this clustering algorithm is to minimize the intra-cluster distance by assigning each data point iteratively to the nearest centroid based on a distance metric, e.g. such as Euclidean Distance (1.3). Given an image dataset  $X$  with a total size of  $N$ , each data point (image)  $x_i \in X$ ,  $i \in 1, \dots, N$ , is mapped with the function  $f$ , with  $f(x_i)$  being a point with a dimension size of  $d$ . The point is obtained via the function  $f$  from a trained convolutional neural network model. *K-Means* iteratively assigns each point  $f(x_i)$  to one specific Cluster  $c_k$ . This cluster is selected based on the distance metric between  $f(x_i)$  and the cluster centroid  $\mu_k$ , which is the mean over all assigned  $f(x_i)$ :

$$\mu_k = \frac{1}{|c_k|} \sum_{x_i \in c_k} f(x_i) \quad (1.6)$$

Centroid  $\mu_k$  is re-calculated once all  $x_i$  is assigned to a cluster  $c_k$ . Although it is possible that the algorithm converges at a very early stage (e.g. no changes in cluster assignment), it is a good practice to set a maximum number of iterations  $t$ , which limits the cluster assignment process. Thus the complexity of *K-Means* is  $\mathcal{O}(t * N * K * d)$ . The objective is defined as:

$$\sum_{k=1}^K \sum_{x_i \in c_k} \|f(x_i) - \mu_k\|^2 \quad (1.7)$$

---

#### Algorithm 1 K-Means Clustering

---

- 1: **input:** number of  $K$  clusters, images  $X$ , mapping function  $f$
  - 2: randomly initialize  $K$  centroids  $\mu_k$
  - 3: **while** centroid  $\mu_k$  has changed for any  $k \in 1, \dots, K$  **do**
  - 4:   assign all mapped images  $f(x_i)$  to nearest  $\mu_k$
  - 5:   recompute all  $\mu_k$
  - 6: **end while**
- 

Despite its efficiency, the main drawback of *K-Means* is the input parameter  $K$ : the number of expected clusters is often known, especially in real-world problems, where label data is not available or hard to obtain. Choosing an inappropriate  $K$  results in poor performance in terms of cluster accuracy. Figure 1.12 illustrates the *Elbow method* for finding the optimal input

parameter  $K$ . The x- and y-axis represents the number of clusters  $K$  and the sum of squared distance between each point to its cluster centroid, respectively. Increasing the number of clusters will reduce the sum of squared distances of all clusters. If there exists an optimal  $K$  from the dataset, this is often shown as a *bend* as illustrated in Figure 1.12.

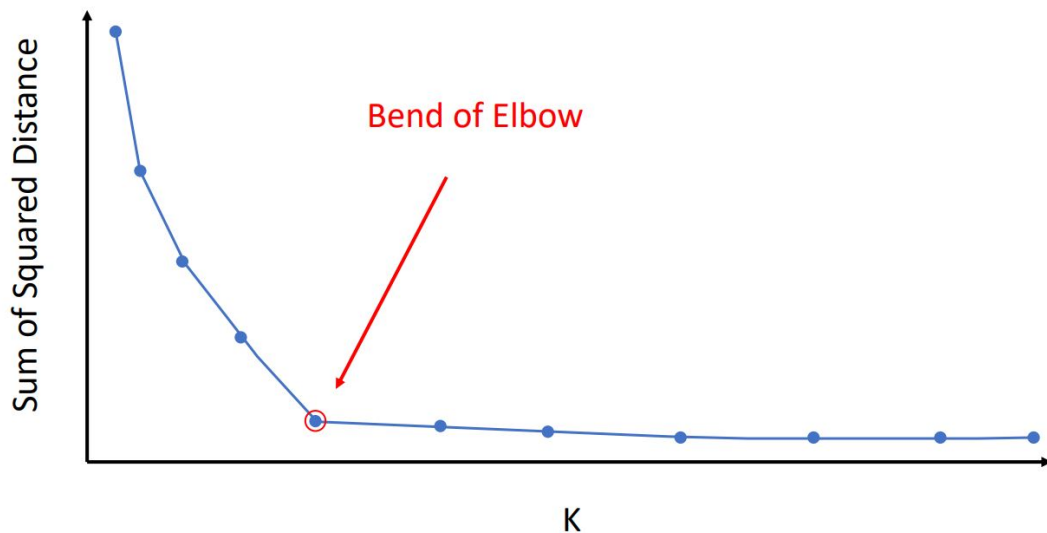


Fig. 1.12 Elbow technique for  $K$ -Means.

Elbow technique for finding the best input parameter  $K$ . The bend (in red) indicates that  $K$  is most likely to have optimal clusters.

However, finding  $K$  using the elbow method on deep visual features may output different results. Caron et al. [14] use  $K$ -Means to cluster ImageNet [28] dataset. Visual features are first extracted via convolutional neural network. In one experiment, the authors show that the optimal  $K$  is 10.000. However, one would expect the optimal  $K$  to be 1.000 since the dataset contains exactly 1.000 classes. Their method thus suggests an over-clustering of ImageNet dataset, which may be desired in practice as this means, that the resulting clusters are more fine-grained. For instance the class label *horse* can be clustered as *white horse* and *zebras*.

### 1.3.3 Minimum Cost Multicut Problem

Minimum Cost Multicut Problem, also called *Correlational Clustering* takes an input graph and separate the data through cutting its edges. Assuming an undirected graph  $G = (V, E)$  is given, where nodes  $v \in V$  represent images and edges  $e \in E$  encode their respective connectivity. Additionally, a given real valued costs  $c : E \rightarrow \mathbb{R}$  is defined on all edges, which represent the node affinities or similarity. The similarity measure (refer to Section 1.3.1) is crucial and is calculated in a pairwise manner between all nodes. The goal is to determine edge labels  $y : E \rightarrow \{0, 1\}$  defining a graph decomposition such that every partition of the graph corresponds to exactly one class. For instance an edge label of 0 means join of two nodes while 1 represents a cut, leading to decomposition of the nodes. To infer such an edge labeling, we can solve instances of the *Minimum Cost Multicut Problem* with respect to the graph  $G$  and costs  $c$ , defined as follows [23, 27]:

$$\min_{y \in \{0,1\}^E} \sum_{e \in E} c_e y_e \quad (1.8)$$

$$s.t. \quad \forall C \in \text{cycles}(G) \quad \forall e \in C : y_e \leq \sum_{e' \in C \setminus \{e\}} y_{e'} \quad (1.9)$$

The objective is to minimize (1.8) with respect to the assigned real valued costs of the edges and the corresponding cycle inequality constraint in Eq. (1.9). The cycle inequality constraint ensures that the edge labeling  $y$  induces a decomposition of  $G$ . In [23], it was shown to be sufficient to enforce Eq. (1.9) on all *chordless* cycles. A visualization can be found in Figure 1.13. Left illustrates an example of a constraint violation while right depicts an example of satisfied constraint, thus leading to correct decomposition of  $G$ . In the example, a complete graph  $G$  with 3 nodes A, B and C is shown. Node A and B belong to the same class (marked as blue). A single cut violates the constraint 1.9 thus a minimum of two cuts is required to decompose  $G$ .



Typically, if cut probabilities between pairs of nodes are available, the costs are computed using the *logit* function  $\text{logit}(p) = \log \frac{p}{1-p}$  to generate the real valued edge costs. This decision boundary or threshold  $\tau$  is learned from the data. Threshold  $\tau$  represents the decision boundary for multicut, which will be explained in detail in section 2.3.3. With these costs set appropriately, e.g.  $\tau$  is set correctly, the optimal solution of minimum cost multicut problems not only yields an optimal cluster assignment but also estimates the number of clusters automatically. Furthermore, this problem is able to generate small clusters and does not necessarily provide balanced sized clusters. However, we later show in Chapter 2 that this highly depends on the distribution of the data points in the learned feature or embedding space. Threshold  $\tau$  may still fail to correctly separate the boundaries. In order to overcome this problem,  $\tau$  can be derived directly from the optimization function, which is also one key contribution of this work. This will be discussed in Section 2.3.3.

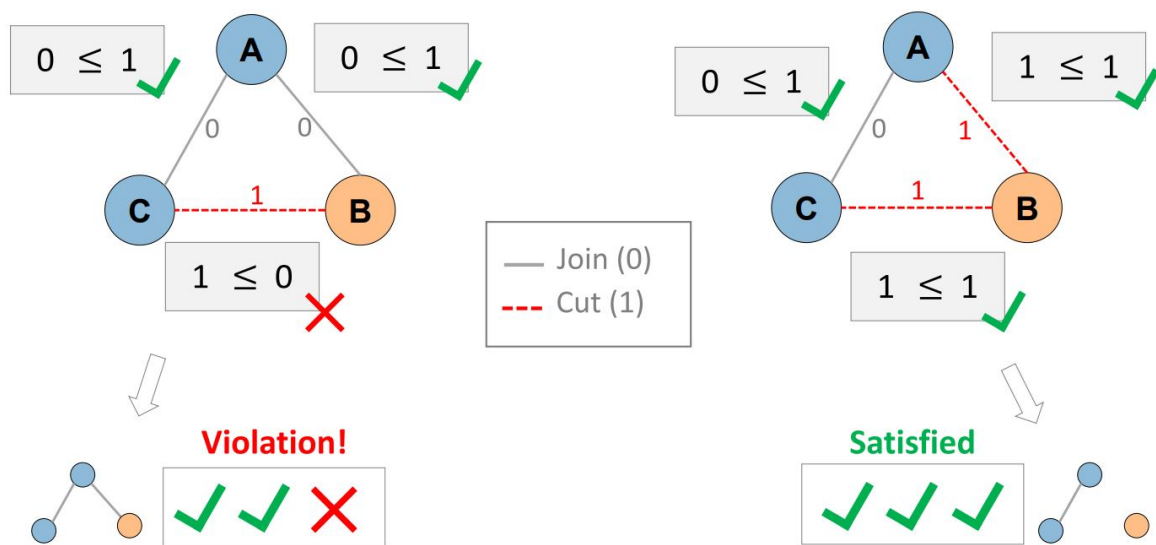


Fig. 1.13 Visualization of Cycle Constraint.

Visualization of cycle constraint for *Minimum Cost Multicut Problem*. A graph  $G$  with 3 nodes are shown. Left example violates the cycle inequality constraint (Equation 1.9). Right satisfies due to cut of two edges, which results in separation clusters.

### 1.3.4 Minimum Cost Lifted Multicut Problem

An extension with long range edges is called the *Minimum Cost Lifted Multicut Problem*. These lifted edges are added as a second set of edges in the graph and thus provide only additional information. The regular edges remain unchanged and define the possible solutions to the clustering problem.

For a given, undirected graph  $G = (V, E)$  and an additional edge set  $F \subseteq \binom{V}{2} \setminus E$  and any real valued cost function  $c : E \cup F \rightarrow \mathbb{R}$ , the 01 linear program written below is an instance of the *Minimum Cost Lifted Multicut Problem (LMP)* w.r.t.  $G, F$  and  $c$  [84]:

$$\min_{y \in Y_{EF}} \sum_{e \in E \cup F} c_e y_e \quad (1.10)$$

with  $Y_{EF} \subseteq \{0, 1\}^{E \cup F}$  the set of all  $y \in \{0, 1\}^{E \cup F}$  with

$$\forall C \in \text{cycles}(G) \forall e \in C : y_e \leq \sum_{e' \in C \setminus \{e\}} y_{e'} \quad (1.11)$$

$$\forall vw \in F \forall P \in vw\text{-paths}(G) : y_{vw} \leq \sum_{e \in P} y_e \quad (1.12)$$

$$\forall vw \in F \forall C \in vw\text{-cuts}(G) : 1 - y_{vw} \leq \sum_{e \in C} (1 - y_e) \quad (1.13)$$

The above inequalities Eq. (1.11) make sure that, as before, the resulting edge labeling is actually inducing a decomposition of  $G$ . Eq. (1.12) enforces the same constraints on cycles involving edges from  $F$ , i.e. so called *lifted* edges, and Eq.(1.13) makes sure that nodes that are connected via a lifted edge  $e \in F$  are connected via some path along original edges  $e' \in E$  as well. Thus, this formulation allows for a generalization of the cost function to include long range information without altering the set of feasible solutions.

Figure 1.14 illustrates an example of the regular and the lifted multicuts. Node A, B, C and D belongs to the same cluster, however the negative pairs AB and CD results in separation. Due to the positive pair AD, the regular multicuts (left) results in two clusters, (A, D) and (B, C), respectively. On the other hand, a lifted edge (right) provides additional, long range edge between node A and D with a strong similarity value. This result in one single cluster (A, B, C, D) when solving this via lifted multicuts.

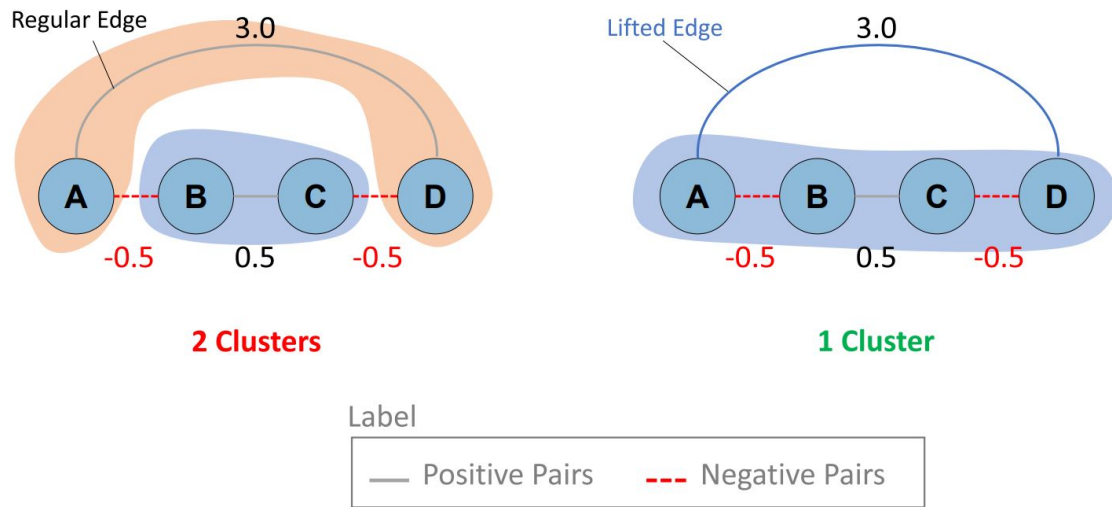


Fig. 1.14 Comparison Regular vs. Lifted Multicuts.

Example of for nodes, which belongs to the same class. Left shows the regular multicuts and right lifted multicuts, respectively. The negative pairs A, B and C, D results will separate the graph into two clusters using regular multicuts. On the other hand, lifted multicuts will join everything together correctly.

### Optimization

The *Minimum Cost Multicut Problem* (1.8) as well as the *Minimum Lifted Multicut Problem* (1.10) are NP-hard [6] and even APX-hard [27, 66]. Nonetheless, instances have been solved within tight bounds, for example in [2] using a branch-and-cut approach. While this can be reasonably fast for some easier problem instances, it can take arbitrarily long for others. Thus, primal heuristics such as the one proposed in [84, 79, 7] or [8] are often employed in practice and show convincing results in various scenarios [84, 149, 72]. Furthermore, another challenge is the memory size that is required to construct  $G$ . For instance, a clustering problem of images with a total dataset size of  $n$ , constructing a complete graph  $G$  results in total  $\frac{n(n-1)}{2}$  edges. This is often infeasible when considering large dataset such as ImageNet [134]. We will later propose an approach to solve large graphs in Chapter 3.

## 1.4 Contribution

We first evaluate models trained with *Triplet Loss* functions for image clustering problems based on two approaches: *K-Means* and *Minimum Cost Multicuts*. We also investigate the behaviour of label noise applied on triplets and reveals interesting behaviour of different *Triplet Losses* on both clustering algorithms. Based on this observation, we proposed a simplification of Triplet Loss from [171], which has several advantages: it is more robust against label noises, performs the best and, most importantly, the decision for cut and join on *Minimum Cost Multicut Problem* is directly learned during the optimization process, which is a significant parameter to tune. This important finding is presented in Chapter 2.

With the proposed simplification of the Triplet Loss, a new algorithm is then proposed in order to scale up the *Minimum Cost Multicut Problem* on clustering problem, which is presented in Chapter 3. One key disadvantage of the *Minimum Cost Multicut Problem* is the fact that it does not scale well on very large datasets. Since a graph is build, the number of edges  $E$  of a complete graph is equal to  $|E| = \frac{n(n-1)}{2}$  where  $n$  is the total dataset size. We show in Chapter 3, that our proposed algorithm is able to cluster a large number of data under a few seconds by utilizing data parallelism: the given dataset is splitted in disjoint sets and distributed on multiple computing threads. Each individual thread contains only a small subset of the data from the dataset, thus leading to a significantly smaller graph. Clustering task is applied on each individual graph in parallel and once the these tasks are solved, the results are merged together. We show that the runtime decreases when using more compute threads on a machine. Yet, no significant drop in performance (e.g. clustering accuracy) is observed. Furthermore, a theoretical proof for the speedup and memory complexity is provided as well.

In Chapter 4, we then apply the *Minimum Cost Multicut Problem* on a real-world problem where a Multiple-Object-Tracking problem (MOT) is treated as a classical clustering problem. We compare our proposed Triplet Loss approach with a fully unsupervised approach based on an convolutional AutoEncoder.

In Chapter 5, we show that clustering can be utilized as robustness predictor on classification models: first, the embedding space of several pre-trained models are evaluated. We show that that intra- and inter-class distances are not suitable as a direct indicator for a model's robustness. Instead, we propose a combination of two clustering methods and reveal a significant correlation between classification accuracy, robustness and clusterability.

## 1.5 Publications

### Peer Reviewed Conference Papers / Arxiv Papers

- Ho, K., Kardoost, A., Pfreundt, F. J., Keuper, J., & Keuper, M. (2020). A Two-Stage Minimum Cost Multicut Approach to Self-Supervised Multiple Person Tracking. In Proceedings of the Asian Conference on Computer Vision.
- Kardoost, A., Ho, K., Ochs, P., & Keuper, M. (2020). Self-supervised Sparse to Dense Motion Segmentation. In Proceedings of the Asian Conference on Computer Vision.
- Durall, R., Ho, K., Pfreundt, F. J., & Keuper, J. (2020). Latent Space Conditioning on Generative Adversarial Networks. arXiv preprint arXiv:2012.08803.
- Ho, K., Keuper, J., Pfreundt, F. J., & Keuper, M. (2021, January). Learning embeddings for image clustering: An empirical study of triplet loss approaches. In 2020 25th International Conference on Pattern Recognition (ICPR) (pp. 87-94). IEEE.
- Ho, K., Chatzimichailidis, A., Keuper, M., & Keuper, J. (2021, June). MSM: Multi-stage Multicuts for Scalable Image Clustering. In International Conference on High Performance Computing (pp. 267-284). Springer, Cham.
- Ho, K., Pfreundt, F. J., Keuper, J., & Keuper, M. (2022). Estimating the Robustness of Classification Models by the Structure of the Learned Feature-Space. In Thirty-Sixth AAAI Conference on Artificial Intelligence 2022, Workshop on Adversarial Machine Learning and Beyond.



# Chapter 2

## Feature Embedding of Image Data

In this Chapter, we introduce embedding learning methods for clustering image data, which is based on a concept and evaluation we previously published in [64]. The work has been supervised by Prof. Dr. Janis Keuper and Prof. Dr. Margret Keuper.

We first give an overview of two popular clustering techniques, *K-Means* and *Minimum Cost Multicuts* and explain the main differences of both methods. Then, we explain how such embedding for clustering is obtained via Deep Learning methods and we introduce three optimization loss functions. A thorough comparison of both clustering methods on three training losses are compared. The contributions of this chapter are:

- We conduct a thorough study of the clustering behavior of two popular clustering approaches, *K-Means* and *minimum cost multicuts*, applied to learnt embedding spaces from three Triplet Loss formulations on the CIFAR-10 [93] dataset under a varying amount of label noise
- Our study reveals that, while the traditional Triplet Loss [136] is well suited for *K-Means* clustering, its performance drops under the looser assumptions made by minimum cost multicuts.
- We propose a simplification of the Triplet Loss from [171] (2.3), which allows to directly compute the probability of two data points for belonging to disjoint components and is robust against noise in both clustering scenarios.
- Our proposed Triplet Loss variant outperforms both previous versions in terms of clustering performance and stability under label noise on the CIFAR-10 dataset.

## 2.1 Introduction

In computer vision, many different convolutional neural network (CNN) architectures have been proposed [94, 174, 70, 51, 145, 172, 67, 144]. All of them have one thing in common: the main goal is to automatically learn (deep) features, that can be classified. Such classification task assumes a fix number of output classes in order to learn the features. However, in real-world scenarios, the number of classes are often unknown thus clustering task or unsupervised learning are crucial when the goal is to group similar data. Furthermore, clustering also leverages the intrinsic data properties such as data density distributions or pairwise distances instead of annotations in order to group. The main challenge however is to find a good measure for similarity or distance metric. One such a metric could be the Euclidean distance of two data points (or images in computer visions) based on learned features. A simple choice is for instance raw pixel values of an image. Centroid-based algorithms such as *K-Means* can be applied though this is often neither effective nor feasible for images with higher resolutions, and it shows poor generalization. One approach is to utilize deep features of images with convolutional neural network: first, the dimension of the image (e.g. raw image pixels) is reduced into a lower (embedding) space with a non-linear mapping function. Secondly, the learned features are then clustered, which labels are not required. Deep features are learned through optimization of specific loss function with CNNs. For instance classification task often uses cross entropy loss. On the other hand, a popular loss function for learning embeddings, where the target number of classes are unknown, is the Triplet Loss [136]: three images are given such that the CNN learns to organize images of the same class closer to one another in the embedding space than images of different classes. Using the resulting embedding as features, one can run traditional clustering methods such as *K-Means* clustering as for example done in [164]. The clustering performance highly depends on the learned feature, e.g. how *close* or *far* similar or dissimilar data points are in the embedding space. Furthermore, when selecting a clustering algorithm, assumptions about the distribution of the data are being made: *K-Means* clustering assumes that the data points are evenly distributed around the cluster centroids. Such assumption may not be suitable for real world problems as a strict distinction between classes is often not possible as shown in Figure 2.1. Images of class *trucks*, *pickup trucks* and *cars* are projected in the embedding space based on some features. The left part of the figure shows the embedding space of the three classes that are well distributed. This assumption is made when performing the centroid based clustering *K-Means*, where the data (images) are distributed around the centroids. However, in reality, the distribution may appear in a smooth transition between *trucks*, *pickup trucks* and *cars*: this transition is spread in the embedding space.



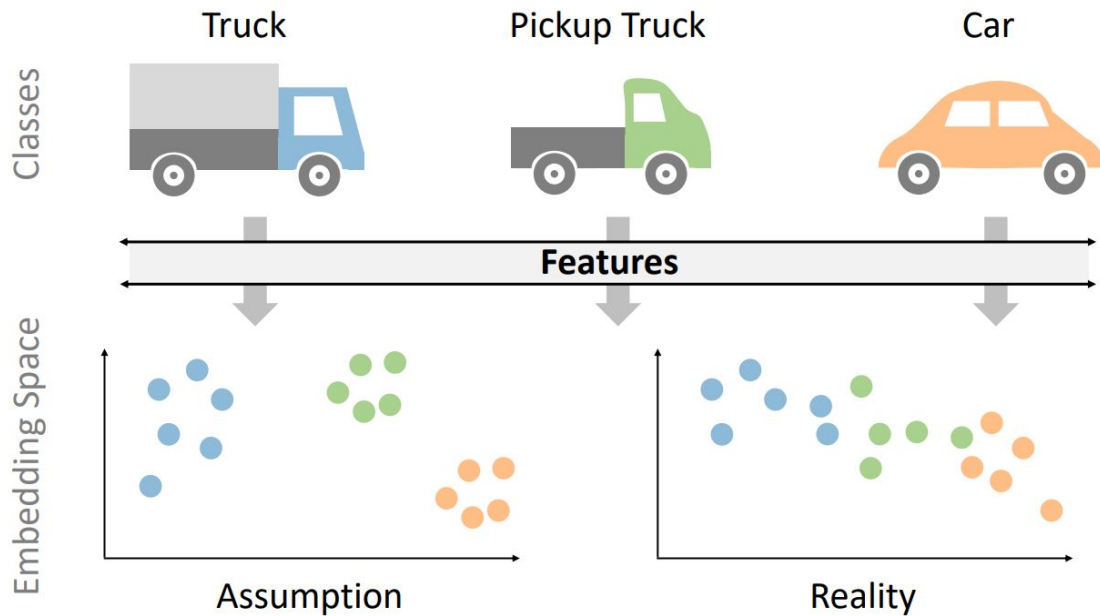


Fig. 2.1 Embedding Space: Assumption vs. Reality.

Images of class *cars*, *trucks* and *pickup trucks* are projected in the embedding space based on their features. Centroid-based clustering such as *K-Means* assume the data to be spread evenly across its centroid (shown in left) while in reality, it may appear to have a smooth transition, e.g. between *trucks* and *cars*, there is *pickup truck* as shown on the right.

Though in both cases, *K-Means* may still yield more or less the same clustering results, we argue that optimization of models (e.g. model training) should consider such behaviour. Additionally, *K-Means* also requires one to specify the number of clusters beforehand. Heuristics such as the elbow method are employed in order to determine the optimal number of clusters. In many practical tasks, this specific scenario might still be unrealistic because, for example, the number of objects to be grouped is simply unknown or because data points from the same class lie on a more complex manifold. This motivates us to additionally consider a graph-based approach, where no data specific knowledge is required, i.e. the *Minimum Cost Multicut Problem*, also known as *correlation clustering* [23, 27]. When features are learned through optimization of some loss functions, the training parameters can be utilized to derive parameters for the graph-based clustering, which will be explained further in the next section. This has the advantage that clustering can be done directly without the search for other parameters (such as the number of clusters  $K$  in *K-Means*). Furthermore, if the number of clusters are unknown, loss functions such as cross entropy loss are not suitable for optimizing the model.

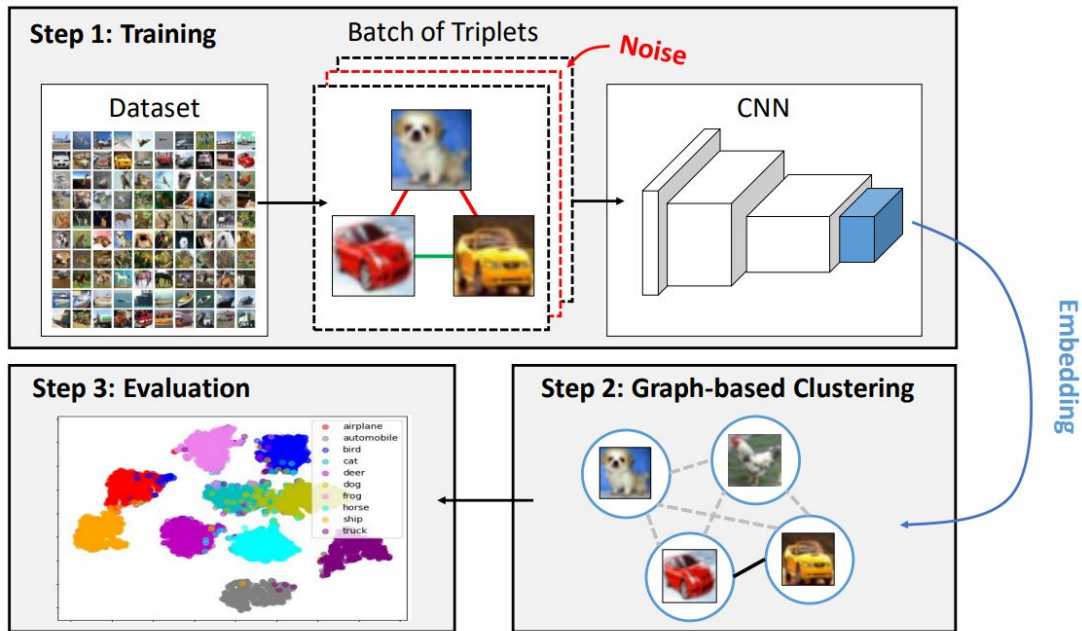


Fig. 2.2 Summary of Experiment Setup: Investigation of three Triplet Losses.

Summary of the experiment setup in three steps: 1) dataset is trained using the Triplet Loss. Here, we add random noises (in red) by selecting wrong samples in the training data. 2) We cluster data using graph-based approach based on the learned embedding features (in blue). 3) Evaluation of clustering accuracy.

In the context of these two common clustering techniques, we want to study the properties of embeddings resulting from two common variants of the Triplet Loss [136, 171] and investigate their susceptibility to label noise in the training data. By label noise, we mean wrong pairs of images that are sampled during training (more details in the next section). Additionally, we propose and study a third variant of the Triplet Loss, which shows promise in the context of both *minimum cost multicut*s as well as *K-Means* clustering and can be understood as a simplification of the loss proposed in [171]. Both share the desired property to directly allow the extraction of pairwise cut probabilities between data points from the embedding space without an intermediate learning step. Specifically, we train a CNN on the CIFAR-10 image classification dataset [93] to learn discriminative features using the three variants of the Triplet Loss, where we apply noise to the training labels for positive and negative samples. We evaluate the resulting embeddings by comparing the resulting clustering performance using *minimum cost multicut*s and *K-Means* clustering. Figure 2.2 illustrates our experimental setup.

## 2.2 Related Work

### 2.2.1 Clustering

Many clustering approaches on computer vision problems are based on dimensionality reduction, where a non-linear mapping function is applied. The key challenge is to learn distinctive features without using label information. One popular way is to use an autoencoder, where an input image is encoded into a embedding of lower dimension and then the decoder attempts to reconstruct its original. The embedding is then used as feature space for the clustering methods: For instance, Xie et al. [164] first train an autoencoder and then use the same dataset and fine-tune it by training it again using a KL-divergence loss. Another approach based on autoencoder is [41], which uses the reconstruction loss along with relative entropy to jointly train the network. Similar approaches can be found in [165, 152, 76]. Recently, generative models have been proposed for clustering tasks [118, 40]. A large scale study on clustering is proposed by Caron et al. [14], which iteratively groups the features with a *K-Means* during the optimization. The parameter  $k$  was estimated via elbow heuristics.

### 2.2.2 Correlation Clustering

Correlation Clustering, also referred to as the *Minimum Cost Multicut Problem* [23, 27] is a popular choice when the number of clusters are unknown. One such practical scenario is multiple object tracking, where pedestrians are tracked by just providing their detections [85, 140, 24, 56, 58]. Correlation clustering allows to group the data points based on pairwise cut probabilities without any cluster size bias and optimizes the number of clusters along with the data association. The crucial part there is to define or learn cut probabilities based on features. For instance [148] uses DeepMatching [159] as a similarity measure, followed by a logistic regression. An alternative is to use features from embeddings learnt through a Siamese network [149]. The training is based on pairs of images where the network outputs a binary decision, e.g. same or different person. This is closely related to our work, since we also want to obtain discriminative features by training a CNN with the Triplet Loss. The main advantage here is that the feature space is directly optimized from training data thus can be mapped to Euclidean space and the distances correspond to similarity measures. However, instead of a binary output as done in [149] (e.g. same or different), we want our network to learn an embedding of a fixed vector length, e.g. 32 dimensions.

### 2.2.3 Deep Embedding Learning

The main idea of learning embeddings is to attract similar data points to one another in a lower dimensional space while pushing dissimilar samples away from one another. While the Contrastive Loss [46] fixes the positive and negative pairs by a fixed distance, it can be restrictive to variations in the embedding space [163]. In contrast, the Triplet Loss [136] captures the relative similarity of pairs of data points instead of absolute similarities. It has been widely used for embedding learning [120, 173, 59, 31]. However, Zhang et al. [171] highlighted three major issues and thus proposed an *Improved Triplet Loss* by enforcing intra- and intercluster constraints. We compare these two Triplet Loss formulations and propose a simplification of the formulation from Zhang et al., which outperforms both other formulations in practice.

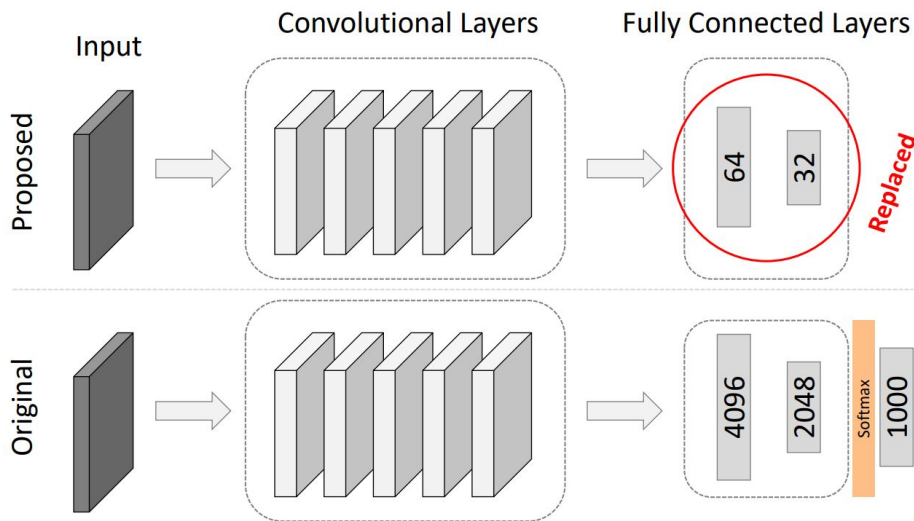


Fig. 2.3 AlexNet Architecture: Proposed vs. Original.

Top: Proposed model architecture. Bottom: original Alexnet [93] architecture. Softmax layers (in orange) and classification head is removed in our proposed model in order to learn feature vectors from images. The size of the fully connected layers are replaced (in red).

## 2.3 Contribution

In this Chapter, we describe the setup of our study. First, we introduce the network architecture and the different variations of the Triplet Loss in Chapter 2.3.1 and 2.3.2. Then, in Chapter 2.3.3, we explain the correlation clustering method, also called the *Minimum Cost Multicut Problem*. Chapter 2.4 describes the used dataset as well as the evaluation metric.

### 2.3.1 CNN-Architecture

We use AlexNet [94] as a CNN-backbone as done in [14]. Furthermore, we also replaced the local response layers with batch normalization as done in [14]. However, in order to reduce the feature dimensionality, we changed the size of the last two fully-connected layers from 4096 to 64 and 32, respectively. This is also due to the fact that the model was introduced for image classification on ImageNet [134], which has significant larger input size. The images are often resized or cropped to a dimension of 256 by 256 in width and height. Our experiments on the other hand evaluates the CIFAR-10 dataset, which all images have a resolution of 32 by 32 in width and height. Furthermore, we removed the classification head in order to obtain feature vectors from images only. The architecture change is shown in Figure 2.3.

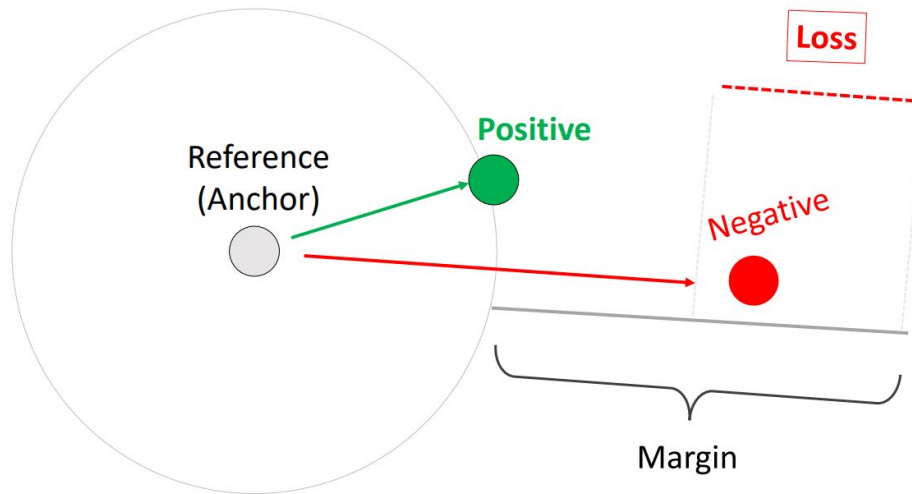


Fig. 2.4 Visualization of Triplet Loss.

Visualization of Triplet Loss (Equation 2.1). An anchor is utilized to compare positive and negative match. The margin determines the minimum distance, resulting in loss value (red dashed line).

### 2.3.2 Loss Function

The Triplet Loss is a cost function that uses a reference point, called *anchor* in order to compare a positive pair and a negative pair, thus requiring in total three data points in one single comparison. Unlike classification tasks where cross entropy is used to predict class labels directly, a distance is learned from input data. This is also called *metric learning* and the advantage is that the similarity between two data points such as two images can be directly compared via Euclidean distance: the further they are apart, the less they are similar (and vice versa). The main goal is to maximize the distance between the negative pairs, e.g. images from different classes, while minimizing the positive pairs (images from the same classes, respectively). Figure 2.4 visualizes one triplet: the reference point (*anchor*) is depicted as light gray circle. The green and red lines show the distance between the positive (green) and negative (red) pair. Margin (gray line) is a parameter that has to be specified by the user. It represents the degree of difficulty for triplets during the optimization and it only allows non-negative values. When positive samples are exactly far away from the negative samples by the value margin, the loss is zero. Any further distance (resulting in negative loss) will not be considered. The main idea of margin is that the training can focus on more

difficult triplets. In this example, the margin is larger than the distance difference between the positive and negative pairs, thus the loss becomes a positive value.

Given any model architecture (such as a convolutional neural network) with trainable parameters  $\theta$ , we want to map an input image  $x_i$  with a non-linear function  $f_\theta : X \rightarrow Z$ , with  $f(x_i) \in \mathbb{R}^d$ . In our case,  $x_i$  is the image of the  $i$ -th sample from the dataset with a total number of  $n$  samples,  $x_i \in X_{i=1}^n$  and  $d$  is the dimension of the embedding space.  $d$  is much smaller than the dimension of the input image. Given a set of three images,  $x_i^a$ ,  $x_j^n$  and  $x_k^p$ , the embedding features are learned by simply minimizing the Triplet Loss [136] over the parameters  $\theta$  of our deep neural network. Here, the parameter  $\alpha$  sets the margin of the similarity difference between the positive sample  $x_k^p$  and negative sample  $x_j^n$  and the anchor image  $x_i^a$ :

$$L_{triplet} = \sum_{i=1}^n [\|f(x_i^a) - f(x_i^p)\|^2 - \|f(x_i^a) - f(x_i^n)\|^2 + \alpha]_+ \quad (2.1)$$

Figure 2.4 visualizes Equation 2.1. Our approach is based on the assumption that embedding features, learned from the regular Triplet Loss (2.1) can produce high variances in inter- and intra-cluster distances, because it only considers relative differences between the distances of positive and negative pairs. This objective is suitable for K-Means clustering. Yet, the attempt to learn whether two data points should belong to the same or to a different class from their pairwise distances might fail, when the intra- and inter cluster samples are equally far away. This is shown in Figure 2.5 where the correct decision boundaries are marked by green lines. In contrast, the red line, at the same Euclidean distance as the green lines, indicates a false separation of data. This motivates us to consider losses that preserve the distance equally between the positive pairs during the optimization. Similar to [171], we therefore add an additional term to equation (2.1), which we denote as *Triplet Loss\_2* [171]:

$$L_{triplet\_2} = L_{triplet} + [\|f(x_i^a) - f(x_i^p)\|^2 - \beta]_+ \quad (2.2)$$

The additive term in equation (2.2) introduces an additional parameter  $\beta$ , which sets the maximum distance between the positive pairs, e.g. the intra-cluster distance. As the regular Triplet Loss only considers the distance *difference* between the positive and negative pairs, set by the parameter  $\alpha$ , we propose a third loss function, which considers the absolute distance for the positive and negative pairs (instead of distance difference):

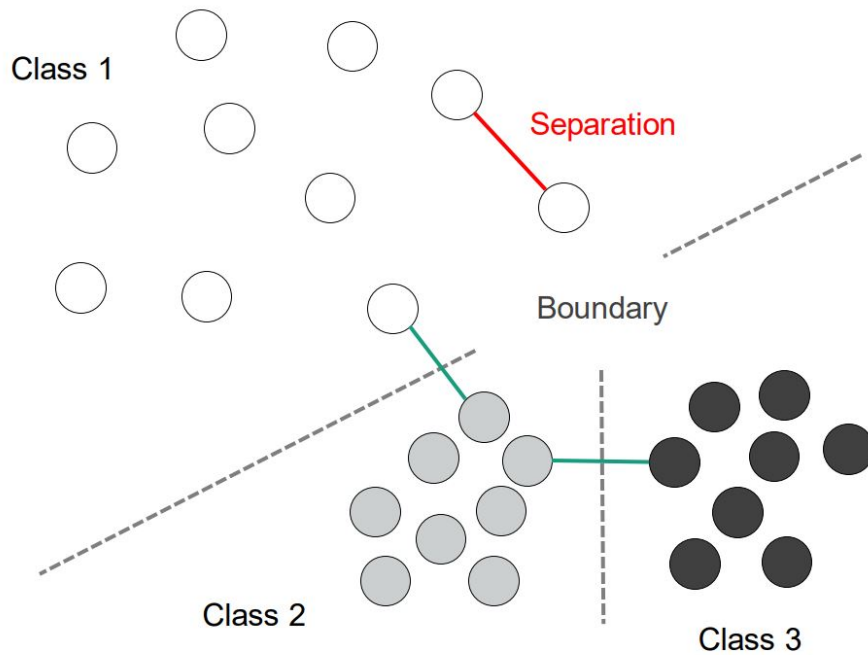


Fig. 2.5 Decision Boundary in Embedding Space.

Visualization of a possible data distribution trained with Triplet Loss [136] (Equation 2.1): different intra-cluster distances over different classes make it impossible to learn one distance threshold at which data points should belong to different components. A logistic regression model could thus not learn cluster boundaries for graph partition. The correct decision boundaries are marked by green lines while the red line shows a wrong separation of data. Our aim is to propose new Triplet Loss to stabilize such distances.

$$L_{triplet\_3} = [\alpha - \|f(x_i^a) - f(x_j^n)\|^2]_+ + [\|f(x_i^a) - f(x_k^p)\|^2 - \beta]_+ \quad (2.3)$$

We argue that this variant of the Triplet Loss is more intuitive than Triplet Loss\_2 because it directly pushes positive pairs within a certain margin  $\beta$  while driving negative pairs apart with a minimum distance  $\alpha$ . Within these margins, it still allows for varying distances, which is in contrast to for example Siamese approaches or the contrastive loss [46]. Despite having an additional hyper-parameter  $\beta$ , we will show later that this is in fact beneficial for our proposed end-to-end framework for feature learning and clustering approach (Section 2.5).



### 2.3.3 Pairwise Cut Probabilities for Multicuts

We formulate the following image clustering task as a *Minimum Cost Multicut Problem* as described in Section 1.3.3: each node in the graph  $G$  represents an image  $x_i$  that we want to assign to a class label (instead of edge labels as presented previously). The weight of the edges between two nodes represents the similarity between the two images. The similarity is computed based on the Euclidean distance of in embedding feature, for instance using a CNN-model:

$$d_{i,j} = \|f(x_i) - f(x_j)\| \quad (2.4)$$

The more similar two images are, the less  $d_{i,j}$  becomes. Once the similarity of two images is obtained, we seek to estimate their probability  $p \in [0, 1]$  to belong to distinct classes. A decision function can be learned through a logistic regression using the label information, e.g. positive pairs are mapped to the value 0 while negative pairs are mapped to 1 as shown previously in Figure 2.5. The threshold value  $\tau$  is therefore represented as the boundary between two classes: if a distance (similarity) between two nodes is below  $\tau$ , it will be joined. Consequently, two nodes with a distance larger than  $\tau$  will be separated.

However, with the Triplet Loss<sub>2</sub> (Equation 2.2) and the proposed Triplet Loss<sub>3</sub> (Equation 2.3), the distance threshold  $\tau$  can be automatically derived from the training parameters  $\alpha$  and  $\beta$ , if they are set correctly. That is, if the intra-cluster distances were set exactly half of the inter-cluster distances during the optimization with the two losses, the decision boundary can be automatically estimated. Since  $\beta$  restricts the maximum distance of positive pairs to exactly  $\beta = \alpha/2$ , the distance threshold  $\tau$  of the logistic function is computed as:

$$\tau = \sqrt{(\alpha + \beta)/2} \quad (2.5)$$

Thus the additional step of learning  $\tau$  from data is not necessary anymore. The embedding space that is essentially optimized for correlational clustering. Note that this is not possible for the Triplet Loss from Equation (2.1) because it only considers relative distances.

## 2.4 Experiments and Results on CIFAR-10

In this Chapter, we present the experiments and the results of our study. CIFAR-10 [93] is a popular dataset for image classification tasks. It contains 50.000 train and 10.000 test data samples of tiny images, which are 32 by 32 pixels in width and height. Each sample is assigned a label that belongs to one of the ten classes: *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*. All models are trained for 100 epochs with a batch size of 100 and a learning rate of 0.001 using AdamOptimizer [88]. First, we compare the performance of the *Minimum Cost Multicuts* and *K-Means* clustering using different Triplet Losses in Chapter 2.4.1. Then, we present some insights related to inter- and intra-cluster distances in Chapter 2.4.2. In Chapter 2.4.3, we present our study on the feature learning under label noise. The results are shown in Table 2.1. In Chapter 2.4.4, we present some qualitative results.

### 2.4.1 Evaluation of Cluster Accuracy

We compare the CNN models that are optimized with the three different losses. Specifically, we train an AlexNet model as described previously in Section 2.3.1 on CIFAR-10 dataset with the three Triplet Losses. Then, embedding features of images from the test dataset are extracted and we evaluate two different clustering methods, *K-Means* and *correlational clustering*. Here, clustering the embedding features with *K-Means* require the parameter  $K$  to be set correctly. On CIFAR-10, this is set to  $K = 10$  (thus introducing external knowledge to the clustering process). On the other hand, *correlational clustering* requires the threshold parameter  $\tau$  in order to cluster the data correctly. For Triplet Loss (2.1), an additional regression model is trained using the label information in order to estimate the threshold, while for (2.2) and (2.3), the threshold is computed directly from the optimization parameters using equation (2.5).

**Evaluation Metric.** While *K-Means* outputs exactly  $K = 10$  clusters, the number of clusters resulting from *correlational clustering* is arbitrary large. In order to evaluate the performance of the two clustering algorithms, we resort to the clustering accuracy metric using the label information from the dataset. The former clustering algorithm is straightforward since it  $K = 10$  represents the number of different classes from CIFAR-10 dataset. The latter is calculated based on the Hungarian algorithm [95], where the best match between the predicted and the true labels are found. That is, if more than 10 clusters are found, anything above  $K = 10$  is considered as false positive, thus lowers the clustering performance.

All experiments are executed five times with different random seeds and we report the average number over the clustering accuracy.

**Results.** Figure 2.6 shows the clustering accuracy for different variants of the Triplet Loss. The average clustering performances for both, *K-Means* and *Minimum Cost Multicuts* (denoted as correlational clustering), over five runs are depicted above the bars, while the standard deviations are shown as black lines. First, we observe that Triplet Loss\_2 (2.2) and Triplet Loss\_3 (2.3) outperform (2.1) on multicut clustering (left). However, *k-means* (right) performs better on all our experiments given the fact that  $K = 10$  is known. The highest performance is achieved when we train the CNN-model with the Triplet Loss\_3 (2.3), where the average accuracy is 80.5% (red). For *k-means*, the Triplet Loss\_2 shows worse performance than the regular one [136], while the proposed, simpler version, Triplet Loss\_3, performs best in both scenarios.

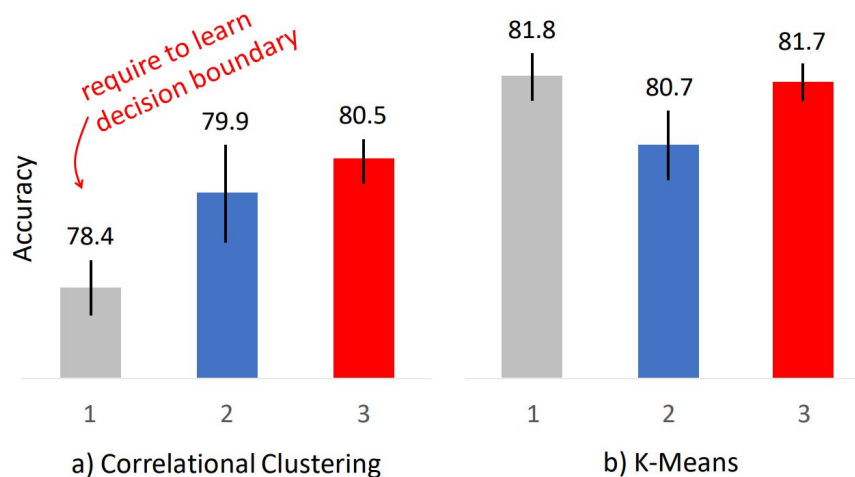


Fig. 2.6 Clustering Performance of three different Triplet Losses.

Comparison of three different losses on clustering performance. The numbers on the x-axis represent the triplet loss type while the y-axis shows the average cluster accuracy on five runs. The black line indicates the standard deviation. Triplet Loss\_2 (2.2) and our Triplet Loss\_3 (2.3) perform better than the regular Triplet Loss for minimum cost multicuts. However, K-Means consistently achieves better results given the fact that the parameter  $K$  is set correctly (shown in right). For K-Means, the Triplet Loss\_2 shows worse performance than the regular one [136], while the proposed, simpler version, Triplet Loss\_3, performs best in both scenarios.

### 2.4.2 Inter- and Intra-cluster distances

In this experiment, we investigate the inter- and intra-cluster distances from the dataset that is embedded in the trained neural network on the three Triplet Losses. Figure 2.7 compares the cluster distances of the samples. The red curve represents the average pairwise distances of the samples within a cluster (intra-cluster distance) while the blue curve shows the average distances of one cluster to its nearest cluster (inter-cluster distance). Furthermore, the color range represents the standard deviation  $\sigma$  and  $2\sigma$  of the cluster distances. The y-axis shows the Euclidean distance while the x-axis depicts the 10 classes of the evaluated dataset CIFAR-10.

**Results.** As illustrated in Figure 2.7 a), both distances show significant variances when trained with the regular Triplet Loss (2.1). These large variances are represented as large blue and red bars. Furthermore, class *cat* and *dog* show a significant overlap in their inter- and intra cluster distances, which prevents the logistic regression from setting the right decision boundary as explained in Figure 2.5. This also explains, why the clustering performance is the lowest among the three Triplet Losses (refer to Figure 2.6 a). In contrast, Triplet Loss\_2 (2.2) and Triplet Loss\_3 (2.3) produce more consistent and stable results. At the same time, the cluster performance is also higher, as shown previously. More importantly, a clear separation of the inter- and intra-cluster distance is observed for both methods (b and c). This validates our hypothesis that the embedding space from the trained model using Triplet Loss\_2 and Triplet Loss\_3 allows a better separation of data compared to the regular Triplet Loss.

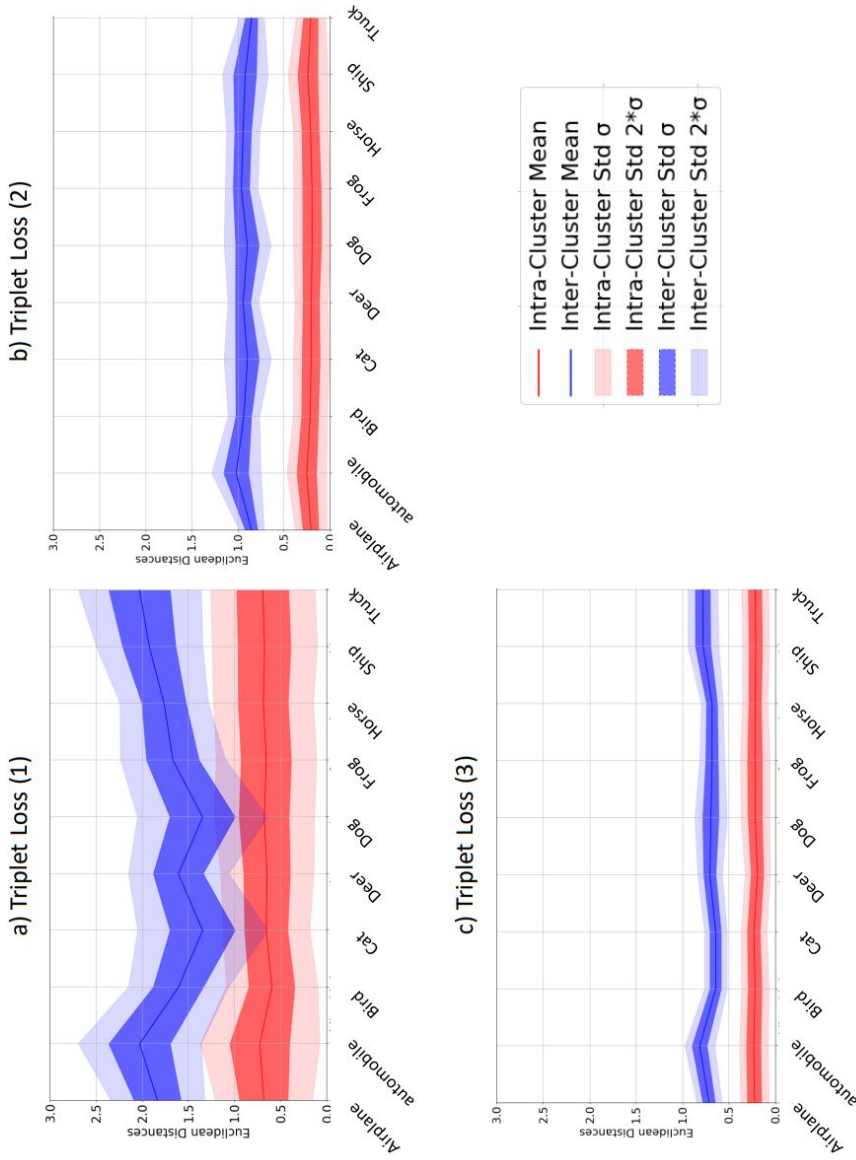


Fig. 2.7 Inter- and Intra Cluster Distance trained with different Triplet Losses.

Inter- and intra cluster distance of AlexNet trained with all three Triplet Loss variants (equation (2.1), (2.2) and (2.3)): the average distance shows significant variance when trained with a) Triplet Loss\_1 (2.1), especially for the class *cat* and *dog*. The overlap of the distances prevents the logistic regression model to learn the cluster boundaries. Both other variants, b) and c), produce more consistent and stable results, thus higher clustering accuracy.

### 2.4.3 Triplet Loss with Label Noise

In this experiment, we investigate the sensitivity of the different Triplet Losses towards label noise. Specifically, we randomly select *wrong* triplets during the training process of our CNN-model and evaluate the clustering performance based on the embedding features, trained on all three loss variants from Subsection 2.3.2. Figure 2.8 depicts three examples of triplets. The experiments were conducted repeatedly five times with different seeds and we report the mean cluster accuracies in %. The parameters  $\alpha$  and  $\beta$  are fixed to 0.8 and 0.4, respectively.

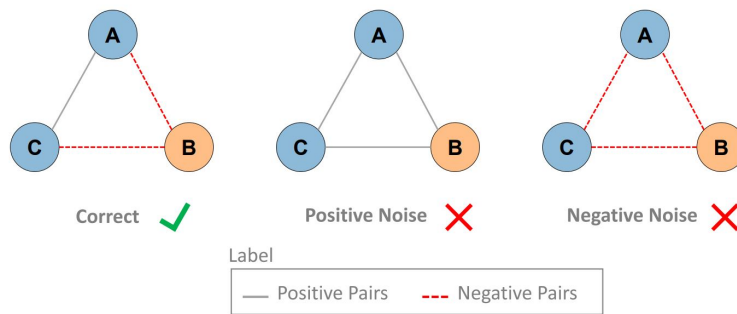


Fig. 2.8 Label Noise for Triplet Loss.

Illustration of three different triplets, where node A and C belonging to the same class. A positive noise indicates that all three nodes shares a positive edge label while a negative triplet consists of only negative edge labels.

**Results.** Table 2.1 shows our complete evaluation with various setups: we applied different amounts of label noise on triplets for positive and negative pairs. The x-axis represents the noise for the negatives while the y-axis indicates noise on positive pairs within the triplet. These *wrong pairs* are retrieved randomly. All results are reported as average clustering accuracy over five runs of training. In Table 2.1 top, we present the performance using *Minimum Cost Multicuts* while the bottom rows show the results of *K-Means* clustering. Note that *K-Means* requires to specify the number of clusters  $K$ , beforehand (on CIFAR-10, we know  $k = 10$ ), while *Minimum Cost Multicut* do not require this dataset specific knowledge. Without noise added, considering the CNN-model trained with the same loss function, *K-Means* seems more stable against noise and outperforms the *Minimum Cost Multicuts* on average by 1-2% and the highest clustering accuracy when no noises are added. In Figure 2.9, we give a more detailed analysis of these results.

**Remark:** When sampling the triplets randomly on balanced dataset with  $K$ -clusters, the chance to get a true positive and true negative pair is  $\frac{1}{K}$  and  $\frac{K-1}{K}$  respectively.

Noise %	Triplet Loss (2.1)				Minimum Cost Multicuts Triplet Loss (2.2)				Triplet Loss (2.3)						
	random	7%	5%	2%	0%	random	7%	5%	2%	0%	random	7%	5%	2%	0%
20%	73.85	73.64	74.43	76.28	75.18	60.13	65.61	68.05	67.26	64.39	75.23	74.39	75.00	72.80	75.27
10%	77.90	77.02	76.93	78.04	77.87	74.23	75.64	75.23	74.75	75.48	77.10	76.53	76.47	76.06	76.29
5%	78.65	76.73	78.30	77.73	78.87	75.62	76.27	76.06	75.80	76.63	77.07	78.94	80.16	77.44	77.75
0%	78.14	77.76	78.31	78.56	<b>78.44</b>	77.07	77.46	77.47	80.41	<b>79.96</b>	80.25	80.38	80.65	80.42	<b>80.51</b>

Noise %	Triplet Loss (2.1)				K-Means Clustering Triplet Loss (2.2)				Triplet Loss (2.3)						
	random	7%	5%	2%	0%	random	7%	5%	2%	0%	random	7%	5%	2%	0%
20%	79.83	80.26	80.29	80.60	80.55	71.71	74.70	74.86	74.62	74.61	79.66	80.54	77.83	78.20	80.53
10%	81.55	80.95	81.10	81.53	81.21	75.33	75.80	75.89	75.54	76.48	81.93	81.78	80.59	80.45	82.06
5%	81.62	80.82	81.27	81.19	81.68	76.79	77.01	76.14	76.97	76.93	82.07	82.11	81.98	81.90	82.08
0%	81.59	81.33	81.53	81.51	<b>81.82</b>	80.25	80.80	81.11	81.39	<b>80.73</b>	82.15	82.14	81.93	81.64	<b>81.72</b>

Table 2.1 Clustering Performance of three Triplet Losses.

Evaluation of clustering accuracy using **minimum cost multicuts** (top) and **K-Means** (bottom) based on the embedding features of the CNN-model trained on three different Triplet Loss variants. The average accuracies of five runs are reported in %. Furthermore, the numbers in bold are the reported numbers from Figure 2.6. The x-axis shows the amount of label noise on the negative samples up to 7%, while y-axis shows the amount of label noise on the positive samples up to 20%, respectively. The column *random* selects the negative sample without employing the labels. On the CIFAR-10 dataset, the chances are 90.0% to retrieve correct negative samples.

### Clustering Performance vs. Sample Noise

Figure 2.9 shows the clustering accuracy against the percentage of noise that is applied to the sampling. Top left, we only add noise to the positive pairs while selecting correct negative samples and evaluate using *Minimum Cost Multicuts*, i.e. without introducing knowledge on the number of classes. Our first observation is that Triplet Loss<sub>2</sub> (2.2) is the most sensitive to noise among all three loss variants. This is shown in Figure 2.9 in blue. The regular Triplet Loss (2.1) and Triplet Loss<sub>3</sub> (2.3) still achieve an average clustering accuracy of 75.0%, respectively, even though 20% of wrong samples are used for training. A similar behavior can be observed in top right, where we evaluate the same embeddings using K-Means clustering.

The observations are slightly different when adding label noise to the negative pairs, Figure 2.9, second row. Even when introducing 10% noise, which corresponds to drawing negative samples completely at random in a balanced 10 class classification problem, all loss variants are relatively robust, especially for *K-Means* clustering (Figure 2.9, second row, right). Yet, the proposed Triplet Loss<sub>3</sub> again performs best.

In the bottom row of Figure 2.9, we consider an equal amount of noise on both positive and negative samples (corresponding to the diagonal in Table 2.1). In this setting, the proposed Triplet Loss<sub>3</sub> again shows higher stability than the two previous variants when clustering using *Minimum Cost Multicuts* Figure 2.9 (bottom left). For *K-Means* clustering, the improvement over the Triplet Loss [136] is marginal. This result is actually expected: The traditional Triplet Loss [136] creates an embedding such that, for every data point, data points from the same class are closer than points from any other class. This fits well with the K-Means clustering objective, assigning every point to the nearest cluster center, regardless their absolute distance. Yet, it can be problematic in the context of correlation clustering, where the minimum absolute distance between two clusters matters.



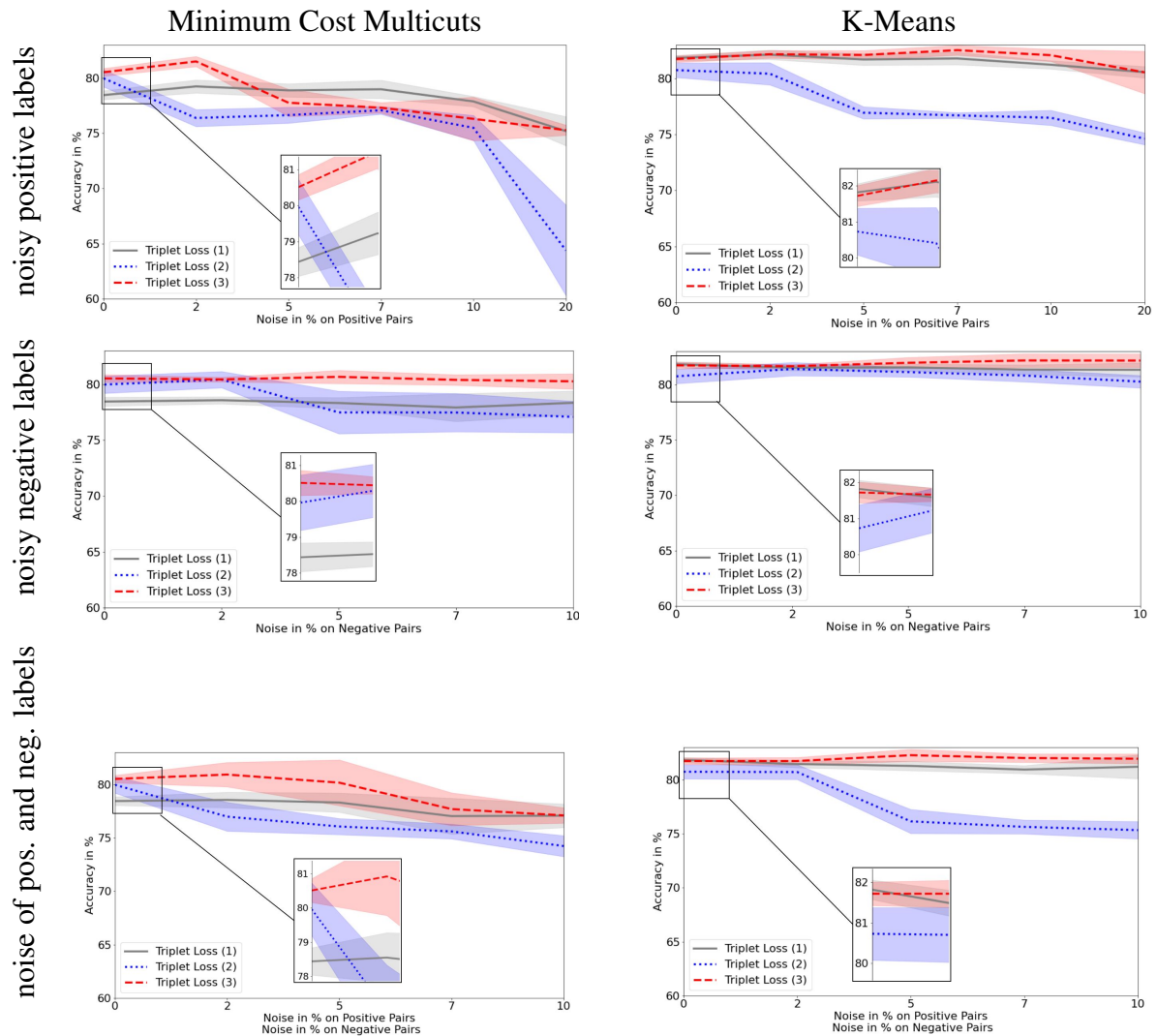


Fig. 2.9 Cluster Accuracy vs. Sample Noise.

Average cluster accuracy against the percentage of noise, that is applied to the sampling. Noises on positive and negative pairs are applied on first and second row, respectively while the last row is evaluated on an equal amount of label noise of both, positive and negative pairs. The first column shows the results of the minimum cost multicuts while the second column employs K-Means. The amount of noise (i.e. wrong pairs) is indicated in x-axis.

### 2.4.4 Qualitative Results

Figure 2.10 shows a TSNE-visualization [112] of the embedding features learned from the CNN-model using the Triplet Loss<sub>3</sub> (2.3) variant. We use the *Minimum Cost Multicuts* approach to cluster CIFAR-10 test dataset. In the particular experiment example, the total number of clusters is 44 while the cluster accuracy is 80.27%. The different colors represent the found class labels while in the ground truth, there are only 10 classes on the CIFAR10 dataset (which is shown in the legend). Any other found clusters are considered as false positives and thus lower the cluster accuracy. However, there are in fact 34 small clusters that contain less than 10 images. Three examples of such mini clusters are shown in bottom left and right as well as on the top left corner. Even though there are false positives shown in the examples of the smaller clusters, the *Minimum Cost Multicuts* approach explores meaningful sub-clusters within a class label, which may be desirable on real-world scenarios. For instance, instead of finding the class *horse* (in cyan), a subclass *white-horses* is also found, as shown in Figure 2.10, bottom right.

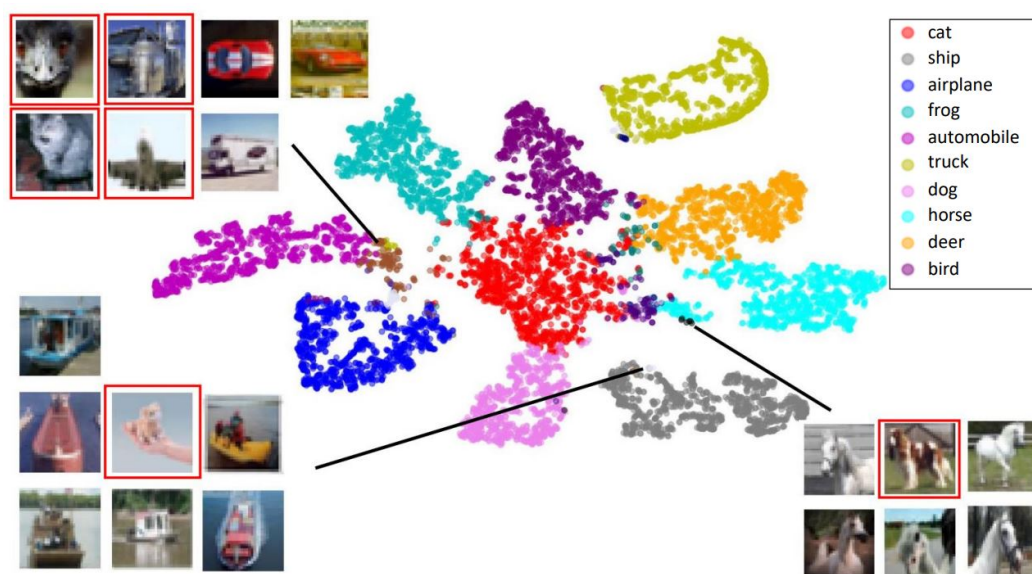


Fig. 2.10 TSNE-Visualization CIFAR-10 with a Multicuts.

TSNE-Visualization of the clusters on CIFAR-10 using a multicut approach. In this example, the cluster accuracy is 80.27% trained on a CNN-model with the Triplet Loss (2.3) and the total number of clusters is 44. The color represents the 10 largest clusters found. There are 33 small clusters (< 10 items), for instance bottom left and right are two clusters shown containing 7 images each. False positives within the clusters are marked as red. The cluster on the upper left corner contains images from bird, cat, automobile (x3), planes and trucks.

## 2.5 Conclusion

In this Chapter, we presented an extensive study on three different variations of the Triplet Loss. Specifically, we have studied the clustering behavior of *K-Means* and *Minimum Cost Multicuts*, applied to learnt embedding spaces from three Triplet Loss formulations on the CIFAR-10 [93] dataset under a varying amount of label noise. We find that, while the traditional Triplet Loss [136] is well suited for *K-Means* clustering, its performance drops under the looser assumptions made by *Minimum Cost Multicuts*. We proposed a simplification of the Triplet Loss from [171], which allows to directly compute the probability of two data points for belonging to disjoint components. In a line of experiments on the CIFAR-10 dataset, we show that this proposed loss is robust against label noise in both clustering scenarios and outperforms both previous Triplet Loss versions in terms of clustering performance and stability. Our hypothesis is that the embedding space trained with the proposed Triplet Loss yields allows better separation of the data. This is validated in another experiment, where the intra- and inter-cluster distances are analyzed: the regular Triplet Loss shows large overlap thus making the separation of data challenging. In the last experiment, a qualitative result is presented. We showed that the *minimum cost multicuts* is able to find sub-classes, which are not defined from the original dataset and we argue that such clustering approach is generally favourable over centroid-based clustering since the distribution of the data in the embedding space is arbitrary and the number of clusters is unknown. With our proposed Triplet Loss, a CNN is trained specific for the *Minimum Cost Multicut Problem*, where threshold  $\tau$  is directly derived from the training parameter  $\alpha$  and  $\beta$ .

## 2.6 Limitation

Currently, the sampling method is done in a supervised way (e.g. triplets are selected using the label information). Furthermore, it has been shown in [163] that the selection of triplets also influences the performance greatly, which we have not investigated in this work. Another problem is that the propose graph-based approach does not scale on larger datasets due to the large size of the graph, which we previously addressed in Section 1.3.4. Not only is solving the problem NP-Hard, it also requires a significant large amount of memory to create graph  $G$ . In the next chapter, we will introduce a novel approach to solve this problem by proposing a multi-stage multicuts that is able to solve large graphs and show theoretical proof.



# Chapter 3

## Scalable Multicuts

In this Chapter, we introduce a novel approach to cluster large number of data using *Minimum Cost Multicuts*. The content of this chapter is based on the approach, that we previously published in [60]. In addition to this, a theoretical proof is provided to verify our claim. The full proof is contributed by the co-author, Avraam Chatzimichailidis, while I contributed the idea, algorithmic implementation and experimental evaluation. The work has been supervised by Prof. Dr. Janis Keuper and Prof. Dr. Margret Keuper.

The rest of this chapter is structured as follows: an introduction is provided in Section 3.1, followed by related work in Section 3.2. The main idea of our approach is explained in Section 3.3 and evaluations of the proposed method are provided in Section 3.4.

### 3.1 Introduction

Clustering data based on some feature measure has been a major interest in machine learning, especially in the area of computer vision. So far, we have shown in the previous chapter, that if pairwise similarities between data points in a set are given, one can partition (e.g. cluster) the data. Such similarity can be obtained via metric learning, such as using the Triplet Loss. The advantage is that the number of clusters does not have to be determined beforehand. This is important as data specific knowledge remains unknown for real world problems. *Correlational Clustering*, also called *Minimum Cost Multicuts*, solves such clustering tasks globally and in a deterministic way. Data points (e.g. nodes) form a graph  $G$  with a cost on the edges between each other, where these costs are often computed using some features based on deep learning models (explained in 2.3.3). The graph is partitioned based on this cost. Previous experiment of clustering task on CIFAR-10 has shown promising results in terms of clustering performance and stability. Our proposed Triplet Loss 2.3 yields best

performance on graph-based clustering and is, at the same time, less susceptible against label noise. Although  $G$  can be either a complete graph or a graph with arbitrary number of edges, solving the *Minimum Cost Multicut Problem* is known to be NP-hard. With the increase in data size (e.g. nodes in  $G$ ), solving such a task remains a big challenge.

**Contributions.** To overcome this challenge, we propose MSM (Multi-Stage Multicuts) to solve the *Minimum Cost Multicut Problem* on large graphs. Specifically, MSM utilizes data parallelism to achieve significant speedup while preserving the performance on image clustering tasks. This straightforward method divides a single, large clustering problem into small disjoint sets across different CPU threads on a shared memory system. This has two main advantages: (1) the clustering tasks become smaller on each thread and (2) the optimization on each set is done concurrently. Our key contributions of this paper are summarized as follows:

- We provide MSM: Multi-Stage Multicuts approach based on *Minimum Cost Multicuts*, that is capable of solving large graphs using features from deep neural networks. MSM runs concurrently on multiple CPU threads, allowing to solve large image clustering problems.
- Theoretical proof for the speedup as well as the memory complexity.
- The performance of MSM is evaluated on multiple scenarios based on image clustering tasks on CelebA, CIFAR10 and CIFAR100 dataset.

## 3.2 Related Work

**Image Clustering.** The goal of image clustering is to assign class labels to a given set of data based on their semantic features. These semantic features for clustering are often obtained via convolutional neural networks (CNN) [94, 21], where dimensions are successively reduced via non-linear mapping functions on each layer. A joint optimization approach is proposed in [14, 15], where these features are learned and clustered via assigned pseudo-labels. However, [170] shows that this method leads to unstable training and propose a decomposing feature clustering approach to tackle this issue. [45] propose a Graph-Convolutional Network approach with LSTM for a density-aware face clustering. [105] seeks fairness in clustering by hiding sensitive features that are based on min-max game strategy. Other methods, such as autoencoders [164, 41], generative models [118, 40] or transformer [16] have also been proposed to solve image clustering tasks.

**Minimum Cost Multicuts.** *Minimum Cost Multicuts*, also called Correlational clustering [23, 27] is a graph-based clustering technique, where data points are represented in a graph and pairwise similarities between the nodes are utilized to optimize (e.g. cluster) the overall problem. This technique has shown success in various computer vision applications, such as multiple object tracking [85], motion and image segmentation [78, 162], pose estimation [127] and image clustering [64]. The *Minimum Cost Multicut Problem* is known to be NP-hard [27]. However, the use of heuristic solvers [83, 8] yield reasonable results in practice.

**Parallel Computing.** Parallel computing describes the process of partitioning a problem into smaller ones and solving these sub problems simultaneously. Recently data parallelism [26, 92] and model parallelism [48, 71] have been explored in the context of deep learning. In [122], a parallel *Multicuts* approach for binary graphs is proposed. In order to scale to bigger datasets, our proposed MSM makes use of the data parallel approach, where each worker holds a copy of the full model locally, as well as a fraction of the whole dataset.

### 3.3 MSM: Multi-Stage Multicuts

In this Chapter, we explain our proposed algorithm: Multi-Stage Multicuts (MSM), which allows parallelization across multiple CPU threads (worker) with shared memory in order to solve a large image clustering problem. An overview of MSM is shown in Figure 3.1. A dataset is too large to be solved on a single CPU thread. First, the number of stages and the available resources have to be specified. At stage one, all available resources are utilized: the large dataset is divided into equally large, disjoint sets. On each CPU thread, clustering tasks are solved simultaneously and the results are then forwarded to the next stage (marked as red). Here, not all the available compute resources are utilized anymore. The clustering tasks are done on the intermediate results from the previous stage. At the final stage, which consists of only one single worker, the results are sent back to all previous stages (marked as green). Through this work, we will use the term *worker* for CPU threads.

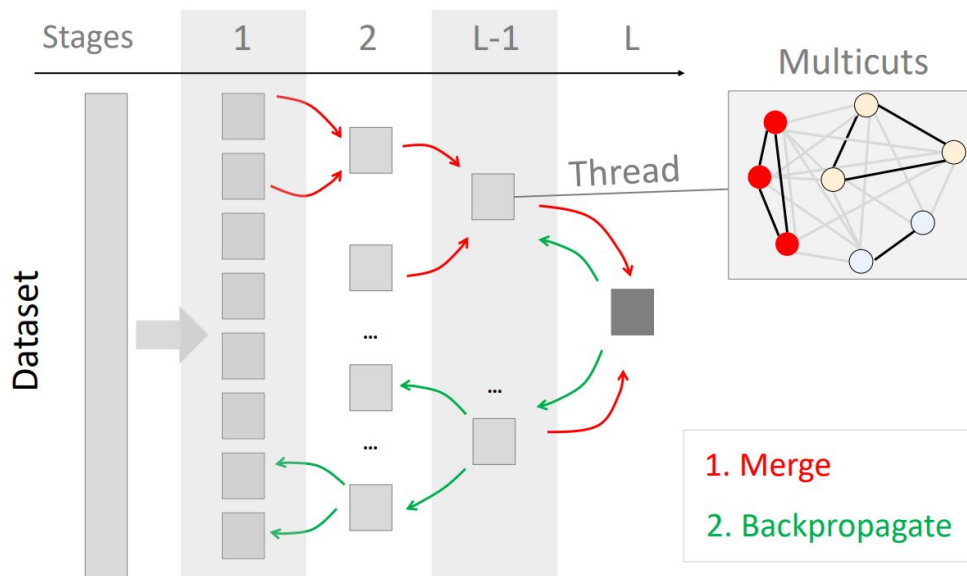


Fig. 3.1 Overview of MSM: Multi-Stage Multicuts.

Our Multi-Stage Multicuts (MSM) procedure: at stage 1, we evenly divide a given dataset into small, disjoint sets and distribute this to individual computing units to solve the *Minimum Cost Multicut Problem* concurrently. The results are forwarded to the next units in the following stage, where the intermediate results are being merged (red arrow). This process is repeated until the final stage  $L$  is reached. The merge result is sent back to all previous computing units via backpropagation (green arrow).



This Chapter is structured as follows: formal definitions of MSM is first described in Chapter 3.3.1. In Chapter 3.3.2, we briefly recap the *Minimum Cost Multicut Problem*, which we already formally introduced previously in Chapter 1.3.3. Specifically, we emphasize how the number of edges can affect the algorithm speed. We discuss the challenge of the *Minimum Cost Multicut Problem* in Chapter 3.3.2 on a single CPU thread and how to overcome this issue. In Chapter 3.3.3, we introduce our proposed algorithm for solving large graphs on multiple CPU threads and the theoretical proof for the speedup and memory complexity is provided in 3.3.4.

### 3.3.1 Notation and Methods

Given a set of available workers  $|S_k|$  at stage  $k$ , each individual CPU thread is represented as  $s_{i,k} \in S_k$  with  $\forall i \in \mathbb{N} : 1 \leq i \leq w$ . At stage  $S_1$ , the number of workers is set to  $|S_1| = w$ . Furthermore, a defined number of stage  $L = k$  is given, where the number of workers are reduced successively:

$$|S_k| < |S_{k-1}| \quad (3.1)$$

A dataset  $X$  with a total size of  $|X|$  is randomly divided into disjoint sets across the workers  $|S_1|$ . At stage one, each worker  $s_{i,1} \in S_1$  holds  $n = \frac{|X|}{|S_1|}$  samples of  $X$ . Since MSM is based on pairwise comparison of data, the permutation of the splits as well as the distribution of the classes within a batch will not affect the clustering performance in noticeable way (we show this in our experiments).

In order to obtain the cluster labels  $y_{i,k}$ , each individual worker solves an instance of *Minimum Cost Multicut Problem* with respect to the graph  $G$ , which is explained previously in subsection 1.3.3. Each cluster label  $y_{i,k}$  is then forwarded during the next stage to  $S_{k+1}$  to compute the joint solution (red arrow). Details are explained in 3.3.3. This process is repeated until the final stage  $L$  is reached. The results of  $S_{L-1}$  are forwarded to one single worker  $|S_L| = 1$ . Cluster labels are then sent back to all other workers via backpropagation (green arrow).

The following section is divided into three parts. Section 3.3.2 describes the image clustering task using *Multicuts* with different graph sizes. Then, our algorithm is introduced in Section 3.3.3. The theoretical proof of our proposed algorithm is presented in Section 3.3.4.

### 3.3.2 Image Clustering with Multicuts

The advantage of *Multicuts* clustering is the fact that pairwise comparisons are performed. In contrast, centroid-based clustering methods such as *k-means* assumes data of same clusters to be distributed evenly around its center or have the same density. However, such assumptions do not perform well on real-world face clustering [45]. We assume that each pair of nodes of the undirected graph  $G$  has at most one edge with a cost  $c$ . Consequently, a complete graph with  $n$  nodes has in total  $|E| = \frac{n(n-1)}{2}$  edges.

#### Sparsity

In context of image clustering using *Multicuts*, it is desired to have as many edges as possible during the graph creation (e.g. complete) as more pairs of images are being compared with. Figure 3.2 a) depicts the final cluster accuracy (in blue) on the CIFAR-10 [93] Test dataset as well as the file size (in red) against the sparsity of the graph: 1.0 represents a complete graph while 0.5 means that half of the edges are being randomly removed in order to save memory. This experiment uses a single worker for solving the the *Minimum Cost Multicut Problem* as done in Section 2.4.1: a convolutional neural network model is trained using the Triplet Loss\_3 (Equation 2.3) and the embedding features of images are extracted and clustered. The experiments were conducted over five runs. We observe the clustering accuracy converges to a value at average 80%, which is consistent with the previous reported numbers. Since this is the case, it is safe to assume that at sparsity=1.0 (e.g. complete graph), the highest clustering accuracy is reached. When using very few edges only (sparsity at around 0.01), performance drops significantly. The red box depicts an area with higher standard deviation. At the same time, the size of the graph file is increasing. Figure 3.2 b) illustrates the runtime in seconds and file size of a complete graph (e.g. sparsity=1.0) for different dataset size. One can observe that the larger the dataset size, the longer the runtime becomes for solving the problem and more memory it consumes.

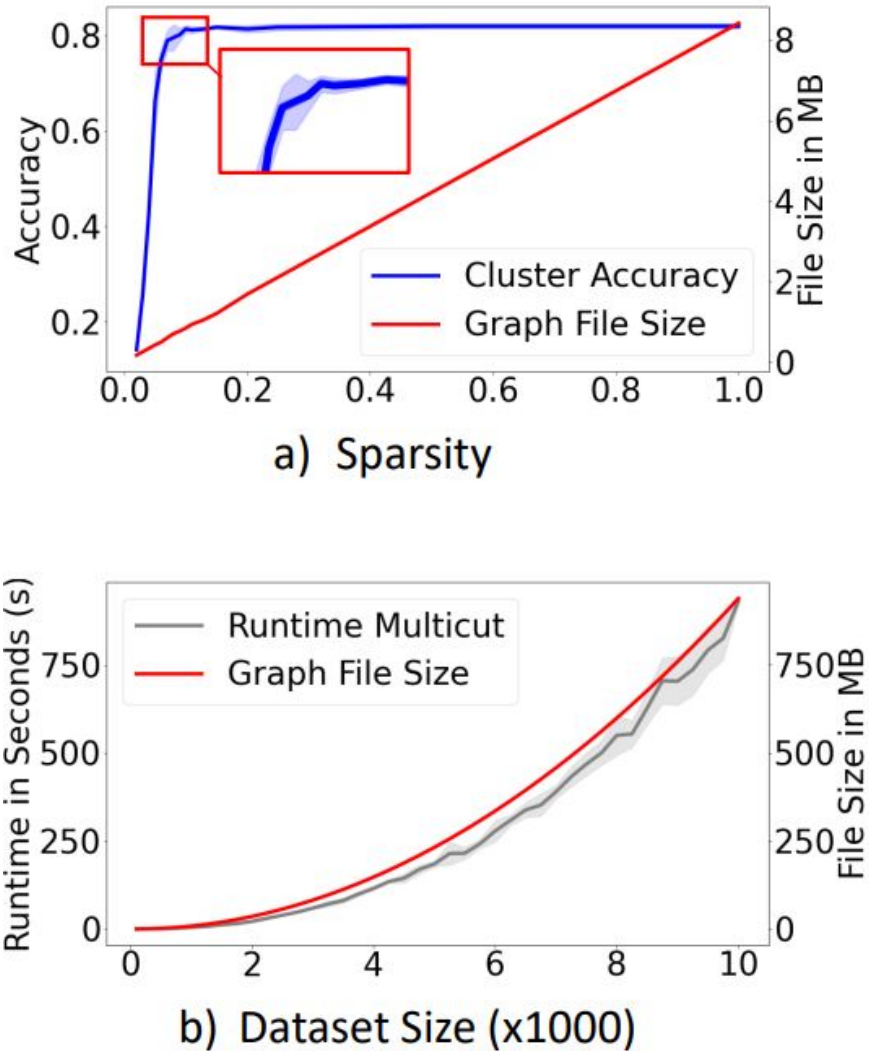


Fig. 3.2 Multicuts on Single CPU Thread.

*Minimum Cost Multicuts* for image clustering task on a single worker. In a), a subset of CIFAR10 (1000 images) is used to evaluate the clustering performance as well as the graph size for different sparsity setups. A sparsity of 0.5 means that half of the edges are removed during the construction of  $G$ . In b), a complete graph (e.g. sparsity=1.0) is being evaluated for different dataset size. Graph file is stored uncompressed as an edge list.

### 3.3.3 MSM: Multi-Stage Multicuts

Our approach is based on the idea to use multiple CPU threads (worker) on a shared memory system to solve the image clustering problem. The dataset  $X$  is decomposed into disjoint sets with equal size, reducing the size of graph  $G$  in order to solve the minimum cost multicut problem in a reasonable amount of time. Figure 3.2 b) shows the relation between runtime and number of data (e.g. nodes in  $G$ ).

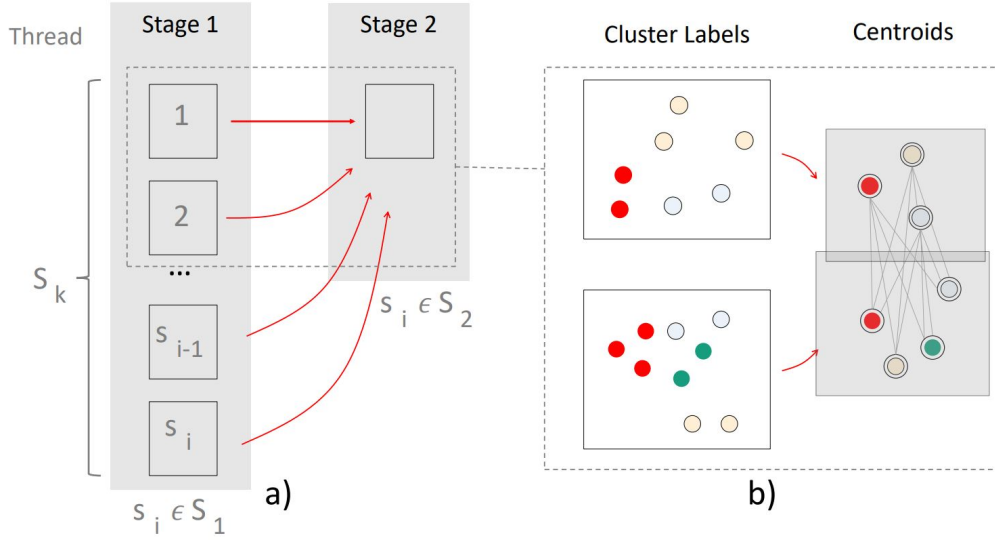
At the initial stage  $k = 1$ , each worker of  $s_{i,1} \in |S_1|$  with  $|S_1| = w$  carries  $n = \frac{|X|}{w}$  samples of  $|X|$ , which represents the number of nodes in  $G$ . The higher the available resource (number of workers  $w$ ), the faster MSM processes at  $S_1$  since  $n$  gets smaller. The output class labels  $y_{i,k} \in Y_k$  for each sample are forwarded to stage  $S_{k+1}$  and the joint solution is computed by worker  $s_{i,k+1} \in S_{k+1}$ .

#### Merge Process

An illustration is shown in Figure 3.3 a): the results of two disjoint sets are merged together. At stage  $S_2$ , the goal is to find the same cluster in workers from the previous stage  $s_{i,1} \in S_1$  and  $s_{i+1,1} \in S_1$ . Figure 3.3 b) depicts the merge process: the centroid of the individual clusters using their cluster labels is computed. The centroids are used as new nodes, allowing the next stage to form a graph of centroids and the clustering process is repeated until the last stage is reached. The size of the graph (thus the runtime) of the next stage mainly depends on the number of output clusters  $y_i$  from previous stage as well as the number of incoming disjoint sets (e.g. number of workers), that are being merged. The number of samples for  $s_{i,k} \in S_k$  with  $k > 1$  is defined as follows:

$$n_{i,k} = \sum_{j \in I_i^k} \text{unique}(y_{j,k-1}) \quad (3.2)$$

where  $I_i^k$  represents the set of all workers in the prior stage that are joint together to worker  $s_{i,k}$ . The full MSM algorithm is described in Algorithm 2

Fig. 3.3 MSM with Two-Stages ( $L = 2$ ).

Combining results of disjoint sets from two different workers: a) two sets are being merged for the next stage. In b), the centroids of different clusters from previous stage are being used to form a new minimum cost Multicut problem.

---

**Algorithm 2** MSM: Multi-Stage Multicuts
 

---

- 1: **Input:** data  $X$ , number of workers  $w$ , Stages  $L$
  - 2: Set  $k = 1$
  - 3: Split data  $X$  to worker  $s_i \in S_1$
  - 4: Extract embedding  $z_n$  via deep neural network
  - 5: Obtain clusters  $y_n$  based on  $z_n$  using Multicuts
  - 6: **while** Final Stage  $L$  not reached **do**
  - 7:    $S_k$  sends  $y_n$  and  $z_n$  to  $S_{k+1}$
  - 8:    $S_{k+1}$  compute centroids:  $\tilde{z}_n$
  - 9:   Obtain new clusters  $y_n$  based on  $\tilde{z}_n$  using Multicuts
  - 10:    $k = k + 1$
  - 11: **end while**
  - 12: **for** All stages **do**
  - 13:   Backpropagate cluster labels  $y_n$
  - 14:    $y_n = \text{Translate } y_n$
  - 15: **end for**
  - 16: **Output:**  $y_n$
-

### 3.3.4 Theoretical Proof

Here, we formally show the theoretical speed up of MSM. Given a set of  $n$  nodes  $V$ , we split  $V$  into  $k$  disjoint subsets  $V_1, V_2, \dots, V_k$ . ( $V_i \cap V_j = \emptyset$  for  $i \neq j$  and  $\bigcup_{i=1}^k V_i = V$ )

We denote the resulting graph that is built from set  $V_i$  as  $G_i(V_i, E_i)$ . The aim is to minimize the minimum cost multicut problem with respect to  $G_i$  as defined in equation 1.8:

$$\min_{y \in \{0,1\}^E} \sum_{e \in E_i} c_e y_e \quad (3.3)$$

$$z_i^{(k+1)} = f(V_i^{(k)}, y_i^{(k)}) \quad (3.4)$$

$$V_l^{(k+1)} = \bigcup_{i \in I_l^{(k+1)}} z_i^{(k+1)} \quad (3.5)$$

The function  $f$  computes the centroids of the nodes in  $V_i^{(k)}$ .  $I_l^{(k+1)}$  is the set of indices of workers in stage  $k$  whose results are combined for worker  $l$  in stage  $k+1$ .

The function  $h^{(k+1)}(C_l^{(k)}) = y_l^{(k+1)}$  relates the indices found by the Multicuts with index in  $I_l^{(k+1)}$  to the indices found by the Multicut of worker  $l$  in the next stage. The set of indices that are combined for worker  $l$  in the stage  $k+1$  is  $C_l^{(k)} = \{y_i^{(k)} | i \in I_l^{(k+1)}\}$ . This is done in order to backpropagate the indices found in the last stage to the input data with  $h^{-1}(y_l^{(k+1)})$ . In our case the function  $h$  is a lookup table that keeps track of all the indices at a given stage.

For a fully connected Multicut with number of nodes  $|V| = n$  the algorithm scales in  $\mathcal{O}(\frac{n^2(n-1)}{2})$  [148]. For  $L$  stages with  $S_k$  workers at stage  $k$  one obtains the following complexity

$$\sum_{k=1}^L \underbrace{\sum_{i=1}^{S_k} \frac{n_{k,i}^2 (n_{k,i} - 1)}{2}}_{R_k} \quad (3.6)$$

where  $n_{k,i}$  is the number of nodes for worker  $i$  in stage  $k$ .

Let  $g_k = \sum_{k=1}^L n_{k,i}$  be the number of nodes of all workers in stage k. Note that  $g_1 = n$ , the size of the original dataset.

**Assumption 1** In order to find an upper bound we assume a balanced data allocation for each worker in each stage, therefore

$$n_{i,k} = \frac{g_k}{S_k} \quad (3.7)$$

The total number of nodes in stage k,  $g_k$ , does not grow in size in the next round,  $g_{k+1} \leq g_k$ .

The following relationship holds

$$g_k = p_{k-1} g_{k-1} \quad (3.8)$$

with  $0 < p_k \leq 1$ .

**Assumption 2** The series of  $(p_k)_{1 \leq k}$  is monotonically increasing and satisfies the following inequality

$$p_k \geq p_{k-1}^{\frac{k-1}{k}} \quad (3.9)$$

This is in line with what is being observed in the case of continually decreasing workers. The workers in the first stage are able to cluster most of the dataset, resulting in a much smaller p than what is achieved in later stages.

**Assumption 3** The number of workers  $S_k$  is decreasing with each stage k,  $S_{k+1} < S_k$ .

In stage k one has a complexity of

$$\begin{aligned}
R_k &\stackrel{(3.7)}{=} \sum_{i=1}^{S_k} \frac{\left(\frac{g_k}{S_k}\right)^2 \left(\frac{g_k}{S_k} - 1\right)}{2} \\
&= S_k \frac{\left(\frac{g_k}{S_k}\right)^2 \left(\frac{g_k}{S_k} - 1\right)}{2}
\end{aligned} \tag{3.10}$$

Since  $1 \leq S_k$  we have

$$S_k \frac{\left(\frac{g_k}{S_k}\right)^2 \left(\frac{g_k}{S_k} - 1\right)}{2} \leq \frac{1}{S_k^2} \frac{g_k^2 (g_k - 1)}{2} \tag{3.11}$$

Given  $g_1 = n$  one can observe that

$$g_k = n \prod_{i=1}^{k-1} p_i \leq n p_{k-1}^{k-1} \tag{3.12}$$

With this relationship we can simplify expression (3.11)

$$\frac{1}{S_k^2} \frac{g_k^2 (g_k - 1)}{2} \leq \frac{1}{S_k^2} \frac{(n p_{k-1}^{k-1})^2 (n p_{k-1}^{k-1} - 1)}{2} \tag{3.13}$$

since  $p_i \leq 1$  we obtain for stage k the following expression

$$R_k \leq \frac{1}{S_k^2} \frac{(n p_{k-1}^{k-1})^2 (n p_{k-1}^{k-1} - 1)}{2} \leq \frac{p_{k-1}^{3(k-1)} n^2 (n - 1)}{S_k^2} \tag{3.14}$$

If we set this back into expression (3.6), we obtain for an L-stage Multicut

$$\sum_{k=1}^L R_k \leq \frac{n^2 (n - 1)}{2} \sum_{k=1}^L \frac{p_{k-1}^{3(k-1)}}{S_k^2} \tag{3.15}$$

Note that we have  $S_L = 1$  and  $\frac{1}{S_k} \leq 1$ . Since  $S_{k+1} < S_k$  we can write



$$\frac{n^2(n-1)}{2} \sum_{k=1}^L \frac{P_{k-1}^{3(k-1)}}{S_k^2} \leq \frac{n^2(n-1)}{2} \sum_{k=1}^L \frac{P_{k-1}^{3(k-1)}}{(L+1-k)^2} \quad (3.16)$$

Since we have  $p_0 = 1$ , the first term is split off the sum

$$\frac{n^2(n-1)}{2} \sum_{k=1}^L \frac{p_{k-1}^{3(k-1)}}{S_k^2} = \frac{n^2(n-1)}{2} \left( \underbrace{\sum_{k=2}^L \frac{p_{k-1}^{3(k-1)}}{(L+1-k)^2}}_{(I)} + \frac{1}{L^2} \right) \quad (3.17)$$

Now focus on term (I) from equation (3.17):

$$\sum_{k=2}^L \frac{p_{k-1}^{3(k-1)}}{(L+1-k)^2} = \sum_{k=1}^{L-1} \frac{p_k^{3k}}{(L-k)^2} \quad (3.18)$$

Denote  $l_k = p_k^{3k}$ . Since  $p_k$  is monotonically increasing, with  $p_k$  satisfying (3.9), we also have  $l_k$  monotonically increasing. Therefore we have that the maximum of  $(l_k)_{1 \leq k \leq L-1}$  is  $l_{L-1} = \max(\{l_k : k = 1, \dots, L-1\})$ . Thus we have

$$\sum_{k=1}^{L-1} \frac{l_k}{(L-k)^2} < l_{L-1} \sum_{k=1}^{L-1} \frac{1}{(L-k)^2} \quad (3.19)$$

One can rearrange the term in the sum so that one obtains

$$l_{L-1} \sum_{k=1}^{L-1} \frac{1}{(L-k)^2} = p_{L-1}^{3(L-1)} \sum_{k=1}^{L-1} \frac{1}{k^2} \quad (3.20)$$

We have that

$$\sum_{k=1}^{L-1} \frac{1}{k^2} = \frac{\pi^2}{6} - \psi^{(1)}(L) \quad (3.21)$$

Here  $\psi^{(1)}(x)$  represents the first derivative of the digamma function.

$$\psi^{(1)}(x) = \sum_{k=0}^{\infty} \frac{1}{(k+x)^2} > \frac{1}{x^2} \quad (3.22)$$

Using the inequality in (3.22) this allows us to put an upper bound on expression (3.21)

$$\sum_{k=1}^{L-1} \frac{1}{k^2} = \frac{\pi^2}{6} - \psi^{(1)}(L) < \frac{\pi^2}{6} - \frac{1}{L^2} \quad (3.23)$$

Putting this result back into equation (3.20)

$$\underbrace{p_{L-1}^{3(L-1)} \sum_{k=1}^{L-1} \frac{1}{k^2}}_{(I)} < p_{L-1}^{3(L-1)} \left( \frac{\pi^2}{6} - \frac{1}{L^2} \right) \quad (3.24)$$

Putting (3.24) into (3.17) and (3.17) into (3.15) we obtain the following upper bound for the complexity for an L-stage Multicut under Assumptions 1-3:

$$\sum_{k=1}^L \sum_{i=1}^{S_k} \frac{n_{k,i}^2 (n_{k,i} - 1)}{2} < \frac{n^2(n-1)}{2} \underbrace{\left( p_{L-1}^{3(L-1)} \left( \frac{\pi^2}{6} - \frac{1}{L^2} \right) + \frac{1}{L^2} \right)}_{h(p_{L-1}, L)} \quad (3.25)$$

Inequality (3.25) gives an upper bound to the complexity of our algorithm. It also relates the complexity of the L-stage Multicut to the regular approach that has complexity of  $\frac{n^2(n-1)}{2}$ . Whether the L-stage Multicut has lower computational complexity depends on the function  $h(p_{L-1}, L)$ . If  $h(p_{L-1}, L) \leq 1$  the L-stage Multicut has lower complexity than a single Multicut on the entire dataset. The function  $h(p_{L-1}, L)$  can be seen in Figure 3.4 for different numbers of stages  $L$ .

The memory complexity of the naive Multicut approach on a dataset of size  $n$  scales with a complexity of  $\frac{n^2(n-1)}{2}$ .

On the other hand, in the MSM approach, the memory scales at most like  $\frac{n^2(n-1)}{2} \max_{k \in [1, L]} \frac{(\prod_{i=1}^{k-1} p_k)^3}{s_k^2}$ . This allows a reduction in the very first stage by  $\frac{1}{s_1^2}$ . In later stages  $p$  is much smaller than one, thus the memory requirement on different workers are still much smaller than the naive approach. This allows to train much bigger datasets than what has previously been possible.

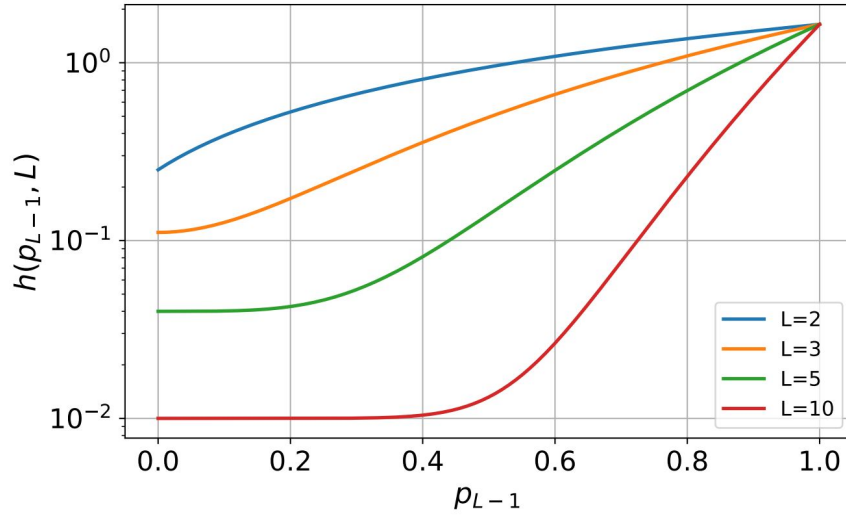


Fig. 3.4 Visualization of the function  $h(p_{L-1}, L)$  for different stages  $L$ .

### 3.4 Experiments

In this section, we evaluate MSM algorithm on various datasets (Table 3.1) for image clustering tasks. They vary in terms of dataset size and number of unique classes. Attribute *haircolors* of CelebA [108] dataset is selected for this clustering tasks, which consists of small number of class. On the other hand, the train and test dataset is swapped in order to evaluate a significant large graph with more than 100k nodes. While CIFAR-10 and CIFAR-100 are similar dataset, they vary in number of unique classes. In other words, there are less samples for each class and more unique classes in CIFAR-100, which makes the learning more challenging.

## Runtime

We measure the runtime in seconds as well as the cluster accuracy, finding the best matches between the predicted cluster and true label as described previously in Section 2.4.1. All experiments are run on the same hardware and setup with multiple runs. If not specified, we report the mean number over five runs as well as its standard deviation. The features are extracted from models based on weakly-supervised learning. We specify this in the according subsection of each experiment. The aim is to show, that, for a given problem, MSM can speedup clustering using *Multicuts* clustering method without any noticeable performance drop. The rest of this Chapter explains the experiments on the datasets: first, a two-stage approach on CIFAR-10 dataset is evaluated in Chapter 3.4.1. Then, we compare different sparsity setup and stages on a significant larger dataset in the following Chapters. Lastly in Chapter 3.4.5, we evaluate a 4-Stage approach on CIFAR-100 dataset.

DATASET	TRAIN SIZE	TEST SIZE	# CLASSES
CELEBA [108]	12.293	101.642	5
CIFAR10 [93]	50.000	10.000	10
CIFAR100 [93]	50.000	10.000	100

Table 3.1 MSM: Datasets used for image clustering tasks.

### 3.4.1 Two-Stage Approach on CIFAR-10 Dataset

In this experiment, we evaluate our MSM on CIFAR-10 based on a two-stage approach ( $L = 2$ ), meaning that we split the dataset across  $S_1$  workers during the first stage as shown in Figure 3.3 a). Each worker  $s_i$  solves the *Minimum Cost Multicut Problem* on a smaller dataset and the results (cluster labels) are forwarded to the next stage. All intermediate results are merged in one worker ( $S_2 = 1$ ) to compute the final results. An example of the merge process with two workers from previous stage is illustrated in Figure 3.3 b).

## Embeddings

We train a deep neural network with our proposed Triplet Loss (Equation 2.3) for multicuts. The threshold for cuts and joins are derived from the training parameters. We also compute all the embeddings offline. This way, the GPUs are not required to extract the embeddings during the clustering task.

## Results

We are comparing this method in terms of runtime as well as clustering accuracy on the full test dataset. As shown previously in Figure 3.2, one single worker takes on average  $930.64 \pm 8.44$  seconds to solve the problem. Increasing the number of workers by 2 using MSM ( $S_1 = 2$ ) reduces the runtime of the problem by more than half ( $361.29 \pm 16.4587$  seconds). While the runtime is decreasing, the cluster accuracy remains the same, which is  $80.54\% \pm 0.36\%$ , as shown in Figure 3.5. Furthermore, we observe that additional computation is required for the merge process (shown as gray dashed line). The more workers are used, the more time it takes for combining the results. However, the major time are spent on the clustering task for each individual worker (shown as red line).

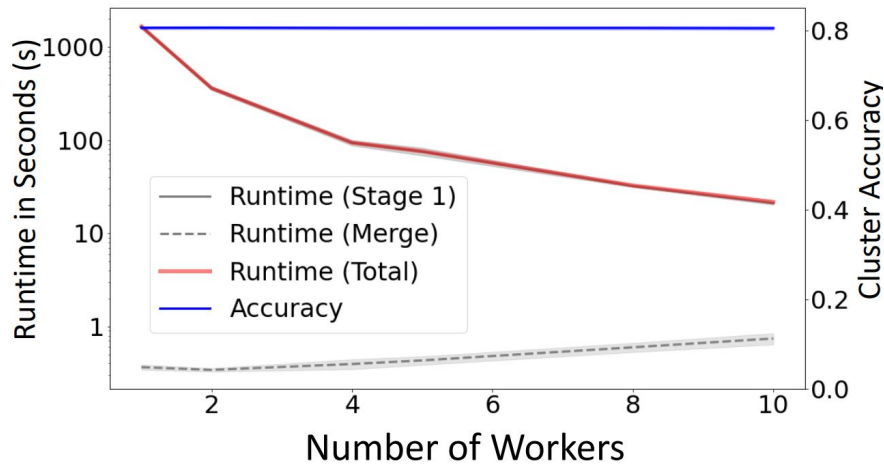


Fig. 3.5 MSM on CIFAR-10: Runtime and Cluster Accuracy.

Runtime and Cluster Accuracy on CIFAR-10 test dataset using different number of workers in first stage ( $S_1$ ). The Multicut runtime is reduced significantly when more workers are utilized. The runtime for merge (e.g. second stage) remains very low and the cluster accuracy is consistently around 80%.

### 3.4.2 Sparsity is Influencing the Runtime on Merge

In this experiment, we evaluate MSM on CelebA dataset [108]. We use the attribute *haircolors* for the image clustering task, which consists of five following classes: *bald*, *gray\_hair*, *blond\_hair*, *black\_hair* and *brown\_hair*.

#### Embeddings

We trained a deep neural network as in the previous experiment, using a modified Triplet Loss (Equation 2.3). Here we swapped the train and test (see Table 3.1). We show the performance of MSM on a larger image clustering problem. The top row of Figure 3.6 shows the performance of MSM on CelebA dataset for a two stage clustering  $L = 2$  (as in section 3.4.1). However, in this particular experiment, we enforce sparsity, where the number of total edges are reduced. Having less edges will speed up the algorithm but tends to produce more clusters thus decreasing the total cluster accuracy (as shown previously in Figure 3.2). We also use up to  $S_1 = 40$  workers for this image clustering task. The grey line depicts the runtime of each individual, concurrent worker for the clustering task via Multicuts while the dashed line shows the runtime for merging the results (e.g. the last stage  $|S_L|$ ). The sum of both (total runtime) is represented as red line.

#### Results

The runtime decreases when adding more workers to MSM. Moreover, only a small drop in clustering performance is observed (blue line). However, Figure 3.6 top left shows, that at roughly  $S_1 = 25$  workers, the runtime of merge process (dashed line) of MSM begins to overtake the runtime of each individual *Multicuts* process (gray line).

This effect is amplified by further removing more edges from the initial graph (sparsity=0.01) at  $S_1$ . This is illustrated in Figure 3.6 top right. Not only does the performance of the clustering task drop, but also the total runtime increases when using more workers. When increasing the sparsity of the graph (=overcluster), the merge process in  $S_2 = 1$  becomes the bottleneck and no performance gain is achieved but rather the opposite instead.

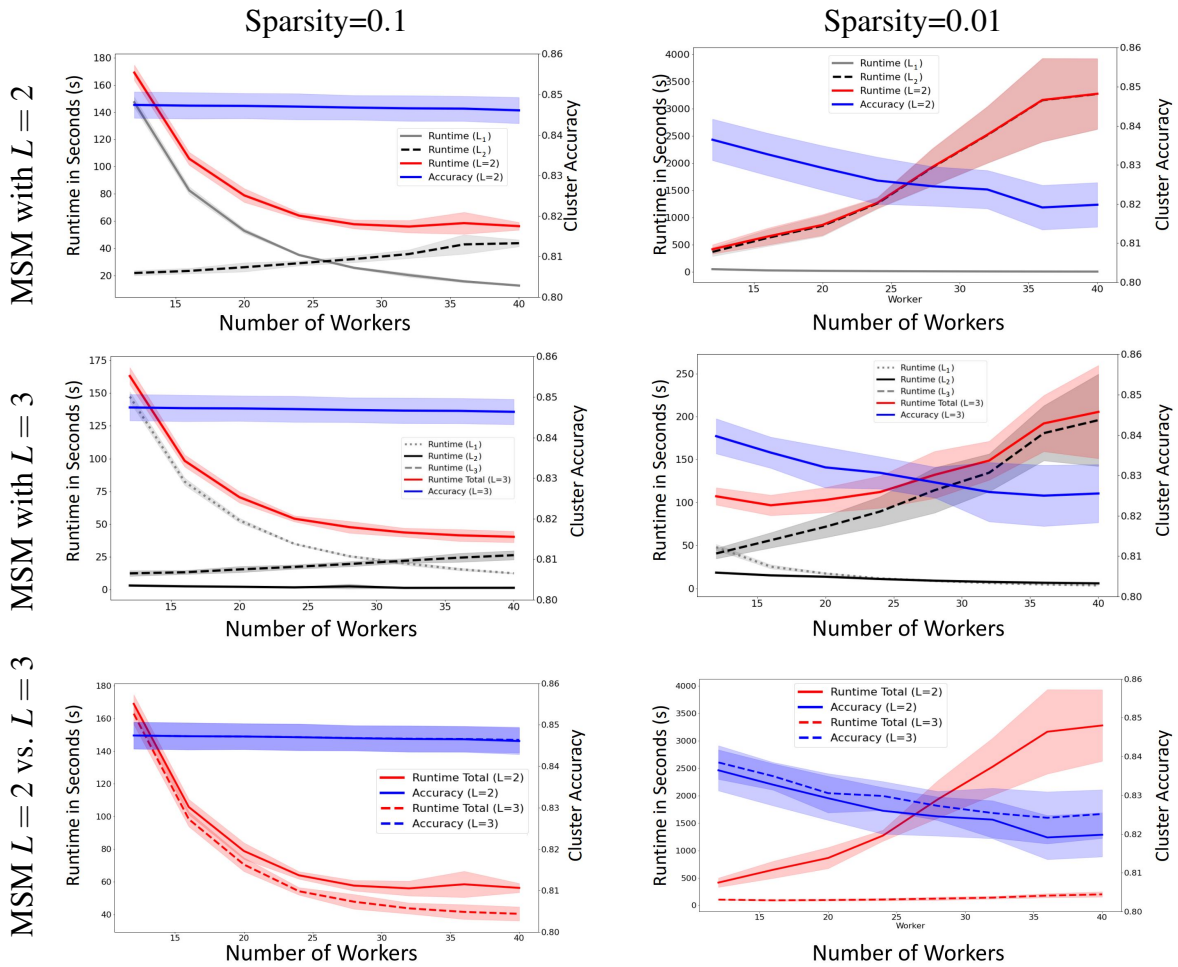


Fig. 3.6 MSM: Evaluation on CelebA Dataset.

Evaluation of MSM on CelebA Dataset:  $L = 2$  vs.  $L = 3$  using different (sparse) input graphs on CelebA Dataset. We compare the performance in terms of runtime (in seconds) and cluster accuracy. First row: MSM with two stages ( $L = 2$ ). Second row: MSM with three stages ( $L = 3$ ). Third row: comparison of  $L = 2$  and  $L = 3$ . The columns represent different sparse graph for the initial stage. When using a two stage approach ( $L = 2$ ) with graphs as depicted on first row, second column, adding more workers will increase the runtime significantly since we merge all results to one single worker in the last stage. We can avoid such *bottleneck* by adding an additional stage ( $L = 3$ ), which is shown in the second row, second column (be aware of the scaling).



### 3.4.3 MSM with 3-Stages on CelebA

When the output of  $S_k$  becomes too large, sending all the intermediate solutions to one single worker (e.g.  $S_{k+1} = 1$ ) for processing the final result is not beneficial for MSM as it creates a bottleneck. We therefore investigate the effects of MSM with  $L = 3$  stages on image clustering tasks. Specifically, we ran the same experiments as previously. On each intermediate stage, we reduce the number of workers by half. For instance if we set  $S_1 = 40$ , then  $S_2 = 20$  and  $S_3 = 1$ , respectively. This way, we successively reduce the number of data on each layer.

#### Results

Figure 3.6 second row illustrates the performance of MSM with  $L = 3$  stages. The total runtime slightly improves compared to MSM with  $L = 2$  stages. However, on second row right, we see a significant performance increase for  $L = 3$ . Similarly, we also observe a drop in cluster accuracy when increasing the number of workers. The last row shows the direct comparison between  $L = 2$  vs.  $L = 3$  stages MSM.

### 3.4.4 Qualitative Results

Figure 3.7 shows a TSNE [112] visualization of the embedding space of CeleA dataset and the clustering results of both methods. The ground truth is displayed in a) while b) and c) shows *k-means* and the *Minimum Cost Multicuts* with its clustering performance of 85.51% and 85.17%, respectively. While *k-means* outputs exactly give clusters, the *Minimum Cost Multicuts* is able to discover new clusters (in red). Interestingly, this new cluster is located between the class *bald* and *black\_hair*. Looking closer to some examples of the clusters, which are shown in 3.8, one can observe that it mostly contains person from certain ethnicity, e.g. *dark skin* and *bald*. Such cluster is not defined on this dataset, however, the *Multicuts* clustering is able to discover it.

On the other hand, the input parameter of *K-Means* algorithm was set to  $K = 5$ . Although this information is known from the dataset, the optimal number of  $K$  is verified in another experiment. Figure 3.9 shows the Elbow method for finding the optimal number of clusters on the dataset, which is  $K = 5$ . The experiment is repeated five times with different random seed on each run with a search range of  $1 < K < 15$ , which is shown in x-axis. Instead of showing the sum squared distance between the data point and its cluster center (y-axis), the cluster performance is evaluated, similar as done in [15]. However, in contrast to the authors method, which suggest an over-clustering, our experiment shows an optimal number for the dataset is  $k = 5$ , which is consistent with the class labels.

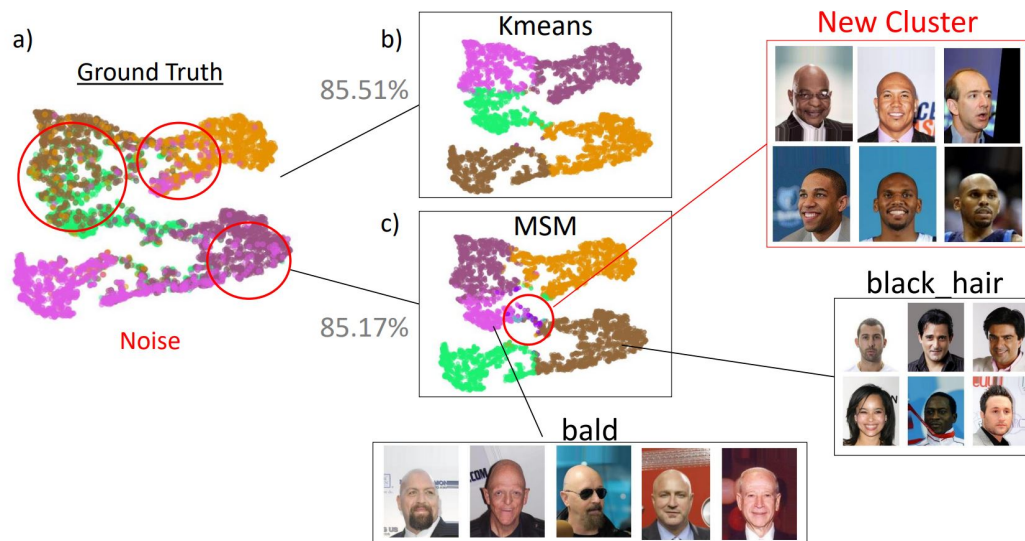


Fig. 3.7 MSM on CelebA dataset: Discovery of new Cluster.

While the dataset has 5 different classes, MSM is able to detect a new cluster.



Fig. 3.8 More Examples of new Clusters.

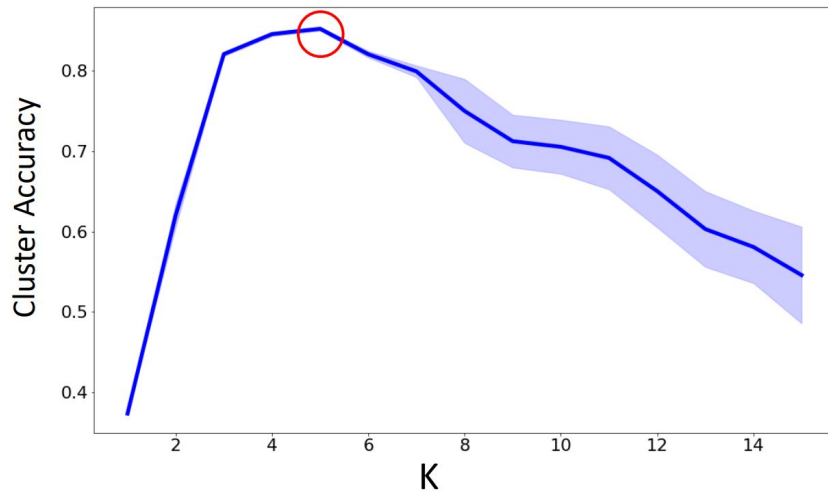


Fig. 3.9 Use ground truth in order to find the optimal  $k$  on CelebA dataset.

The optimal parameter for  $K$  for  $K$ -Means for the CelebA dataset is 5.

### 3.4.5 MSM with 4-Stages on CIFAR100

CIFAR100 Test dataset has in total 10,000 samples with 100 unique, balanced clusters. These 100 unique clusters are further grouped into 20 super clusters, for instance the class *trees* consists of  $tree = \{maple, oak, palm, pine, willow\}$  Figure 3.10 shows the cluster accuracy as well as the total runtime in seconds. We use the embedding from [89] and evaluated MSM for  $L = 3$  and  $L = 4$ .

#### Results

While the clustering accuracy remains very stable (average of  $75.7\% \pm 0.4\%$ ), we observe different runtimes for  $L3$  and  $L4$ . The fastest clustering solution ( $115.83s \pm 1.75s$ ) in this experiment is obtained with  $L = 4$  and 12 workers for the initial stage ( $S_1 = 12$ ), which is circled in back on Figure 3.10. Increasing the number of workers slows down MSM. The best clustering accuracy is  $75.7\%$  with 578 unique clusters in total.

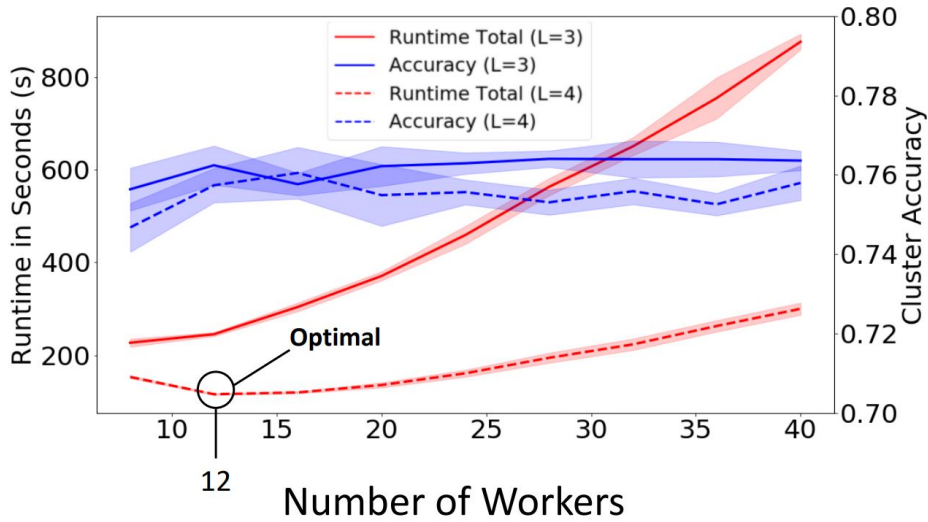


Fig. 3.10 MSM on CIFAR100 Dataset with  $L = 3$  vs.  $L = 4$ .

Evaluation of MSM on CIFAR100 dataset:  $L = 3$  vs.  $L = 4$ . We compare the performance in terms of runtime (in seconds) and cluster accuracy. The optimal runtime for  $L = 4$  is achieved with  $S_1 = 12$  workers.

Figure 3.11 shows the embedding space of the CIFAR100 dataset. Example of the cluster *tree* is displayed as well. Despite having a clustering performance of 75.7%, the distribution of the images across the clusters varies significantly. This is shown in Figure 3.12. While the test dataset contains 10,000 images with 100 class labels, it is expected to have 100 clusters with each 100 data since the dataset is balanced as well. However, the histogram shows a significant large number of single clusters (484) and a very few very large clusters with more than 100 images. These super large clusters contribute significantly to the error rate as they contain disproportionately many images. Thus the individual cluster accuracies are between 29% – 48%. On the other hand, the small clusters containing only a single image contribute very little to the error rate. For instance, 484 single clusters contributes only  $\frac{484}{10,000} = 4.84\%$  to the error rate.

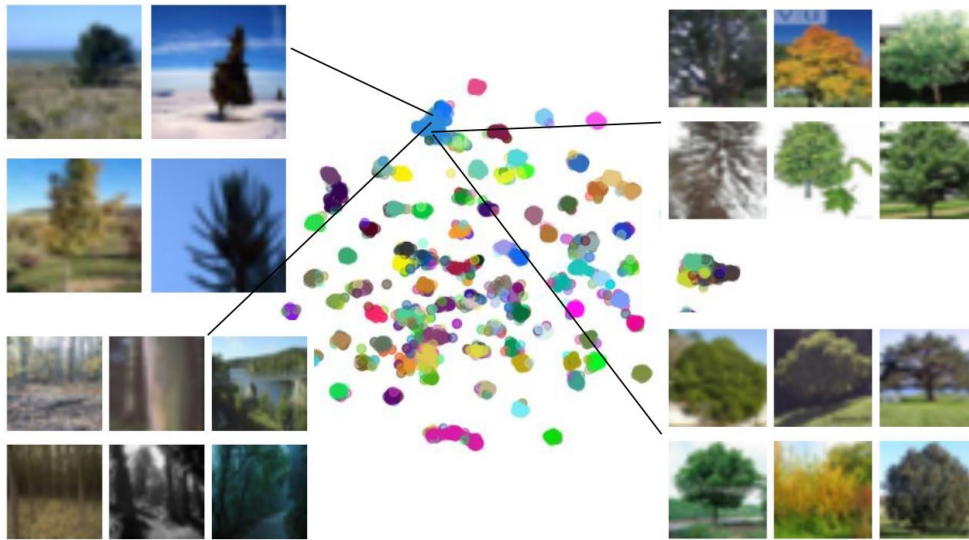


Fig. 3.11 MSM on CIFAR100 Dataset: example cluster *tree*.

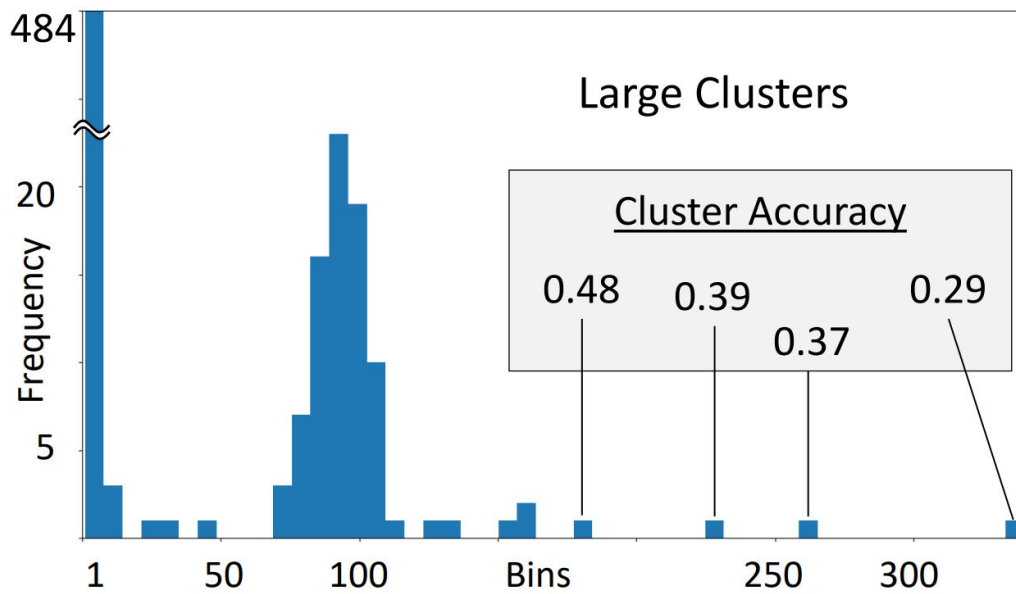


Fig. 3.12 MSM on CIFAR100 Dataset: Cluster Histogram.

The optimal distribution for CIFAR100 is around 100. The errors are contributed by some large clusters, which have low cluster accuracy.

## 3.5 Conclusion & Future Work

In this Chapter, we presented MSM, a Multi-Stage parallel *Multicuts* algorithm for clustering data based on a given graph of pair-wise distances between entities. This algorithm utilized the previously introduced Triplet Loss for a graph-based clustering. We showed that MSM provides good scalability on shared memory systems, while preserving the accuracy of the originally sequential *Multicuts* clustering. The theoretical proof for the proposed algorithm is provided as well.

The advantage of the *Minimum Cost Multicut Problem* is the fact, that it is possible to perform pairwise comparisons of data. This allow us to overcome certain assumptions of data distributions, such as a priors known numbers of clusters or priors on cluster density and shape as in popular methods like *k-means*. However, optimizing such a data requires significant amount of resources such as memory and computation time. We observe that for image clustering task, MSM provides significant speedup using without any noticeable drop in cluster accuracy. We reported the runtime in seconds on different datasets with different number of classes, for example MSM is clusters a dataset (CelebA with 5 classes) of 100.000 images in one minute. Our aim is to provide a *Multicuts* clustering approach that is capable of solving a large graph within a reasonable amount of time. We believe that *Multicuts* clustering approaches enable new possibilities for research towards unsupervised learning. Later in Chapter 5, we show that using MSM algorithm on large dataset such as ImageNet [134], we are able to not only cluster dataset with a large number of unique classes, but we are also able to utilize clustering as a performance measure for robustness of models.

### Limitation

MSM introduces different hyperparameters such as sparsity of the input graphs, number of available resources and number of stages. One interesting research direction for the future is to investigate a heuristic for automatically finding the optimal setup for a given problem and dataset. Currently, the number of stages is set manually.

# Chapter 4

## Application on Multiple Object Tracking

In this Chapter, we apply the clustering approach on Multiple Object Tracking (MOT) application. The content of this chapter is based on the approach, that we previously published in [62, 63]. While I contributed the idea, algorithmic implementation and experimental evaluation, the work has been supervised by Prof. Dr. Janis Keuper and Prof. Dr. Margret Keuper. The co-authors Amirhossein Kardoost and Franz-Josef Pfreundt in [62] contributed to insightful discussions.

### 4.1 Introduction

The objective of multiple object tracking is to find a trajectory for each individual object of interest in a given input video. Specific interest has been devoted to the specific task of multiple person tracking [169, 55, 148, 149, 109]. Most successful approaches follow the *Tracking-By-Detection* paradigm. First, an object (pedestrian) detector is used in order to retrieve the position of each person within each frame. Secondly, the output detections of same persons across video frames are associated over space and time in order to form unique trajectories. Since objects might get occluded during the video sequence or the detector might simply fail on some examples, successful approaches are usually based not solely on spatial but also on appearance cues. These are learned from annotated data, for example using Siamese networks for person re-identification [149]. Such association of data and re-identification of persons can be treated as a clustering task of detections.

## 4.2 Motivation

Supervised approaches for person re-identification require large amounts of sequence specific data in order to achieve good performance. For this reason, multiple object tracking benchmarks such as MOT [115] are providing a training sequence recorded in a sufficiently similar setting for every test sequence. The results of our experiments in table 4.1 confirm this dependency and show the high variance in the quality of supervised approaches, depending on the data used for training. The standard approach to solve this problem is to incorporate additional annotated training data. For example, [168, 37] showed that additional data is key to improving the overall tracking performance. Thus, publicly available, annotated training data currently seems not to be sufficient for training reliable person re-identification networks. Furthermore, recording and labeling sufficient data in a setting close to a final test scenario usually comes at a high price. Hence, the need for methods with a low amount of supervision becomes obvious and motivates us to propose a multiple object tracking method based on self-supervision. While self-supervised learning methods [90] have been successfully exploited in other vision tasks [124, 113, 53, 167, 102, 157], a direct application to tracking is non-trivial: Learning suitable object appearance metrics for object tracking in a self-supervised way is challenging since, compared to classical clustering problems, visual features of the same person may change over time due to pose and viewpoint changes and partial occlusion. Other issues, such as frequent and long range full occlusion or background noises, makes pedestrian tracking even more challenging.

In this Chapter, we propose an approach for learning appearance features for multiple object tracking without utilizing human annotations of the data. Our approach is based on two observations: I) given an image sequence, many data associations can be made reliably from pure spatio-temporal cues such as the intersection over union (IoU) of bounding boxes within one frame or between neighboring frames. II) Resulting tracklets, carry important information about the variation of an object's appearance over time, for example by changes of the pose or viewpoint. In our model, we cluster the initial data based on simple spatial cues using the recently successful minimum cost multicut approach [148]. The resulting clustering information is then injected into a convolutional AutoEncoder to enforce detections with the same, spatio-temporally determined label to be close to one-another in the latent space (see Fig.4.1). Thus, the resulting latent data representation is encoding not only the pure object appearance, but also the expected appearance variations within one object ID. Distances between such latent representations can serve to re-identify objects even after long temporal distances, where no reliable spatio-temporal cues could be extracted. We use the resulting information in the minimum cost lifted multicut framework, similar to the formulation of



Table 4.1 Relative Tracking Performance on Transfer Task.

Results for training with one training sequence using *GT annotations*<sup>1</sup> for the tracklet generation, and evaluating on other training sequences with different viewpoints and resolutions. This table shows the relative MOTA changes for non-matching sequences on MOT17, FRCNN in comparison to the baseline (bold). Columns represent the training sequence, rows the test sequence. The tracking performance heavily depends on the employed training data and can become unstable across domains.

		Train (Supervised)						
		MOT-02	MOT-04	MOT-05	MOT-09	MOT-10	MOT-11	MOT-13
Test	MOT-02	<b>100.0</b>	-0.3	-0.3	-19.2	-9.1	-12.5	-9.5
	MOT-04	0.0	<b>100.0</b>	0.0	-19.3	-4.9	-11.5	-4.9
	MOT-05	-0.6	-1.2	<b>100.0</b>	-3.2	-5.1	-3.4	-5.1
	MOT-09	-0.2	-0.4	-0.2	<b>100.0</b>	-2.9	-0.5	-2.5
	MOT-10	0.8	0.6	1.2	0.6	<b>100.0</b>	0.6	0.4
	MOT-11	0.0	-0.2	-0.2	0.2	-1.2	<b>100.0</b>	-1.4
	MOT-13	0.4	-1.1	-0.4	-3.8	-0.8	-2.7	<b>100.0</b>

Tang [149], whose method is based on Siamese networks trained in a fully supervised way. To summarize, our contributions are:

- We present an approach for multiple object tracking, including long range connections between objects, which is completely supervision-free in the sense that no human annotations of person IDs are employed.
- We propose to inject spatio-temporally derived information into convolutional AutoEncoder in order to produce a suitable data embedding space for multiple object tracking and compare this with the previously proposed Triplet Loss.
- We evaluate our approach on the challenging MOT17 benchmark and show competitive results without using training annotations.

The rest of the chapter is structured as follows: Section 4.3 discusses the related work on multiple object tracking. Our self-supervised approach on multiple object tracking is explained in Chapter 4.4. In Chapter 4.5, we show the tracking performance of the proposed method in the MOT Benchmark [115] and conclude in Section 4.6.

<sup>1</sup>Specifically, we mine GT tracklets from the detections with IoU > 0.5 with the GT as e.g. done in [98].

## 4.3 Related Work

### 4.3.1 Multiple Object Tracking

In Multiple Object Tracking according to the *Tracking by Detection* paradigm, the objective is to associate detections of individual persons, which may have spatial or temporal changes in the video. Thus re-identification over a long range remains a challenging task. Multiple object tracking by linking bounding box detections (*tracking by detection*) was studied, e.g., in [126, 4, 69, 3, 38, 169, 57, 150, 57, 55]. These works solve the combinatorial problem of linking detections over time via different formulations e.g. via integer linear programming [142, 158], MAP estimation [126], CRFs [96], continuous optimization [4] or dominant sets [151]. In such approaches, the pre-grouping of detections into tracklets or non-maximum suppression are commonly used to reduce the computational costs [69, 160, 3, 38, 169, 161, 57, 150]. For example Zamir et al. [169] use generalized minimum clique graphs to generate tracklets as well as the final object trajectories. Non-maximum suppression also plays a crucial role in disjoint path formulations, such as [126, 158, 17]. In the work of Tang et al. [148], local pairwise features based on DeepMatching are used to solve a multicut problem. The affinity measure is invariant to camera motion and thus makes it reliable for short term occlusions. An extension of this work is found in [149], where additional long range information is included. By introducing a lifted edge in the graph, an improvement of person re-identification has been achieved. Similarly, [65] uses lifted edges as an extension to the disjoint path problem. [12] exploits the tracking formulation using a Message Passing Networks (MPNs). In [85], low-level point trajectories and the detections are combined to jointly solve a co-clustering problem, where dependencies are established between the low-level points and the detections. Henschel et al. [56] solves the multiple object tracking problem by incorporating additional head detection to the full body detection while in [58], they use a body and joint detector to improve the quality of the provided noisy detections from the benchmark. Other works that treat Multiple Object Tracking as a graph-based problem can be found in [55], [84, 86, 96] and [169]. In contrast, [111] introduces a tracklet-to-tracklet method based on a combination of Deep Neural Networks, called *Deep Siamese Bi-GRU*. The visual appearance of detections are extracted with CNNs and RNNs in order to generate a tracklet of individuals. These tracklets are then split and re-connected such that occluded persons are correctly re-identified. The framework uses spatial and temporal information from the detector to associate the tracklets. The approach in [9] exploits the bounding box information by learning from detectors first and combined with a re-identification model trained on a siamese network. While the state of the art approaches in MOT17 Challenge are all based on supervised learning [56, 87, 139, 18], there are similar

works in [104, 110], which attempt to solve person re-identification (ReID) problems in an unsupervised manner.

### 4.3.2 Self-supervised learning

Self-supervised learning aims to generate pseudo labels automatically from a pretext task, and then employs these labels to train and solve for the actual downstream task. This is especially useful when no labeled data is available. Thus self-supervised approaches can be applied to many specific real-world problems. An extensive review of recent methods is presented in [77]. For instance [124] uses a motion-based approach to obtain labels to train a convolutional neural network for semantic segmentation problems. Another work on self-supervision based on motion can be found in [113] The idea of Doersch et al. [30] is to predict the position of eight spatial configurations given an image pair. In [125] semantic inpainting task is solved using a context encoder to predict missing pixels of an image. Hendrycks et al. [53] use a self-supervised method to improve the robustness of deep learning models. Lee et al. [102] propose an approach to improve object detection by recycling the bounding box labels while Ye et al. [167] use a progressive latent model to learn a customized detector based on spatio-temporal proposals.

## 4.4 Method

The proposed approach is based on the idea to learn, from simple spatial data associations between object detections in image sequences, which appearance variations are to be expected within one object for the task of multiple object tracking. An overview of our workflow implementing this idea is given in Fig. 4.1.

### Step 1

The object detection bounding boxes are extracted along with their spatial information such that spatial correspondences between detections in neighboring frames can be computed. Based on these simple spatial associations, detections can be grouped into tracklets in order to obtain cluster labels using clustering approaches such as correlation clustering as introduced in 1.3.3.

### Step 2

A convolutional AutoEncoder is trained to learn the visual features of detections. The objective is to learn a latent space representation which can serve to match the same object in different video frames. Thus, the information about spatial cluster labels from the first stage is used as the centroid of latent features. Distances between latent representations of data samples and their centroids are minimized in the convolutional AutoEncoder using a clustering loss.

Lastly, the data are transformed into the latent space of the trained AutoEncoder to extract pairwise appearance distances which are expected to encode the desired invariances. Such pairwise appearance distances are used to not only provide additional grouping information between nearby detections, but also for detections with long temporal distance. The final detection grouping is computed using minimum cost lifted multicuts [84].

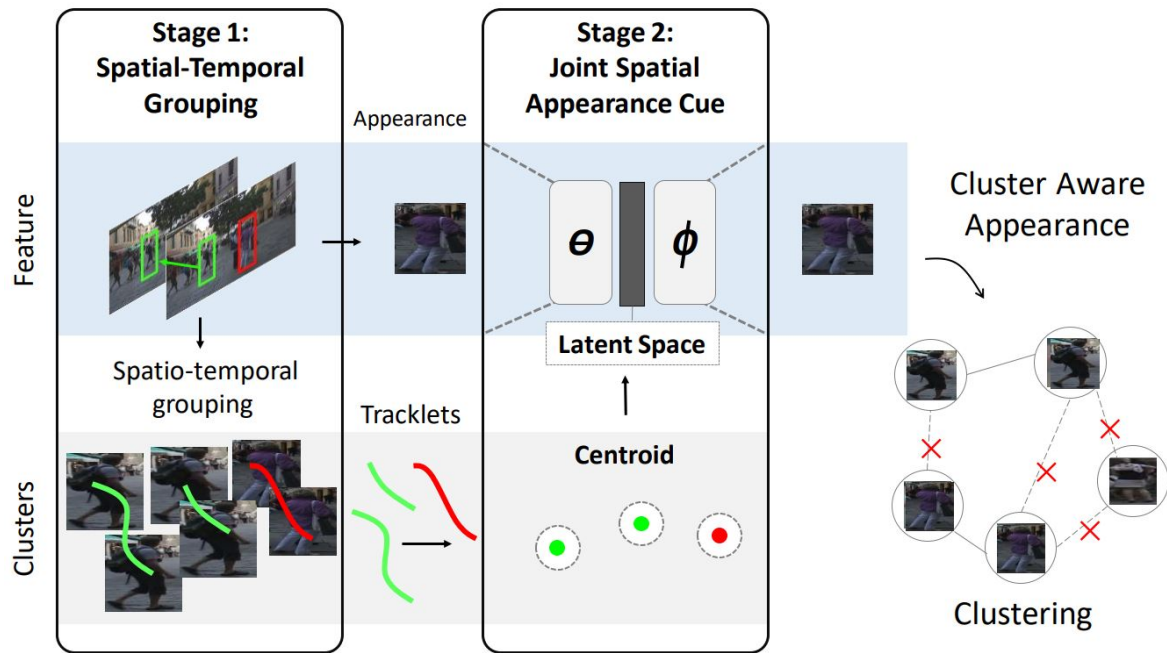


Fig. 4.1 Multiple Object Tracking in Two Steps.

Summary of our approach in two steps: 1. First, weak cluster labels (tracklets) are obtained from spatio-temporal vicinity using *Minimum Cost Multicuts* [23]. 2. Then, visual features are learned by an AutoEncoder, with an additional data association loss within the tracklets. The AutoEncoder provides a stable appearance embedding while the additional loss forces detections within one tracklet to have similar embeddings. This facilitates to extract affinities between detections to compute the final tracking with re-identification using lifted multicuts [149].

This section is divided into three subsections: Section 4.4.1 describes the *Minimum Cost (Lifted) Multicuts* approach employed for obtaining the initial spatial cluster labels (e.g. tracklets), as well as for the generation of the final tracking result. Section 4.4.2 describes the feature learning process using a convolutional AutoEncoder and cluster labels, and section 4.4.4 describes the computation of the joint spatial and appearance metrics used in the final data association step within the minimum cost *Lifted Multicuts* framework.

### 4.4.1 Multicut Formulation

We follow Tang [149] and phrase the multiple target tracking problem as a graph partitioning problem, more concretely, as a minimum cost (lifted) multicut problem. This formulation can serve as well for an initial tracklet generation process, which will help us to inject cues learned from spatial information into the appearance features, as it can be used to generate the final tracking result using short- and long-range information between object detections.

While the plain *Minimum Cost Multicut Problem* (refer to Chapter 1.3.3 for the formal definition) has shown good performance in multiple object tracking scenarios with only short range information available [148], the cost function actually has a rather limited expressiveness. In particular, when we want to add connectivity cues between temporally distant bounding boxes, we can only do so by inserting a direct edge into the graph. This facilitates solutions that directly connect such distant nodes even if this link is not justified by any path through space and time. This limitation is alleviated by the formulation of minimum cost *Lifted Multicuts* [84].

### Spatio-Temporal Tracklet Generation

Since the proposed approach is self-supervised in a sense that no annotated labels from the dataset are used in the training process, it is challenging to effectively learn such probabilities. To approach this challenge, we first extract reliable point matches between neighboring frames using DeepMatching [132] as done before e.g. in [148, 149, 85]. Instead of learning a regression model on features derived from the resulting point matches, we simply assume that the intersection over union (IoU) of retrieved matched within pairs of detections (denoted by  $\text{IoU}_{\text{DM}}$ ) is an approximation to the true IoU. Thus, when  $\text{IoU}_{\text{DM}} > 0.7$ , we can be sure we are looking at the same object in different frames. While this rough estimation is not suitable in the actual tracking task since it clearly over-estimates the cut probability, it can be used to perform a pre-grouping of detections that definitely belong to the same person. The computation of pairwise cut probabilities used in the lifted multicut step for the final tracking task is described in section 4.4.4.

### 4.4.2 Deep Convolutional AutoEncoder

A convolutional AutoEncoder takes an input image, feed forward it into a latent space and reconstructs it with the objective to learn meaningful features in an unsupervised manner. Unlike training with Triplet Loss, where specific image pairs from different classes are selected, the AutoEncoder requires no training labels at all. It consists of two parts: the encoder  $f_\theta(\cdot)$  and a decoder  $g_\phi(\cdot)$ , where  $\theta$  and  $\phi$  are trainable parameters of the encoder and decoder, respectively. For a given input video, there are in total  $n$  detections  $x_i \in X_{i=1}^n$ , the objective is to find a meaningful encoding  $z_i$ , where the dimension of  $z_i$  is much lower than  $x_i$ . The used convolutional AutoEncoder first maps the input data into a latent space  $Z$  with a non-linear function  $f_\theta : X \rightarrow Z$ , then decodes  $Z$  to its input with  $g_\phi : Z \rightarrow X$ . The encoding and reconstruction is achieved by minimizing the following loss equation:

$$L_{ae} = \min_{\theta, \phi} \sum_{i=1}^N L_2(g(f(x_i)), x_i) \quad (4.1)$$

where  $L_2$  is the least-squared loss  $L_2(x, y) = \|x - y\|^2$ . Similar to the work of [165], we add an additional clustering term to minimize the distance between learned features and their cluster center  $\tilde{c}_i$  from the spatio-temporal tracklet labels.

$$L_{ae\_c} = \min_{\theta, \phi} \sum_{i=1}^N L_2(g(f(x_i)), x_i) \lambda + L_2(f(x_i), \tilde{c}_i) (1 - \lambda) \quad (4.2)$$

The parameter  $\lambda \in [0, 1]$  balances between reconstruction and clustering loss. When choosing  $0 < \lambda < 1$ , the reconstruction part (Eq. (4.1)) can be considered to be a data-dependent regularization for the clustering. To compute the centroid  $c_i$ , the whole dataset is passed through the AutoEncoder once:

$$\tilde{c}_i = \frac{1}{N} \sum_{i=1}^N f(x_i) \quad (4.3)$$

We use a deep AutoEncoder with five convolutional and max-pooling layers for the encoder and five de-convolutional and upsample layers for the decoder, respectively. Furthermore, batch normalization is applied on each layer and initialized using Xavier Initialization [42]. The input image size is halved after each layer while the number of filters are doubled.

The size of latent space is set to 32. The input layer takes a colored image with dimension  $128 \times 128$  in width and height and we applied ReLu activation functions on each layer. An overview of the architecture is provided in 4.2.

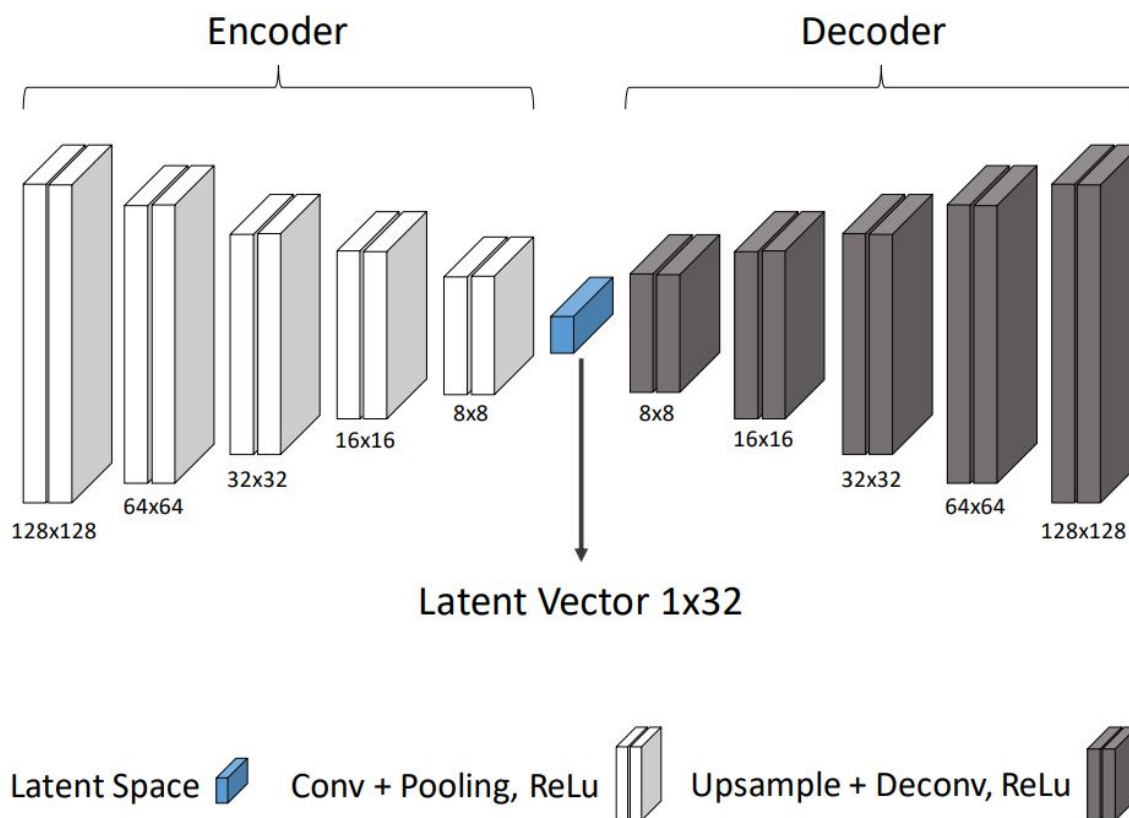


Fig. 4.2 AutoEncoder Architecture.

Convolutional AutoEncoder with five convolutional layers in encoder and de-convolutional layers in decoder, respectively. The center contains the latent space with a vector size of total 32, which is used as feature (latent) vector for similarity measure.



### 4.4.3 AutoEncoder combined with Proposed Triplet Loss

The convolutional deep AutoEncoder can be extended along with the previously proposed Triplet Loss 2.3. This additive term can be optimized jointly with  $L_{ae\_c}$  (4.2):

$$L_{ae\_Triplet} = L_{ae\_c} + L_{Triplet\_3} \quad (4.4)$$

Figure 4.3 illustrates the training overview, incorporating the proposed Triplet Loss. First, bounding boxes from video sequences are retrieved: the green line represents positive pair, which is selected purely based on spatio-temporal features from two adjacent frames (e.g. frame 1 and 2). The one with highest intersection over union (IoU) is selected. In contrast, the negative pair (in red) is selected within the same frame. This is done at random with the condition that no overlap between the anchor and negative bounding box is allowed. All three additive lost terms are depicted in blue.

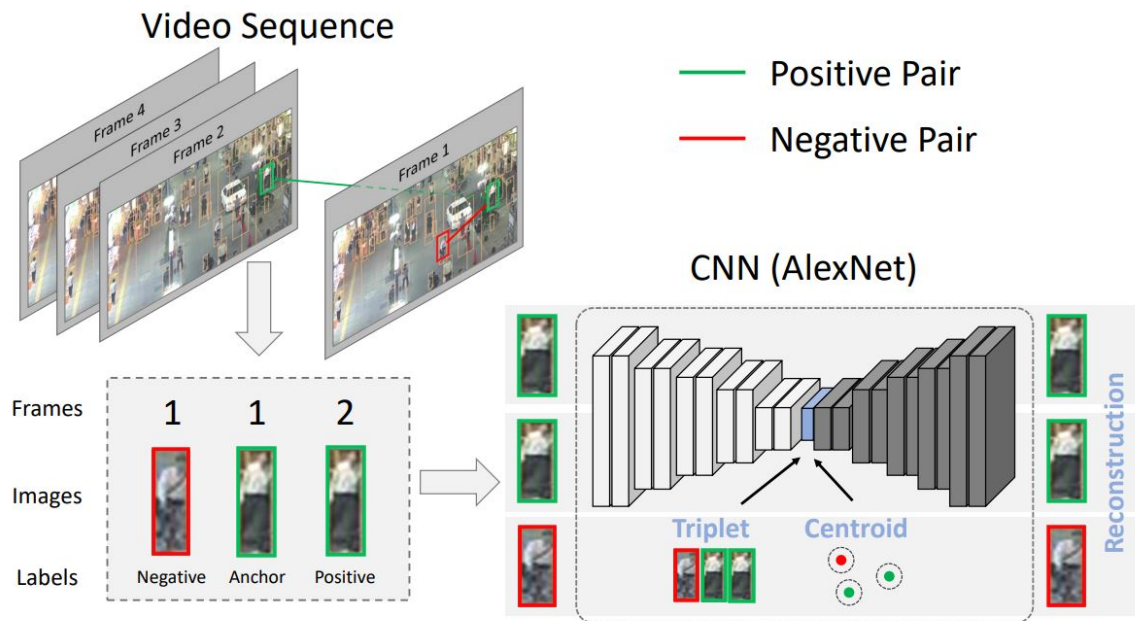


Fig. 4.3 Apply proposed Triplet Loss 2.3 on multiple person tracking.

Three images are selected in order to train a CNN to learn visual features. Positive pair (green) is selected based on spatio-temporal features between two adjacent frames, e.g. intersection of union. Negative pair (red) is selected randomly within the same frame, where no overlap of its bounding box positions are allowed.



Fig. 4.4 Comparison with and without Clustering Loss.

Nearest neighbor of the query detection (left most detection) within 46 frames with a step size of 5 frames of the sequence MOT17-09-SDP without ( $L_{ae}$ ) and with ( $L_{ae_c}$ ) the self-supervised clustering loss. Without clustering loss, the detections on the girl are spread over several clusters and a false association is made by the nearest neighbor. These mistakes are corrected by the clustering loss. Furthermore, optimizing with  $L_{ae_c}$  shows lower the Euclidean distance (in blue).

#### 4.4.4 AutoEncoder-based Similarity Measure

We use the trained AutoEncoder to estimate the similarity of two detections  $x_i$  and  $x_j$  of a video sequence based on the Euclidean distance in the latent space:

$$d_{i,j} = \|f(x_i) - f(x_j)\| \quad (4.5)$$

Figure 4.4 shows the nearest neighbor of a selected frame  $t$  (left box marked in red) from the sequence MOT17-09 and frame  $t + 5 \cdot k$ . The example illustrates that the location of detections with the same ID are close to one another in the latent space even over a long distance of up to 40 frames. Yet, false positives can appear. The example also shows that change in appearance affects the AutoEncoder distance, further denoted  $d_{AE}$ . For instance in the first row, frame 1 and frame 6 are very similar due to the same detection position of the person within the bounding box as well as the direction the girl is looking to. At frame 41, the girl (in Fig. 4.4) slightly turned towards another person. Although the correct nearest neighbour was retrieved, the distance  $d_{AE}$  almost doubled (in blue: distance 4.83 compared to 2.96 at frame 6). Another observation is that the position of the bounding box influences the latent space distance. Such behavior easily allows for false positive associations. In the second row, in the first detection from the left (frame 5), the detection of the person is slightly shifted to the left. At frame 15, 20 or 25, the position is slightly zoomed and  $d_{AE}$  increases. Yet, it is overall more stable and less false positive associations are made.

The distance is directly used to a binary logistic regression to compute the cut probability of the respective edge in graph  $G$ . The label that is used for the regression comes from the DeepMatching IoU. If  $\text{IoU}_{DM}(x_i, x_j) < T_{low}$  for a threshold  $T_{low}$ ,  $x_i$  and  $x_j$  most certainly belong to different objects. If  $\text{IoU}_{DM}(x_i, x_j) > T_{high}$  for a different threshold  $T_{high}$ , they are very likely to match. Formally, we estimate a probability  $p_e \in [0, 1]$  between two detections using a feature vector  $f^{(e)}$  by regressing the parameters  $\beta$  of a logistic function:

$$p_e = \frac{1}{1 + \exp(-\langle \beta, f^{(e)} \rangle)} \quad (4.6)$$

Thus, the costs  $c_e$  can intuitively be computed by the logit. To robustly estimate these probabilities, we set  $T_{low}$  and  $T_{high}$  most conservatively to 0.1 and 0.7, respectively.

From this partial, purely spatially induced labeling, we can estimate cut probabilities for all available features combinations, i.e. possible combinations of  $\text{IoU}_{DM}$  and  $d_{AE}$  within nearby frames and only  $d_{AE}$  for distant frames.

Table 4.2 Multiple Object Tracking: Ablation Study on Performance

Tracking Performance using different features on the MOT17 Training Dataset. The third column refers to the frame distance over which bounding boxes are connected in the graph.  $d_{AE}$  represents the AutoEncoder latent space distance while  $d_{AE+C}$  includes the clustering term, respectively. Our proposed approach includes lifted edges [149] between frames of distance 10, 20 and 30.

No	Features	Distance	MOTA	IDs	FP	FN
1	IoU <sub>DM</sub>	1-3	47.2	3,062	7,868	167,068
2	$d_{AE}$	1-3	35.2	4,378	10,213	203,868
3	$d_{AE+C}$	1-3	37.6	3,830	8,951	197,308
4	Combined (1+2)	1-3	49.4	1,730	7,536	161,057
5	Combined (1+3)	1-3	49.4	1,713	7,786	161,084
6	IoU <sub>DM</sub>	1-5	47.2	2,731	12,195	163,055
7	$d_{AE}$	1-3	35.8	4,623	<b>6,867</b>	204,697
8	$d_{AE+C}$	1-5	35.2	4,378	10,213	203,868
9	Combined (6+7)	1-5	49.7	1,567	9,067	158,788
10	Combined (6+8)	1-5	49.8	1,569	8,869	158,715
11	AutoEncoder ( $L_{ae_c}$ )	<b>1-5</b>	<b>50.2</b>	1,458	<b>8,466</b>	157,936
12	Triplet Loss ( $L_{ae\_Triplet}$ )	<b>1-5</b>	<b>50.2</b>	<b>1,381</b>	8,794	<b>157,462</b>

## 4.5 Experiments and Results

We evaluate the proposed method on the MOT17 Benchmark [115] for multiple person tracking. The dataset consists of 14 sequences, divided into train and test sets with 7 sequences each. For all sequences, three different detection sets are provided, from the detectors SDP[166], DPM[36] and FRCNN [131], thus yielding 21 sequences in both data splits. While SDP and FRCNN provide reliable detections, the performance of the DPM detector is relatively noise and many detections are show poor localization.

The settings between the training and testing scenes are very similar such as moving/static camera, place of recording or view angle, such that learning-based methods usually train on the most similar training sequence for every test sequence. For the evaluation, we use the standard CLEAR MOTA metric [99]. We reported Tracking Accuracy (MOTA), Precision (MOTP), number of identity switches (IDs), mostly tracked trajectories ratio (MT) and mostly lost trajectories (ML).

Table 4.3 Multiple Object Tracking Comparison.

Tracking result compared to other methods on the MOT17 dataset. The best performance is marked in bold.

Sequence	Method	<b>MOTA</b>	IDs	FP	FN
eHAF17[140]	Supervised	<b>51.8</b>	1,834	33,212	<b>236,772</b>
AFN17[138]	Supervised	51.5	2,593	22,391	248,420
YOONKJ17[138]	Supervised	51.4	2,593	29,051	243,202
NOTA[19]	Supervised	51.3	2,285	20,148	252,531
jCC[85]	Supervised	51.2	<b>1,802</b>	25,937	247,822
AutoEncoder ( $L_{ae\_c}$ )	Self-Supervised	48.1	2,328	17,480	272,602
Triplet Loss ( $L_{ae\_Triplet}$ )	Self-Supervised	47.9	2,082	<b>15,827</b>	276,179

After providing our implementation details, we report an ablation study on the training sequences of MOT17 in section 4.5.1. Our final results are discussed in section 4.5.2.

### Implementation Details

Our implementation is based on the Tensorflow Deep Learning Framework. We use a convolutional AutoEncoder in order to extract features by optimizing the equation (4.2). Thus no pre-training or any other ground truth is required. Furthermore, our pre-processing step is only limited to extracting the provided detections from all sequences and resizing them to the corresponding size of the AutoEncoder input layer. Thus the detections from the MOT17 dataset are directly fed to the AutoEncoder. For each sequence from the dataset (MOT17-01 to MOT17-14 with the detector SDP, FRCNN and DPM), one individual model is trained with the same setup and training parameters. However, it is important to note that the number of detections for each individual person varies significantly: This is due to the fact that individual pedestrians are captured in a scene over many frames while others are quickly passing by or simply missed by the detector. While some pedestrians are staying in the scene for a long time, others are passing by quickly out of the scene. This results in different cluster sizes. To balance this, randomized batches of detections are applied during the training, where each batch contains only images from one single frame. This way, one iteration of training contains only detections from unique persons. The initial learning rate is set to  $LR = 0.001$  and decays exponentially by a factor of 10 over time. The balancing parameter between reconstruction and clustering loss is set to  $\lambda = 0$  at the beginning in order to first learn the visual features of the video sequences. After five epochs, the cluster information is

included in the training, e.g.  $\lambda$  is set to 0.95 to encode the appearance variations from the spatio-temporal clusters into the latent space of the AutoEncoder.

### From Clusters to Tracklets

To transform detection clusters into actual tracks, we follow the procedure proposed in [148], i.e. from all detections within one cluster, we select the one with the best detection score per frame. Clusters containing less than 5 detections are completely removed and gaps in the resulting tracklets are filled using bilinear interpolation.

#### 4.5.1 Ablation Study

We investigated feature setups in the *Minimum Cost Multicuts* framework. The cut probability between pairs of nodes are computed using a logistic regression function. Adding new features directly affects the edge cost between pairs thus resulting in different clustering performances. Here, we investigate the extent to which our proposed appearance model improves the tracking performance.

#### Comparison of different setups

Table 4.2 shows the evaluated setups and the resulting tracking performance scores. The column *Features* lists the added features to the logistic regression model. The temporal distances over which bounding boxes are connected in the graph are marked in the column *Distances*. The tracking accuracy of experiment 1 and 6, which uses  $\text{IoU}_{\text{DM}}$  only, is 47.2%. Experiment 2+3 and 7+8 compare the different AutoEncoder models: the Euclidean distance ( $d_{\text{AE}}$ ) from the AutoEncoder latent space is computed in order to estimate the similarity of each pair detections. Here,  $d_{\text{AE}}$  denotes the latent space distance before adding the clustering loss while  $d_{\text{AE}+\text{C}}$  denotes the distance after training of the AutoEncoder with the clustering loss, i.e. our proposed appearance method.

#### Best performance with proposed method

The benefit from using the clustering loss on the model training is obvious: for both distances (1-3 and 1-5 frames), the performance is significantly higher. For distance 1-3,  $d_{\text{AE}+\text{C}}$  has a tracking accuracy of 37.6 compared to  $d_{\text{AE}}$  (35.2) and for distance 1-5, the MOTA scores are 35.2 and 35.8 for  $d_{\text{AE}+\text{C}}$  and  $d_{\text{AE}}$ , respectively. Although the scores are lower than using

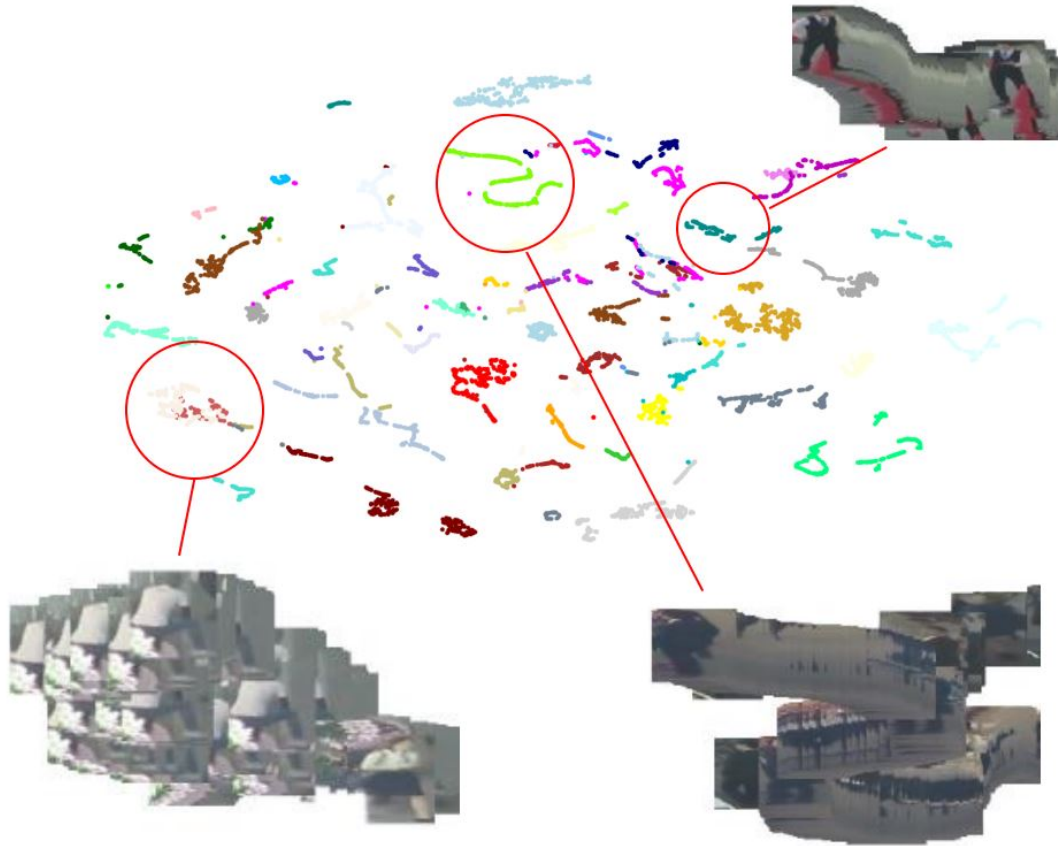


Fig. 4.5 TSNE Visualization of AutoEncoder Latent Space.

TSNE Visualization of the latent space of the trained AutoEncoder for the sequence MOT17-04 FRCNN. The colors represent the assigned person IDs. As the appearance changes for example due to pose changes, the latent representations vary smoothly.

$\text{IoU}_{\text{DM}}$ , combining them both together increases the performance further. This is shown in experiment 4+5 and 9+10, where the best score is achieved with in experiment 10 (proposed method). We also observe that the number of identity switches (IDs) is reduced with our setup. Finally, we add lifted long range edges and solve the resulting *Minimum Cost Lifted Multicuts Problem* on  $G$ . Our best performance is achieved using the setup of experiment 11 with a MOTA of 50.2% using all model components.

### TSNE-Visualization

Figure 4.5 shows the TSNE-Visualization [112] of the latent space from the sequence MOT17-04-FRCNN. Our proposed AutoEncoder learned the visual features without supervision. The different colors represent the cluster labels. As shown in the example circled on the bottom left, similar looking persons are very closed to one another in the latent space: The sitting person in white shirt and the lady, wearing a white shirt (example in bottom left). The visualization also shows, that the same person may change the appearance over time (example on the bottom right). In the latent space, the *snake*-like shape may indicate that the viewpoint or pose of a person may have changed over time, causing a continuous appearance change. When standing still, the change is minimal, which is also observed in the example on the top right corner. While for nearby frames, we can compute pairwise cues based on the distance between latent feature representation ( $d_{AE}$ ), as well as on spatial cues (IoU<sub>DM</sub>), spatial information can not be used to associate detections over longer temporal distances. However, to facilitate the re-identification of objects after being fully or partly occluded, such long-range information is needed. In these cases, we have to purely rely on the learned latent space distance  $d_{AE}$ .

## 4.5.2 Results

### Tracking Performance on test data

Here, we present and discuss our final tracking results on the MOT17 test dataset. Compared to the performance on the training dataset, the MOTA score of our proposed approach is slightly lower (Training: 50.2% vs. Testing: 48.1%), which is within the observed variance between different sequences, neglecting excessive parameter tuning. The best performance is achieved in conjunction with the SDP-detector while the performance on the noisier DPM detections are weaker (detailed tables are provided in the supplementary material). While supervised approaches can also train their models w.r.t. the overlap of provided detections with the ground truth and thus compensate for poor detector quality, our self-supervised approach depends on reasonable object detections.



### Comparison with other tracking approaches

We compare our method with five other reported tracking methods eHAF17[140], AFN17[138], YOONKJ17[138], NOTA[19] and jCC[85]. Unlike better performing approaches on the MOT benchmark, these methods use the provided public detections directly without employing any repair mechanism in order to improve the bounding box quality.

We consider a tracking method as supervised when ground truth data is used (for example label data for learning a regression function) or if any pre-trained model is included in the approach. Table 4.3 gives an overview of the scores in different metrics that is being evaluated. The best on each category is marked in bold. Our proposed method is competitive given the fact that no pre-trained model or any other ground truth is employed. Without using any of the provided human annotations and with identical parameter settings for all sequences and detectors, our resulting MOTA scores are shown to be competitive. Note especially the relatively low number of ID switches.

Yet, as discussed above, our model currently has no mechanism that would allow to *repair* erroneous detection such as it is done e.g. in [9], [12] or [37] or by the current state-of-the-art approach Lif\_T[65].

When comparing more closely the average MOTA scores we achieve per detector over all sequences, our proposed method reaches 46.9% on the SDP detector while [9] reach 47.1%. For a state-of-the-art detector, our method performs thus competitive with supervised one. Yet, on the noisy DPM detections, our approach is outperformed by 10% (49.0 [9] vs. 34.3 (Ours)), decreasing the total average significantly.

## 4.6 Conclusion

In this Chapter, we applied the clustering technique on a real-world problem. Specifically, we presented an approach towards tracking of multiple persons without the supervision by human annotations based on deep convolutional AutoEncoder. First, we group the data based on their spatial-temporal features to obtain weak clusters (tracklets). The clustering is done using the *Lifted Minimum cost multicut*s, where similarity measures are first solely based on spatial-temporal features. Combining the visual features learned from an AutoEncoder with these tracklets, we are able to automatically create robust appearance cues enabling multiple person tracking over a long distance. The result of our proposed method achieves a tracking accuracy of 48.1% on the MOT17 benchmark.



# Chapter 5

## Clustering as Robustness Predictor

In this Chapter, we use clustering as a indicator for measuring robustness of deep neural networks. The content of this chapter is based on the approach, that we previously published in [61]. While I contributed the idea, algorithmic implementation and experimental evaluation, the work has been supervised by Prof. Dr. Janis Keuper and Prof. Dr. Margret Keuper. The co-authors Avraam Chatzimichailidis and Franz-Josef Pfreundt in [61] contributed to insightful discussions.

### 5.1 Introduction

We want to apply previously introduced *Minimum Cost Multicuts* and *K-Means* clustering in order to predict, whether a model is robust against corruptions of input data, since deep learning approaches have shown rapid progress on computer vision tasks. Much work has been dedicated to train ever deeper models with improved validation and test accuracies and efficient training schemes [174, 67, 106, 68]. Recently, this progress has been accompanied by discussions on the robustness of the resulting model [29]. Specifically, the focus shifted towards the following two questions:

1. How can we train models that are robust with respect to specific kinds of perturbations?
2. How can we assess the robustness of a given model?

These two questions represent fundamentally different perspectives on the same problem. While the first question assumes that the expected set of perturbations is known during model training, the second question rather aims at estimating a models behavior in unforeseen cases and predict its robustness without explicitly testing on specific kinds of corrupted data.

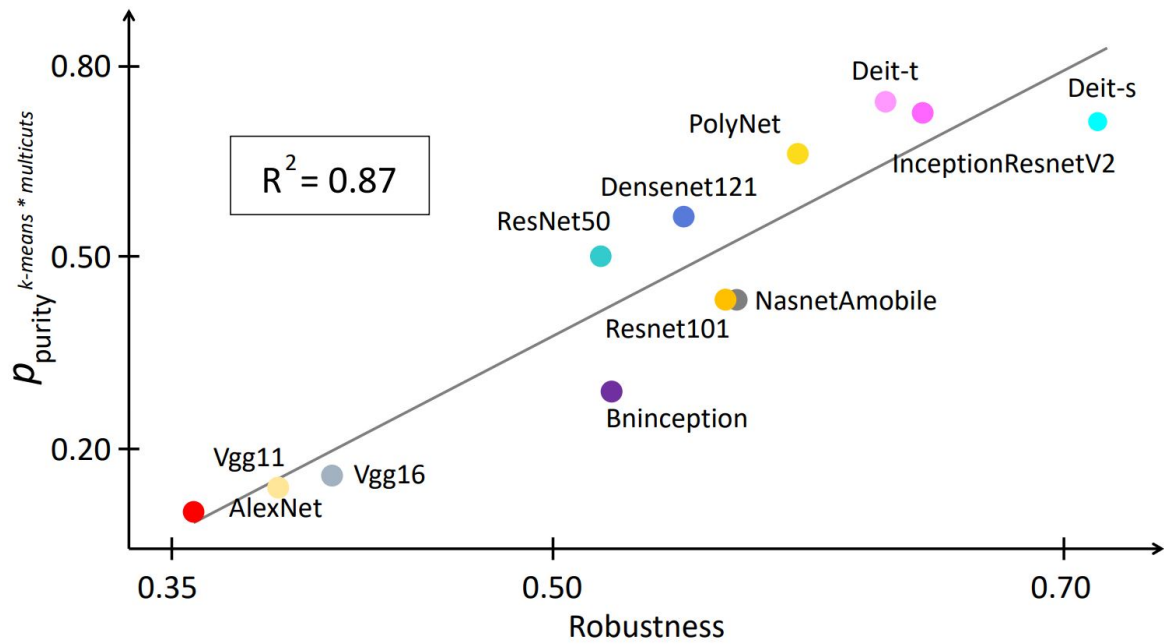


Fig. 5.1 Clustering as a Predictor.

Predicting the robustness of models using our proposed cluster purity indicator ( $p_{purity}$ ): The correlation between  $p_{purity}$  of models trained on the original ImageNet with the measured test accuracy on ImageNet-C is  $R^2 = 0.87$ .

In this Chapter, we address the second research question. We argue that the clustering performance in a model’s latent space can be an indicator for a model’s robustness. For this purpose, we introduce cluster purity as a robustness measure in order to predict the behavior of models against data corruption and adversarial attacks. Specifically, we evaluate various classification models [94, 174, 70, 51, 145, 172, 73, 153] on the ImageNet-C [52] dataset of corrupted ImageNet images where we measure the robustness of a model as the ratio between the accuracy on corrupted data and clean data. The key result of this paper is illustrated in figure 5.1: it shows that the model robustness is strongly correlated to the relative clustering performance on the models’ latent spaces, i.e. the ratio between the cluster purity and the classification accuracy, both evaluated on clean data. The clusterability of a model’s feature space can therefore be considered as an easily accessible indicator for model robustness.

In summary, our work contributes the following:

- We study the feature spaces of several ImageNet pre-trained models including the state-of-the-art CNN models [174, 70, 51, 145, 172] and the recently proposed transformer models [153] and evaluate their model robustness on the ImageNet-C dataset and against adversarial attacks.
- We show that intra- and inter-class distances extracted from classification models are not suitable as a direct indicator for a model's robustness.
- We provide a study of two clustering methods, *K-Means* and the *Minimum Cost Multicut Problem* (MP) and analyze the correlation between classification accuracy, robustness and clusterability.
- We show that the relative clustering accuracy, i.e. the ratio between classification and clustering performance, is a strong indicator for the robustness of the classification model under ImageNet-C corruptions.

This Chapter is structured as follows: We first review the related work on image classification, model robustness and deep clustering approaches in Chapter 5.2, then we propose the methodology for the feature space analysis in Chapter 5.3. Our experiments and results are discussed in Chapter 5.4.1.

## 5.2 Related Work

### Image Classification

Convolutional neural networks (CNN) have shown great success in computer vision. In particular, from the classification of handwritten characters [101] to images [93], CNN-based methods consistently achieve state-of-the-art in various benchmarks. With the introduction of ImageNet [134], a dataset with higher resolution images and one thousand diverse classes is available to benchmark the classification accuracy of ever better performing networks [94, 174, 70, 51, 145, 172], ranging from small and compact network [67] to large models [144] with over 100 millions of parameters.

### Transformers

Recently, transformer network architectures, which were originally introduced in the area of natural language processing [155], have been successfully applied to the image classification task [20, 32]. The performance of transformer networks is competitive despite having no convolutional layers. However, transformer models require long training times and large amounts of data [32] in order to generalize well. A more efficient approach for training has been proposed in [153], which is based on a teacher-student strategy (distillation). Similarly, [16] uses the same strategy on self-supervised tasks.

### Model Robustness

Convolutional neural networks are susceptible to distribution shifts [129] between train and test data [121, 39, 52, 135]. This concerns both visible input domain shifts by for example considering corrupted, noisy or blurred data, as well as imperceptible changes in the input, induced by [117, 44, 97]. These explicitly maximize the error rate of classification models [146, 11] and thereby reveal model weaknesses. Many methods have been proposed to improve the adversarial robustness by specific training procedures, e.g. [117, 75]. In contrast, input distribution shifts induced by various kinds of noise as modeled in the ImageNet-C [52] dataset mimic the robustness of a model in unconstrained environments, for example under diverse weather conditions. This aspect is crucial if we consider scenarios like autonomous driving, where we want to ensure robust behaviour for example under strong rain. Therefore, we focus on the latter aspect and investigate the behaviour of various pre-trained models under ImageNet-C corruptions but also evaluate the proposed robustness measure on adversarial perturbations [117, 75].

## Clustering

Clustering approaches, deep clustering approaches in particular, have shown to benefit from well structured feature spaces. Such approaches therefore aim at optimizing the latent representations for example using variational autoencoders or Gaussian mixture model or *K-Means* priors [128, 164, 41, 40, 14]. [14] iteratively groups points using *K-Means* during the latent space optimization. Conversely, we are investigating the actual feature space learned from image classification tasks using clusterability as a measure for its robustness. Therefore, we apply clustering approaches on pre-trained feature spaces. Further, while the above mentioned methods rely on a *K-Means*-like clustering, i.e. data is clustered into a given number of clusters, we also evaluate clusters from a similarity driven clustering approach, the *Minimum Cost Multicut Problem* [6].

The Multicut Problem, aka. Correlation Clustering, groups similar data points together by pairwise terms: data (e.g. images) are represented as nodes in a graph. The real valued weight of an edge between two nodes measures their similarity. Clusters are obtained by cutting edges in order to decompose the graph and minimize the cut cost. This problem is known to be NP-hard [27]. In practice, heuristic solvers often perform reasonably [83, 8]. Correlational Clustering has various applications in computer vision, such as motion tracking and segmentation [85, 162], image clustering [62] or multiple object tracking [149, 62].

### 5.3 Feature Space Analysis

Our aim is to establish indicators for a model’s robustness from the structure of its induced latent space. Therefore, we first extract latent space samples, i.e. feature representations of input test images. The latent space structure is subsequently analyzed using two different clustering approaches. *K-Means* is clustering data based on distances to a fixed number of cluster means and can therefore be interpreted as a proxy of how well the latent space distribution can be represented by a univariate Gaussian mixture model. The *Minimum Cost Multicut Problem* formulation clusters data points based on their pairwise distances and therefore imposes less constraints on the data manifold to be clustered. Figure 5.2 gives an overview of the methodology. First, we briefly recap classification models as feature extractors in Chapter 5.3.1. The *K-Means* and *Minimum Cost Multicut Problem* on the image clustering task are explained in Chapter 5.3.2. In Chapter 5.3.3, we review evaluation metrics for measuring the clustering performance and in Chapter 5.3.4, we present our proposed metrics for robustness estimation.

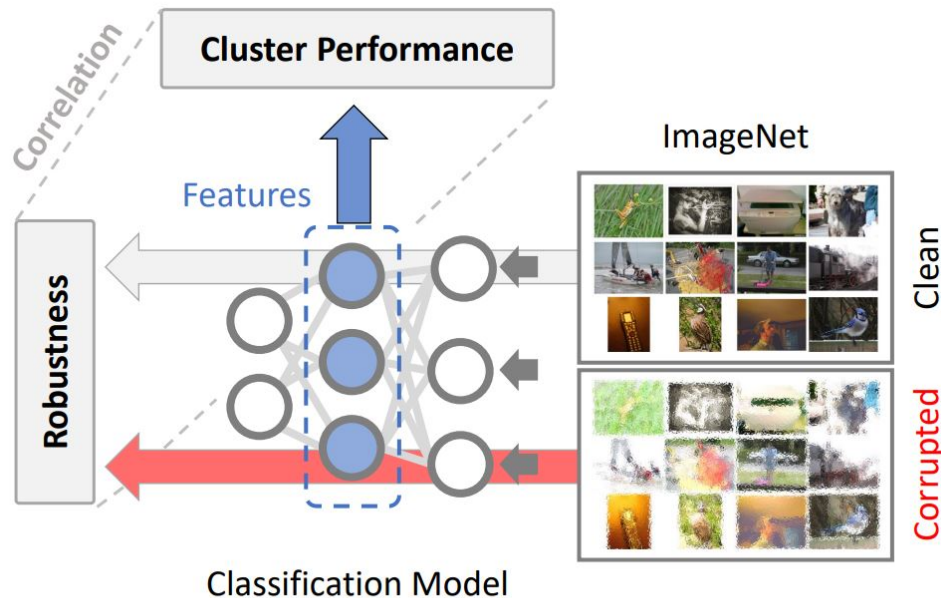


Fig. 5.2 Robustness Predictor: Experiment Setup.

The robustness of a model is measured by its relative classification performance, which is the ratio between clean and corrupted (in red arrow) data.. The latent space or features (in blue) of various classification models is sampled using ImageNet images. The feature representations are then clustered with the *K-Means* and *Multicuts* clustering approaches. The correlation is visualized in 5.1.



Table 5.1 List of Classification Models.

**Classification models:** all models are trained and evaluated on the ImageNet [134] dataset, sorted by performance. We report the Top1 classification accuracy in %. The first ten models are based on convolutional layers while the last two are transformer networks.

MODEL	FEATURES	PARAM	TOP1 %
ALEXNET [94]	4096	61.1M	56.4
VGG11 [174]	4096	132.9M	69.0
VGG16 [174]	4096	138.4M	71.6
BNINCEPTION [73]	1024	11.3M	73.5
NASNETAMOBILE [174]	1056	5.3M	74.1
DENSENET121 [70]	1024	7.9M	74.6
RESNET50 [51]	2048	25.6M	76.0
RESNET101 [51]	2048	44.5M	77.4
INCR50 [145]	1536	55.8M	80.2
POLYNET [172]	2048	95.3M	81.0
DEIT-TINY [153]	192	5.9M	74.5
DEIT-SMALL [153]	384	22.4M	81.2

### 5.3.1 Extracting Features from Classification Models

Classification models with multiple classes are often trained with softmax cross-entropy and it has been shown that features, learned from vanilla softmax cross-entropy achieve a high performance in transfer accuracy [91]. In order to obtain the learned features from images, the last layer of the trained model (classifier) is removed, which is often done for instance in transfer learning [137, 141] or clustering tasks [164]. The model encodes an image  $x_i$  with a function  $f_{\theta}(\cdot)$ , with pre-trained parameters  $\theta$ . Table 5.1 shows the different classification models with their according feature dimensions as well as the number of parameters and their top 1 classification accuracy in %. We investigate models which vary significantly in their architectures, including CNNs and transformer models, their number of parameters, ranging from 3.5M to 138M, as well as their test accuracy, ranging from 56.4% to 81.2% top-1 scores. We use features extracted from the full ImageNet test set as latent space samples for our analysis as shown in 5.2.

### 5.3.2 Latent Space Clustering

**K-Means** is a simple and effective method to cluster  $N$  data points into  $K$  clusters  $S_k$ ,  $k = 1, \dots, K$ . As  $K$  is set a priori, this method produces exactly the number of defined clusters by minimizing the intra-cluster distance:

$$\sum_{k=1}^K \sum_{x_i \in S_k} \|f(x_i) - \mu_k\|^2 \quad (5.1)$$

where the centroid  $\mu_k$  is computed as the mean of features  $\frac{1}{|S_k|} \sum_{x_i \in S_k} f(x_i)$  in cluster  $k$ .

**The Minimum Cost Multicut Problem** is a graph-based clustering approach. Considering an undirected graph  $G = (V, E)$ , with  $v \in V$  being the images  $x_i$  of the dataset  $X$  with  $|V| = N$  samples, a complete graph with  $N$  nodes has in total  $|E| = \frac{N(N-1)}{2}$  edges. A real valued cost  $w : E \rightarrow \mathbb{R}$  is assigned to every edge  $e \in E$ . While the decision, whether an edge is joined or cut, is made based on the edge label  $y : E \rightarrow \{0, 1\}$ , the decision boundary can be derived from training parameters as we proposed in Chapter 2, directly learned from the dataset [62, 149] or simply estimated empirically (via parameter search). The inference of such edge labels is defined previously in Chapter 1.3.3 (Equation 1.8).

Practically, the edge costs are computed from pairwise distances in the feature space. The distance  $d_{i,j}$  between two features  $f(x_i)$  and  $f(x_j)$  is calculated from the pre-trained model or encoder  $f$ , where  $x_i$  and  $x_j$  are two distinct images from the test dataset, respectively, as

$$d_{i,j} = \|f(x_i) - f(x_j)\|^2. \quad (5.2)$$

A logistic regression model estimates the probability from the pairwise distance  $d_{i,j}$  of the edge between  $f(x_i)$  and  $f(x_j)$ . This cut probability is then converted into real valued edge costs  $w$  using the *logit* function  $\text{logit}(p) = \log \frac{p}{1-p}$  such that similar features are connected by an edge with positive, i.e. attractive weight and dissimilar features are connected by edges with negative, i.e. repulsive weight. The decision boundary (i.e. the threshold on  $d$ , which indicates when to cut or to join) is estimated empirically in our experiments.

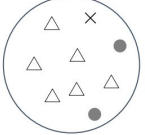

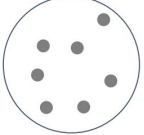
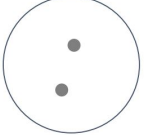




	Cluster 1	Cluster 2	Cluster 3	Cluster 4	
Majority					
					
Accuracy	6/9 = 66%	6/8 = 75%	7/7 = 100%	0/2 = 0%	19/26 = 73%
Purity	6/9 = 66%	6/8 = 75%	7/7 = 100%	2/2 = 100%	21/26 = 80%

Fig. 5.3 Evaluation Metric for Clusters.

Evaluation metrics with 4 clusters with 3 unique classes. **Cluster Accuracy:** The best match for class *dark circle* is cluster 3, since it contains the most frequent items from the same class. Cluster 4 is considered as false positive. **Purity score** on the other hand does not penalize cluster 4. Thus, the purity score is higher than the cluster accuracy (80% vs. 73%).

### 5.3.3 Cluster Quality Measures

We use two popular external evaluation metrics (i.e. label information are used) to measure the clustering performance: Cluster Accuracy (ACC) and Purity Score. The former metric is calculated based on the Hungarian algorithm [95], where the best match between the predicted and the true labels are found. The purity score assigns data in a cluster to the class with the most frequent label [74]. Formally, given a set of  $K$  clusters  $CL_k$  and a set of classes  $L$  with a total number of  $N$  data samples, the purity is computed as follows:

$$\frac{1}{N} \sum_{k \in K} \max_{\ell \in L} |CL_k \cap \ell| \quad (5.3)$$

The advantage of using this metric is two-fold: on one hand, it is suitable if the dataset is balanced and on the other hand, purity score does not penalize having a large number of clusters. Figure 5.3 depicts an example of both metrics.

### 5.3.4 Performance Measure

Next, we derive a measure based on the latent space clustering performance, that allows to draw conclusions on a model's robustness without evaluating the model on corrupted data. Thereby, we measure a model's robustness as its relative classification accuracy, i.e. the ratio between its classification accuracy on corrupted data and on clean data:

$$Robustness = \frac{\text{Model}_{\text{ACC}_{s,c}^*}}{\text{Model}_{\text{ACC}}} \quad (5.4)$$

Parameters  $c$  and  $s$  are corruption type and severity level (or intensity), respectively for non-adversarial attacks such as ImageNet-C. The aggregated value over all severity levels  $s \in \tilde{S}$  on all corruption types  $c \in CORR$  is calculated as follows:

$$ACC_{all}^* = \frac{1}{|CORR|} \sum_{c \in CORR} \frac{1}{|\tilde{S}|} \sum_{s=1}^{|\tilde{S}|} ACC_{s,c}^* \quad (5.5)$$

According to equation 5.4, perfectly robust models therefore have a robustness of 1, smaller values indicate lower robustness. Based on the above considerations on model robustness and clustering performance, we propose to consider the relative clustering performance as an indicator for the model robustness and show empirically that there exists a strong correlation between both. The relative clustering performance, i.e. the ratio between clustering performance and classification accuracy  $Model_{ACC}$  is defined as follows:

$$p = \frac{\text{clustering performance}}{Model_{ACC}} \quad (5.6)$$

Here, we consider the clustering accuracy  $C_{ACC}$  and purity score  $C_{purity}$  as a performance measures for our experiments, i.e.

$$p_{ACC} = \frac{C_{ACC}}{Model_{ACC}} \quad \text{and} \quad p_{purity} = \frac{C_{purity}}{Model_{ACC}}$$

respectively.

**Correlation Metrics.** The degree of correlation is computed based on the coefficient of determination  $R^2$  and Kendall rank correlation coefficient  $\tau$ , respectively with a value of 1.0 being perfectly correlated while 0 means no correlation at all. An example for  $R^2$  is illustrated in Figure 5.1 and  $\tau$  in Figure 5.12.

**Baseline Indicator: Class Overlap  $\Delta$ .** Our hypothesis is that an initial well-separated feature space of a classification model provides a good estimate regarding the model robustness. A simple method to determine such a separation would be to observe the intra- and inter-class distances between data samples in the feature space. If an overlap between classes exists, they are not well separated, which may indicate weak models. We define this setting as a baseline in order to show that latent space clustering provides significantly more information.

To investigate this, we define the overlap  $\Delta$  between the intra- and inter-class distances as follows:

$$\Delta = (\mu_{intra} + \sigma_{intra}) - (\mu_{inter} + \sigma_{inter}) \quad (5.7)$$

$\mu$  and  $\sigma$  represent the mean and standard deviation of the intra- and inter-class distances.

## 5.4 Experiments

This Chapter is structured as follows: we first explain the setup of our experiments in 5.4.1. Then, we present the clustering results in Chapter 5.4.2 where we analyse the clustering accuracy and purity for the two considered clustering approaches on the feature spaces of the different models. Section 5.4.3 shows that the intra- and inter-class distances cannot directly be used as robustness indicators. In Section 5.4.4, we consider the relationship between the model classification robustness under corruptions and the relative clustering performance of the considered clustering methods and metrics. We show that both clustering accuracy and cluster purity, computed on the feature spaces of clean data, allow to derive indicators for a model’s expected robustness under corruptions. Thereby, the purity score is more stable than the clustering accuracy and the information provided by *K-Means* clustering and *Multicuts* complement one another. In Section 5.4.5, we evaluate the proposed robustness indicator in the context of adversarial attacks.

### 5.4.1 Setup

Our experiments are based on the ImageNet [134] dataset. We use our MSM approach for *Minimum Cost Multicuts Problem*, which we introduced in Chapter 3. All models were pre-trained on the original training dataset. We evaluate 10 CNN-based models and 2 transformer architectures *deit-t* and *deit-s* (t stands for tiny and s for small). An overview is provided in Table 5.1. On *Multicuts*, the decision, whether a distance  $d_{i,j}$  needs to be cut or joined is based on the threshold parameter (decision boundary or  $\tilde{\tau}$ ). Setting it too high results in less clusters while choosing a small value will output a large number of small clusters. This threshold is crucial for the clustering performance and we provide the relationship between this hyper-parameter and the different performance metrics in Figure 5.4. Since all models are pre-trained on classification tasks, the decision boundary multicuts cannot be derived from the training parameters as described in Chapter 2. We therefore empirically estimated on a smaller dataset first and once the optimal threshold  $\tilde{\tau}$  is found, it is applied to the complete test dataset. In our experiments, we chose the threshold with the highest cluster accuracy.

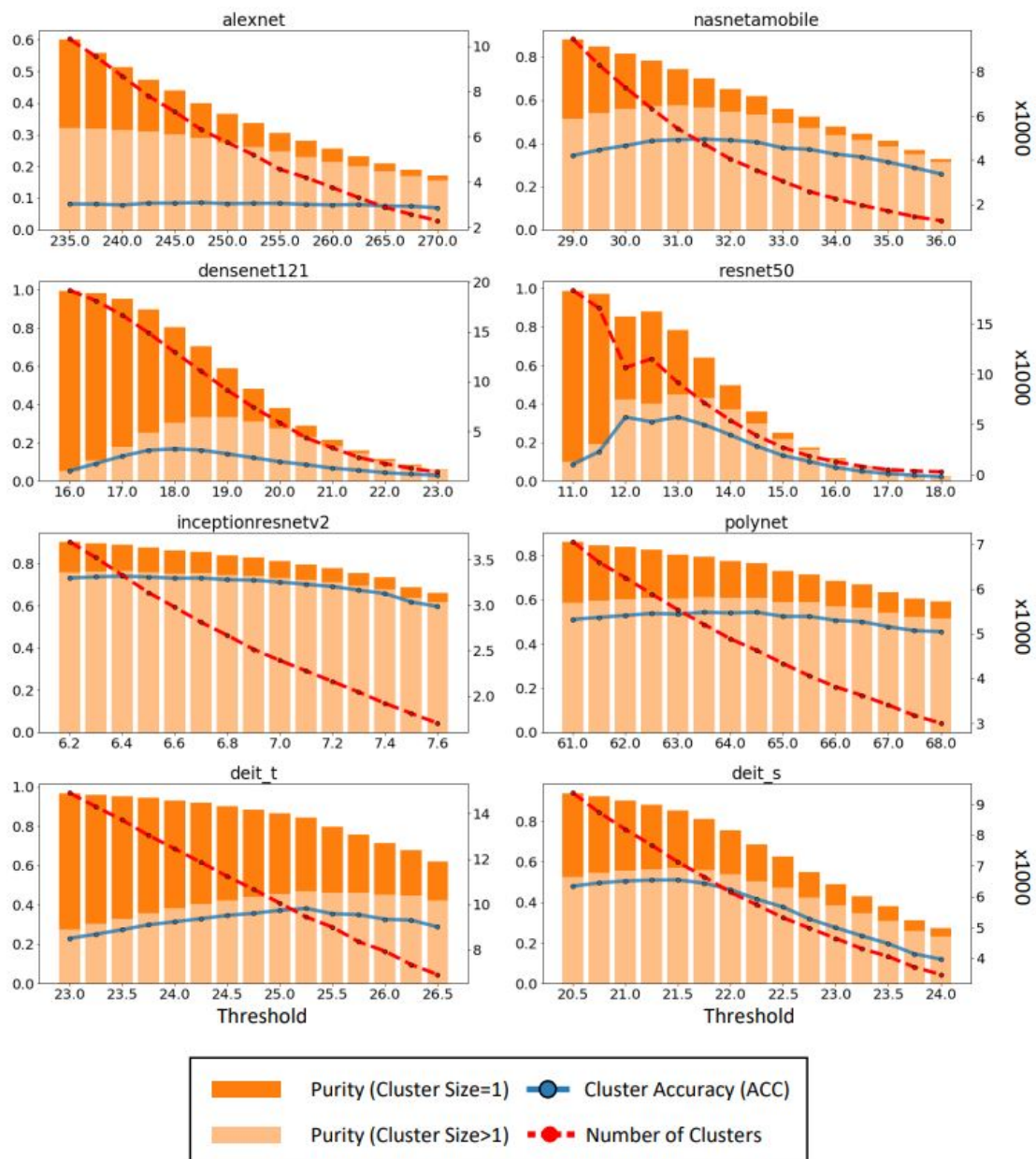


Fig. 5.4 Find Threshold on Pre-Trained models.

**Threshold on *Multicuts*:** In our experiments, we selected the threshold with the highest cluster accuracy (blue line). Blue line represents the cluster accuracy and orange bar the purity score, divided by cluster types, e.g. Cluster containing only one single image (dark orange) and clusters with more than one image within a cluster (light orange). Furthermore, the red line shows the total number of unique clusters.

**ImageNet-C.** We evaluate the robustness of the considered models against corruptions using the *ImageNet-C* [52] dataset and report the model accuracy for classification and clustering tasks. Figure 5.5 illustrates an example of considered image corruptions: the first row shows the different severity levels  $s = 1, \dots, 5$  of the corruption *brightness*, with 1 being the lowest and 5 the strongest corruption. The second row shows other kinds of image perturbations  $c$  at severity level 5 such as *fog*, *frost*, *gaussian blur*, *jpeg\_compression* or *pixelate*. Each corruption  $c$  has 5 severity levels  $s = 1, \dots, 5$ . All models are trained on the clean dataset and the numbers are evaluated on the full test dataset, as done in [52].



Fig. 5.5 ImageNet-C Examples.

ImageNet-C dataset: first row shows the original image and the corruption *brightness* for different severity levels. Second row: examples of other corruption types at severity level 5.

## 5.4.2 Classification vs. Clustering

Table 5.2 summarizes the evaluation in three categories: classification, *K-Means* and *Multicuts*. Additionally, Figure 5.6, 5.7, 5.8 and 5.9 provide details of the experiments on ImageNet-C. There are in total  $|\text{CORR}| = 19$  corruption types with each  $|\tilde{S}| = 5$  severity levels on ImageNet-C. For the classification task, the numbers are reported in top 1% accuracy for all five levels of corruption (denoted as 1 – 5). On *K-Means* and *multicuts*, we report the clustering metrics as presented in Chapter 5.3.3.

### Transformer has highest accuracy

The transformer *deit-s* shows the highest top 1% accuracy on the classification task both on clean and on corrupted data for all severity levels. *Inceptionresnetv2* and *polynet* perform only slightly worse on clean data but are more strongly affected by the ImageNet-C data corruptions than *deit-s*. *Alexnet* shows the worse performance across all corruption levels. Although *resnet50* outperforms *bninception*, *nasnetamobile* and *densenet121* it is less robust against corruption. This is also illustrated later Chapter 5.4.4 in Figure 5.10 (right).

### Purity and Cluster Accuracy

Considering the clustering accuracy and purity, the *K-Means* and the *Multicuts* behave significantly different from one another. *K-Means* clustering achieves about 70% accuracy for models with the highest clean classification accuracy. Yet, its accuracy is much better for the *deit-t* latent space than for example for the *densenet121* induced latent space, although the clean classification accuracy of both networks is comparable.

### *K-Means* works well on Transformers

Overall, the *K-Means* clustering works surprisingly well on the transformer models. The *Multicuts* clustering showed the highest clustering accuracy on the *inceptionresnetv2* model. The cluster purity was comparably high for the best transformer model *deit-s*. Note that our goal is to derive from the clustering performance an indicator for model robustness, i.e. we expect clustering to be less accurate when models are less robust to noise.



Table 5.2 Evaluation of Robustness.

Evaluation of robustness on classification and clustering tasks with the *ImageNet C* dataset, evaluated on corruption severity levels 1 to 5. Column *CLEAN* represents the classification performance of the models on the clean dataset. Columns 1-5 show the classification accuracy (top 1%) under different severity levels of over all 19 corruptions and column  $ACC_{all}^*$  shows the mean over corruptions on all 5 severity levels. On *K-Means*, *ACC* and *purity* are clustering performance on clean test data. The numbers on *Multicuts* are evaluated on a subset. The best score on each column is marked in **bold**.

MODEL	CLASSIFICATION ACCURACY (TOP 1%)					K-Means			MULTICUTS				
	CLEAN	1	2	3	4	5	$ACC_{all}^*$	ACC PURITY	$ACC_{all}^*$	ACC PURITY	$ACC_{all}^*$		
ALEXNET	56.4	35.9	25.4	18.9	12.7	8.0	20.2	14.6	18.4	8.0	8.0	28.1	2.6
VGG11	69.0	47.3	35.3	25.7	16.7	10.1	27.0	28.0	32.8	12.4	15.8	27.2	2.5
VGG16	71.6	50.9	38.6	28.5	18.7	11.4	29.6	32.3	37.5	14.4	19.3	27.6	2.8
BNINCEPTION	73.5	59.4	48.4	38.8	27.2	17.7	38.3	40.5	44.0	18.0	11.5	46.6	7.9
NASNETAMOBILE	74.1	60.7	51.3	43.7	33.6	22.5	42.4	41.0	45.3	23.6	41.9	70.2	19.3
DENSENET121	74.6	60.2	50.9	42.2	31.4	21.0	41.1	48.9	52.1	23.4	16.8	80.5	8.3
RESNET50	76.0	60.1	49.8	40.1	28.9	18.8	39.6	55.8	58.6	24.9	29.3	64.3	11.1
RESNET101	77.4	63.6	54.4	45.5	34.0	22.9	44.1	59.1	61.9	29.3	28.6	53.7	16.6
INCEPTIONRESNETV2	80.2	68.8	60.8	53.5	43.5	31.6	51.7	<b>70.0</b>	<b>71.2</b>	37.4	<b>71.3</b>	<b>81.3</b>	<b>39.8</b>
POLYNET	81.0	68.0	58.9	49.9	38.2	26.3	48.3	67.8	69.7	34.8	54.4	76.6	24.1
DEIT-T	74.5	63.3	55.9	48.7	38.9	28.1	47.0	57.4	60.0	31.7	33.0	91.9	19.5
DEIT-S	<b>81.2</b>	<b>72.1</b>	<b>66.1</b>	<b>60.2</b>	<b>51.3</b>	<b>39.7</b>	<b>57.9</b>	68.8	70.8	<b>43.4</b>	49.4	81.1	29.6

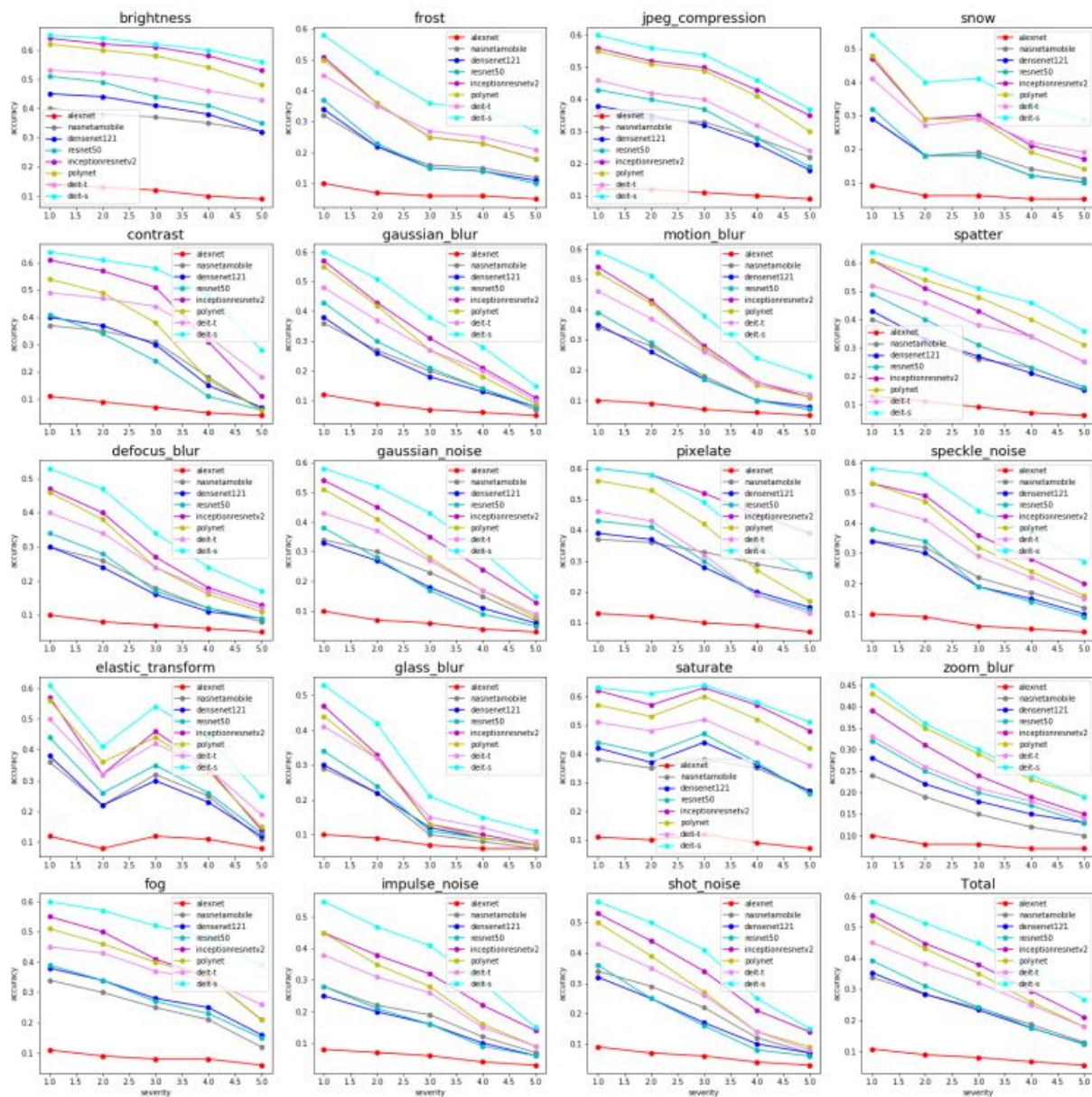


Fig. 5.6 *K-Means*: cluster accuracy on ImageNet-C, grouped by severity levels

Cluster accuracy decreases as severity level goes up.

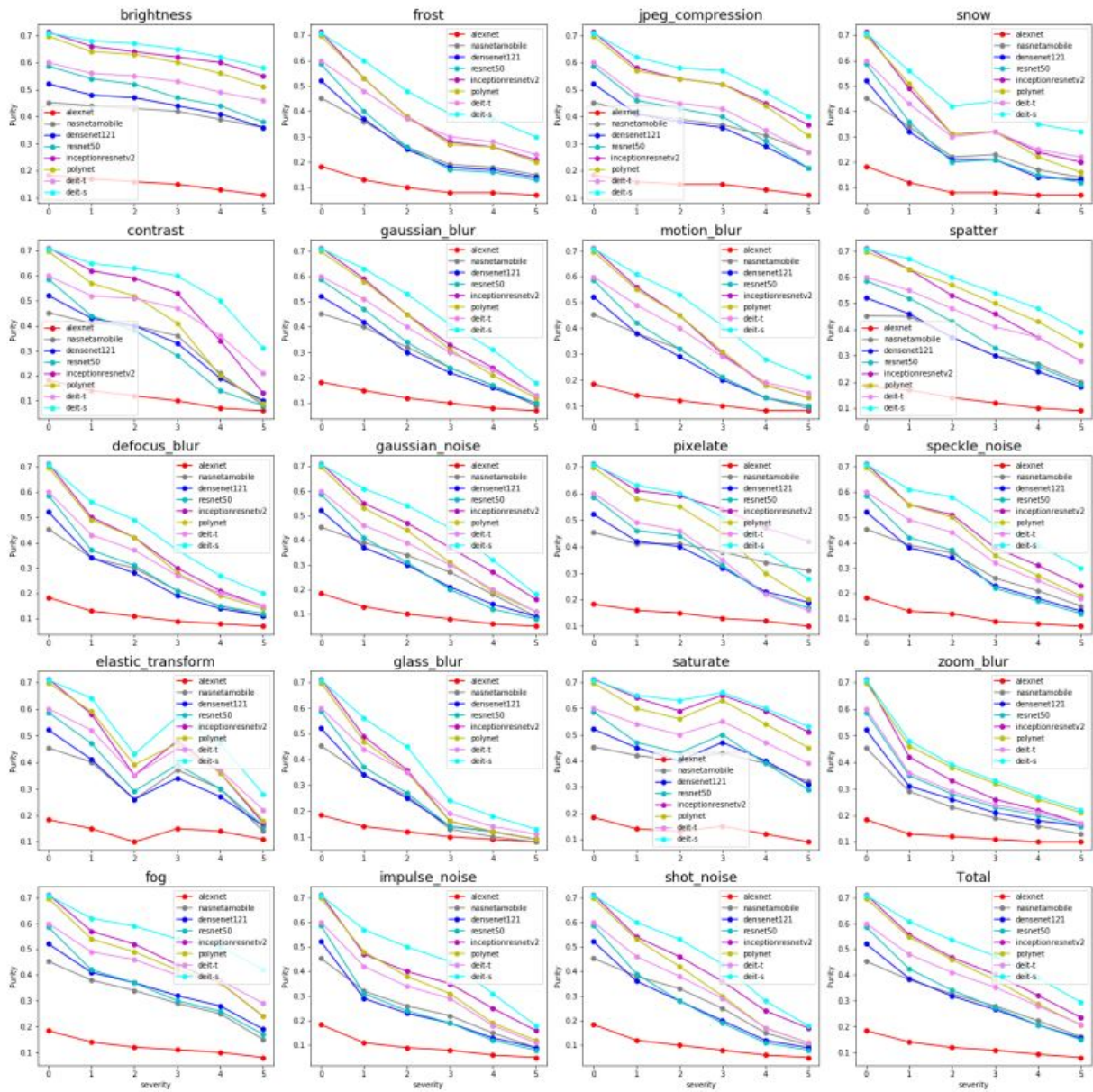


Fig. 5.7 *K-Means*: cluster purity on ImageNet-C, grouped by severity levels

Cluster purity decreases as severity level goes up.

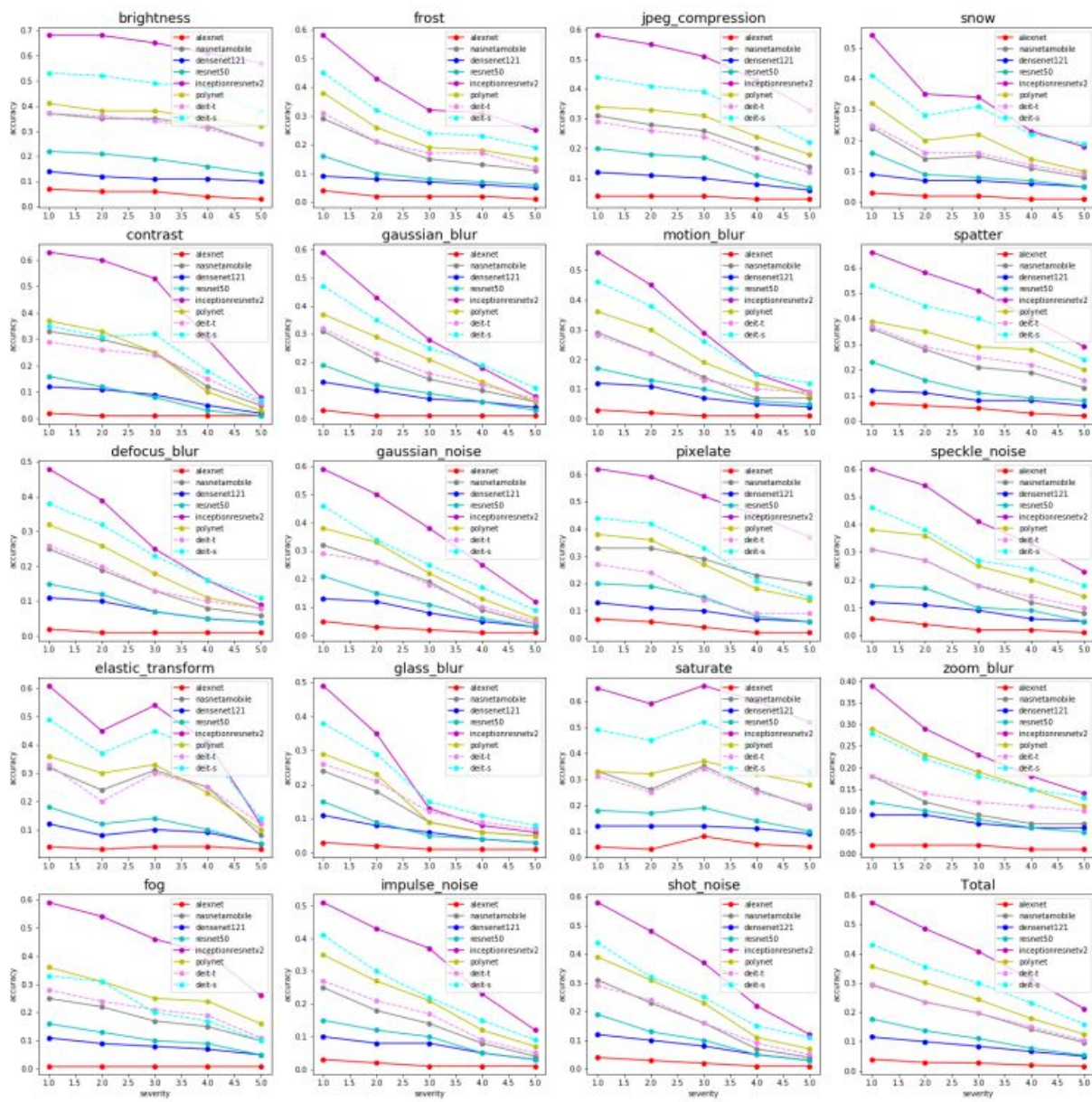


Fig. 5.8 *Multicuts*: cluster accuracy on ImageNet-C, grouped by severity levels

Cluster accuracy decreases as severity level goes up.

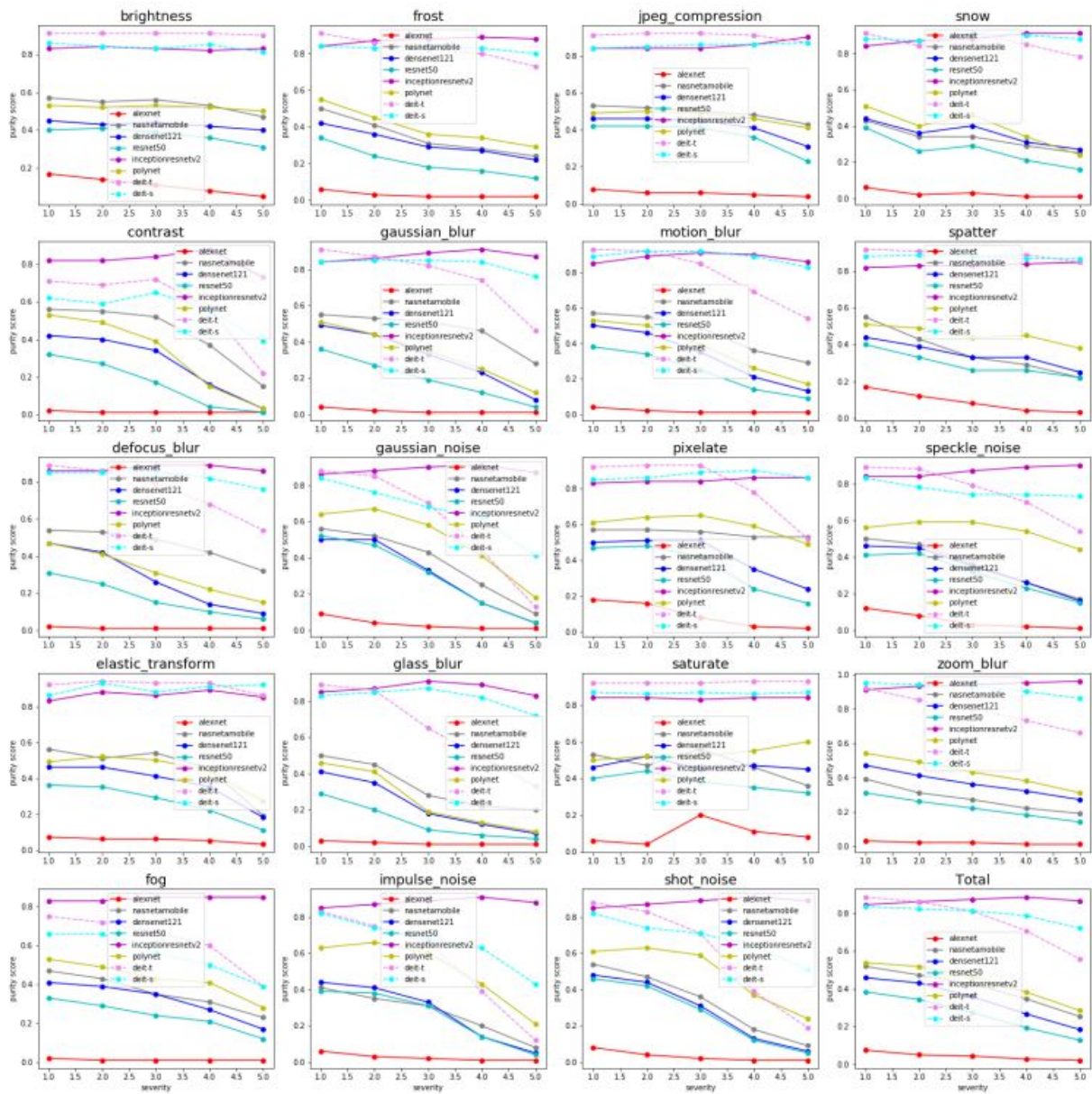


Fig. 5.9 *Multicuts*: cluster purity on ImageNet-C, grouped by severity levels

Cluster purity remains stable as severity level goes up for some models.

Table 5.3 Baseline Indicator for Model Robustness.

**Baseline** indicators for model robustness: The table shows the correlation between overlap  $\Delta$  and model robustness for different corruption severity levels. Second row shows the rank correlation  $\tilde{\tau}$  between the actual model robustness rank and the predicted rank using  $\Delta$ .

	SEVERITY					
METRIC	1	2	3	4	5	TOTAL
$R^2$	0.27	0.29	0.27	0.25	0.26	0.27
$\tilde{\tau}$	0.48	0.52	0.52	0.52	0.52	0.48

### 5.4.3 Baseline Indicators: Intra- and Inter class-distances

Table 5.3 shows the correlation (as  $R^2$ ) and the ranking correlation  $\tilde{\tau}$  between the class overlap baseline indicator  $\Delta$ , which we detailed in section 5.3.3, and the model robustness, grouped by severity level. We use equation 5.4 to calculate the robustness for severity level  $s$  over all corruptions and compare them with  $\Delta$ . The last column shows the correlation on all corruption levels. All 12 models are considered. The rank correlation  $\tilde{\tau}$  is calculated by comparing the model’s robustness rank and the overlap  $\Delta$  ranking. Initial well-separated feature spaces (thus a low  $\Delta$ ) should have a high correlation with their model’s robustness. Despite its simplicity, this metric  $\Delta$  correlates poorly with a highest score of  $R^2 = 0.29$  and  $\tilde{\tau} = 0.52$ . This observation rejects the simple hypothesis about the overlap of intra- and inter-class distances and it suggests that using  $\Delta$  is not sufficiently informative as an indicator for model robustness.

### 5.4.4 Robustness Indicators: Clustering Measures

In the following we evaluate our proposed clustering driven robustness indicator. Specifically, we want to investigate the effects of different clustering measures on the correlation coefficient  $R^2$ . Table 5.4 gives an overview of the strength of correlation on different severity levels and clustering metrics on *K-Means* and *multicuts*. Column  $\Delta$  shows the correlation on robustness using the overlap of intra- and inter-class distances as previously discussed. Furthermore, the columns *ACC* and *P* are showing the correlation between the model robustness and the clustering accuracy and purity, respectively. The last column shows the combination of both clustering methods in one metric. *K-Means* and *multicuts* have an  $R^2$  value of  $R^2 = 0.83$  and  $R^2 = 0.55$  for clustering accuracy on all corruption levels. On the purity score, both methods show a slightly higher correlation of  $R^2 = 0.83$  and  $R^2 = 0.71$ , respectively for the

Table 5.4 Correlation by Severity Levels.

**Correlation with different metrics and severity levels:** the reported numbers are the coefficient of determination ( $R^2$ ) on different clustering metrics. Column  $\Delta$  is the overlap (from Table 5.3). Column *ACC* and *Purity* (denoted as *P*.) are used to compute the correlation coefficient  $R^2$ . The last column is the combination of both clustering methods, i.e. last column *Purity* is equation 5.8. The highest score is marked in bold.

METRIC: $R^2$		K-MEANS		MULTICUTS		COMBINED	
SEVERITY	$\Delta$	ACC	P.	ACC	P.	ACC	P.
1	0.27	<b>0.85</b>	<b>0.85</b>	0.48	0.67	0.54	0.82
2	0.29	<b>0.87</b>	<b>0.87</b>	0.51	0.70	0.58	0.86
3	0.27	0.84	0.83	0.55	0.73	0.61	<b>0.87</b>
4	0.25	0.79	0.79	0.58	0.72	0.64	<b>0.87</b>
5	0.26	0.75	<b>0.84</b>	0.57	0.68	0.64	<b>0.84</b>
<b>ALL</b>	0.27	0.83	0.83	0.55	0.71	0.62	<b>0.87</b>

sum over all corruptions (last row of Table 5.4). This indicated that latent space clusterability of clean test images K-Means is a valid indicator for model robustness under corruptions. However, we show that both clustering methods are complementary when combining their purity scores with

$$P_{\text{purity}^{K\text{-Means-multicuts}}} = \frac{C_{\text{purity}}^{K\text{-Means}} \cdot C_{\text{purity}}^{\text{multicut}}}{\text{Model}_{\text{ACC}}}. \quad (5.8)$$

This measure shows the highest correlation with the model robustness with  $R^2 = 0.87$  (refer to Figure 5.1 in Chapter 5.1 for the full correlation plot). Additionally, the combination of purity scores of both methods also yields more consistent results across different severity levels.

Predicted			Actual		
$100 \times p_{ACC}^{k-means}$	Model Name	Rank	Rank	Model Name	$100 \times$ Robustness
25.9	AlexNet	12	12	AlexNet	35.8
40.6	Vgg11	11	11	Vgg11	39.1
45.1	Vgg16	10	10	Vgg16	41.3
55.1	BNInception	9	9	Resnet50	52.1
55.3	NasnetAmobile	8	8	BNInception	52.1
65.5	Densenet121	7	7	Densenet121	55.1
73.4	Resnet50	6	6	Resnet101	57.0
76.4	Resnet101	5	5	NasnetAmobile	57.2
77.0	Deit-t	4	4	PolyNet	59.6
83.7	PolyNet	3	3	Deit-t	63.1
84.7	Deit-s	2	2	InceptionResnetV2	64.5
87.3	InceptionResnetV2	1	1	Deit-s	71.3

Fig. 5.10 Robustness Ranking.

Change in robustness ranking based on predicted (left) vs. actual (right) model robustness on ImageNet-C using clustering metric  $p_{ACC}^{K-Means}$  on total corruptions  $\tilde{\tau} = 0.79$ . Top is the least robust model ( $Rank = 12$ ) while the bottom shows the most robust model ( $Rank = 1$ ). The highest score is marked in bold.

**Model Ranking.** Next, we evaluate whether our proposed robustness indicator is able to retrieve the correct ranking in terms of model robustness for our set of classification models. The rank correlation is measured as the Kendall rank coefficient  $\tilde{\tau}$ . Table 5.5 shows the results for different setups. Here, *K-Means* shows a more consistent and better correlation with highest rank correlation of  $\tilde{\tau} = 0.82$  on *ACC* and *Purity*. Again, all clustering metrics outperform the  $\Delta$  baseline. Figure 5.10 illustrates one example of the change of rank between the predicted (left) and actual (right) model robustness. The prediction is done using  $p_{ACC}^{K-Means}$ , which has a rank correlation of  $\tilde{\tau} = 0.79$ . Our proposed measure is able to rank different models according to their robustness. The three worse performing models (*alexnet*, *vgg11* and *vgg16*) are correctly retrieved. The largest ranking gap of 3 positions is observed for *nasnetamobile* and *resnet50*. In this particular example, the value for *alexnet* is calculated as follows:  $\frac{14.6}{56.4} * 100 = 25.9$  and  $\frac{20.2}{56.4} * 100 = 35.8$  for predicted and actual values, respectively.



Table 5.5 Rank Correlation by Severity.

**Rank correlation with different metrics and severity levels:** the reported numbers are the rank coefficient ( $\tilde{\tau}$ ) on different clustering metrics. Column  $\Delta$  is the overlap (from Table 5.3). Column *ACC* and *Purity* (denoted as *P.*) are the used compute the rank correlation coefficient  $\tilde{\tau}$ . The last column is the combination of both clustering methods, i.e. last column *Purity* is Equation 5.8. The highest score is marked in bold.

METRIC: $\tilde{\tau}$		K-MEANS		MULTICUTS		COMBINED	
SEVERITY	$\Delta$	ACC	P.	ACC	P.	ACC	P.
1	0.48	<b>0.79</b>	<b>0.79</b>	0.61	0.52	0.73	0.73
2	0.52	<b>0.82</b>	<b>0.82</b>	0.64	0.55	0.76	0.76
3	0.52	<b>0.82</b>	<b>0.82</b>	0.70	0.61	<b>0.82</b>	0.76
4	0.52	<b>0.82</b>	<b>0.82</b>	0.70	0.61	<b>0.82</b>	0.76
5	0.52	<b>0.82</b>	<b>0.82</b>	0.70	0.61	<b>0.82</b>	0.76
<b>ALL</b>	0.48	<b>0.79</b>	<b>0.79</b>	0.67	0.58	<b>0.79</b>	0.73

**Latent Space Visualization.** Umap [114], a scalable dimensionality reduction method similar to the popular technique TSNE[154], has been applied to features on 10 randomly selected classes of the ImageNet dataset for visualization. Figure 5.11 shows one example of the corruption *brightness* for 2 different models: the first column shows features without any corruptions (clean). As the severity level increases, a collapse is observed for instance on *alexnet*: well-separated clusters (i.e. different colors) are being pulled into a direction in the latent space as the severity increases. The model with the highest robustness, i.e. *deit-s*, preserves the clusters well, which explains the high relative clustering performance. This verifies our assumption on the correlation between clusterability and robustness of classification models, that were evaluated in ImageNet-C dataset.

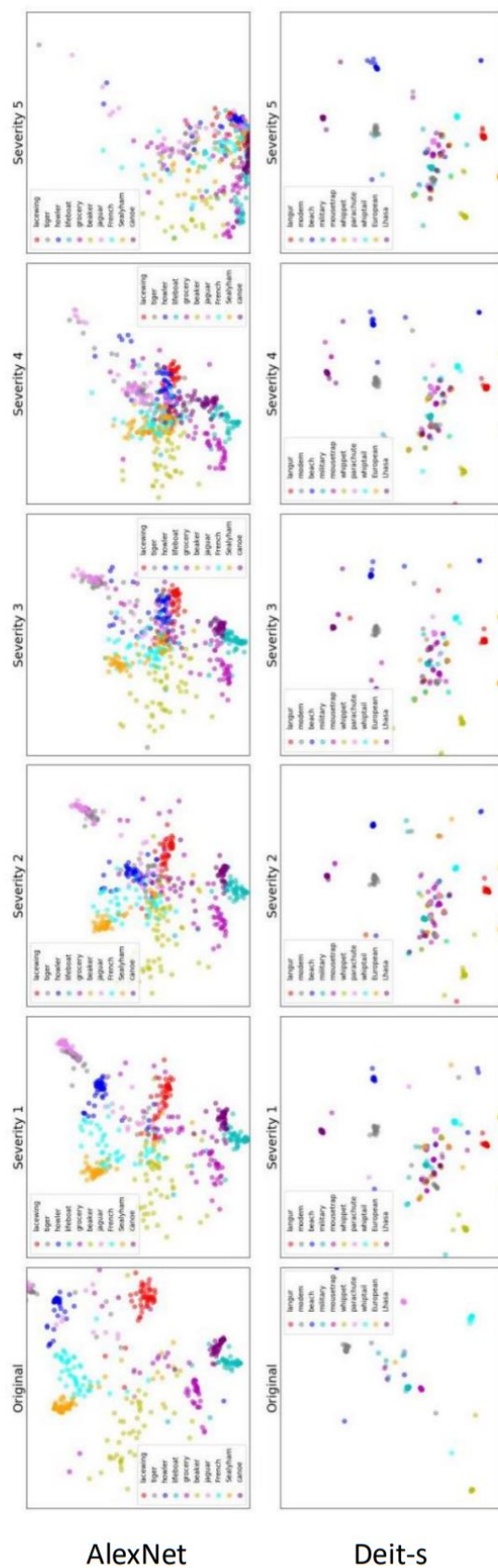


Fig. 5.11 UMap Visualization: CNN vs. Transformer.

Visualization of feature space on *alexnet* and *deit-s* using umap. The colors correspond to the class labels, where only 10 classes were selected at random. First column shows the initial, clean latent space from the classification model. Each new column depicts the corresponding severity level of the corruption *brightness*. While *alexnet* collapses as the severity increases, the most robust model *deit-s* preserves the clusters very well even after significant corruptions and thus our proposed clusterability of latent space provides a good indicator about the model robustness.

### 5.4.5 Adversarial Robustness

So far, we have shown that our proposed approach can effectively indicate the robustness of classification models towards visible image corruptions and shifts in the data distributions provided by the ImageNet-C benchmark. Here, we extend this evaluation to intentional, non-visible corruptions induced by adversarial attacks. Using the proposed clustering metric  $P_{\text{purity}^{K\text{-Means-multicuts}}}$  as an estimator, we evaluate all 12 models with ImageNet test dataset under two adversarial attacks: *DeepFool* [117], *FGSM* [44] and *PGM* [97] with different perturbation sizes  $\epsilon$ . Figure 5.12 shows the results of all three attacks across all 12 models: left (a) represents the correlation of determination  $R^2$  while on the right (b) the classification accuracy, respectively.  $\epsilon$  (x-axis) is the perturbation size of the attacks. For small epsilon, we expect lower correlations since the model accuracy should hardly be affected. As epsilon increases, some models are more robust than others, i.e. better preserve their classification accuracy. In this range, we see a relatively strong correlation of the proposed indicator and the relative robust accuracy, albeit weaker than the correlation with robustness to corruptions, with  $R^2 = 0.66$ ,  $R^2 = 0.44$  and  $R^2 = 0.44$  for *DeepFool* and *FGSM* and *PGM*, respectively. When epsilon becomes too large, the correlation becomes weaker. Our method therefore works well for adversarial examples within a certain range of epsilons. In contrast, no gradients need to be computed (e.g. clustering with *K-Means*), thus requiring less compute resources as opposed to *FGSM* and *PGM*.

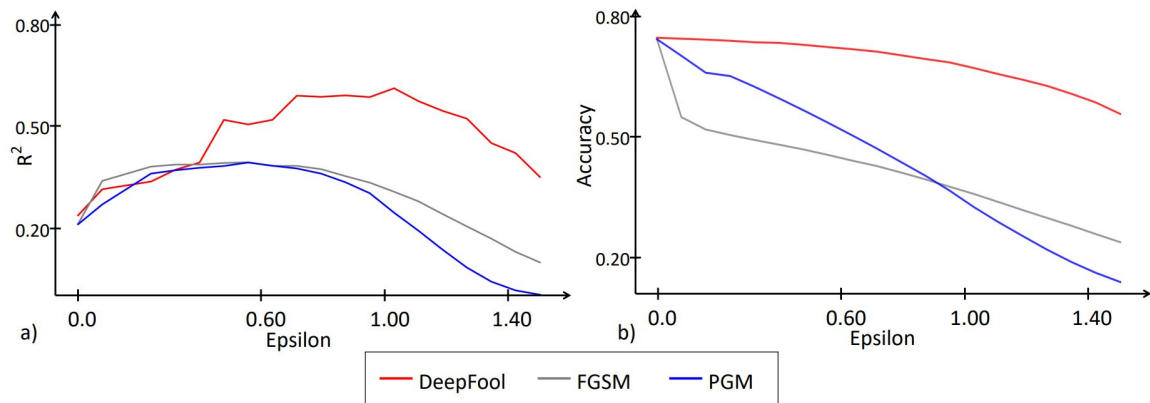


Fig. 5.12 Correlation on Adversarial Robustness.

Correlation of our proposed clustering metric on different adversarial attacks with different strengths ( $\epsilon$ ). Dashed line represents the classification accuracy and solid line the coefficient of determination  $R^2$ , respectively.

## 5.5 Conclusion

In this Chapter, we presented a study of the feature space of several pre-trained models on ImageNet including state-of-the-art CNN models and the recently proposed transformer models and we evaluated the robustness on ImageNet-C dataset and extended our evaluation on adversarial robustness as well. We propose a novel way to estimate the robustness behavior of trained models by analyzing the learned feature-space structure. Specifically, we presented a comprehensive study of two clustering methods, *K-Means* and the *Minimum Cost Multicuts Problem* on ImageNet, where the classification accuracy, clusterability and robustness are analyzed. We show that the relative clustering performance gives a strong indication regarding the model's robustness. We argue that if data in a feature space are distributed in a way that can be clustered effectively, then the model are less susceptible against data corruptions. While *K-Means* assumes that the data are well distributed around its centroid, *Minimum Cost Multicuts* on the other use pairwise comparisons to obtain the clusters. Both considered clustering methods show complementary behaviour in our analysis: the coefficient of determination is  $R^2 = 0.87$  when combining the purity scores of both methods. Our experiments also show that this indicator is lower, albeit still significant for adversarial robustness ( $R^2 = 0.66$  and  $R^2 = 0.44$ ). Additionally, our proposed method is able estimate the order of robust models ( $\tau = 0.79$ ) on ImageNet-C. This novel method is simple yet effective and allows the estimation of robustness of any given classification model without explicitly testing on any specific test data. To the best of our knowledge, we are the first to propose such technique for estimating model robustness.

# Chapter 6

## Conclusion

The *Minimum Cost Multicut Problem* has been proven to be effective on various computer vision tasks, especially when clustering a large number of image data. Unlike *K-Means*, where data are assigned to a fixed number (e.g.  $K$  clusters) of predefined cluster centers, pairwise comparisons are made and objective is solved as one graph problem by cutting or joining of nodes. To obtain such pairwise distance, an embedding space from the dataset is calculated, for instance the Euclidean Distance from two data points. The key aspect of this approach is to find a good representation of similarity. Such feature or embedding space can be obtained via supervised, weak-supervised (Triplet Loss) or completely unsupervised (AutoEncoder). The main contribution of this thesis is to learn such embedding space for the *Minimum Cost Multicut Problem* and its application on Computer Vision problems.

### 6.1 Summary of Contribution

**Triplet Loss for Minimum Cost Multicut Problem** This thesis first introduces a metric learning technique for the *Minimum Cost Multicut Problem* in Chapter 2. We evaluated two variants of Triplet Losses, that is well-known in the literature. A simplification of the Triplet Loss is then proposed, which enables a more robust graph-based clustering against label noises. Experiments on CIFAR10 dataset validate our initial assumption regarding the embedding space, trained on regular Triplet Loss. A comprehensive study on the other variants of the function were then conducted: not only does the proposed version of Triplet Loss performs better on image clustering tasks, it allows the embedding space to be optimized specifically for the *Minimum Cost Multicut Problem*. Thus the threshold for cut and join can be directly derived from the training parameters. This threshold has to be learned from additional training otherwise (for instance using logistic regression).

### **MSM - Multi-Stage Multicuts**

While the *Minimum Cost Multicut Problem* performs well when using a correct distance measure, such as the Euclidean distance trained an embedding space, many practical applications may not be suitable as when the dataset becomes very large. We therefore proposed a scalable version of the *Minimum Cost Multicut Problem* in Chapter 3, which is called *MSM*. Since the size of the graph grows quadratic to the number of data points (nodes), splitting the complete dataset in disjoint small sets reduces the total size significantly, thus leading to speedup. Yet, the performance of the actual clustering task is not affected significantly. Our experiments on CelebA dataset shows, that MSM with 40 computing threads can cluster a dataset of over 100k images under just one minute. This allows the *Minimum Cost Multicut Problem* to be able to scale up when sufficient compute resources are available.

### **Multiple Object Tracking**

We then applied *Minimum Cost Multicut Problem* on the Multiple Object Tracking problem in Chapter 4. Additionally, long-range distances are included as well. First, an AutoEncoder approach is proposed to learn an embedding space. During the optimization, additional spatio-temporal features are included. We also compared this supervision free approach with our proposed Triplet Loss and experiments on MOT17 dataset show, that both methods are similar and competitive.

### **Robustness Predictor**

Finally, Chapter 5 utilizes clustering approach in order to predict the robustness of classification models. Specifically, several ImageNet pretrained models are evaluated on various forms of image corruptions, including adversarial attacks. The experiments show that the intuitive intra- and inter class distances from pretrained models are not suitable as direct indicators for model robustness. Instead, the relative clustering accuracy of the combined method, *K-Means* and *Minimum Cost Multicut Problem*, shows a very strong indication for model robustness under ImageNet-C corruptions. Both considered clustering methods show complementary behaviour in our analysis.

## 6.2 Future Work

*Minimum Cost Multicut Problem* has been widely studied in computer vision. The main advantage is that no prior knowledge is required in order to decompose the given graph. However, the performance is highly depending on the similarity measure. This can be obtained by training an embedding space for a given task from a dataset. While this thesis proposes a simplification of the previously proposed Triplet Loss [171], that is suitable to learn such embedding space for the *Minimum Cost Multicut Problem*, we point out some future research directions and possible extensions to this work:

### Architectures

Embedding space can be obtained via training, such as optimizing the Triplet Loss. When the task specific training is done, a simple distance measure represents the similarity of the data, e.g. the closer the points are together, the more similar they are. This is essential for the *Minimum Cost Multicut Problem* as it requires pairwise distances for decomposition of the given graph. In our experiments, such as in Chapter 2, 3 or 4, we used AlexNet [94] architecture, as we followed the approach as done in [14, 119]. However, it is unclear, how other architecture such as VGG [174], Densenet [70] or Resnet [51] affect the overall clustering performance. Recently, transformers [13, 10, 153, 32, 16] have been consistently achieving state-of-the-art performance in classification tasks. It is worthwhile to research on transformer-based architecture as well for metric learning, such as using Triplet Loss.

### MSM

In Chapter 3, one key problem with multicut is highlighted: scalability. When a dataset becomes too large, it is infeasible to decompose the large graph in a realistic runtime on a modern computer. Furthermore, the size of the graph becomes too large to fit on a memory. We presented *MSM* to overcome this problem. By dividing the whole dataset into small, disjoint sets, the size of the graph is reduced greatly. While small performance drop is observed, the overall speedup gain is significant. However, additional hyperparameters are introduced such as the number of stages or sparsity of graph. Furthermore, for a given number of available resources, it is still to be investigated, whether utilizing all compute resources will lead to optimal performance. When introducing additional stages, overhead is generated when merging the intermediate results. Furthermore, the output of the intermediate stages are ultimately affected by the total number of clusters, as they become the new number *nodes* for the next stage. Therefore, it is not clear yet, how the number of available resources, number of stages, sparsity and prior knowledge about the dataset affect the proposed algorithm.

### **Real-Time Tracking**

In Chapter 4, we applied our proposed method on multiple person tracking. This has been done in the past already [85, 149]. Specifically, the tracking problem is treated as clustering tasks, where detection from persons are considered as nodes in graph  $G$ . The graph decomposition is done on the whole dataset, which makes this approach completely offline. In practice, it is desired to have an effective, real-time (online) tracking algorithm. One possible solution would be to set a fixed window size of frames at timestamp  $t - 1$  and solve the *Minimum Cost Multicut Problem*. Then, similar as done in our proposed MSM, the task is repeated at timestamp  $t$  and the intermediate result is merged. Enabling *Minimum Cost Multicut Problem* on real-time tracking problems can be a promising direction of future work.

### **Explainable AI**

As deep neural networks becomes better in specific computer vision tasks, much attention has been given toward explainability of classification models recently [143, 5, 49, 50]. While feature attributions show the effect of each feature on the model prediction, we provided an empirical approach towards explainability of classification models by looking at the embedding space of the model (see Chapter 5). One key finding is the relationship between clusterability and robustness against corruptions. This can be considered as a complementary tool for existing techniques.



# References

- [1] Aggarwal, C. C., Hinneburg, A., and Keim, D. A. (2001). On the surprising behavior of distance metrics in high dimensional space. In *International conference on database theory*, pages 420–434. Springer.
- [2] Andres, B., Kröger, T., Briggman, K. L., Denk, W., Korogod, N., Knott, G., Köthe, U., and Hamprecht, F. A. (2012). Globally optimal closed-surface segmentation for connectomics. In *ECCV*.
- [3] Andriluka, M., Roth, S., and Schiele, B. (2010). Monocular 3d pose estimation and tracking by detection. In *CVPR*.
- [4] Andriyenko, A., Schindler, K., and Roth, S. (2012). Discrete-continuous optimization for multi-target tracking. In *CVPR*.
- [5] Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., and Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, 10(7):e0130140.
- [6] Bansal, N., Blum, A., and Chawla, S. (2004). Correlation clustering. *Machine Learning*, 56(1–3):89–113.
- [7] Beier, T., Andres, B., Köthe, U., and Hamprecht, F. A. (2016). An efficient fusion move algorithm for the minimum cost lifted multicut problem. In *ECCV*.
- [8] Beier, T., Kroeger, T., Kappes, J. H., Kothe, U., and Hamprecht, F. A. (2014). Cut, glue & cut: A fast, approximate solver for multicut partitioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 73–80.
- [9] Bergmann, P., Meinhardt, T., and Leal-Taixe, L. (2019). Tracking without bells and whistles. *arXiv preprint arXiv:1903.05625*.
- [10] Bertasius, G., Wang, H., and Torresani, L. (2021). Is space-time attention all you need for video understanding? *arXiv preprint arXiv:2102.05095*.
- [11] Biggio, B. and Roli, F. (2018). Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331.
- [12] Brasó, G. and Leal-Taixé, L. (2020). Learning a neural solver for multiple object tracking. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6247–6257.

- [13] Carion, N., Massa, F., Synnaeve, G., Usunier, N., Kirillov, A., and Zagoruyko, S. (2020). End-to-end object detection with transformers. In *European Conference on Computer Vision*, pages 213–229. Springer.
- [14] Caron, M., Bojanowski, P., Joulin, A., and Douze, M. (2018). Deep clustering for unsupervised learning of visual features. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 132–149.
- [15] Caron, M., Bojanowski, P., Mairal, J., and Joulin, A. (2019). Unsupervised pre-training of image features on non-curated data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2959–2968.
- [16] Caron, M., Touvron, H., Misra, I., Jégou, H., Mairal, J., Bojanowski, P., and Joulin, A. (2021). Emerging properties in self-supervised vision transformers. *arXiv preprint arXiv:2104.14294*.
- [17] Chari, V., Lacoste-Julien, S., Laptev, I., and Sivic, J. (2015). On pairwise costs for network flow multi-object tracking. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5537–5545.
- [18] Chen, J., Sheng, H., Zhang, Y., and Xiong, Z. (2017). Enhancing detection model for multiple hypothesis tracking. In *Conf. on Computer Vision and Pattern Recognition Workshops*, pages 2143–2152.
- [19] Chen, L., Ai, H., Chen, R., and Zhuang, Z. (2019). Aggregate tracklet appearance features for multi-object tracking. *IEEE Signal Processing Letters*, 26(11):1613–1617.
- [20] Chen, M., Radford, A., Child, R., Wu, J., Jun, H., Luan, D., and Sutskever, I. (2020a). Generative pretraining from pixels. In *International Conference on Machine Learning*, pages 1691–1703. PMLR.
- [21] Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020b). A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.
- [22] Choi, Y., Choi, M., Kim, M., Ha, J.-W., Kim, S., and Choo, J. (2018). Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8789–8797.
- [23] Chopra, S. and Rao, M. (1993). The partition problem. *Mathematical Programming*, 59(1–3):87–115.
- [24] Chu, P. and Ling, H. (2019). Famnet: Joint learning of feature, affinity and multi-dimensional assignment for online multiple object tracking. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 6172–6181.
- [25] Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223.

- [26] Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M. a., Senior, A., Tucker, P., Yang, K., Le, Q., and Ng, A. (2012). Large scale distributed deep networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1223–1231. Curran Associates, Inc.
- [27] Demaine, E. D., Emanuel, D., Fiat, A., and Immorlica, N. (2006). Correlation clustering in general weighted graphs. *Theoretical Computer Science*, 361(2–3):172–187.
- [28] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.
- [29] Djolonga, J., Yung, J., Tschannen, M., Romijnders, R., Beyer, L., Kolesnikov, A., Puigcerver, J., Minderer, M., D’Amour, A., Moldovan, D., et al. (2020). On robustness and transferability of convolutional neural networks. *arXiv preprint arXiv:2007.08558*.
- [30] Doersch, C., Gupta, A., and Efros, A. A. (2015). Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430.
- [31] Dong, X. and Shen, J. (2018). Triplet loss in siamese network for object tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 459–474.
- [32] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.
- [33] Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017.
- [34] Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010). The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338.
- [35] Fard, M. M., Thonet, T., and Gaussier, E. (2020). Deep k-means: Jointly clustering with k-means and learning representations. *Pattern Recognition Letters*, 138:185–192.
- [36] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645.
- [37] Feng, W., Hu, Z., Wu, W., Yan, J., and Ouyang, W. (2019). Multi-object tracking with multiple cues and switcher-aware classification. *arXiv preprint arXiv:1901.06129*.
- [38] Fragkiadaki, K., Zhang, W., Zhang, G., and Shi, J. (2012). Two-granularity tracking: Mediating trajectory and detection graphs for tracking under occlusions. In *ECCV*.
- [39] Geirhos, R., Temme, C. R. M., Rauber, J., Schütt, H. H., Bethge, M., and Wichmann, F. A. (2018). Generalisation in humans and deep neural networks. *arXiv preprint arXiv:1808.08750*.

- [40] Ghasedi, K., Wang, X., Deng, C., and Huang, H. (2019). Balanced self-paced learning for generative adversarial clustering network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4391–4400.
- [41] Ghasedi Dizaji, K., Herandi, A., Deng, C., Cai, W., and Huang, H. (2017). Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization. In *Proceedings of the IEEE international conference on computer vision*, pages 5736–5745.
- [42] Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feed-forward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256.
- [43] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014a). Generative adversarial nets. *Advances in neural information processing systems*, 27.
- [44] Goodfellow, I. J., Shlens, J., and Szegedy, C. (2014b). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*.
- [45] Guo, S., Xu, J., Chen, D., Zhang, C., Wang, X., and Zhao, R. (2020). Density-aware feature embedding for face clustering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6698–6706.
- [46] Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE.
- [47] Haeusser, P., Mordvintsev, A., and Cremers, D. (2017). Learning by association—a versatile semi-supervised training method for neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 89–98.
- [48] Harlap, A., Narayanan, D., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., and Gibbons, P. B. (2018). Pipedream: Fast and efficient pipeline parallel DNN training. *CoRR*, abs/1806.03377.
- [49] Haug, J., Pawelczyk, M., Broelemann, K., and Kasneci, G. (2020). Leveraging model inherent variable importance for stable online feature selection. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1478–1502.
- [50] Haug, J., Zürn, S., El-Jiz, P., and Kasneci, G. (2021). On baselines for local feature attributions. *arXiv preprint arXiv:2101.00905*.
- [51] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.
- [52] Hendrycks, D. and Dietterich, T. (2019). Benchmarking neural network robustness to common corruptions and perturbations. *arXiv preprint arXiv:1903.12261*.

- [53] Hendrycks, D., Mazeika, M., Kadavath, S., and Song, D. (2019). Using self-supervised learning can improve model robustness and uncertainty. In *Advances in Neural Information Processing Systems*, pages 15637–15648.
- [54] Hendrycks, D., Zhao, K., Basart, S., Steinhardt, J., and Song, D. (2021). Natural adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15262–15271.
- [55] Henschel, R., Leal-Taixé, L., Cremers, D., and Rosenhahn, B. (2017). Improvements to frank-wolfe optimization for multi-detector multi-object tracking. *arXiv preprint arXiv:1705.08314*.
- [56] Henschel, R., Leal-Taixé, L., Cremers, D., and Rosenhahn, B. (2018). Fusion of head and full-body detectors for multi-object tracking. In *Computer Vision and Pattern Recognition Workshops (CVPRW)*.
- [57] Henschel, R., Leal-Taixé, L., and Rosenhahn, B. (2014). Efficient multiple people tracking using minimum cost arborescences. In *GCPR*.
- [58] Henschel, R., Zou, Y., and Rosenhahn, B. (2019). Multiple people tracking using body and joint detections. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0.
- [59] Hermans, A., Beyer, L., and Leibe, B. (2017). In defense of the triplet loss for person re-identification. *arXiv preprint arXiv:1703.07737*.
- [60] Ho, K., Chatzimichailidis, A., Keuper, M., and Keuper, J. (2021a). Msm: Multi-stage multicuts for scalable image clustering. In *International Conference on High Performance Computing*, pages 267–284. Springer.
- [61] Ho, K., Chatzimichailidis, A., Pfreundt, F.-J., Keuper, J., and Keuper, M. (2021b). Estimating the robustness of classification models by the structure of the learned feature-space. In *The AAAI-22 Workshop on Adversarial Machine Learning and Beyond*.
- [62] Ho, K., Kardoost, A., Pfreundt, F.-J., Keuper, J., and Keuper, M. (2020a). A two-stage minimum cost multicut approach to self-supervised multiple person tracking. In *Proceedings of the Asian Conference on Computer Vision*.
- [63] Ho, K., Keuper, J., and Keuper, M. (2020b). Unsupervised multiple person tracking using autoencoder-based lifted multicuts. *arXiv preprint arXiv:2002.01192*.
- [64] Ho, K., Keuper, J., Pfreundt, F.-J., and Keuper, M. (2020c). Learning embeddings for image clustering: An empirical study of triplet loss approaches. *arXiv preprint arXiv:2007.03123*.
- [65] Hornakova, A., Henschel, R., Rosenhahn, B., and Swoboda, P. (2020). Lifted disjoint paths with application in multiple object tracking. *arXiv preprint arXiv:2006.14550*.
- [66] Horňáková, A., Lange, J.-H., and Andres, B. (2017). Analysis and optimization of graph decompositions by lifted multicuts. In *ICML*.

- [67] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.
- [68] Hu, J., Shen, L., and Sun, G. (2018). Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141.
- [69] Huang, C., Wu, B., and Nevatia, R. (2008). Robust object tracking by hierarchical association of detection responses. In *ECCV*.
- [70] Huang, G., Liu, Z., Van Der Maaten, L., and Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.
- [71] Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q. V., and Chen, Z. (2018). Gpipe: Efficient training of giant neural networks using pipeline parallelism. *CoRR*, abs/1811.06965.
- [72] Insafutdinov, E., Pishchulin, L., Andres, B., Andriluka, M., and Schieke, B. (2016). Deepcut: A deeper, stronger, and faster multi-person pose estimation model. In *European Conference on Computer Vision (ECCV)*.
- [73] Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR.
- [74] Jain, H., Grover, R., and LIET, A. (2017). Clustering analysis with purity calculation of text and sql data using k-means clustering algorithm. *IJAPRR*, 4(44557):47–58.
- [75] Jakobovitz, D. and Giryes, R. (2018). Improving dnn robustness to adversarial attacks using jacobian regularization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 514–529.
- [76] Ji, P., Zhang, T., Li, H., Salzmann, M., and Reid, I. (2017). Deep subspace clustering networks. In *Advances in Neural Information Processing Systems*, pages 24–33.
- [77] Jing, L. and Tian, Y. (2019). Self-supervised visual feature learning with deep neural networks: A survey. *arXiv preprint arXiv:1902.06162*.
- [78] Kardoost, A., Ho, K., Ochs, P., and Keuper, M. (2020). Self-supervised sparse to dense motion segmentation. In *Proceedings of the Asian Conference on Computer Vision*.
- [79] Kardoost, A. and Keuper, M. (2018). Solving minimum cost lifted multicut problems by node agglomeration. In *ACCV 2018, 14th Asian Conference on Computer Vision*, Perth, Australia.
- [80] Karmaker, S. K., Hassan, M. M., Smith, M. J., Xu, L., Zhai, C., and Veeramachaneni, K. (2021). Automl to date and beyond: Challenges and opportunities. *ACM Computing Surveys (CSUR)*, 54(8):1–36.
- [81] Kawaguchi, K. (2016). Deep learning without poor local minima. *Advances in neural information processing systems*, 29.

- [82] Kawaguchi, K. and Kaelbling, L. (2020). Elimination of all bad local minima in deep learning. In *International Conference on Artificial Intelligence and Statistics*, pages 853–863. PMLR.
- [83] Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell system technical journal*, 49(2):291–307.
- [84] Keuper, M., Levinkov, E., Bonneel, N., Lavoué, G., Brox, T., and Andres, B. (2015). Efficient decomposition of image and mesh graphs by lifted multicuts. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1751–1759.
- [85] Keuper, M., Tang, S., Andres, B., Brox, T., and Schiele, B. (2018). Motion segmentation & multiple object tracking by correlation co-clustering. *IEEE transactions on pattern analysis and machine intelligence*, 42(1):140–153.
- [86] Keuper, M., Tang, S., Zhongjie, Y., Andres, B., Brox, T., and Schiele, B. (2016). A multi-cut formulation for joint segmentation and tracking of multiple objects. *arXiv preprint arXiv:1607.06317*.
- [87] Kim, C., Li, F., Ciptadi, A., and Rehg, J. M. (2015). Multiple hypothesis tracking revisited. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4696–4704.
- [88] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
- [89] Kolesnikov, A., Beyer, L., Zhai, X., Puigcerver, J., Yung, J., Gelly, S., and Houlsby, N. (2020). Big transfer (bit): General visual representation learning. In *European conference on computer vision*, pages 491–507. Springer.
- [90] Kolesnikov, A., Zhai, X., and Beyer, L. (2019). Revisiting self-supervised visual representation learning. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1920–1929.
- [91] Kornblith, S., Lee, H., Chen, T., and Norouzi, M. (2020). What’s in a loss function for image classification? *arXiv preprint arXiv:2010.16402*.
- [92] Krizhevsky, A. (2014). One weird trick for parallelizing convolutional neural networks. *CoRR*, abs/1404.5997.
- [93] Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.
- [94] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105.
- [95] Kuhn, H. W. (2005). The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 52(1):7–21.
- [96] Kumar, R., Charpiat, G., and Thonnat, M. (2014). Multiple object tracking by efficient graph partitioning. In *Asian Conference on Computer Vision*, pages 445–460. Springer.

- [97] Kurakin, A., Goodfellow, I., and Bengio, S. (2016). Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*.
- [98] Leal-Taixé, L., Canton-Ferrer, C., and Schindler, K. (2016). Learning by tracking: Siamese cnn for robust target association. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 33–40.
- [99] Leal-Taixé, L., Milan, A., Reid, I., Roth, S., and Schindler, K. (2015). MOTChallenge 2015: Towards a benchmark for multi-target tracking. *arXiv:1504.01942 [cs]*. arXiv: 1504.01942.
- [100] LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995.
- [101] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.
- [102] Lee, W., Na, J., and Kim, G. (2019). Multi-task self-supervised object detection via recycling of bounding box annotations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4984–4993.
- [103] Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T. (2018a). Visualizing the loss landscape of neural nets. *Advances in neural information processing systems*, 31.
- [104] Li, M., Zhu, X., and Gong, S. (2018b). Unsupervised person re-identification by deep learning tracklet association. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 737–753.
- [105] Li, P., Zhao, H., and Liu, H. (2020). Deep fair clustering for visual learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9070–9079.
- [106] Liu, C., Zoph, B., Neumann, M., Shlens, J., Hua, W., Li, L.-J., Fei-Fei, L., Yuille, A., Huang, J., and Murphy, K. (2018a). Progressive neural architecture search. In *Proceedings of the European conference on computer vision (ECCV)*, pages 19–34.
- [107] Liu, H., Simonyan, K., and Yang, Y. (2018b). Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.
- [108] Liu, Z., Luo, P., Wang, X., and Tang, X. (2015). Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*.
- [109] Luo, W., Xing, J., Milan, A., Zhang, X., Liu, W., Zhao, X., and Kim, T.-K. (2014). Multiple object tracking: A literature review. *arXiv preprint arXiv:1409.7618*.
- [110] Lv, J., Chen, W., Li, Q., and Yang, C. (2018). Unsupervised cross-dataset person re-identification by transfer learning of spatial-temporal patterns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 7948–7956.
- [111] Ma, C., Yang, C., Yang, F., Zhuang, Y., Zhang, Z., Jia, H., and Xie, X. (2018). Trajectory factory: Tracklet cleaving and re-connection by deep siamese bi-gru for multiple object tracking. *arXiv preprint arXiv:1804.04555*.



- [112] Maaten, L. v. d. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605.
- [113] Mahendran, A., Thewlis, J., and Vedaldi, A. (2018). Cross pixel optical-flow similarity for self-supervised learning. In *Asian Conference on Computer Vision*, pages 99–116. Springer.
- [114] McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.
- [115] Milan, A., Leal-Taixé, L., Reid, I., Roth, S., and Schindler, K. (2016). MOT16: A benchmark for multi-object tracking. *arXiv:1603.00831 [cs]*. arXiv: 1603.00831.
- [116] Mirkes, E. M., Allohbi, J., and Gorban, A. (2020). Fractional norms and quasinorms do not help to overcome the curse of dimensionality. *Entropy*, 22(10):1105.
- [117] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. (2016). Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582.
- [118] Mukherjee, S., Asnani, H., Lin, E., and Kannan, S. (2019). Clustergan: Latent space clustering in generative adversarial networks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 4610–4617.
- [119] Noroozi, M. and Favaro, P. (2016). Unsupervised learning of visual representations by solving jigsaw puzzles. In *European conference on computer vision*, pages 69–84. Springer.
- [120] Oh Song, H., Xiang, Y., Jegelka, S., and Savarese, S. (2016). Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4004–4012.
- [121] Ovadia, Y., Fertig, E., Ren, J., Nado, Z., Sculley, D., Nowozin, S., Dillon, J. V., Lakshminarayanan, B., and Snoek, J. (2019). Can you trust your model’s uncertainty? evaluating predictive uncertainty under dataset shift. *arXiv preprint arXiv:1906.02530*.
- [122] Pan, X., Papailiopoulos, D., Oymak, S., Recht, B., Ramchandran, K., and Jordan, M. I. (2015). Parallel correlation clustering on big graphs. *arXiv preprint arXiv:1507.05086*.
- [123] Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, L., Shazeer, N., Ku, A., and Tran, D. (2018). Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR.
- [124] Pathak, D., Girshick, R., Dollár, P., Darrell, T., and Hariharan, B. (2017). Learning features by watching objects move. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2701–2710.
- [125] Pathak, D., Krahenbuhl, P., Donahue, J., Darrell, T., and Efros, A. A. (2016). Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2536–2544.

- [126] Pirsiavash, H., Ramanan, D., and Fowlkes, C. C. (2011). Globally-optimal greedy algorithms for tracking a variable number of objects. In *CVPR*.
- [127] Pishchulin, L., Insafutdinov, E., Tang, S., Andres, B., Andriluka, M., Gehler, P. V., and Schiele, B. (2016). Deepcut: Joint subset partition and labeling for multi person pose estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4929–4937.
- [128] Prasad, V., Das, D., and Bhowmick, B. (2020). Variational clustering: Leveraging variational autoencoders for image clustering. In *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10. IEEE.
- [129] Quiñero-Candela, J., Sugiyama, M., Lawrence, N. D., and Schwaighofer, A. (2009). *Dataset shift in machine learning*. Mit Press.
- [130] Recht, B., Roelofs, R., Schmidt, L., and Shankar, V. (2019). Do imagenet classifiers generalize to imagenet? In *International Conference on Machine Learning*, pages 5389–5400. PMLR.
- [131] Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.
- [132] Revaud, J., Weinzaepfel, P., Harchaoui, Z., and Schmid, C. (2015). Deep convolutional matching. *CoRR*, abs/1506.07656.
- [133] Ronneberger, O., Fischer, P., and Brox, T. (2015). U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer.
- [134] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252.
- [135] Saikia, T., Schmid, C., and Brox, T. (2021). Improving robustness against common corruptions with frequency biased models. *arXiv preprint arXiv:2103.16241*.
- [136] Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.
- [137] Sharif Razavian, A., Azizpour, H., Sullivan, J., and Carlsson, S. (2014). Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813.
- [138] Shen, H., Huang, L., Huang, C., and Xu, W. (2018). Tracklet association tracker: An end-to-end learning-based association approach for multi-object tracking. *arXiv preprint arXiv:1808.01562*.
- [139] Sheng, H., Chen, J., Zhang, Y., Ke, W., Xiong, Z., and Yu, J. (2018a). Iterative multiple hypothesis tracking with tracklet-level association. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1.

- [140] Sheng, H., Zhang, Y., Chen, J., Xiong, Z., and Zhang, J. (2018b). Heterogeneous association graph fusion for target association in multiple object tracking. *IEEE Transactions on Circuits and Systems for Video Technology*, 29(11):3269–3280.
- [141] Shin, H.-C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., and Summers, R. M. (2016). Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, 35(5):1285–1298.
- [142] Shitrit, H. B., Berclaz, J., Fleuret, F., and Fua, P. (2011). Tracking multiple people under global appearance constraints. In *ICCV*.
- [143] Simonyan, K., Vedaldi, A., and Zisserman, A. (2013). Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*.
- [144] Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.
- [145] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31.
- [146] Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., and Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*.
- [147] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. (2019). Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2820–2828.
- [148] Tang, S., Andres, B., Andriluka, M., and Schiele, B. (2016). Multi-person tracking by multicut and deep matching. In *European Conference on Computer Vision*, pages 100–111. Springer.
- [149] Tang, S., Andriluka, M., Andres, B., and Schiele, B. (2017). Multiple people tracking by lifted multicut and person re-identification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3539–3548.
- [150] Tang, S., Andriluka, M., and Schiele, B. (2014). Detection and tracking of occluded people. *IJCV*.
- [151] Tesfaye, Y. T., Zemene, E., Pelillo, M., and Prati, A. (2016). Multi-object tracking using dominant sets. *IET Computer Vision*, 10(4):289–297.
- [152] Tian, F., Gao, B., Cui, Q., Chen, E., and Liu, T.-Y. (2014). Learning deep representations for graph clustering. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*.
- [153] Touvron, H., Cord, M., Douze, M., Massa, F., Sablayrolles, A., and Jégou, H. (2020). Training data-efficient image transformers & distillation through attention. *arXiv preprint arXiv:2012.12877*.

- [154] Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9(11).
- [155] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *arXiv preprint arXiv:1706.03762*.
- [156] Vincent, P., Larochelle, H., Lajoie, I., Bengio, Y., Manzagol, P.-A., and Bottou, L. (2010). Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of machine learning research*, 11(12).
- [157] Vondrick, C. M., Shrivastava, A., Fathi, A., Guadarrama, S., and Murphy, K. (2018). Tracking emerges by colorizing videos.
- [158] Wang, X., Turetken, E., Fleuret, F., and Fua, P. (2014). Tracking interacting objects optimally using integer programming. In *ECCV*.
- [159] Weinzaepfel, P., Revaud, J., Harchaoui, Z., and Schmid, C. (2013). Deepflow: Large displacement optical flow with deep matching. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1385–1392.
- [160] Wojek, C., Roth, S., Schindler, K., and Schiele, B. (2010). Monocular 3d scene modeling and inference: Understanding multi-object traffic scenes. In *ECCV*.
- [161] Wojek, C., Walk, S., Roth, S., Schindler, K., and Schiele, B. (2013). Monocular visual scene understanding: Understanding multi-object traffic scenes. *IEEE TPAMI*.
- [162] Wolf, S., Bailoni, A., Pape, C., Rahaman, N., Kreshuk, A., Köthe, U., and Hamprecht, F. A. (2020). The mutex watershed and its objective: Efficient, parameter-free graph partitioning. *IEEE transactions on pattern analysis and machine intelligence*.
- [163] Wu, C.-Y., Manmatha, R., Smola, A. J., and Krahenbuhl, P. (2017). Sampling matters in deep embedding learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2840–2848.
- [164] Xie, J., Girshick, R., and Farhadi, A. (2016). Unsupervised deep embedding for clustering analysis. In *International conference on machine learning*, pages 478–487. PMLR.
- [165] Yang, B., Fu, X., Sidiropoulos, N. D., and Hong, M. (2017). Towards k-means-friendly spaces: Simultaneous deep learning and clustering. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3861–3870. JMLR. org.
- [166] Yang, F., Choi, W., and Lin, Y. (2016). Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2129–2137.
- [167] Ye, Q., Zhang, T., Ke, W., Qiu, Q., Chen, J., Sapiro, G., and Zhang, B. (2017). Self-learning scene-specific pedestrian detectors using a progressive latent model. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 509–518.

- [168] Yoon, Y.-c., Boragule, A., Song, Y.-m., Yoon, K., and Jeon, M. (2018). Online multi-object tracking with historical appearance matching and scene adaptive detection filtering. In *2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, pages 1–6. IEEE.
- [169] Zamir, A. R., Dehghan, A., and Shah, M. (2012). Gmcp-tracker: Global multi-object tracking using generalized minimum clique graphs. In *Computer Vision—ECCV 2012*, pages 343–356. Springer.
- [170] Zhan, X., Xie, J., Liu, Z., Ong, Y.-S., and Loy, C. C. (2020). Online deep clustering for unsupervised representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6688–6697.
- [171] Zhang, S., Gong, Y., and Wang, J. (2016). Deep metric learning with improved triplet loss for face clustering in videos. In *Pacific Rim Conference on Multimedia*, pages 497–508. Springer.
- [172] Zhang, X., Li, Z., Change Loy, C., and Lin, D. (2017). Polynet: A pursuit of structural diversity in very deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 718–726.
- [173] Zhuang, B., Lin, G., Shen, C., and Reid, I. (2016). Fast training of triplet-based deep binary embedding networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5955–5964.
- [174] Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710.

