Zurich University
of Applied Sciences

School of
Engineering

InIT Institute of Applied
Information Technology

Master Thesis
Fall 2017

Learning to Cluster – Investigations in Deep Learning for End-to-End Clustering

Author Benjamin Meier

Supervisors Dr. Thilo Stadelmann
Dr. Oliver Dürr

Date February 28, 2018

DECLARATION OF ORIGINALITY

Master's Thesis for the School of Engineering

By submitting this Master's thesis, the student attests of the fact that all the work included in the assignment is their own and was written without the help of a third party.

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree Courses at the Zurich University of Applied Sciences (dated 29 Januar 2008) and subject to the provisions for disciplinary action stipulated in the University regulations (Rahmenprüfungsordnung ZHAW (RPO)).

Town/City, Date: Signature:

…… ……..

The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all Master's theses submitted.

Zusammenfassung

Viele Anwendungen, die auf grossen Datenmengen basieren, nutzen Clustering-Algorithmen, um die Daten in verschiedene Gruppen zu unterteilen. Die Grössen dieser Gruppen können stark variieren und haben je nach Anwendungsfall eine komplett unterschiedliche Bedeutung: Beispielsweise ein Online-Händler könnte seine Kunden in 5 verschiedene Gruppen unterteilen, wobei jede Gruppe ein unterschiedliches Kaufverhalten hat. Für diese Gruppen könnte nun jeweils eine eigene und gezielte Marketing-Strategie angewendet werden. Eine Cluster-Analyse mit Audio-Daten kann genutzt werden, um grosse Musiksammlungen zu gruppieren, um dann beispielsweise Nutzern einer Musikplattform ähnliche Musik, die sie noch nicht gehört haben, zu empfehlen. Hierzu können auch die Nutzerprofile selber geclustert werden und Empfehlungen aufgrund von Nutzerprofilen im gleichen Cluster gemacht werden. Cluster-Analysen bei Text-Daten können angewendet werden, um Dokumente zu organisieren, um ähnliche Patente zu finden oder auch um einem Anwalt bei der Recherche zu helfen.

Wie wird nun eine solche Cluster-Analyse durchgeführt? Bisherige Ansätze basieren hauptsächlich auf manuell angefertigten Features für spezifische Datentypen (Bilder, Audio, ...) und der nachfolgenden Anwendung von Algorithmen wie k-means oder DBSCAN. Diese Algorithmen funktionieren für niedrig-dimensionale Daten sehr gut, jedoch wird bei hoch-dimensionalen Daten das notwendige Feintuning bei der Dimensionsreduktion und für die notwendigen Hyperparameter sehr schnell kompliziert und aufwendig. Neuere Ansätze basieren auf künstlichen neuronalen Netzen, welche unter anderem sehr effektiv in der Dimensionsreduktion sind. Oftmals bilden die neuronalen Netzwerke jedoch nur einen Teil des Gesamtproblemes ab.

In dieser Arbeit wird eine end-to-end Lösung für das Clustering-Problem basierend auf einem neuronalen Netzwerk untersucht. Das heisst der gesamte Clustering-Task wird ausschliesslich von einem neuronalen Netzwerk ausgeführt. Es wird eine Architektur vorgeschlagen, die viele verschiedene Datentypen clustern kann: Es werden signifikante Features für das definierte Ähnlichkeitskriterium mit einem Embedding-Netzwerk extrahiert und während dem supervised Training angewendet, um die Daten zu clustern. Für das Training des Clusterings wird dementsprechend dem Netzwerk anhand von Beispielen spezifiziert was ein Cluster ist und das Netzwerk lernt dann wie Cluster auf den entsprechenden Daten gebildet werden. Dies erlaubt es zu definieren was genau ein Cluster ist: Beispielsweise kann bei Gesichtern nach dem abgebildeten Menschen geclustert werden, oder nach dem Gesichtsaudruck (fröhlich/lächelnd, traurig/weinend, etc.). Audio-Daten können nach dem Sprecher oder nach dem Inhalt geclustert werden. Bei der Evaluierung werden ähnliche Datenpunkte bei einer unbekannten Anzahl von Clustern gruppiert.

Die beschriebene Architektur wird mit verschiedenen Datensets evaluiert: Unter anderem mit 2D-Punktdaten, TIMIT (Sprecher-Clustering) und COIL-100 (Bild-Clustering). Für die 2D-Punktdaten und TIMIT werden gute Resultate erzielt und für andere Datensets kann gezeigt werden, dass das beschriebene Netzwerk in der Lage ist, die Daten zu clustern, wobei die state-of-the-art Resultate für die einzelnen Datensets jeweils noch höher sind.

Abstract

Many data-driven applications use clustering-algorithms to detect different groups in large or small datasets: For instance, an online shop could cluster the customers in 5 different groups based on their buying behavior. Different marketing strategies can then be applied to each group. Clustering for audio data can be applied to group large collections of music. This can be helpful for music platforms to recommend similar music to a user that he/she has not heard about yet and probably likes. This also can be achieved by clustering the user profiles: Recommendations can be done by analyzing what music others in the same cluster listen to. Text clustering is helpful for organizing lots of documents, to help a lawyer find similar patents or the correct law, and similar cases.

How is such a clustering conducted? The currently used approaches are often based on hand-crafted features for specific data types (images, audio, etc.) combined with algorithms like k-means or DBSCAN. These algorithms work well on low-dimensional data types, but the required fine-tuning for the dimension-reduction and the hyperparameters quickly are very complex and time-intensive for high-dimensional data. Newer approaches are based on artificial neural networks which are very effective in the task of dimension-reduction. Often only a part of the problem is conducted by a neural network and not the complete task.

This thesis is about an end-to-end solution for the clustering task based on a neural network. This means the complete clustering task is conducted by a neural network. An architecture is proposed which is able to cluster different data types: Salient features for any similarity criterion specified through weakly labeled training data are extracted by an embedding network trained in a supervised fashion. Therefore, by some example clusters, the proposed network learns what exactly a cluster for a given data type is. This knowledge is used to cluster unknown examples. This method allows it to specify what a cluster is: E.g., for faces one may cluster them by the person and another use case may require it to cluster these faces according to the facial expression. Spoken text (audio) could be clustered according to the speaker or according to the content. During the unsupervised evaluation, the network groups similar data points of any modality into a previously unknown number of clusters.

The proposed architecture is evaluated on several datasets, including 2D-point data, TIMIT (speaker clustering) and COIL-100 (image clustering). For 2D-point data and TIMIT, the architecture reaches promising results, and for other datasets, it can be shown that the network is able to do clustering, but it does not reach state-of-the-art results.

Management Review

In this thesis, it is evaluated if the proposed end-to-end trainable neural network is able to perform clustering of different data types. Clustering high-dimensional data like images or audio is a difficult task, even more if a specific feature should be used for the clustering. Often, it is hard to extract this specific feature or even to formalize it exactly: E.g., it is not that easy to formalize what exactly identifies the speaker in a given piece of audio or how to identify a specific person's face on an image. In these cases it can be a big advantage by using neural networks. End-to-end solutions go a step further and allow an optimization of the complete process.

The proposed solution allows it to use clustering for many data types and specific features, if a dataset with example clusterings is available. If this is the case, the proposed architecture may be trained within a few hours up to a few days. The training has to be done only once. Other solutions do not require a dataset with example clusterings, but then it is not possible to define the features which should be used for the clustering process. For low-dimensional data, this is not a big problem, but with high-dimensional data it is often not that obvious which feature(s) should be used to conduct a clustering: E.g., a given set of images could be clustered according to objects on them (different animals, cars, etc.) or according to the place/environment (city, forest, etc.) on the images. For most data types and datasets, there is not just one single unique and natural feature set available that can be used for clustering tasks.

One might differ between two types of clustering tasks: Finding out something about a given dataset without any labels and prior knowledge, and analyzing a given dataset where are some labels available and one knows what high-level features are interesting for the clustering task. The proposed solution might be used to conduct clusterings, given the interested high-level features are known and they can be described by providing a datasets with example clusterings. Therefore, the proposed method cannot be applied if no prior knowledge (example clusterings) are available.

By using the proposed architecture, many use cases may be simplified: E.g., clustering audio data for grouping music, clustering text for finding similar documents or clustering images for grouping faces, humans, animals, environments etc. Compared to other solutions, the proposed architecture is more flexible and less fine-tuning is required. This might reduce the development time by a huge factor: The similarity of different examples can be defined by just providing example clusterings. Then, the neural network is able to learn how to extract the required features and to conduct the clustering. No formal specification is required.

The network architecture may be trained on a limited dataset (e.g. a dataset of faces) and can then be applied to any other set of face images, also if the persons on the images were not in the training dataset. If already a network was trained for a similar dataset, then the training time may decrease by using the already trained model as initial model.

Nevertheless, more specific clustering methods may reach higher accuracies for a given data type. As can be seen in this thesis, clustering high-dimensional data works well, but the state-of-the-art quality is not reached. On the other side, the proposed architecture can be used for many different data types and the state-of-the-art which is used as a comparison often only works for one specific data type and use case. Therefore, the proposed solution may be especially helpful to improve the development time if a general clustering algorithm is required or if no specific clustering method for a given data type or use case exists. Of course, it is also required that some example clusterings exists.

As can be shown if the clusters represent some kind of class (e.g., speakers in speaker clustering) and the data is complex (e.g., spoken text), then there are about 300-400 classes required to train the network and to reach a high quality. Simpler data types, like the COIL-100 images, require less classes to reach good results. This might be a limiting factor, because it can be very costly to create such a dataset. On the other side, there are already many datasets for free available which can be used to solve many clustering problem statements with the described method. Another possibility is to retrain a model that was trained on a similar but larger dataset. This might decrease the amount of the required data by a large factor.

The runtime performance of the described method was not focused during the thesis. Given the network is already trained, which might take up to a few days, the network is quite efficient: E.g., clustering 20 spoken texts snippets with a length of 1.28 seconds (audio data) by the speaker requires only a few milliseconds. Different data types may require a bit more or less time, but in general, the performance is already quite well. It still can be improved if one focus more on this aspect.

The public available source code of the proposed model allows to easily conduct tests which is helpful while evaluating if a given use case may be solved by the proposed neural network architecture. Only minimal coding skills are required to include a given dataset, to train the model and to finally test if it works well. Many different metrics and possibilities to analyze the final conducted results are included in the software. This allows it to quickly test if the proposed solution may be helpful to solve a given problem statement.

Finally, it can be said that the proposed model has limitations, especially because much data is required to train the model, but as can be shown in this thesis it already can be used to cluster 2D-point data, spoken text and some images. Other clustering tasks probably also work well. This might be quickly evaluated by using the existing software. If the clustering task with the proposed model for a given use case works, it may decrease the development time and therefore reduce the costs.

Preface

First and foremost, I want to thank my supervisors Dr. Thilo Stadelmann and Dr. Oliver Dürr. Both did an excellent work.

The author of this work is not only very interested in the use of neural networks but also in clustering. Therefore, this thesis is the next logical step: A work about an end-to-end neural network-based approach to solving the clustering problem. Dr. Thilo Stadelmann had the initial core idea for such a neural network.

Dr. Thilo Stadelmann and Dr. Oliver Dürr were very helpful in evaluating all ideas, also at a low level, and in fixing mathematical problems. Both supervisors helped to get and evaluate new ideas and to solve problems during the thesis. This thesis would not be possible without the help that they provided me.

I also like to thank the ZHAW Datalab for providing the computational resources. This was extremely helpful to evaluate new ideas very quickly and to conduct many tests.

Finally, I want to thank the software developers of TensorFlow, Keras, and all the other libraries and software products that I used. These libraries made it much easier to implement all the ideas.

Contents

1. Introduction

1.1. Motivation

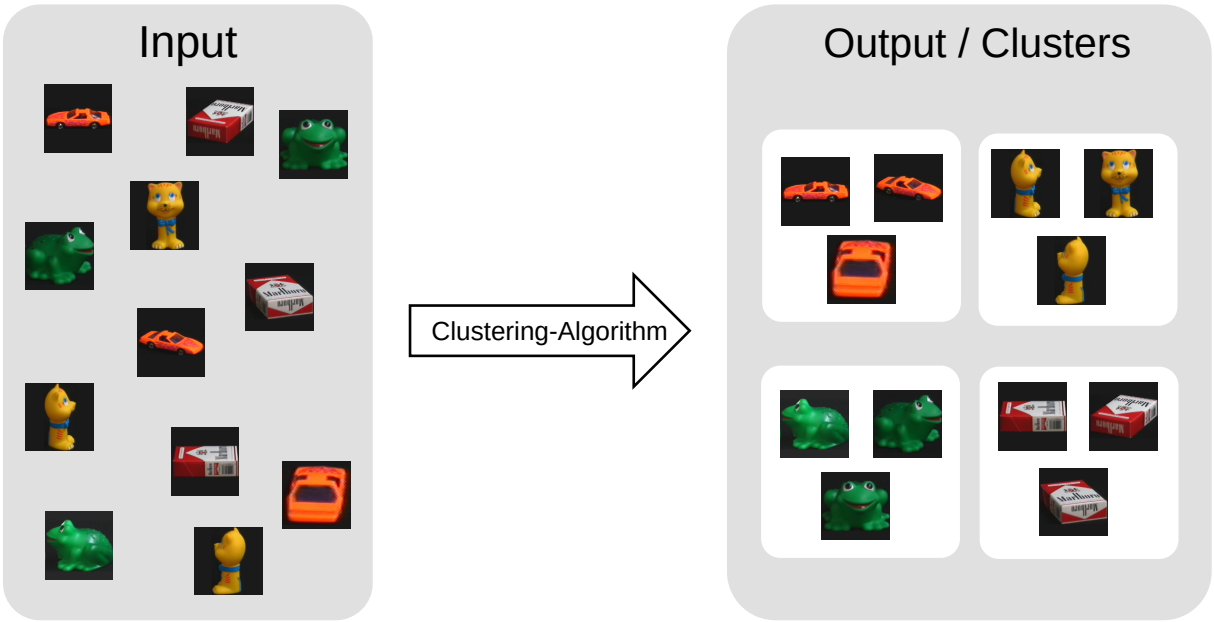


Figure 1.: An example clustering for 12 images of the COIL100 [1] dataset.

What exactly is cluster analysis? This problem statement defines that a set of examples is grouped in a sense that examples in the same group are more similar than examples in different groups. Such groups are called clusters. The similarity measure might be very difficult depending on the data and is a key point for the clustering problem. An example of an input and the output of a clustering algorithm is shown in Figure 1.

In the last decades, there were many attempts made to solve the clustering problem. Many different algorithms [2] [3] [4] [5] exist which sometimes only work on a specific type of data [6] [7]. The clustering problem is very general and may be applied for almost any data type, e.g., for images [8] [9] [10], audio data [11] [12] [13], point data [14], or even text [15] [16] [17]. Often a distance measure (e.g. for hierarchical clustering [18]) and many other parameters are required to obtain good results. High dimensional data often requires a dimension reduction to be used with these algorithms [12] [19]. Therefore, one has to solve the problems of the dimension reduction for the specific data type, the distance measure, and the right clustering algorithm. Popular dimension reduction methods are PCA [20] [21], handcrafted features [22] [23] [24] or embeddings generated by neural networks [12] [25]. PCA [26] can only do linear transformations which are often not sufficient; hand-crafted features are often very hard to obtain, and it is not always sure that they are even optimal [27]. Hand-crafted features are being increasingly replaced by learned features, e.g. for classification tasks [28] [29] [30] and object detection tasks [31] [32] [33] [34]. Low dimensional representations generated by neural networks are often obtained by training a classification network [35], an auto-encoder [36] [37], or more advanced techniques [12]. It may be asked whether this representation is optimal for the clustering task? As can be shown [38], different objects of the same class but of different subclasses may have a very different representation in neural networks up until the very last layer. Therefore, the previous representations may not be very well suitable for clustering. It may also be asked for which clustering algorithm a representation is good, and why it is good? These questions are hard to answer for many representations, especially for representations learned by neural networks.

Nevertheless, state-of-the-art clustering methods reach very accurate results [39] [40], including neural network-based approaches [12] [14].

Conventional clustering algorithms can be categorized into hierarchical and partitional approaches. k-means [3] and expectation maximization (EM) [4] are the best-known partitional algorithms, and agglomerative clustering is a well-known hierarchical clustering approach [41] [42] [43], but there are also other hierarchical clustering approaches proposed [44] [45]. Another possibility is the Hidden Markov Model (HMM)-based models, e.g. the one proposed by Lin et al. [46]. It can be shown that the quality of these algorithms for different problem statements may be very different [47] [48]. Therefore, the best-matching clustering algorithm for a given use case highly depends on the data.

For neural network-based clustering algorithms, there are two major categories: (semi-) supervised methods [12] [14] [49] and unsupervised methods [9] [10]. Unsupervised methods do not use any labels to learn the clustering. This is an advantage because labels are often expensive to obtain. On the other hand, one has to trust the neural network to learn the low-dimensional representation that contains the relevant information. For instance, when images that contain animals in the wild are clustered by using an unsupervised method, does the network focus on the animals or on the background (forest, sea, etc.)? Depending on the given use case, the aspect used for clustering the data may be different, even within the same dataset. For example, one may like to cluster the audio according to the voice [12] [50] for a speaker-recognition task [51] and something else according to its content [52] [53] for a speech recognition task. This means that it is in general not possible to deduce a unique, natural, and correct similarity function or clustering just based on the data. Therefore, unsupervised clustering may not work for any use case because the method used decides the criterion in clustering the data.

Because there are so many clustering algorithms available, and their quality often highly depends on the data type used, it is important to use a well-performing algorithm for a specific task. The clustering quality therefore requires a metric. Unfortunately, there is not a single overall natural metric available such as for the accuracy of classification problems. Therefore, there are many different metrics proposed to measure the quality of a clustering result [50] [54] [55] [56] and which compared to one another [57] [58].

1.2. Requirements for Readers of this Thesis

For a better understanding of the proposed neural network architecture and the thesis in general, it is required that the reader understands what artificial neural networks [59] are and how they function. It is also assumed that the reader roughly knows what an LSTM layer [60] or at least a Recurrent Neural Network (RNN) [61] is.

It is preferred if the reader is already familiar with some classical clustering algorithms (e.g., k-means [3] and DBSCAN [62]) and convolutional neural networks (CNNs) [63], but this is not a strict requirement.

1.3. Structure

In this thesis, a supervised end-to-end neural network architecture is presented that uses a set of examples as input and directly produces a set of clusters as the output. Therefore, the embeddings are trained at the same time as the differentiable clustering "algorithm" itself. The process of creating this architecture is only roughly described because the thesis focuses more on the details of the final model and the clustering quality that can be achieved. Some of the

additionally tested models and loss functions are also roughly explained.

The novel neural network model is described in Chapter 3, and two alternative models which were evaluated but not finally used are listed in Chapter 4. They may be used as in inspiration for further developments. The specialized loss function used is explained in Chapter 5. Possible regularizations and alternative loss functions are described in Chapter 6. These regularizations and loss-terms may be helpful when the architecture is extended. The proposed model requires an embedding network which may vary for different input data types: Some possible embedding networks are described in Chapter 7. Once the model is explained, the training procedure and some details are described in Chapter 8, followed by an introduction to the evaluation methods used in Chapter 9. Some of the datasets implemented and used are then described in Chapter 10. Then there are some experiments which are conducted on the datasets, described in Chapter 11. The results finally obtained are summarized in Chapter 12. A brief description about how the software implementation is done is given in Chapter 13. This allows third persons to reproduce the experiments, extend the code, and to perform further experiments. Finally, there are some conclusions in Chapter 14 followed by discussions in Chapter 15.

2. Related Work

2.1. Clustering Based on Neural Networks

Several clustering approaches based on neural networks exist. Lukic et al. [12] train a neural network on a given TIMIT dataset [64] based on the approach described by Hsu et al. [65]. The supervised speaker clustering method described reaches state-of-the-art results using trained embedding and a conventional hierarchical clustering approach for the given data. However, the embedding used is not explicitly trained for the given task, and it is used in combination with an external clustering algorithm, therefore the training is not conducted in an end-to-end fashion.

Guo et al. [36] use an auto-encoder to generate embeddings which are used for a clustering algorithm. An additional loss, based on class labels, is used for the training. State-of-the-art results can be reached for the MNIST [66] dataset. Peng et al. [67] and Tian et al. [68] also use an auto-encoder based method to learn an embedding and then use a conventional clustering algorithm. Auto-encoder-based embeddings may contain too much information because they try to compress the complete input so that the entire content can be decompressed again. Often there are only a very few aspects which are relevant for the clustering, and they might not even be extremely relevant to reproducing the exact input. So, the embedding may be much smaller and more effective if other training approaches are used.

Kampffmeyer et al. [10] use unsupervised learning methods based on a neural network to solve the clustering task. The proposed method works very well for MNIST [66]. On the other side, it is not possible to give hints to the network about which aspect should be used for the clustering task. Therefore, the network itself chooses a set of features to use for the clustering task. This means that the network decides on the meaning of similar inputs. The approach requires that the network is trained for each set of data that should be clustered or at least, that all possible classes/clusters are available in the training set. Therefore, the clustering process itself is performed during the training. Additionally, the exact number of clusters has to be known because the trained network finally takes one input example, independent of all other examples, to predict the cluster index. Mahkzani et al. [9] use a similar architecture to predict the cluster index. On the other hand, the proposed approach in this thesis does not require that the training data contains all classes, and it does not have to be retrained for new classes. The exact number of clusters does not have been known, but before training the network, a limited range of possible clusters per input for the network must be known.

Yang et al. [49] present a neural network-based clustering approach. They use a task-specific CNN embedding network and an agglomerative clustering-based method to cluster image data. They interpret the agglomerative clustering as a recurrent process which allows an optimization of the entire network, which, therefore, can be used to optimize the embedding. The proposed method does not only work for images, but it is more general. Another big difference to the method proposed in this thesis is the clustering algorithm: [49] uses an agglomerative-based clustering method, whereas the method proposed in this thesis lets the network decide how the data should be clustered. The proposed network learns a differentiable clustering "algorithm" from scratch and can therefore learn more specialized "algorithms".

Borji et al. [14] show that 2D CNNs are able to cluster 2D-point data. The network architecture is based on U-Net [69] and can predict up k clusters, where k is a fixed number. The training is done in a supervised fashion. However, the network is only able to cluster 2D-point data, and this point data has to be discretized. The method proposed in this thesis is able to cluster point data without any discretization and high-dimensional data-like images.

Wang et al. [70] propose an end-to-end trainable unsupervised neural network-based clustering method. Their network detects discriminative features, but they do not have the possibility to cluster the data on a specific feature. On the other side, in the approach described in this thesis, it is possible to give the network information about what kind of feature it should cluster. There are more methods that extract a good representation of images with unsupervised training [71] [72] [73], but the proposed approach is not compared to these methods because it uses supervised training and, therefore, has more information during training time.

2.2. The Proposed Approach: The Differences

The proposed method has to learn some kind of similarity function for different input examples. This exact task is explored in the deep learning subfield Deep Metric Learning [74] [75] [76] [77]. However, the proposed model not focus on this specific task but rather on the entire clustering task, which is learned end-to-end. It cannot even be shown exactly how the proposed network compares two examples because there is not any explicit function or loss used for this similarity on the embedding level. Therefore, the model is not compared against the Deep Metric Learning solutions.

The proposed network uses labels to train the clustering. Many other approaches do not require labels; on the other hand, the proposed method has the advantage that it can learn to cluster on a specific feature. To learn this, it requires clustering examples (=labels). A big difference to many other models is, that the proposed network is trained end-to-end: The complete clustering task is learnt by the neural network.

3. Model

The proposed probabilistic model contains a specialized layer type, which is introduced in the first section. The subsequent section contains a detailed explanation of how the model works.

3.1. The RBDLSTM Layer

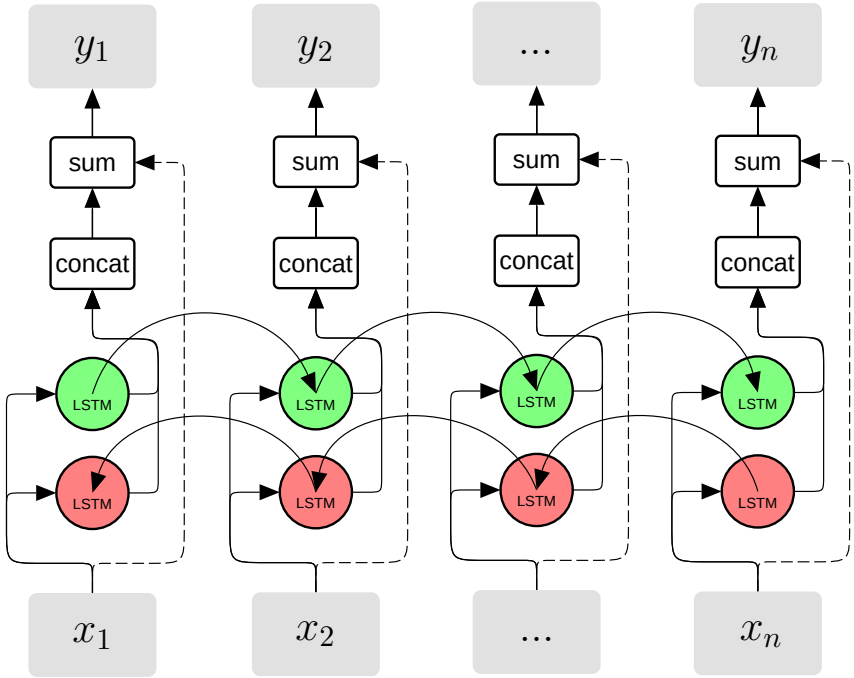


Figure 2.: The RBDLSTM Layer: There are two underlying LSTM layers for both possible directions. Their output vectors are concatenated and then summed up element-wise with the input. The residual connections are dashed. All these sublayers together build an RBDLSTM layer.

An important layer type of the proposed model is a residual bi-directional LSTM [60] layer (RBDLSTM). This layer is visualized in Figure 2. It is based on a bidirectional LSTM layer (BDLSTM) [78] [79] with an additional residual connection that adds a direct connection from the input to the output of the BDLSTM layer. Instead of a single-directional layer, a bidirectional layer is used because in the proposed model, the input sequence does not really have an order, and information has to be exchanged from every example to every example. The underlying BDLSTM layer concatenates the output of the two LSTM networks. Residual connections generally allow much deeper architectures and are very helpful for an efficient back-propagation [80]. This layer is very similar to the residual LSTM layer described in [81], except that it is bi-directional. An important restriction of an RBDLSTM layer is that the input shape must match the output shape. Therefore, the number of internal units for each BDLSTM layer inside the RBDLSTM layer must be equal to half the number of components of the input vectors.

The model architecture described does not only converge very slowly when instead of the RBDLSTM layers, BDLSTM layers are used, but also to a very bad optimum. This is even true for 2D-point data. Therefore, it is essential for the model described. This effect is briefly described in Section 11.2.3.

3.2. The Clustering Model

The probabilistic model described performs end-to-end clustering, and therefore, the input of the model is a set of examples x_i (for $1 \leq i \leq n$), and the output is a set of clusters. There is an output that describes the cluster count and for each input example x_i there is an output distribution for each possible cluster count k ($1 \leq k \leq k_{\max}$) over the cluster indices. Given the most probable cluster count, the most probable cluster index for each example x_i can be calculated easily. There is a limited number k of possible cluster counts, for which the following limit exists: $1 \leq k \leq k_{\max}$. This can easily be extended to allow a lower-limit $k_{\min}$, but this is often not required because $k_{\min} = 1$ is usually a reasonable assumption. The maximum cluster count $k_{\max}$ has to be defined prior to the training process. The complete network architecture is visualized in Figure 3 and described more detailed in Table 1. The network allows using another count of input examples n after the training since it is based on a recurrent architecture.

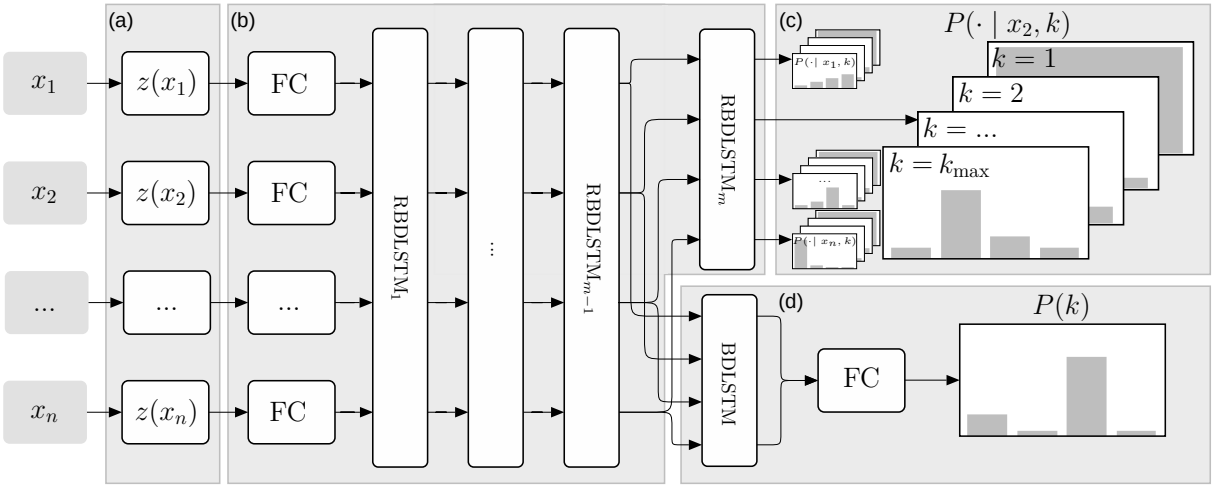


Figure 3.: The complete model, consisting of (a) the embedding network, (b) the clustering network, (c) the cluster-assignment network, and (d) the cluster-count estimating network.

The proposed model architecture contains an initial embedding network (a) that creates an embedding $z_i = z(x_i)$ for each input example x_i. The embedding network may be specific for a given data type or dataset (like in [49]: the CNN used varies for different image datasets). For example, for images, a CNN [29] may be used. Other highly compact data types, like d-dimensional points, may not use any embedding network at all because their representation is already very compact. In such cases, the embedding network just can be replaced by an identity function. In this section, the focus is not on the embedding network but on the complete architecture. In Chapter 7, some embedding networks are described.

Once the embeddings are available, the representations dimension is changed to the required size for the clustering network (b) by using a fully connected layer with a LeakyReLU activation ($\alpha = 0.3$). Usually, this increases the size of the representation. This layer is required because of the output shape restriction of the RBDLSTM layer. It also allows using exactly the same network architecture, except for the embedding network, for many different data types.

These "rescaled" representations are the input for a stacked RBDLSTM network with $m = 14$ layers. These layers are included in the clustering network (b). After each RBDLSTM layer, each vector of each input contains less information about the embedding z_i and more information about the target cluster. Each RBDLSTM layer refines this information by using all available inputs and combining the information in them. Therefore, the clustering "algorithm" itself is encoded in these stacked RBDLSTM layers as a differentiable program. There are several ad-

vantages of using stacked RBDLSTM layers compared to a stacked BDLSTM network without residual connections; for instance, the back-propagation works much better [80] because of the skip-connections. Many clustering algorithms use iterations that refine representations – the same idea is used in the proposed architecture. Each RBDLSTM layer may be seen as an iteration of which the representations are always modified by a delta. BDLSTM layers would always not only modify the input by a delta but create new outputs altogether, so RBDLSTM layers seem to be the more natural choice. It was possible to show that this type of layer performs much better on the given clustering task (see Section 11.2.3). The final representation of z_i after the RBDLSTM_m layer (see Figure 3) is called ξ_i. This vector no longer has to contain specific information about the input example x_i but only about to which cluster the given input example is assigned to.

Table 1.: All layers of the network are described in this table. The input layer is green, and all output layers are blue. n describes the number of inputs. This value is flexible and may be seen as the time axis for the LSTM-based layers. X describes the input shape of an example; this value is fixed during the training and testing time and depends on the data type used and the encoding. E stands for the embedding dimension: It depends on the embedding network used.

	Name/Type	Input Layer	Units	Output Shape
	`input`	-	-	$n \times X$
Embedding-Network (a)				
	`en-embedding`	`input`	-	$n \times E$
Clustering-Network (b)				
	`cn-fully-connected`	`en-embedding`	288	$n \times 288$
	`cn-leaky-relu`$[\alpha = 0.3]$	`cn-fully-connected`	-	$n \times 288$
	$\texttt{rbdlstm}_1$	`cn-leaky-relu`$[\alpha = 0.3]$	288	$n \times 288$
	$\texttt{rbdlstm}_2$	$\texttt{rbdlstm}_1$	288	$n \times 288$
	...	-	-	$n \times 288$
	$\texttt{rbdlstm}_m$	$\texttt{rbdlstm}_{m-1}$	288	$n \times 288$
Cluster-Assignment Network (c): The layers are executed in parallel for each output of `cn-rbdlstm`$_m$. Therefore, the shapes are always "$n \times \ldots$".				
	$\texttt{ca-fully-connected}_1$	$\texttt{rbdlstm}_m$	1	$n \times 1$
	$\texttt{ca-fully-connected}_2$	$\texttt{rbdlstm}_m$	2	$n \times 2$
	...	...	...	...
	$\texttt{ca-fully-connected}_{k_\text{max}}$	$\texttt{rbdlstm}_m$	k_max	$n \times k_\text{max}$
	$\texttt{ca-softmax}_1$	$\texttt{ca-fully-connected}_1$	-	$n \times 1$
	$\texttt{ca-softmax}_2$	$\texttt{ca-fully-connected}_2$	-	$n \times 2$
	...	...	-	...
	$\texttt{ca-softmax}_{k_\text{max}}$	$\texttt{ca-fully-connected}_{k_\text{max}}$	-	$n \times k_\text{max}$
Cluster-Count Estimating Network (d)				
	`cc-bdlstm`	$\texttt{rbdlstm}_{m-1}$	128	$n \times 128$
	`cc-concat[first,last]`	`cc-bdlstm`	-	256
	$\texttt{cc-fully-connected}_1$	`cc-concat`	256	256
	$\texttt{cc-leaky-relu}_1[\alpha = 0.3]$	$\texttt{cc-fully-connected}_1$	-	$n \times 288$
	$\texttt{cc-batch-norm}_1$	$\texttt{c-leaky-relu}_1$	-	256
	`cc-dropout`$[p = 0.5]$	$\texttt{cc-batch-norm}_1$	-	$n \times 288$
	$\texttt{cc-fully-connected}_2$	`cc-dropout`	k_max	k_max
	`cc-softmax`	$\texttt{cc-fully-connected}_2$	-	k_max

Two networks follow the clustering network (b), one is the cluster-assignment network (c) and the other the cluster-count estimating network (d). The cluster-assignment network contains, in

combination with each input vector x_i, a fully connected layer that uses a softmax activation for each possible cluster count $1 \leq k \leq k_{\max}$. This softmax activation describes $P(\ell \mid x_i, k)$ which is the distribution for the cluster index ℓ given a specific total cluster count and input example x_i. For instance, $P(\ell = 2 \mid x_2, k = 3)$ describes the probability that the example x_2 belongs to the cluster 2 given there are 3 clusters in the data. Tests were done to see whether the model's quality increases when there are more fully connected layers between the softmax activation and the RBDLSTM$_m$ layer, but this was not the case. Therefore, there are no fully connected layers except for the one required for the softmax activation.

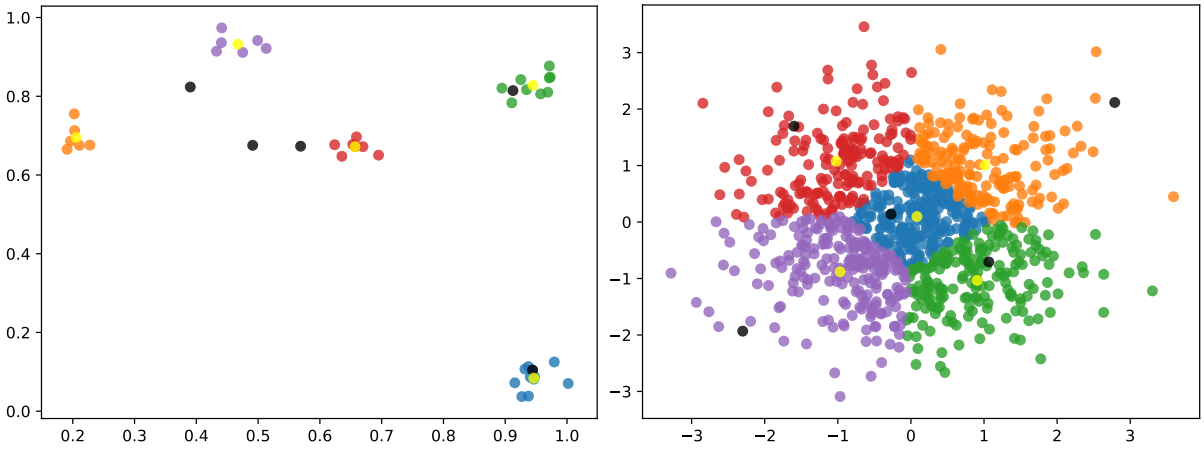
The cluster-count estimating network (d) uses the representation after the RBDLSTM$_{m-1}$ layer as input. It could be seen that this representation leads to better results. The reason for this is that the final representation ξ_i after the RBDLSTM$_m$ layer is optimized for the softmax output of the cluster assignment network. This representation may not be optimal for counting the clusters, so it can be assumed that the layer prior to the last RBDLSTM layer is a better choice. This layer already contains a very abstract representation of the data, but it is still not too specific (compared to ξ_i). This list of vectors is processed in a BDLSTM layer where only the first and the last output vectors are used in further processing. These two vectors are concatenated and then processed by a fully connected layer using a LeakyReLU activation ($\alpha = 0.3$). Then there is a final fully connected layer that uses a softmax activation and produces the cluster-count distribution $P(k)$ ($1 \leq k \leq k_{\max}$). For instance, $P(k = 2)$ describes the probability that there are 2 clusters in the data.

This model finally used has a runtime complexity of $\mathcal{O}(n)$, where n is the number of inputs given that the dimensionality of the data has a fixed size and that the possible cluster counts are fixed.

4. Alternative Models

In the thesis, different core models were evaluated and tested. All models tested used the same input and output data structures/interface as the final model used which is described in Chapter 3. The different models evaluated are based on different core ideas. Not only the runtime complexity is different for the models described, but the structural assumptions on how clustering is performed is also very different for these models.

4.1. The k-Means based Model



(a) (b)

Figure 4.: Figures 4a and 4b visualize the different outcomes of the differentiable k-means algorithm, assuming that there are $k = 5$ clusters in the data. The black points describe the (random) initial cluster centers, and the yellow points represent the final cluster centers. The colored points visualize the cluster assignment. The two examples shown return a plausible clustering.

k-means [3] is a well-known and widely used clustering algorithm. It works very well for low-dimensional data in which the clusters are of similar size and are Gaussian-distributed. It is also important that the cluster count k is known. The algorithm is not that effective for more complex data or when the cluster count is not known.

The idea behind this model is to train a neural network which produces embeddings for the k-means algorithm. The trick is to include the k-means algorithm in the neural network to allow an end-to-end training. This gives the optimization process the possibility to optimize the embeddings for the k-means algorithm. These embeddings are produced by multiple BDLSTM layers.

One problem in this model is that the k-means algorithm is not differentiable. Therefore, the algorithm has to be modified to fit the requirements. Finally, the modified and differentiable (soft) k-means algorithm gets the cluster count k and a set of vectors as input and produces a softmax-distribution for each vector over the k clusters. It is important to mention that this algorithm has to be executed for each possible cluster count k because the cluster count is also a parameter of the algorithm. The implemented (soft) k-means algorithm is executed exactly in ITRS iterations. An important note is that the algorithm initially squeezes all inputs into the range $[-1, 1]$ by using the $\tanh(x)$ function. This is done to handle distances between points more easily because this limits the distances to $\sqrt{d}$ for d-dimensional inputs, which is very

helpful to define an effective soft k-means algorithm. Only $d = 2$ dimensions are used in the tests.

Algorithm 1 The proposed differentiable k-means algorithm which has k number of clusters and a set X of n examples as input. Instead of returning a cluster index per input like the real k-means algorithm, the proposed algorithm returns a cluster index distribution per input. The value ε is a small constant that is used to avoid divisions by 0.

```
procedure SoftKMeans(k, X = {x_1, x_2, ... x_n})
    cluster_centers ← random_uniform(k×dim(x_1), range= [0, 1])
    cluster_assignments ← matrix_zero(n×k)
    X ← tanh(X)
    for i in 1...ITRS do
        for j in 1...n do
            distances ← array_empty()
            for k in 1...k do
                distances[k].append ← inverse_distance(x_j, cluster_centers[k])
            end for
            cluster_assignments[j] ← softmax(distances)
        end for
        for k in 1...k do
            v_sum ← matrix_zero(dim(x_1))
            v_count ← 0
            for j in 1...n do
                v_sum ← v_sum + cluster_assignments[j, k] * x_j
                v_count ← v_count + cluster_assignments[j, k]
            end for
            cluster_center[k] ← v_sum / (v_count + ε)
        end for
    end for
    return cluster_assignments
end procedure
```

The differentiable soft k-means algorithm described (see Algorithm 1) has a critical part in it: The inverse_distance(x, y) function must be differentiable. It should be low for the near points and high for points that are far away. This function is used to calculate, using a softmax activation, the probability that a given point should be assigned to a specific cluster center: This distance is calculated for each input and cluster center. In the real k-means, this function would be $-\infty$ for all cluster centers except for the nearest center. For the nearest center, it would have a real number as result, e.g. 1. By using a softmax-activation, this produces a distribution that contains 1 for the nearest center and 0 for all other centers. Multiple approaches were evaluated, but finally, the following formula was used for the inverse_distance(x, y) function:

$$\text{inverse\_distance}(x, y) = -(1 + \|x - y\|_2)^2$$

Unfortunately, it is quite hard to get a function that works well because the values should be low for all cluster centers, except the nearest, i.e. high for the nearest cluster center, and the function must have good gradients for the back-propagation [82].

The quality of the differentiable k-means algorithm highly depends on the initially chosen points and on what the clusters look like. For Gaussian-distributed clusters and a given cluster count k, in general, the algorithm works as expected as can be seen in Figure 4.

The model used with the differentiable k-means does not use RBDLSTM layers but only BDLSTM layers. Only $m = 2$ BDLSTM layers were used in this model. The model was already discarded before the RBDLSTM layers were used. The cluster-assignment network finally

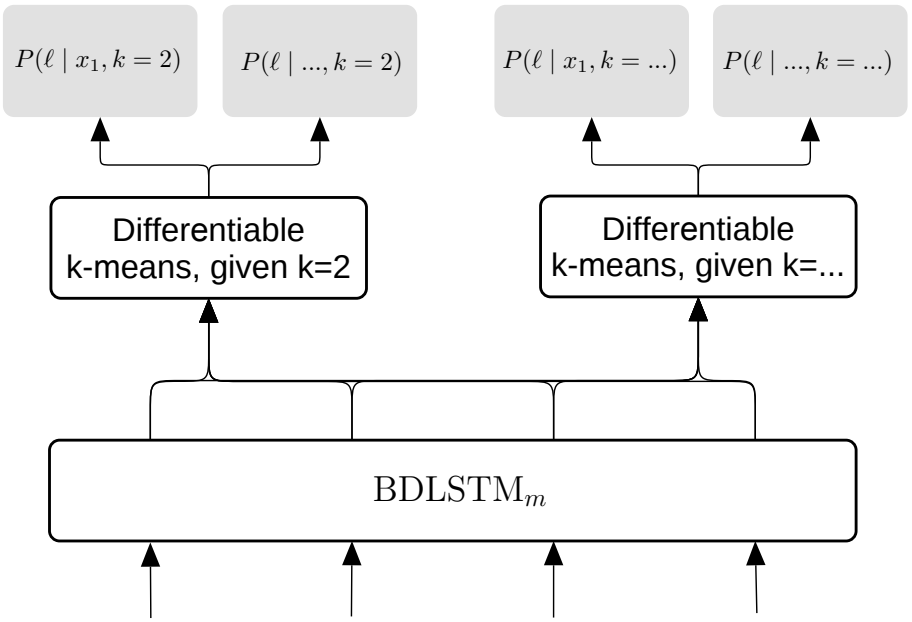


Figure 5.: The cluster-assignment network of the differentiable k-means model, including the BDLSTM$_m$ layer.

obtained of the model is shown in Figure 5. Except for the differentiable k-means on top and the use of $m = 2$ BDLSTM layers, the model is equal to the model described in Chapter 3. For debugging purposes, the network was trained on 2D-point data (see Section 10.2). The representation ξ_i is 2-dimensional because this allows observing the representation of the data that the network generates for the differentiable k-means algorithm. One could assume that the network creates something like Gaussian distributions for each cluster detected, even if the cluster in the input was not Gaussian-distributed. But it could be seen that the network puts all the points on the border and builds some clusters there. This behavior is visualized in Figure 6. Unfortunately, this does not seem to work well for more than 4 clusters. In general, doing back-propagation does not seem to be effective using the differentiable k-means algorithm. The described algorithm, given a fixed number of iterations for the differentiable k-means, is executed in $\mathcal{O}(n)$.

Because the network model used is ineffective, it was no longer applied. Finally, it could be seen that the network chooses a more effective method provided it can decide by itself how it should implement the clustering "algorithm".

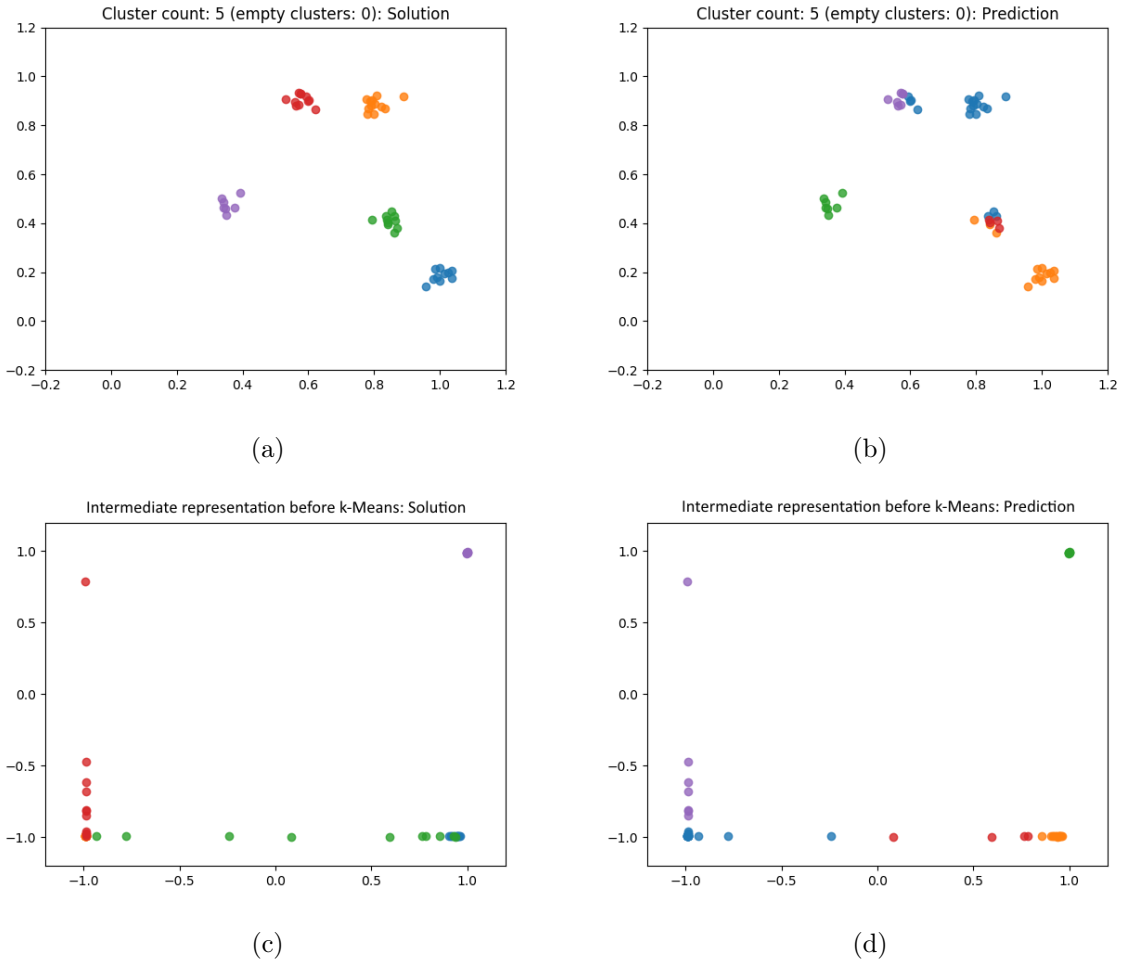


Figure 6.: Figure 6a presents the optimal solution for the input points given, and Figure 6b shows the actual prediction. To get a better feeling of how the network works, there is the 2-dimensional input representation of the soft k-means algorithm (including the $\tanh(x)$ squeeze) visualized, which is equal to $\tanh(\xi_i)$: Once with the expected clusters (=the solution) in Figure 6c and once with the actual prediction by the k-means algorithm in Figure 6d. These two images illustrate how the network processes the points for the differentiable k-means algorithm.

4.2. The Iterative Model

Many classical clustering algorithms work in iterations [3] [4]. Therefore, it might be assumed that a neural network based on iterations may converge to a better solution. Such an algorithm basically contains a state described by the data and a fixed code that is executed multiple times, once for each iteration. In the proposed model, there is a state for each input and also a global state. For the k-means, the global state could represent the current cluster centers, and the state per input could show the current assignment per input example. For this model, there are no assumptions made on how the states are used and what they represent; they are trained by the neural network.

A model based on this idea was developed for testing whether this works well. The model initializes an empty state for each input example. In each iteration, a loop is made over all inputs. Inside the loop, the current input is compared with all other inputs and their states. This list of values is reduced by a BDLSTM layer and produces a new state for the current input. The final step in each iteration is updating the global state by using a BDLSTM layer.

The pseudo-code shown in Algorithm 2 describes the neural network model. The F_1- and F_2 functions each contain a fully connected layer with a LeakyReLU activation ($\alpha = 0.3$) and, therefore, also trainable weights. Furthermore, the BDLSTM layers used contain trainable weights. The code is executed for ITRS iterations.

Algorithm 2 The proposed differentiable iterative algorithm for the clustering model. $c_0, c_1 \in \mathbb{N}_{>0}$ are hyperparameters which define the state size for the global state and for the state per input example.

```
procedure IterativeModel(X = {x_1, x_2, ... x_n})
    global_state ← matrix_zero(c_0)
    states ← matrix_zero(n×c_1)
    for i in 1...ITRS do
        states_new ← matrix_zero(n×c_1)
        for j in 1...n do
            processed_data ← array_empty()
            for l in 1...n do
                if j ≠ l then
                    comparison_result = compare_data((x_j, states[j]), (x_l, states[l]))
                    process_data.append(comparison_result)
                end if
            end for
            processed_data_output = select_first_and_last_vector(BDLSTM(processed_data))
            states_new[j] = F_1(processed_data_output)
        end for
        states = states_new
        global_state = F_2(
          global_state,
          select_first_and_last_vector(BDLSTM(concat((x_1, ... x_n), (states[1], ... states[n))
        )
    end for
    return states[1], ... states[n]
end procedure
```

The described algorithm replaces the clustering network (see the entire model in Section 3) but still uses the same embedding network and softmax-output. Also, the loss function described in Chapter 5 is used. The proposed algorithm in each iteration allow this item to access information

for each item from any other item by using a BDLSTM layer to update its state. This implies a time complexity of $\mathcal{O}(n^2)$, given a constant number of iterations, including a relatively high constant factor (at least compared to the loss function used, which is described in Chapter 5 and which also has a time complexity of $\mathcal{O}(n^2)$). This is a practical problem because it considerably slows down the training time compared to the finally used model which has a time complexity of $\mathcal{O}(n)$. The training for this model is quite slow and almost impractical: After a few days, the quality slowly increased. The final result reached is not better than when using a basic (R)BDLSTM model. Therefore, this model was no longer used. A disadvantage of this model is that it makes relatively high assumptions about how the neural network, and therefore, the differentiable clustering program is structured. On the other hand, it allows implementing a program with the time complexity of $\mathcal{O}(n^2)$, which could lead to more powerful clustering models. Adding residual connections for all states that connect the different iterations could lead to better results, but this model was no longer tested.

5. The Loss Function

The different network outputs are described in Chapter 3. They are used to define an appropriate loss function for the given network architecture. These outputs include a distribution $P(k)$ for the cluster count, and for each combination of an input example x_i and cluster count k, they include a distribution of the cluster index $P(\ell \mid x_i, k)$.

5.1. The Cluster-Count Estimation Loss

For the cluster-count estimating output, the categorical cross-entropy [83] $\mathrm{CCE}(y_c, P(k))$ is used. This term is called L_{cc}. The real cluster count has to be known and must be in the predefined and fixed range $1 \leq k \leq k_{\max}$. The loss term L_{cc} is given by the following expression where y_c is 1 if c is equal to the real cluster count and otherwise, 0:

$$\begin{aligned} L_{\mathrm{cc}} &= \mathrm{CCE}(y_c, P(k)) \\ &= -\sum_{c=1}^{k_{\max}} y_c \log(P(k=c)) \end{aligned}$$

5.2. The Cluster-Assignment Loss

It is not possible to predefine which elements should be assigned to which cluster indices because there are multiple valid solutions for this problem: Every permutation of the cluster indices is an equivalent solution. Therefore, the loss function must take care of this properly. This makes it impossible to just use another categorical cross-entropy for the cluster-assignment outputs. Therefore, the probability that x_i and x_j have the same cluster index (i.e. they are in the same cluster) $P_{ij} = P(\ell_i = \ell_j)$ is calculated because this probability does not make any more assumptions about a fixed cluster index for a given element. For this probability, a weighted version of the binary cross-entropy [83] loss is used. It is called $\mathrm{BCE}_{\mathrm{w}}(y_{ij}, P_{ij})$. For a given clustering, it is simple to get the required labels because it is known when two examples are in the same cluster. If they are in the same cluster, then the target which is called y_{ij} is 1, otherwise, it is 0. This loss requires that each element is compared with every other element, and therefore, there are $\frac{n(n-1)}{2}$ resulting comparisons.

The formulas below show how to derive the probability P_{ij} given the network outputs described in Chapter 3. The formulas also define the loss term L_{ca} which describes the average error on these calculated probabilities compared to the ground truth. The entire loss function is visualized in Figure 7, where the input values are at the bottom, and the loss based on these values is at the top. $P_{ij}(k)$ describes the probability $P(\ell_i = \ell_j \mid k)$.

$$P_{ij}(k) = \sum_{\ell=1}^{k} P(\ell \mid x_i, k) P(\ell \mid x_j, k)$$

By marginalizing over k, the term

$$P_{ij} = \sum_{k=1}^{k_{\max}} P(k) \sum_{\ell=1}^{k} P(\ell \mid x_i, k) P(\ell \mid x_j, k)$$

is obtained for the model probability that x_i and x_j belong to the same cluster. Let $y_{ij} = 1$ if x_i and x_j are from the same cluster (e.g. have the same labels) and 0 otherwise. The loss

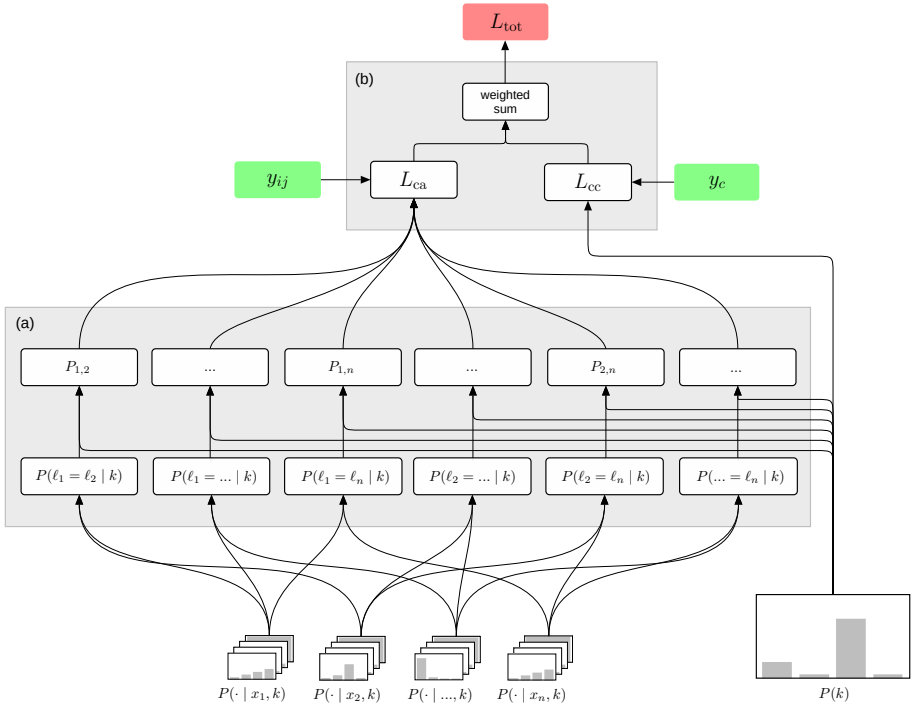


Figure 7.: The used loss function which takes the given network outputs as input: The part (a) of the loss functions is used to calculate all $\frac{n(n-1)}{2}$ combinations of P_{ij} with $1 \leq i < j \leq n$. The sub-sequential part (b) calculates the binary and the categorical cross-entropy for the given inputs and the available labels. The computed sum is the final loss. y_{ij} describes the upper half of the similarity matrix. It contains all y_{ij} for $1 \leq i < j \leq n$: This is the ground truth for P_{ij}. The value y_c describes the true cluster count and is therefore the ground truth for $P(k)$.

component for cluster assignments L_{ca} is then given by the following formula:

$$\begin{aligned} L_{\text{ca}} &= \frac{-2}{n(n-1)} \sum_{i<j} \text{BCE}_w(y_{ij}, P_{ij}) \\ &= \frac{-2}{n(n-1)} \sum_{i<j} (\varphi_1 y_{ij} \log(P_{ij}) + \varphi_2 (1 - y_{ij}) \log(1 - P_{ij})) \end{aligned}$$

The $\text{BCE}_w(y_{ij}, P_{ij})$ loss uses different weights for the available classes. The value φ is defined as the expected value of y_{ij}. This probability is taken over all possible cluster counts for a fixed input example count n. As described in Chapter 8, the cluster count is uniformly distributed in the training: This has a direct effect on φ. For the model proposed, φ is approximated by sampling data until the 95% confidence interval is smaller than 0.005. This approximated probability is called $\widetilde{\varphi}$. This probability is used to calculate an error-weight for outputs with the label 1, which is called w_1, and outputs with the label 0, which is called w_0. Usually, the probability φ (and therefore, also $\widetilde{\varphi}$) is quite small but still higher than 0. $\widetilde{\varphi}$, in general, only makes sense when the value is larger than 0 and smaller than 1, otherwise, the task is trivial. If $\varphi = 0$, then every element is always in its own cluster. On the other hand, if $\varphi = 1$, then all elements are always in the same cluster. The $\sqrt{x}$ function is used to soften the proportion of the weights. This softening results in a more effective training process. Finally, the weights are normalized to a sum of 2 because this is equal to the original error sum where each type of error has the weight 1. The formulas below show how these error weights are derived:

$$\begin{aligned} \varphi_0 &= c\sqrt{\widetilde{\varphi}} \\ \varphi_1 &= c\sqrt{1 - \widetilde{\varphi}} \\ \varphi_0 + \varphi_1 &= 2 \end{aligned}$$

$$\Rightarrow c = \frac{2}{\sqrt{\widetilde{\varphi}} + \sqrt{1 - \widetilde{\varphi}}}$$
$$\Rightarrow \varphi_0 = 2\left(1 + \sqrt{\widetilde{\varphi}^{-1} - 1}\right)^{-1}$$
$$\varphi_1 = 2\left(1 + \sqrt{\frac{\widetilde{\varphi}}{1 - \widetilde{\varphi}}}\right)^{-1}$$

5.3. The Final Loss

The final loss is described by the following formula where the default value of λ is 5.0:

$$L_{\text{tot}} = L_{\text{cc}} + \lambda L_{\text{ca}}$$

The loss function has a runtime and memory complexity of $\mathcal{O}(n^2)$ because all inputs are compared with one another. More exactly: There are $\frac{n(n-1)}{2}$ comparisons made. This might be a limiting factor for the training. At least, the operations performed in the loss functions do not require much memory and computational power, but they still scale quadratically. The final loss for the proposed model does not include any regularizations.

Given that the network perfectly predicts the cluster count k (which is, e.g., often the case for the 2D-point model), then this loss function does not force the network to output something meaningful for $P(\ell \mid x_i, k)$ for all other possible cluster counts. The reason for this is that in this case, only the error over the predicted cluster count is calculated: Back-propagation is then also only done by the predicted cluster count. More general, back-propagation is proportional to the $P(k)$ distribution performed by the output softmax layers. This makes sense, because the network has not to optimize solutions which are not probable.

5.4. Interpreting the Cluster-Assignment Accuracy

Given the computed network output P_{ij} and the ground truth y_{ij}, it is possible to calculate the binary accuracy of these probabilities so that the two examples x_i and x_j are in the same cluster. This should be done carefully because this accuracy may be too pessimistic. This is shown in an example:

Given in the data, there are three clusters and the maximum cluster count is $k_{\max} = 3$. The network predicts $P(k = 1) = 0.1$, $P(k = 2) = 0.3$, and $P(k = 3) = 0.6$, and there are $n \geq 3$ input examples. It is known that there are 3 clusters and that x_1 and x_2 are in the same cluster (i.e. $y_{1,2} = 1$). Their cluster assignment probabilities are:

$$P(\ell = 1|x_1, k = 1) = 1.0$$
$$P(\ell = 1|x_1, k = 2) = 0.3$$
$$P(\ell = 2|x_1, k = 2) = 0.7$$
$$P(\ell = 1|x_1, k = 3) = 0.4$$
$$P(\ell = 2|x_1, k = 3) = 0.3$$
$$P(\ell = 3|x_1, k = 3) = 0.3$$
$$P(\ell = 1|x_2, k = 1) = 1.0$$
$$P(\ell = 1|x_2, k = 2) = 0.4$$
$$P(\ell = 2|x_2, k = 2) = 0.6$$
$$P(\ell = 1|x_2, k = 3) = 0.5$$
$$P(\ell = 2|x_2, k = 3) = 0.1$$
$$P(\ell = 3|x_2, k = 3) = 0.4$$

Now, it is possible to calculate the probability that x_1 and x_2 are in the same cluster:

$$\begin{aligned}
P_{ij} =& P(k=1)(P(\ell=1|x_1,k=1)P(\ell=1|x_2,k=1))+ \\
& P(k=2)(P(\ell=1|x_1,k=2)P(\ell=1|x_2,k=2)+P(\ell=2|x_1,k=2)P(\ell=2|x_2,k=2))+ \\
& P(k=3)(P(\ell=1|x_1,k=3)P(\ell=1|x_2,k=3)+P(\ell=2|x_1,k=3)P(\ell=2|x_2,k=3)+ \\
& P(\ell=3|x_1,k=3)P(\ell=3|x_2,k=3))+ \\
=& 1.0*1.0*0.1+ \\
& 0.3*(0.3*0.4+0.7*0.6)+ \\
& 0.6*(0.4*0.6+0.3*0.1+0.3*0.4) \\
=& 0.496
\end{aligned}$$

Clearly, if the maximum arguments for the cluster count and the predicted cluster indices are used, the network chooses the correct cluster count and the correct cluster assignment for the given examples x_1 and x_2. But as can be seen, the output says that it is not so probable that x_1 and x_2 are in the same cluster. This creates a bad loss and accuracy. It is very important to mention that the loss can only be minimized and be 0 if and only if the perfect clustering is chosen where all probabilities in the output P_{ij} are exactly 0 or 1. Of course, this is unlikely. If all rounded P_{ij} values are correct, then it can be said that the network's output (=the clustering) is also perfect for maximum argument values. Therefore, this is **sufficient**. On the other hand, if the rounded P_{ij}-values are not correct, the maximum arguments of the network outputs (=the clustering) still may be perfect. Therefore, it is **not necessary** that all rounded P_{ij} probabilities are correct for a perfect clustering. So, even with a bad cluster-assignment accuracy, the performance of the network may be still good. This accuracy should only be carefully interpreted because it can be pessimistic, especially for large cluster counts.

6. Regularizations and alternative Loss Functions

Different regularizations and alternative loss functions were evaluated for the different models. The final loss function used outperformed the other approaches, and the regularizations did not improve the result. Some had slightly improved the result for a small specific subset of the data but made other results worse. Because they often make the model much more complex, e.g. by adding a runtime and memory complexity (at the training time) of $\mathcal{O}(n^2)$, where the constant factor is relatively large, they are not used in the final model.

6.1. KL Divergence based Regularization

The use of the Kullback-Leibler (KL) divergence as a regularization is based on the idea of Lukic et al. [12]. The KL-divergence of two discrete probability distributions P and Q is defined as follows:

$$\mathrm{KL}(\mathrm{P} \parallel \mathrm{Q}) = \sum_{i=0}^{k} P_i \log\left(\frac{P_i}{Q_i}\right)$$

The KL-divergence is 0 if and only if the two given distributions are equal. In all other cases, the value is higher. A symmetric version of the KL-divergence is introduced to compare two distributions, P and Q, which are generated by two data records, x_p and x_q. The function $I_s(x_p, x_q)$ returns 1 if both inputs are generated from the same source (in this case, the speaker); otherwise, it is 0. $I_{ds}(x_p, x_q)$ returns 1 when both inputs are from different sources; otherwise, 0. The loss is given by $\mathrm{L}(\mathrm{P}, \mathrm{Q})$:

$$\mathrm{cost}(\mathrm{P} \parallel \mathrm{Q}) = I_s(x_p, x_q)\mathrm{KL}(\mathrm{P} \parallel \mathrm{Q}) + I_{ds}(x_p, x_q)\max(0, \mathrm{margin} - \mathrm{KL}(\mathrm{P} \parallel \mathrm{Q}))$$

$$\mathrm{L}(\mathrm{P}, \mathrm{Q}) = \mathrm{cost}(\mathrm{P} \parallel \mathrm{Q}) + \mathrm{cost}(\mathrm{Q} \parallel \mathrm{P})$$

As described in Chapter 3, the representation after the embedding network is called z_i. To implement this loss, a fully connected layer with 256 units and a LeakyReLU activation ($\alpha = 0.3$) is added, followed by another fully connected layer with 128 units and a softmax activation. This new representation, which is basically a discrete distribution generated by x_i, is called η_i. The neural network still just uses the z_i vectors for the clustering; this means that η_i is only used for this regularization term.

The final regularization term is then defined by the following formula:

$$L_{\mathrm{KL}} = \frac{2}{n(n-1)} \sum_{i<j} \mathrm{L}(\eta_i, \eta_j)$$

This regularization term requires $\mathcal{O}(n^2)$ computational steps and $\mathcal{O}(n^2)$ memory. The regularization term L_{KL} was added to the proposed model with a factor of 0.1. For some models, it improved the network quality, but for the final network architecture used, there was no longer any effect visible. Therefore, the loss term was removed.

6.2. Simplified Center-Loss based Regularization

The original center loss is described in [84]. The version used in this section is simplified compared to this definition. This simplified center loss is applied to the ξ_i-vectors that are described in Chapter 3. These vectors may no longer contain any information about the original input but only about the assigned cluster. For clusters, it is generally not possible to just store a class center as described in [84]; therefore, the regularization is just done by measuring the squared distance from the mean vector of all vectors ξ_i of the same cluster. The average distance to the center is minimized. The following formulas describe this logic where C_i is the set of indices of all inputs in the same cluster as the example with the index i. The set C_i is given by the ground truth. The index i is always in the set C_i included.

$$\zeta(\xi_i) = \frac{1}{|C_i|} \sum_{j \in C_i} \xi_j$$

$$L_{\text{CL}} = \frac{1}{n} \sum_{i=1}^{n} (\xi_i - \zeta(\xi_i))^2$$

The loss term L_{CL} was included with a factor of 0.1. Initially, it had improved the model, but as the model itself was improved during the thesis, this loss was no longer needed. The computational and memory complexities of this regularization term are $\mathcal{O}(n^2)$ given a static graph with a minibatch size greater than 1.

6.3. Cluster-Assignment Regularization

The network always outputs the clustering, assuming that there are 1 up to $k_{\max}$ clusters. When there are $k+1$ clusters and given the network predicts that x_i and x_j are in the same cluster, it makes sense that these two examples are also in the same cluster when there are k clusters. This would imply a more hierarchical style of clustering. Assuming that there are $k+1$ clusters, exactly two of the given clusters should be merged when there are k clusters. With a smaller number of clusters, the probability that two examples are in the same cluster should always (but not necessarily strictly) grow. Finally, for $k = 1$, all probabilities are equal and $P_{ij}(k = 1) = 1$. More formally, this can be written as:

$$P_{ij}(k) \geq P_{ij}(k+1)$$

With the current model, this is not always fulfilled, but a rule to imply this can be included as a soft constraint. This still does not force the network to use this rule, but it makes it much more probable that this equation is fulfilled. As described in Chapter 5, the probability $P_{ij}(k)$ may be calculated. This value described how probable it is that the examples x_i and x_j are in the same cluster, given that there are k clusters. Now it can be calculated how large the probability is that two examples are in the same cluster given $k+1$ clusters and, at the same time, given that there are k clusters, they are not in the same cluster:

$$\Omega_k(i,j) = P_{ij}(k+1)(1 - P_{ij}(k))$$

If the described requirements are fulfilled, then $\Omega_k(i,j)$ is equal to 0, otherwise, it is in the interval $(0,1]$. The loss term L_{CA} is an average of all possible valid $\Omega_k(i,j)$ terms. The loss does not make sense when there is only one possible cluster count k; therefore, it is required that there are at least two possible cluster counts available. They do not even have to have a difference of 1, but this is usually the case. The following formula for L_{CA} assumes that the minimal cluster

count is 1 and the maximal cluster count is $k_{\max}$:

$$L_{\text{CA}} = \frac{1}{k_{\max}-1} \sum_{k=1}^{k_{\max}-1} \frac{2}{n(n-1)} \sum_{i<j} \Omega_k(i,j)$$
$$= \frac{2}{(k_{\max}-1)n(n-1)} \sum_{k=1}^{k_{\max}-1} \sum_{i<j} P_{ij}(k+1)(1-P_{ij}(k))$$

Unfortunately, this regularization causes the network to prefer creating empty clusters. It much prefers creating identical clusterings for different k values, but for higher k values, many clusters are empty. Therefore, another regularization has to be added to avoid empty clusters. This is done by maximizing the maximal probability of a cluster element belonging to this cluster. This implies that at least one element in each cluster should have a large probability for it being in this cluster. Therefore, there should not be any empty clusters. The following formula describes this additional loss term which is called L_{NE}:

$$L_{\text{NE}} = 1 - \frac{1}{k_{\max}} \sum_{k=1}^{k_{\max}} \frac{1}{k} \sum_{c=1}^{k} \max_{i \in \{1 \dots n\}} P(\ell_i = c|k)$$

Without the cluster assignment regularization given, the network predicts the cluster count very well, the clustering for the not-so-probable cluster counts are almost not trained. This makes sense because if the given cluster count is not probable nor even possible, then it should not be trained with the original loss function. This regularization is applied to all outputs for all cluster counts in the network. It does not force the network to create meaningful clusters but only hierarchical clusters. Of course, it must be combined with another loss function to obtain meaningful clusters.

The finally added loss for this regularization, therefore, is given by $L_{\text{CAR}} = L_{\text{CA}} + \lambda_1 L_{\text{NE}}$. The parameter λ_1 is used to regulate the weight of the loss terms; the default value can be set to 1.

The loss component L_{CAR} is in some experiments used. They show that it has more positive effects on the hierarchical clustering if a higher factor is used for the loss components, e.g. $5L_{\text{CAR}}$. It is very interesting that the loss has no significant negative effect, considering the clustering quality in total, but it does have positive effects for the hierarchical clustering. This loss component might be used for further optimizations.

6.4. Rand Index based Loss

The *rand index* (RI) [85] is defined by

$$\text{RI} = \frac{a+b}{C_2^{\text{samples}}}$$

where a describes the number of pairs of elements that are in the same cluster in the ground truth and in the same cluster in the clustering, and b describes the number of pairs of elements which are not in the same cluster in the ground truth and are not in the same cluster in the clustering. C_2^{samples} describes the total number of possible pairs in the data.

The intuition behind this loss is that a good metric for clustering also is a good loss. If the metric is good, then obviously, the clustering is also good. A loss function for a neural network must be differentiable almost everywhere and should be continuous. This is not true for most metrics, including the RI. If an algorithm or function is generalized to a differentiable version, it may have a slightly different behavior, and there is no guarantee that it is a good optimizable loss function for a neural network.

Because a differentiable version of the RI is required, the values a, b, and C_2^{samples} have to be redefined in the following way by using the existing defined and differentiable values (given the notation defined in Chapter 5):

$$\begin{aligned} a &= \sum_{i<j} y_{ij} P_{ij} \\ b &= \sum_{i<j} (1 - y_{ij})(1 - P_{ij}) \\ C_2^{\text{samples}} &= \text{nCr}(\text{n}, 2) = \frac{n(n-1)}{2} \end{aligned}$$

The RI is optimal when it reaches a value of 1. In general, RI is always in the interval of $[0, 1]$. This is also true for the generalized differentiable version of the RI, whose behavior is identical with the original RI if y_{ij} and P_{ij} are discrete values (0 or 1). Because a loss function is minimized, and the RI is optimal when maximized ($= 1$), the calculated result is subtracted from 1. The final loss term L_{RI} is given by the following expression:

$$\begin{aligned} L_{\text{RI}} &= 1 - \frac{\sum_{i<j} y_{ij} P_{ij} + \sum_{i<j} (1 - y_{ij})(1 - P_{ij})}{\left(\frac{n(n-1)}{2}\right)} \\ &= 1 - \frac{2 \sum_{i<j} (y_{ij} P_{ij} + (1 - y_{ij})(1 - P_{ij}))}{n(n-1)} \end{aligned}$$

The loss term L_{RI} would replace L_{ca}. The loss term performed well but much less effectively than L_{ca} (see Chapter 5). For this reason, it was not used for the final model. The computational costs of the loss term are in $\mathcal{O}(n^2)$ and similar to L_{ca}.

7. Embedding Networks

Different data types require different embedding networks: The reason for this is that different data types often need different architectures to reduce the input dimensionality. In this chapter, some possible embedding networks are described. This includes an embedding network that was not used for the final experiments, but which shows the flexibility of the approach.

7.1. Identity-Function

The simplest embedding network is the identity function. In this case, the embedding network is just defined as $z_i = z(x_i) = x_i$. This kind of embedding network makes sense if the given data is already very compact, e.g. as the 2D-point data which is described in Section 10.2.

7.2. CNN

Table 2.: This table describes a general CNN-based embedding network for images with the shape $128 \times 128 \times X$. The network used is a simple feed-forward network with one input and one output. The first layer in the table is the input layer, and the last layer describes the output of the network. The parameter X describes the number of color channels (or features) of the input image.

Name/Type	**Kernel Size**	**Stride**	**Padding**	**Units**	**Output Shape**
`input`	-	-	-	-	$128 \times 128 \times X$
$\texttt{conv}_1$	3×3	1×1	1	32	$128 \times 128 \times 32$
$\texttt{leaky-relu}_1\,[\alpha = 0.3]$	-	-	-	-	$128 \times 128 \times 32$
$\texttt{batch-norm}_1$	-	-	-	-	$128 \times 128 \times 32$
$\texttt{max-pool}_1$	2×2	2×2	0	-	$64 \times 64 \times 32$
$\texttt{conv}_2$	3×3	1×1	1	64	$64 \times 64 \times 64$
$\texttt{leaky-relu}_2\,[\alpha = 0.3]$	-	-	-	-	$64 \times 64 \times 64$
$\texttt{batch-norm}_2$	-	-	-	-	$64 \times 64 \times 64$
$\texttt{max-pool}_2$	2×2	2×2	0	-	$32 \times 32 \times 64$
$\texttt{conv}_3$	3×3	1×1	1	128	$32 \times 32 \times 128$
$\texttt{leaky-relu}_3\,[\alpha = 0.3]$	-	-	-	-	$32 \times 32 \times 128$
$\texttt{batch-norm}_3$	-	-	-	-	$32 \times 32 \times 128$
$\texttt{max-pool}_3$	2×2	2×2	0	-	$16 \times 16 \times 128$
$\texttt{fully-connected}_1$	-	-	-	256	256
$\texttt{leaky-relu}_4\,[\alpha = 0.3]$	-	-	-	-	256
$\texttt{batch-norm}_4$	-	-	-	-	256
$\texttt{fully-connected}_2$	-	-	-	256	256
$\texttt{leaky-relu}_5\,[\alpha = 0.3]$	-	-	-	-	256
$\texttt{batch-norm}_5$	-	-	-	-	256

CNNs are very powerful networks for extracting good features. They are used to process images [29], audio [12], text [86], and also for many other tasks [87] [88].

Because CNNs work that well, especially for images, it makes sense to create an embedding network based on a CNN. Table 2 shows such a possible network. This embedding network is used for all image and audio experiments. It is roughly based on the network described by Lukic et al. [12] but, e.g., uses LeakyReLU ($\alpha = 0.3$) instead of ReLU because this improves the clustering quality of the complete network.

7.3. BDLSTM

LSTM networks are very powerful for processing sequences: They are used for handling text [89], audio [90], and time series in general [91] [92]. A big advantage of LSTMs is that they can handle dynamic long inputs and if the input axis is seen as a time axis, they can remember specific features for a very long time. In this thesis, some tests are conducted on an architecture based on the BDLSTM network described by Egli et al. [93], but it was not possible to achieve any good results. The architecture used is described in Table 3. The biggest disadvantage of a BDLSTM network for the embedding is the processing speed because calculating LSTMs is very complex compared to e.g. CNNs.

Table 3.: This table describes a general LSTM-based embedding network for inputs with the shape $T \times X$. The network input is a time series with T elements and X features. The number of features has to be fixed, and T is dynamic.

Name/Type	**Units**	**Output Shape**
`input`	-	$T \times X$
`bdlstm`$_1$	128	$T \times 128$
`bdlstm`$_2$	128	$T \times 128$
`concat[first,last]`	-	256
`fully-connected`$_1$	256	256
`leaky-relu`$_1$ `[`$\alpha = 0.3$`]`	-	256
`batch-norm`$_1$	-	256
`fully-connected`$_2$	256	256
`leaky-relu`$_2$ `[`$\alpha = 0.3$`]`	-	256
`batch-norm`$_2$	-	256

To decrease the complexity of the LSTM network (i.e. use fewer time steps), also a CNN-LSTM-based architecture [94] was evaluated, but it did not show any significant improvements in quality and time complexity.

8. Training

For different data types and datasets, the training may require much more or less time, e.g. larger datasets require in general more time. In general, the Adadelta [95] optimizer is used with a learning rate of 5.0. The training is done in iterations where each iteration is a minibatch with N generated clusterings. N may depend on the data type used, the input count n, and the available hardware; in general, higher values for N are preferred. Early stopping [96] is used: If there is no new best valid loss for more than 15 000 iterations, the training is stopped. This, in general, increases the training time for good models and decreases the training time for bad models. A test on the validation data is only done every 100th iteration.

8.1. Using Class-Labeled Data

Any data with class labels may be used to train the network. Compared to a classification problem, the requirements are slightly different: For classification problems, it is preferred to have only a few classes and many training examples per class. The data is then split into a test, a validation, and a training set, where each class is present in all sets. The given number of examples per class are often the main limitation. For the supervised clustering problem, it is preferred and often required to have many classes. Of course, many examples per class are also important. Many classes are required to learn an inter-class distance function; therefore, the limitation is often the available number of classes. The network not only has to learn that there are some classes which contain a given feature but also the difference between classes. This also has some implications for the test and training set. The dataset is split in a way that the training, validation, and test set contain disjunct sets of classes, but these sets then include all examples of the given classes. This is required to test if the network really learns the true inter-class distance-function and not just some kind of classification, therefore this function has to be applied to a set of unknown classes. This point is very important because if the data is split per class into training, validation, and test data (like for classification problems), and then the training, and the tests are done on the same classes, but disjunct object sets, the task is much easier. This basically leads the network to overfit to some classes and the network might not even learn a real inter-class distance function.

8.2. Class-Labeled Data vs. Cluster-Labeled Data

In general, the network does not require class-labeled data for the training, but cluster-labeled data. The difference is that each example, in general, is in exactly one class, but it may appear in different clusters. Given the 2D-point $(x = 1, y = 2)$: Depending on neighboring points, this point may be contained in different clusters. It is not possible to assign a fixed class to each point. Of course, it is trivial to view class-labeled data as cluster-labeled data because it can be just defined that one class with all examples is equal to a cluster: y_{ij} is 1 if two examples x_i and x_j are in the same class. One still should remember that clustering is much more general and that it is required to combine information from all input records to detect the clusters in the data.

If only class-labeled data types are used, then an example can be assigned to a fixed cluster without having a look at all other input examples. For instance, audio snippets with different voices are such examples. Each voice is a class. The class could be extracted as independent of all other examples, and then it could be compared to these. For other data types, this property is not given. For instance, for d-dimensional point sets, all examples/points must be observed to know where a cluster is and what points are contained in it. The cluster structure highly

depends on the neighboring data points. Cluster-labeled data, on the other hand, contains a set of clusters where a given example may be for each clustering in a different cluster.

Weakly labeled data may also be used for the training: This usually refers to data where it is only known that two examples, x_i and x_j, are "similar". The value s_{ij} can be defined as 1 when x_i and x_j are similar and otherwise, it is 0. The big difference between y_{ij} and s_{ij} is that $s_{1,2} = 1$ combined with $s_{2,3}$ does **not** imply that $s_{1,3} = 1$ is true, i.e. s_{ij} is not transitive. This always must be true for y_{ij}, therefore y_{ij} is transitive. All examples x_i and the s_{ij} values (=the complete dataset) can be visualized as a (not necessary connected) graph: The examples x_i are the vertices, and if $s_{ij} = 1$, then there is an edge between x_i and x_j. An example for this is shown in Figure 8. To generate clusters for the training, one could sample disconnected subgraphs (e.g., the subgraphs $\{x_1, x_2, x_3, x_4\}$, and $\{x_7, x_8\}$). It is very important that these subgraphs have no direct connection via an edge and that the s_{ij}-values are meaningful for the given data type. For the selected subgraphs which are basically the resulting clusters, it is easy to generate the y_{ij}-values because they are just 1 when the two examples are in the same subgraph, and otherwise, they are 0. It is possible to visualize class-labeled data in this way by creating a disconnected graph for each class where each of these subgraphs is fully connected. Also, density-based clustering for point data may be visualized as such a graph: Each possible point is a vertex, and if two points are near to each other, then they have an edge. This shows that such a graph may even have an infinite number of vertices and edges. The minimum requirement for a dataset used for the proposed model is, therefore, to have the exact s_{ij}-value for all possible pairs of examples.

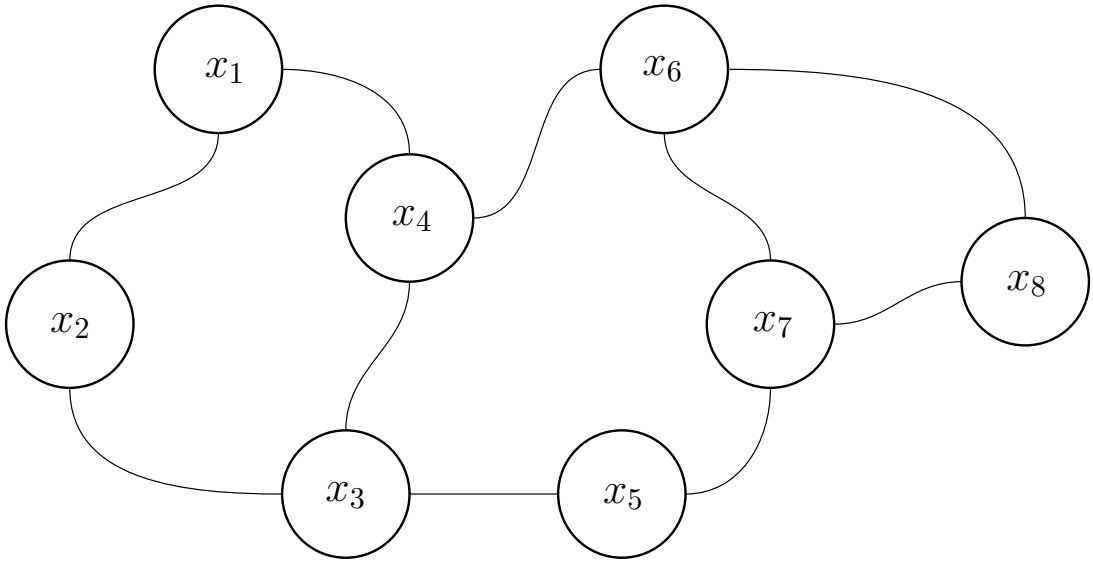


Figure 8.: A possible visualization as a graph of a dataset containing 8 examples $x_1 \dots x_8$ where two examples/vertices x_i and x_j are connected by an edge if $s_{ij} = 1$, which means that they are "similar".

Before the training starts, the data has to be split into a test, a training, and a validation set. This is not done for the point data because new examples can be generated in real time. The input count and the minibatch size highly depend on the data and the available hardware resources. Image data may only allow lower minibatch sizes N than low-dimensional point data does.

8.3. Data Generation

In each iteration, input data is generated: First the number of target clusters is generated. This number is sampled from a uniform distribution over all possible cluster counts. Then for each cluster, a distinct random class is selected. In the experiments, a minimal number of n_m examples per cluster is defined ($n_m = 2$ is used); therefore, for each class, there are n_m examples sampled and added to the set of input examples. Then, until there are enough input examples in total ($=n$), a random class of the defined clusters is sampled and then, a random example of this class is sampled. Sampling examples may already include data augmentation. These steps are different for point data: There, just a set of points may be generated at once (given a cluster count k and a point count n).

Given the input data, it is possible to train the network. To avoid some systematic error, the list of input examples is shuffled before it is used as input for the network. A clustering algorithm and, therefore, also the implemented network should be invariant, according to the order of the input examples. This method allows it to generate extremely many distinct training records for a classification dataset. Already, the input permutation generates $n!$ possibilities. The complete clustering task, in general, is quite difficult to train, and the limitation is often the number of classes.

9. Evaluation

To evaluate the results of the experiments, different methods are used. Metrics are used to measure how good the average clustering quality is; forward dropout [97] is used to measure the confidence of the network's decisions, and of course, also loss and accuracy graphs are generated to get a feeling about quality of the network. The following sections describe the metrics used and how exactly forward dropout is used.

9.1. Metrics

For all experiments, several evaluation metrics are used, including the BBN metric [98], the *misclassification rate* (MR) [50], the *normalized mutual information* (NMI) [99], and many more. In the literature there are many different metrics described [50] [54] [55] [56]. In this document, there are only 3 metrics used: These are the most intuitive and comparable metrics, and many other metrics have a very similar behavior. Nevertheless, the implemented software (see Chapter 13) computes many more scores and metrics. It allows adding custom metrics with a minimal effort.

The value range for all metrics used is $[0, 1]$, where all metrics are better when the value is higher, except for the MR where a lower value is better. The following sections explain how the metrics used are defined and what the intuition behind them is.

9.1.1. BBN

The BBN metric [98] is especially used in speaker clustering and diarization. The number of speakers, which is equal to the number of true clusters, is called N_s. N_c is the number of proposed clusters. The number of examples in the cluster i which are spoken by speaker j (=are contained in the true cluster j) are described by n_{ij}. Finally, $n_{i.}$ describes the number of examples in the proposed cluster i. The BBN metric has a parameter Q, which defines whether small or large clusters are preferred. For the conducted tests $Q = 0$ is used because there is no preference. The BBN metric is defined by the following expression:

$$\mathrm{BBN}_Q = \sum_{i=1}^{N_c} \sum_{j_1}^{N_s} \frac{n_{ij}^2}{n_{i.}} - QN_c$$

The metric is not normalized; therefore, a normalized version is introduced because otherwise, it may be hard to interpret the score given the different cluster counts. The normalized version $\mathrm{BBN}_{\mathrm{norm}}$ (given $Q = 0$) is described in the formula below and has a value range of $[0, 1]$:

$$\mathrm{BBN}_{\mathrm{norm}}(y, \widetilde{y}) = \frac{\mathrm{BBN}_{Q=0}(y, \widetilde{y})}{\mathrm{BBN}_{Q=0}(y, y)}$$

For larger Q-values, the simple normalization used does not always work because it might happen that the value range of the BBN-metric is negative, and the optimal value for a given clustering is equal to zero. Therefore, by using this simple scaling, this would lead to a division by zero. The normalized BBN metric mainly measures the purity of clusters, i.e. pure (=homogeneous) clusters increase the score.

9.1.2. Misclassification Rate (MR)

The MR [50] is used because previous works also use it [12] [35], and it is a very intuitive measure: It assumes that each cluster represents a class, and then, this measure is defined as the fraction of wrongly assigned examples. Because there are many possibilities of assigning a class to a cluster (exactly $k!$ for k clusters), the MR is defined as the minimum possible value. This is especially very useful when the underlying dataset is a classification dataset. Originally, the MR was used in speaker clustering and is defined as

$$\mathrm{MR} = \frac{1}{N}\sum_{k=1}^{N_s} e_j$$

where N defined as the total number of audio pieces to cluster, N_s is the number of speakers, and e_j is the number of audio pieces assigned wrongly to speaker j. This metric is generalized and instead of audio pieces, uses more general objects, and instead of speakers, it uses the true object clusters.

9.1.3. Normalized Mutual Information (NMI)

The NMI metric [99] is a widely used metric [49] [100] in clustering. The *mutual information* (MI) describes the information of one random variable U obtained through a second random variable V. The NMI metric normalizes this value to the range $[0, 1]$ by the square root of the product of the entropy of both random variables. The metric is given by the following expression:

$$\mathrm{NMI}(U, V) = \frac{\mathrm{MI}(U, V)}{\sqrt{H(U)H(V)}}$$

where MI is given by

$$\mathrm{MI}(U, V) = \sum_{i=1}^{|U|}\sum_{j=1}^{|V|} \frac{|U_i \cap V_i|}{N} \log\left(\frac{N|U_i \cap V_i}{|U_i||V_i|}\right).$$

The two values U and V represent an object-to-cluster assignment for N objects. The expression $\frac{|U_i \cap V_i|}{N}$ describes the probability that an example picked at random falls into the class $U_i \cap V_i$.

The MI / NMI metrics are not only used for clustering, but they may be used for any random variables, which makes them quite general. Given this, an intuitive explanation is possible: Given two random variables, e.g., one that describes the current month (U) and the other that describes the weather (V), it is easier to predict the weather (V) when the month (U) is known because the weather distribution highly depends on the month (e.g. it is more probable that there is snow in the winter). In this example, it can be seen that by knowing U, there is some information gained about V. If the information in U fully describes V, and by knowing V, no more information is gained, then the NMI is equal to 1.0. If the information in U is completely independent of the information in V, and therefore, by knowing U does not help in knowing anything about V, then the NMI is equal to 0.0. If this score is used for clustering, the information gain by knowing the true clustering is measured: If there is no information gain (by already having the correct result/clustering), then the score is 1.0, otherwise, it is lower.

9.2. Forward Dropout

Additional to the metrics, also some tests with forward dropout [97] are conducted. These tests show how confident the network is with the given answer. This is very helpful in seeing how stable the network output is.

This kind of test is especially used in the cluster count estimation: It is a very good indicator to see how good the network is able to handle the given data. For instance, considering the cluster count for 2D-point data, the network is very confident.

The basic idea of forward dropout is to use dropout [101] in the forward pass: This randomizes the internal state of the neural network. If the network is very confident with its decisions, the output (of e.g. a classification) is only changed a little bit. On the other side, if the decisions inside the network are very unstable, then the output may change a lot. The changes of the output may be used to describe how certain the network is about its decisions.

For the tests conducted, not only forward dropout is used but also random permutations for the inputs. The clustering network should be invariant according to the order of the inputs; therefore, this additional randomization of the input is used.

Finally, $n_r = 1000$ runs are executed for a given collection of examples, and the cluster count probabilities $P(k)$ of the cluster estimations are collected. This means that there are $k_{\max}$ values per run stored. An example for $k_{\max} = 5$ is shown in Table 4.

Table 4.: Forward Dropout test: $n_r = 1000$ randomized runs are conducted on a set of examples. The resulting cluster count distribution is shown in this table. Because $k_{\max} = 5$, there are five columns.

Run	$P_r(k=1)$	$P_r(k=2)$	$P_r(k=3)$	$P_r(k=4)$	$P_r(k=k_{\max}=5)$
$r=1$	0.0	0.0	0.2	0.3	0.5
$r=2$	0.0	0.0	0.0	0.3	0.7
...	...	...	...	...	...
$r=n_r=1000$	0.0	0.0	0.1	0.3	0.6

Given the list of all probabilities, it is possible to create the distributions for each column $P_r(k=c)$. The 5 resulting distributions and a final distribution over all possible cluster counts are shown in Figure 9. The modus of the 5 distributions is green. The final distribution uses the modus of the distribution per cluster count k and visualizes the final network decisions. The most probable cluster count is marked green. The more concentrated a distribution is, the more confident the network is about a decision. In the most confident cases, the network always returns the same value for $P_r(k=c)$, and in the least confident case, the value of $P_r(k=c)$ is uniformly distributed over $[0,1]$.

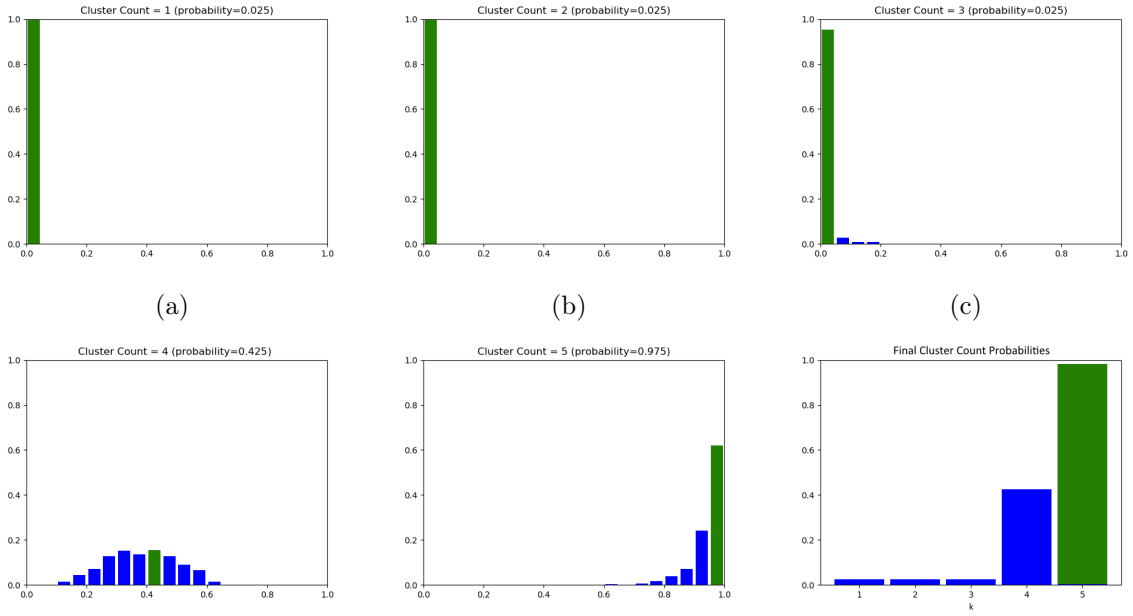


Figure 9.: The Figures 9a - 9e show the distribution for $P_r(k = 1)$, ..., $P_r(k = 5)$: The title contains the cluster count k and the final probability. The probabilities in the x-axis are discretized to 20 possible values. The x-axis shows the output values of $P_r(k = x)$, and the y-axis shows frequency of the given probability in the data. The final probability for the given cluster count is calculated as the modus and is marked green. As can be seen, the network is very sure that there are not $k = 1$ or $k = 2$ clusters in the given data. Also, $k = 3$ clusters are very unlikely. It can be seen by the broad curve in Figure 9d that the probability for $k = 4$ clusters is not very stable, therefore the network is not very sure about this decision. It still can be seen that the probability for $k = 4$, independent of the confidence, is very low. The output probability for $k = 5$ clusters is much higher and also more confident (i.e. the distribution is more concentrated). The resulting probabilities for each possible cluster count are visualized in Figure 9f. The x-axis describes the cluster count, and the y-axis represents the modus of the corresponding distribution (see Figures 9a - 9e). It is important to note that the sum of these final probabilities in Figure 9f is **not** normalized.

10. Datasets

Different datasets were implemented for conducting the experiments, in which 2D-point data, audio data, and images were used. This chapter contains a description for some of the datasets used: What splits are used to create the test, the training and the validation set, and what kind of data augmentation was used.

All datasets are implemented quite generally and may be reused for other projects. Even more datasets were implemented than listed here, but they were only used in a very few experiments and, therefore, are not relevant in this thesis.

10.1. Data Augmentation

Data augmentation is used for most of the datasets. The software library used is described in Chapter 13. This section gives an overview of the different effects of the data augmentation methods used. Table 5 visualizes the different data augmentation methods applied to a given image.

Table 5.: The different methods used for data augmentation. The input image is taken from the LFW-crop dataset (see Section 10.9) and shows the actress Sarah Wynter. The image has a resolution of 128×128 pixels and has 3 color channels.

Method	**Configuration**	**Result**
None	-	
Random Distortion	A grid with size 4×4 and a magnitude of 8	
Flip left/right	-	
Flip top/bottom	-	

10.2. 2D-Point Data

The 2D-point dataset is used to perform the basic tests for a clustering algorithm. The implemented dataset contains almost an infinite number of different clustering possible because the data can just be generated at runtime. Therefore, this dataset is optimal for evaluation purposes. For a good clustering algorithm, it should not be a problem to solve this clustering task. The parameters for generating data are the number of data points n and the number of clusters k. The default target domain is $[0, 1] \times [0, 1]$

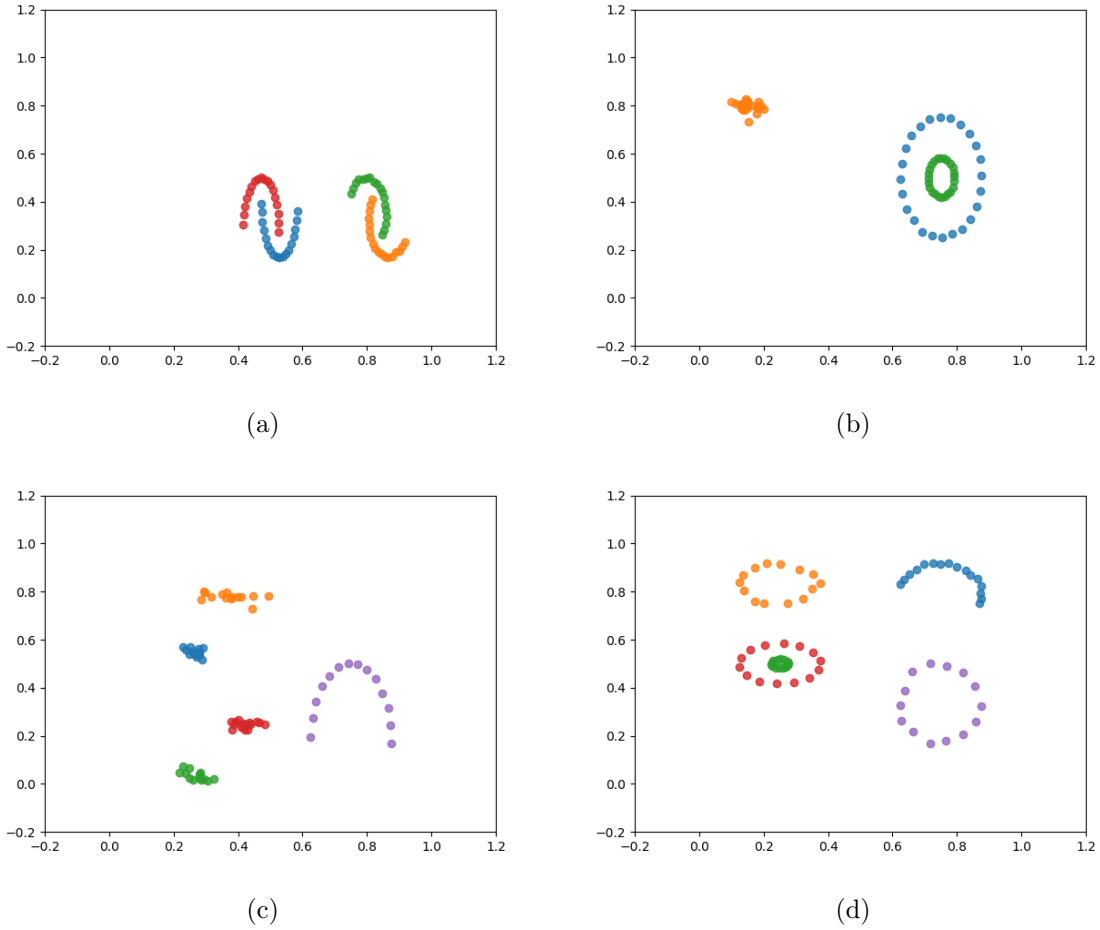
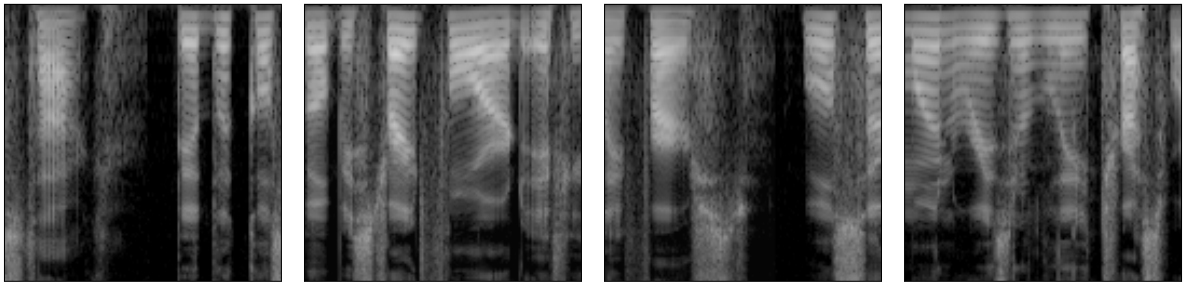


Figure 10.: Different cluster examples with 2D-points which are generated.

Figure 10 shows some example clusters shown that were generated by the algorithm. As can be seen, it is possible to generate moon shapes, circles, and Gaussian distributions. The clusters never overlap.

10.3. TIMIT



(a) (b) (c) (d)

Figure 11.: Four examples of data records from the TIMIT dataset, each with a length of 1.28 seconds.

TIMIT [64] is a well-known speech corpus dataset. It may be used for many different tasks, but for the experiments in this thesis, it is used for speaker clustering. The data preprocessing used is very similar to [12]. TIMIT contains 630 speakers, where each speaker has 10 spoken sentences. Per speaker, all 10 audio pieces are ordered by their filename and are then concatenated to 1 long audio clip. This audio clip is then converted to a mel-spectrogram, for which the following settings are used:

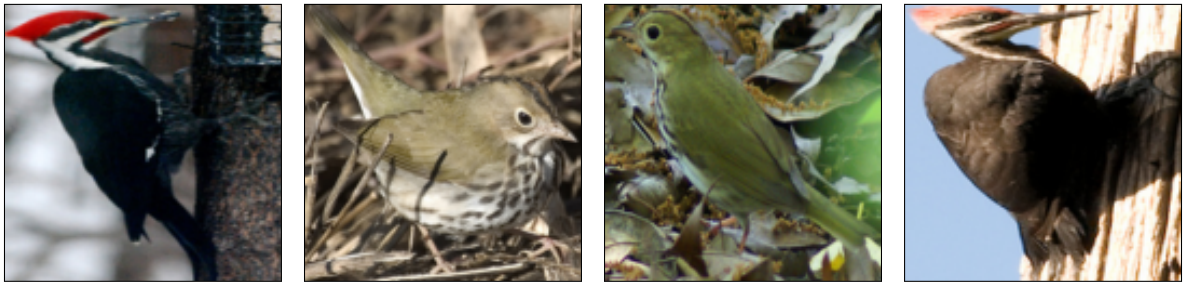
- Sample rate: 16kHz
- Mel coefficients: 128
- Window length: 320
- Hop length: 160
- FFT window size: 512

This leads to a matrix with the dimensions $128 \times N_s$ per speaker, where $10N_s$ is the speaker-specific length of the concatenated audio piece in milliseconds. Different experiments may require audio pieces of different lengths: These pieces are randomly sampled at runtime from the $128 \times N_s$ matrix. For instance, if a 2.56-second audio snippet of a speaker is required, then the result will be a 128×256 matrix.

The training set used contains 430 speakers, and the test and validation set each contains 100 speakers. The `testlist200` defined by [**?**] is used to define the test and validation set: The first 100 speakers are used for the test set, and the second 100 are for the validation set.

Example records of the dataset are visualized in Figure 11. The implemented neural networks handle these inputs as 2D images with the size $128 \times T$ and 1 color channel. For the experiments described in this document, $T = 128$ is always used.

10.4. Caltech-UCSD Birds 200



(a) (b) (c) (d)

Figure 12.: Four examples of data records from the Caltech UCSD Birds 200 dataset which are rescaled to 128×128 pixels without applying data augmentation.

The Caltech-UCSD Birds 200 [102] dataset contains 6033 color images of 200 different birds. These images are rescaled to 128×128 pixels, and at runtime, the following data augmentation pipeline is used:

1. Random distortion based on a grid with the size 4×4 and a magnitude of 8
2. Flip left/right with a probability of 50%
3. Flip top/bottom with a probability of 50%

160 birds/classes are used for the training, 20 for the validation, and 20 for testing. The complete list of birds is shuffled with a fixed seed, and then, the different sets are extracted.

Example records of the dataset are visualized in Figure 12. Usually, the tested neural networks handle these inputs as 2D images with the size 128×128 and 3 color channels.

10.5. CIFAR-100

(a) (b) (c) (d)

Figure 13.: Four examples of data records from the CIFAR-100 dataset without applying data augmentation.

CIFAR-100 [103] is a dataset with 100 classes, where each class contains 600 color images, each with a resolution of 32×32 pixels. A similar dataset exists, called CIFAR-10 [103], which has only 10 classes but with more examples per class. At runtime, the following data augmentation pipeline is used:

1. Random distortion based on a grid of size 4×4 and a magnitude of 8
2. Flip left/right with a probability of 50%

80 classes are used for the training, 10 for the validation, and 10 for testing. The complete list of classes is shuffled with a fixed seed, and then the different sets are extracted.

Examples of the CIFAR-100 dataset records are visualized in Figure 13. Usually, the neural networks tested handle these inputs as 2D images with the size 32×32 and 3 color channels.

10.6. COIL-100

(a) (b) (c) (d)

Figure 14.: Four examples of data records from the COIL-100 dataset without applying data augmentation.

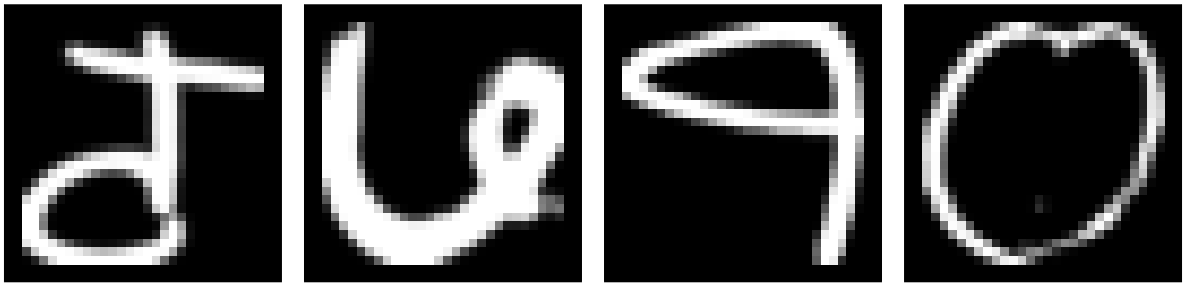
The COIL-100 [1] dataset consists of 100 classes where each class contains 72 color images, each with a resolution of 128×128 pixels. This dataset is used by other approaches [49] [104] [105] to evaluate image clustering. At runtime, the following data augmentation pipeline is used:

1. Random distortion based on a grid of size 4×4 and a magnitude of 8
2. Flip left/right with a probability of 50%

80 classes are used for the training, 10 for the validation and 10 for testing. The complete list of classes is shuffled with a fixed seed, and then the different sets are extracted.

Example records of the COIL-100 dataset are visualized in Figure 14. Usually, the neural networks tested handle these inputs as 2D images with the size 32×32 and 3 color channels.

10.7. Devanagari Characters



(a) (b) (c) (d)

Figure 15.: Four examples of data records from the Devanagari character dataset without applying data augmentation.

The Devanagari character [106] dataset contains 46 classes and a total of 92 000 images. The grayscale images have a resolution of 32×32 pixels. Random distortion based on a grid with the size 4×4 and a magnitude of 8 is applied for data augmentation.

36 classes are used for the training, 6 for the validation, and 6 for testing. The complete list of classes is shuffled with a fixed seed, and then the different sets are extracted.

Examples of the Devanagari character dataset records are visualized in Figure 15. Usually, the neural networks tested handle these inputs as 2D images with the size 128×128 and 1 color channel.

10.8. FaceScrub

(a) (b) (c) (d)

Figure 16.: Four examples of data records from the FaceScrub dataset without applying data augmentation.

FaceScrub [107] is a dataset with the face images of 530 people. In total, there are 50 092 images in the dataset. All face images are rescaled to 128×128 pixels and handled as color images (some images in the dataset are only in grayscale). In general, the quality of the images is very different, and some are not even available because the dataset only contains a list of links, and the images have to be downloaded. Therefore, some cleanup is required before the dataset is used. At runtime, the following data augmentation pipeline is used:

1. Random distortion based on a grid of size 4×4 and a magnitude of 8
2. Flip left/right with a probability of 50%

424 classes are used for the training, 53 for the validation, and 53 for testing. The complete list of classes is shuffled with a fixed seed, and then the different sets are extracted.

Example records of the dataset are visualized in Figure 16. Usually, the tested neural networks handle these inputs as 2D images with the size 128×128 and 3 color channels.

10.9. Labeled Faces in the Wild

(a) (b) (c) (d)

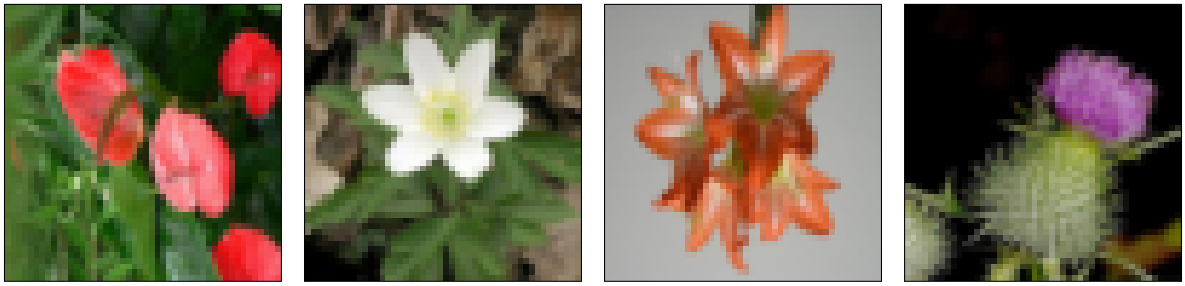
Figure 17.: Four examples of data records from the LFW-crop dataset.

Labeled Faces in the Wild (LWF) [108] is a dataset with face images that is widely used for validating algorithms and for doing tests [8] [109] [110] [111] but not to train them. The dataset contains 5749 classes and a total of 13 233 images. Most classes contain only one image.

Many images contain relatively much context around the face. This is, for instance, not the case with the images of the FaceScrub dataset (see Section 10.8). Due to that LFW should be used to validate the model which is trained with FaceScrub, the LFW-funneled images are used as a base and from these 256×256 images, the center 128×128 pixels are extracted. In this work, this version of the LFW dataset is called "LFW-crop". This dataset is only used for testing; therefore, no data augmentation is applied to these images.

Examples of the LFW-crop dataset records are visualized in Figure 17. Usually, the tested neural networks handle these inputs as 2D images with the size 128×128 and 3 color channels.

10.10. Flowers-102



(a) (b) (c) (d)

Figure 18.: Four examples of data records from the dataset without applying data augmentation.

The 102-Category-Flower or Flowers-102 [112] dataset contains color images of 102 different flowers. Each class of these 102 classes contains 40-258 images. The images are rescaled to 48×48 pixels. At runtime, the following data augmentation pipeline is used:

1. Random distortion based on a grid with the size 4×4 and a magnitude of 8
2. Flip left/right with a probability of 50%
3. Flip top/bottom with a probability of 50%

82 classes are used for the training, 10 for the validation, and 10 for testing. The complete list of classes is shuffled with a fixed seed, and then the different sets are extracted.

Examples of the Flowers-102 dataset records are visualized in Figure 18. Usually, the neural networks tested handle these inputs as 2D images with the size 48×48 and 3 color channels.

10.11. Tiny ImageNet

(a) (b) (c) (d)

Figure 19.: Four examples of data records from the dataset without applying data augmentation.

The Tiny ImageNet dataset is a subset of the ImageNet [113] dataset. Initially, it was created as a part of an image classification challenge in the CS231n course by Fei-Fei Li, Andrej Karpathy, and Justin Johnson.[1] Then the dataset was also used by people outside of this course, e.g. there was a Kaggle competition.[2] All images are already downscaled to 64×64 pixels and have 3 color channels. There are 200 classes available, and each class contains 600 images. At runtime, the following data augmentation pipeline is used:

1. Random distortion based on a grid of size 4×4 and a magnitude of 8
2. Flip left/right with a probability of 50%

160 classes are used for the training, 20 for the validation, and 20 for testing. The complete list of classes is shuffled with a fixed seed, and then the different sets are extracted.

Examples of the Tiny ImageNet dataset records are visualized in Figure 19. Usually, the neural networks tested handle these inputs as 2D images with the size 64×64 and 3 color channels.

[1] http://cs231n.stanford.edu/
[2] https://www.kaggle.com/c/tiny-imagenet/leaderboard

10.12. Cars-196

(a) (b) (c) (d)

Figure 20.: Four examples of data records from the dataset without applying data augmentation.

The Cars-196 [114] dataset contains 16 185 images of 196 different cars. In the dataset, the image resolution and quality vary considerably. All images are rescaled to 128×128 pixels. At runtime, the following data augmentation pipeline is used:

1. Random distortion based on a grid of size 4×4 and a magnitude of 8
2. Flip left/right with a probability of 50%

146 classes are used for the training, 25 for the validation, and 25 for testing. The complete list of classes is shuffled with a fixed seed, and then the different sets are extracted.

Examples of records of the Cars-196 dataset are visualized in Figure 20. Usually, the tested neural networks handle these inputs as 2D images with the size 128×128 and 3 color channels.

10.13. SUN-397

(a) (b) (c) (d)

Figure 21.: Four examples of data records from the Sun-397 dataset without applying data augmentation.

The SUN-397 [115] dataset contains images of different scenes. There are 397 scenes/classes and a total of 108 754 color images. All images are rescaled to 128×128 pixels. At runtime, the following data augmentation pipeline is used:

1. Random distortion based on a grid of size 4×4 and a magnitude of 8
2. Flip left/right with a probability of 50%

317 classes are used for the training, 40 for the validation, and 40 for testing. The complete list of classes is shuffled with a fixed seed, and then the different sets are extracted.

Examples of the Sun-397 dataset records are visualized in Figure 21. Usually, the neural networks tested handle these inputs as 2D images with the size 32×32 and 3 color channels.

11. Experiments

Many parameter sets and network architectures have been evaluated and tested in the thesis. In total, there were more than 150 different experiments conducted. Therefore, only the most important and informative experiments which were conducted on the final model are described in this chapter. As described in Chapter 13, the source code for all experiments is online available. The evaluation metrics and methods used are described in Chapter 9.

All experiments described use 1-5 clusters with a minimum cluster size of $n_m = 2$. This means that $k_{\max} = 5$. Not only the training was conducted using this configuration but also the tests. The cluster count is sampled uniform and randomly. All experiments contain the resulting values for the NMI metric, the BBN_{norm} metric, and the MR metric. Given the configuration described, the mean metrics for random clusterings can be calculated. This value still depends on the input example count n. Therefore, the corresponding mean value for $n = 20$ or $n = 72$ is computed for all experiments. These mean metrics for random clusterings are shown in Table 6. They can be used as a lower-bound reference: Metrics worse than this should never appear.

Table 6.: Mean metrics for random clusterings. The values are calculated by measuring the metrics for 5000 random clusterings.

Metric	**Best/Worst Value**	**Mean Value for $n = 20$**	**Mean Value for $n = 72$**
NMI	1.0/0.0	0.3411 ± 0.0049	0.2329 ± 0.0222
BBN_{norm}	1.0/0.0	0.5479 ± 0.0033	0.4839 ± 0.0158
MR	0.0/1.0	0.4236 ± 0.0031	0.4853 ± 0.0148

In all experiments, all graphs are smoothed by using an averaging sliding window approach. The sliding window has a length of 0.5% of the iteration count of the graphs. This is done because the graphs show the values for each iteration (where an iteration is equivalent to a mini-batch, often with a small size of $N = 25$) and, therefore, have a high variance. If epochs could be used, this problem would not exist because then, each datapoint would represent the average over many minibatches.

The experiments described are chosen to show the strengths and weaknesses of the model. Many experiments were conducted on image datasets to see whether image clustering works well. Some tests on other data types were conducted and finally, also a regularization term was tested. The objective of this section is to show what already works well and what does not work well. The intention is not to show how the final model architecture was derived and improved over time; therefore, all tests use the final model architecture.

11.1. Hardware

All experiments are conducted inside a Docker container with the following resources:

- NVIDIA Titan X Pascal with 12 GiB memory
- Intel Xeon E5-2650 CPU
- 48 GiB of RAM

11.2. 2D-Point Data

11.2.1. Description

This experiment is conducted on the final model and uses the dataset described in Section 10.2. No embedding network is used (see Section 7.1).

11.2.2. Training

The minibatch size used is $N = 200$ and the training is done for 128 700 iterations, where each iteration basically contains one sampled minibatch. The input example count is $n = 72$. The training was ended by early stopping after 123 hours. The training took comparably much time because the network performed well and improved for a long time. The quality was already very high after about half the time, but it was still increasing. There is no training set, no validation set, and no test set: The data is generated in real time; therefore, there are almost an infinite number of different training records available.

11.2.3. Results

Some training graphs are shown in Figure 22. These visualize the improvement over time in the loss terms, the cluster-count estimation accuracy, and the metrics on the validation set. By using early stopping, not the latest network configuration on the graphs is used for further tests. Only the training set graph is plotted because there is no validation set for the 2D-point data.

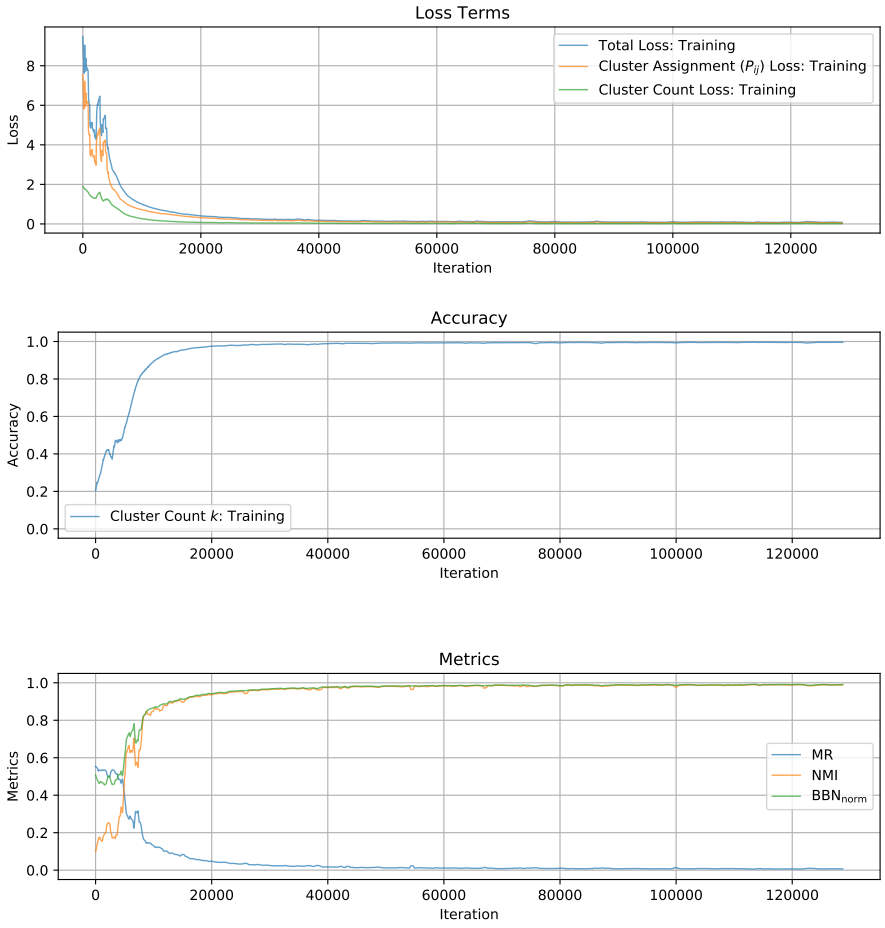


Figure 22.: 2D-Point Data: Training graphs

The finally metrics achieved using the best network configuration are shown in Table 7.

Table 7.: 2D-Point Data: The metrics achieved by performing 300 clusterings.

Metric	**Mean**
NMI	0.9925 ± 0.0015
BBN_{norm}	0.9937 ± 0.0014
MR	0.0038 ± 0.0009

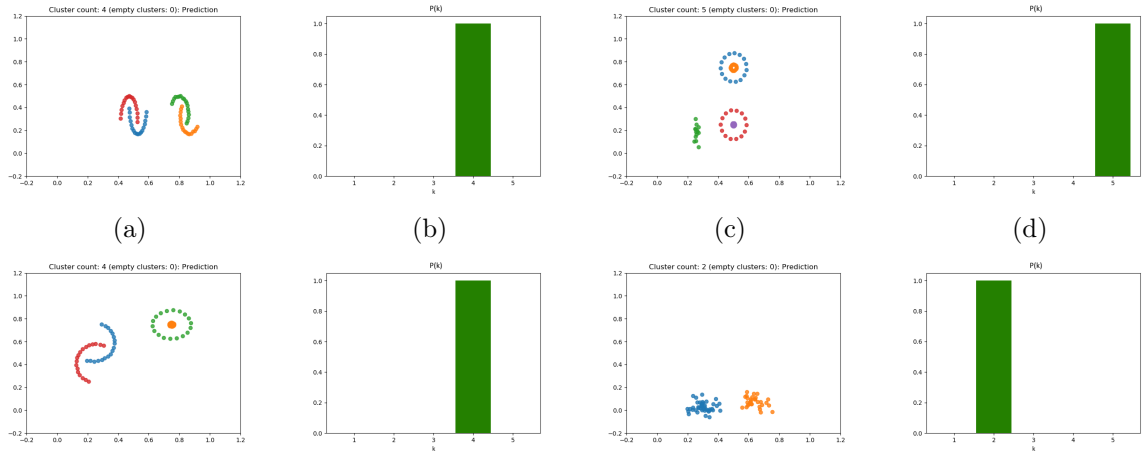
Results achieved with forward dropout could show that this model is always very confident in its decisions. The same training is done for a model with BDLSTM layers, instead of RBDLSTM layers (see Section 3.1). Except for the cluster count estimation, there is no convergence at all, even after 90 hours of training and 66 000 training iterations. The metrics achieved are shown in Table 8. It can be seen that these are much worse than in the model using the RBDLSTM layers.

Table 8.: 2D-Point Data model using BDLSTM layers instead of RBDLSTM layers: The metrics achieved by performing 300 clusterings.

Metric	**Mean**
NMI	0.2461 ± 0.0045
BBN_{norm}	0.4758 ± 0.0071
MR	0.5158 ± 0.0032

11.2.4. Example

To complete the results of the experiment described, some example clustering is given in Figure 23. As can be seen, the results are usually very good.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 23.: Four clustering examples of the 2D-point data experiment. The left image always shows the resulting clusters, and the right image shows the cluster count distribution $P(k)$. The scores are optimal for all examples.

11.2.5. Comparison with Classical Algorithms

As can be seen in Figure 23, the clusters generated may have many different shapes. But one might wonder how well other algorithms like k-means [3] or DBSCAN [62] work on this dataset. For this reason, the metrics are calculated for these two algorithms: The result is shown in Table 9. It can be seen that the proposed model outperforms both algorithms.

Another interesting point for comparing algorithms might be their runtime behavior: The proposed model has a runtime complexity of $\mathcal{O}(n)$ (see Chapter 3). The k-mean's runtime complexity is $\mathcal{O}(ndki)$ where d describes the dimensionality of the inputs, k is the number of clusters, and i is the number of iterations. By assuming that d, k, and i are fixed for an implementation (given a specific and fixed data type), the effective runtime for n inputs is $\mathcal{O}(n)$. DBSCAN, on the other hand, has a complexity of at least $\mathcal{O}(n \log n)$. The asymptotic complexity of the proposed model is therefore comparable with the k-means and is optimal by assuming that conducting clustering n examples requires accessing each example at least once. DBSCAN has a worse asymptotic runtime complexity.

Table 9.: 2D-Point Data: The metrics achieved by performing 300 clusterings with different algorithms.

Approach	**NMI**	**BBN$_{\text{norm}}$**	**MR**
k-means	0.7965 ± 0.0142	0.8333 ± 0.0093	0.1784 ± 0.0152
DBSCAN	0.6768 ± 0.0201	0.7356 ± 0.0136	0.2651 ± 0.0149
Proposed model	0.9925 ± 0.0015	0.9937 ± 0.0014	0.0038 ± 0.0009

11.2.6. Conclusions

By the model architecture proposed, the task of clustering 2D-points can be seen as solved. The results are very accurate, and the network is very confident. In particular, this experiment shows that the network architecture is able to do simple clustering. This allows for conducting tests on more complex data types.

11.3. TIMIT

11.3.1. Description

This experiment is conducted on the final model and uses the dataset described in Section 10.3. The CNN-embedding network described in Section 7.2 is used. This test is conducted to see whether or not the network is able to cluster complex data types.

11.3.2. Training

The minibatch size used is $N = 25$ and the training was done for 72 500 iterations, where each iteration basically contains one sampled minibatch. The input example count is $n = 20$. The training was ended by early stopping after 18 hours.

11.3.3. Results

Figure 24 shows some of the training graphs. These visualize the improvement over time for the loss terms, the cluster-count estimation accuracy, and the metrics on the validation set. By using early stopping, not the latest network configuration on the graphs is used for further tests. It can be seen that the training process is very smooth, and there is no significant overfit.

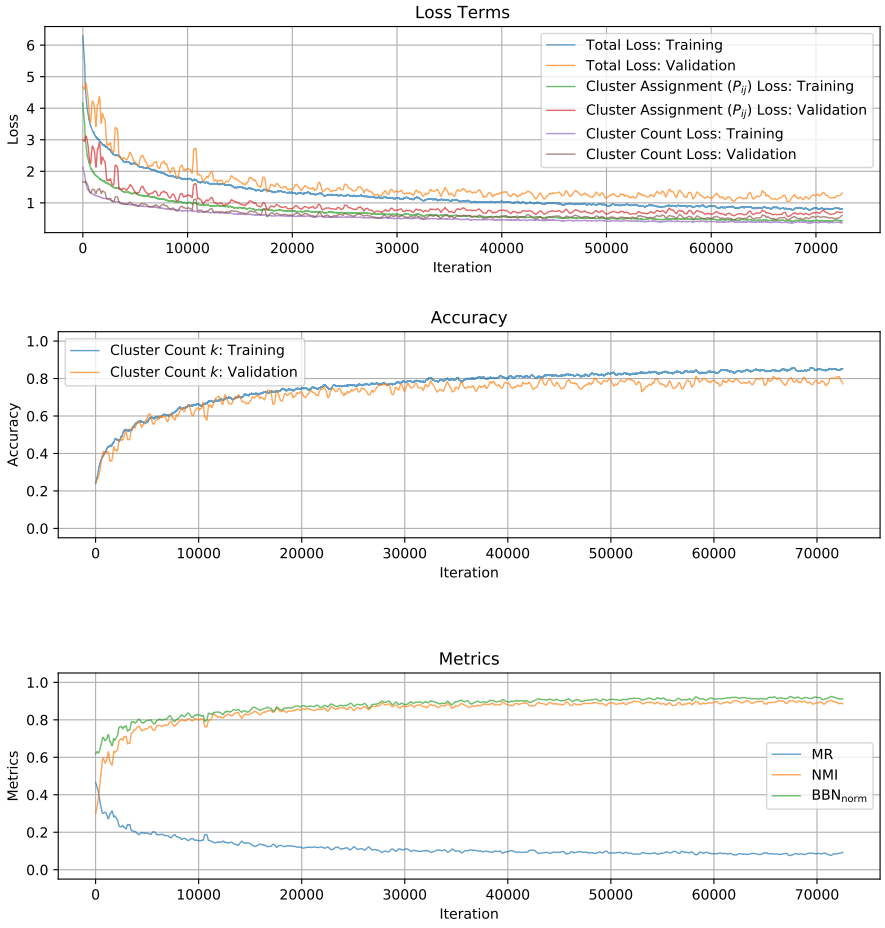


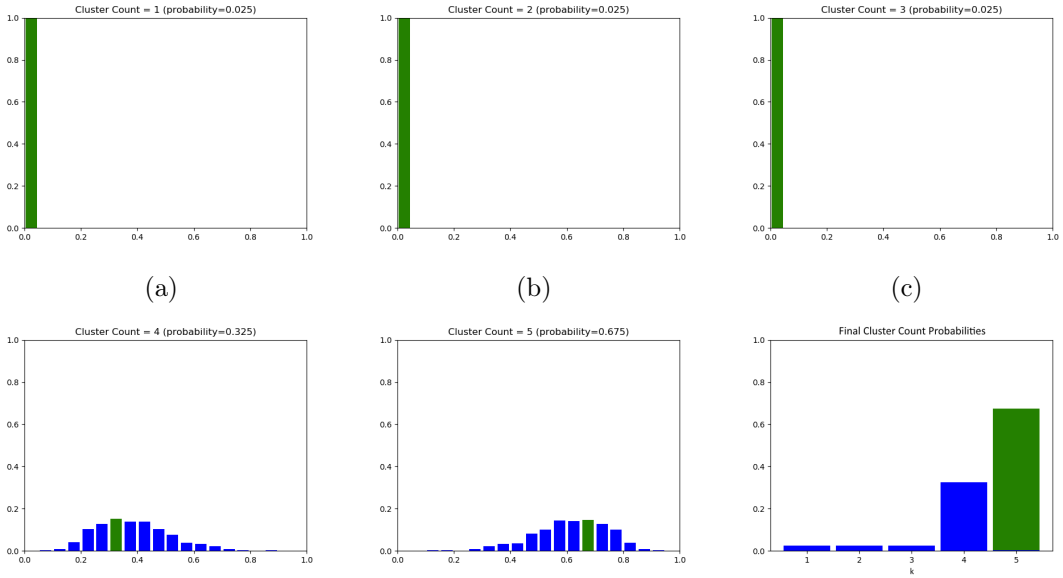
Figure 24.: TIMIT: Training graphs

Table 10 shows the metrics finally achieved by using the best network configuration on the test data.

Table 10.: TIMIT: The metrics achieved by performing 300 clusterings on the test set.

Metric	**Mean**
NMI	0.9283 ± 0.0072
BBN_{norm}	0.9350 ± 0.0060
MR	0.0603 ± 0.0056

By the results achieved with forward dropout, it could be shown that the network is very confident for a small number of clusters. When there are 4 or more clusters, then the confidence of the network decreases, but it is still sufficiently high. This effect is shown in Figure 25.



(a) (b) (c)

(d) (e) (f)

Figure 25.: It can be seen that the network is very confident in the probabilities that there are $k = 4$ or $k = 5$ clusters present. On the other hand, the overall probability that there are $k = 5$ clusters present is significantly higher than that of $k = 4$ clusters being present. The final cluster count prediction is shown in Figure 25f. The mode of each cluster count prediction and the final cluster count predicted are marked in green.

11.3.4. Example

To complete the results of the described experiment, an example of the clustering is given in Table 11. The predicted cluster count distribution $P(k)$ is shown in Figure 26. The scores for the example shown are perfect: NMI=1.0, BBN_{norm}=1.0, MR=0.0.

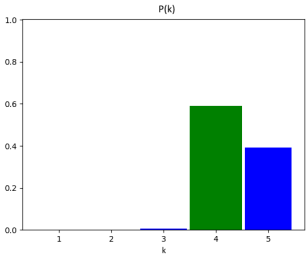
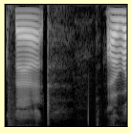
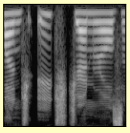
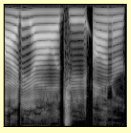
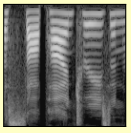
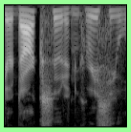
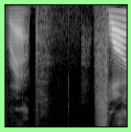
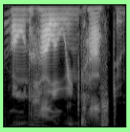
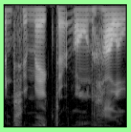
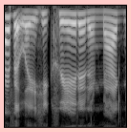
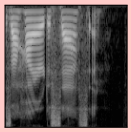
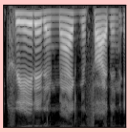
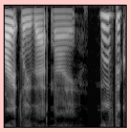
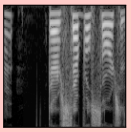
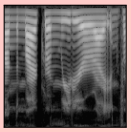
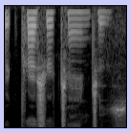
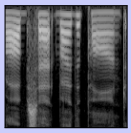
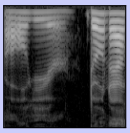
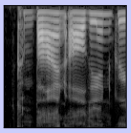
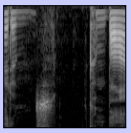


Figure 26.: The predicted cluster count distribution $P(k)$. The true cluster count is marked in green.

Table 11.: The network's proposed clusters for a given set of TIMIT audio snippets of the test set. Each row is a cluster proposed by the network, and each speaker has another color for his/her snippets (=ground truth). For each snippet, the TIMIT speaker name and the audio start, and the end positions are given in milliseconds. The snippets are encoded as a mel-spectrogram.

Cluster	Objects					
0	 FGMB0 2288-2416	 FGMB0 2542-2670	 FGMB0 3290-3418	 FGMB0 631-759	 FGMB0 664-792	
1	 MKAH0 1549-1677	 MKAH0 1615-1743	 MKAH0 2610-2738	 MKAH0 58-186		
2	 FREH0 1186-1314	 FREH0 2120-2248	 FREH0 514-642	 FREH0 713-841	 FREH0 931-1059	 FREH0 977-1105
3	 FSXA0 1569-1697	 FSXA0 1645-1773	 FSXA0 1955-2083	 FSXA0 286-414	 FSXA0 3204-3332	

11.3.5. Comparison with the State-of-the-Art

Lukic et al. [12] reach a perfect MR of 0.0 for more speakers than tested in this experiment. They outperform the proposed model. Nonetheless, the scores reached with this experiment are also competitive. Experiments with more speakers in a network input gave a worse result; therefore, the speaker/cluster count is limited to $k_{\max} = 5$.

11.3.6. Conclusions

Using a spectrogram of some spoken text, the clustering of speakers works very well with the architecture described. It still can be improved, but the experiment shows that this task can definitely be solved by the network architecture proposed. Therefore, the model is able to not only cluster simple data types like 2D-point data but also more complex data like spectrograms.

11.4. COIL-100

11.4.1. Description

This experiment is conducted on the final model and uses the dataset described in Section 10.6. The CNN-embedding network described in Section 7.2 is used. This experiment is conducted to see whether or not the model is able to cluster simple images.

11.4.2. Training

The minibatch size used is $N = 25$, and the training was done for 32 100 iterations, where each iteration basically contains one sampled minibatch. The input example count is $n = 20$. The training was ended by early stopping after 8 hours.

11.4.3. Results

Some of the training graphs are shown in Figure 27. These visualize the improvement over time in the loss terms, the cluster-count estimation accuracy, and the metrics on the validation set. By using early stopping, not the latest network configuration on the graphs is used for further tests, but the configuration at the iteration 17 100. As can be seen, there is much overfitting: The network stops generalizing, and overfits to the training data. The reason for this is that the same embedding network is used as for other image datasets and TIMIT. It seems to be too complex. Therefore, more tests were conducted with less complex embedding networks and much more dropout, but the overall quality did not increase. Note that early stopping here is an important factor because as can be seen, the quality of the final used configuration at iteration 17 100 is not that bad.

With a less complex embedding network with more dropout and by using early stopping, the best model achieved an NMI of 0.862, a BBN_{norm} of 0.846, and an MR of 0.149. Different models with different embedding networks were tested, but the overall quality of the initial model described in this test could not be outperformed. For some tests, it could be seen that there is less overfit but also less quality in the final test.

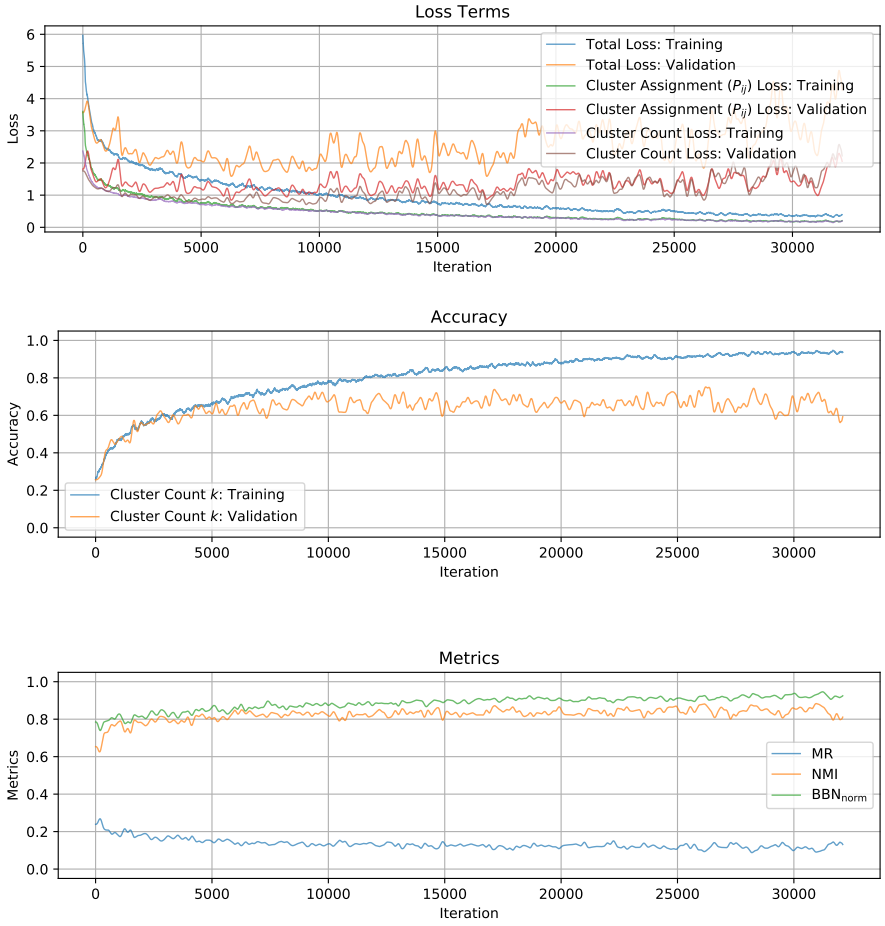


Figure 27.: COIL-100: Training graphs

The metrics finally achieved on the test data by using the best network configuration are shown in Table 12.

Table 12.: COIL-100: The metrics achieved by performing 300 clusterings on the test set.

Metric	**Mean**
NMI	0.8674 ± 0.0113
BBN_{norm}	0.9035 ± 0.0521
MR	0.1155 ± 0.0080

By the results achieved with forward dropout, it could be shown that the network is very confident for a small number of clusters. When there $k = 4$ or more clusters, then the confidence in the network decreases. This effect is shown in Figure 28. In general, the confidence is lower than in the TIMIT experiment, for instance, but for small cluster numbers still not too bad.

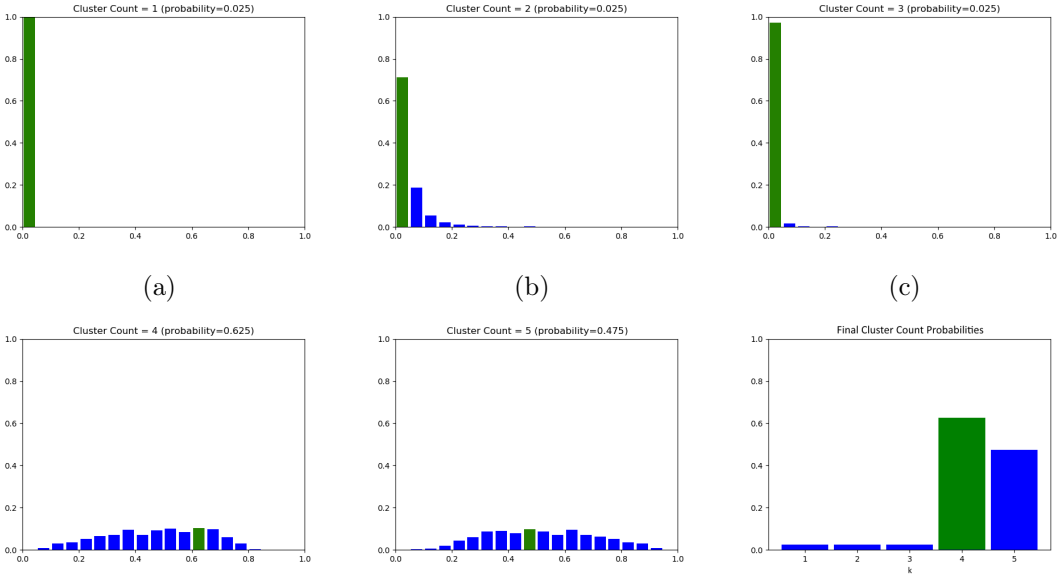


Figure 28.: It can be seen that the network is very confident and that there are more than $k = 3$ clusters present. On the other hand, it can be seen that the network has problems in deciding on $k = 4$ or $k = 5$ clusters. The mode for each cluster count prediction and the cluster count finally predicted are marked in green. The decision is finally $k = 5$, but as can be seen by the broad distribution, the confidence in $k = 4$ and $k = 5$ is very low.

11.4.4. Example

To complete the results of the experiment described, an example of the clustering given in Table 13. The predicted cluster count distribution $P(k)$ is shown in Figure 29. The metrics achieved for the example shown are NMI=0.933, BBN_{norm}=0.917, and MR=0.050.

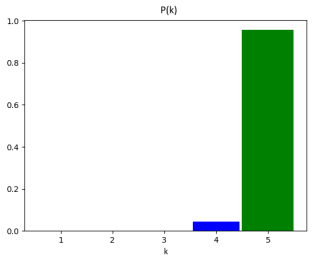
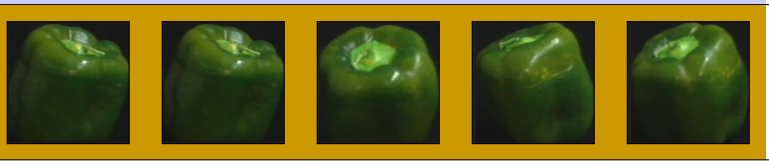


Figure 29.: The predicted cluster count distribution $P(k)$. The true cluster count is marked in green.

Table 13.: The network's proposed clusters for a given set of COIL-100 images of the test set. Each row is a cluster proposed by the network, and each class has different color (=ground truth).

Cluster	Objects
0	
1	
2	
3	
4	

11.4.5. Comparison with the State-of-the-Art

The results achieved are quite good, but the state-of-the-art is better. For instance, [49] reach an NMI of 0.985 (compared to this experiment: 0.867) by using the image intensities as an input. Many others approached using COIL-100 to perform clustering. Some of them are shown in Table 14.

Table 14.: The different clustering algorithms evaluated on COIL-100. k-means and Yang et al. [49] use image intensities as input.

Approach	NMI
Yang et al. [49]	0.985
CLR2_2 by Nie et al. [104]	0.941
LRAE1 by Chen et al. [105]	0.768
LRAE2 by Chen et al. [105]	0.783
LRAE3 by Chen et al. [105]	0.921
k-means	0.822
Proposed model	0.867

11.4.6. Conclusions

The clustering performed by the proposed network is not bad, but it can be improved. Nevertheless, it can be seen that the network architecture works adequately. Unfortunately, clustering these images do not work as well as clustering the TIMIT spectrograms. A reason for this could be the much smaller number of classes for the training.

11.5. Tiny ImageNet

11.5.1. Description

This experiment is conducted on the final model and uses the dataset described in Section 10.11. A modified version of the CNN-embedding network described in Section 7.2 is used: The width and height of the convolutional layers are reduced by a factor of 2 to handle the input images of the size 64×64 pixels. This test is used to see whether image datasets other than COIL-100 work better for image clustering.

11.5.2. Training

The minibatch size used is $N = 25$, and the training was done for 44 500 iterations where each iteration basically contains one sampled minibatch. The input example count is $n = 20$. The training was ended by early stopping after 10 hours.

11.5.3. Results

Some of the training graphs are shown in Figure 30. These visualize the improvement over time in the loss terms, the cluster-count estimation accuracy, and the metrics on the validation set. Due to using early stopping, not the latest network configuration on the graphs is used for further tests. As can be seen, there is almost no overfit, but unfortunately, the overall quality, even of the training data, is not that great.

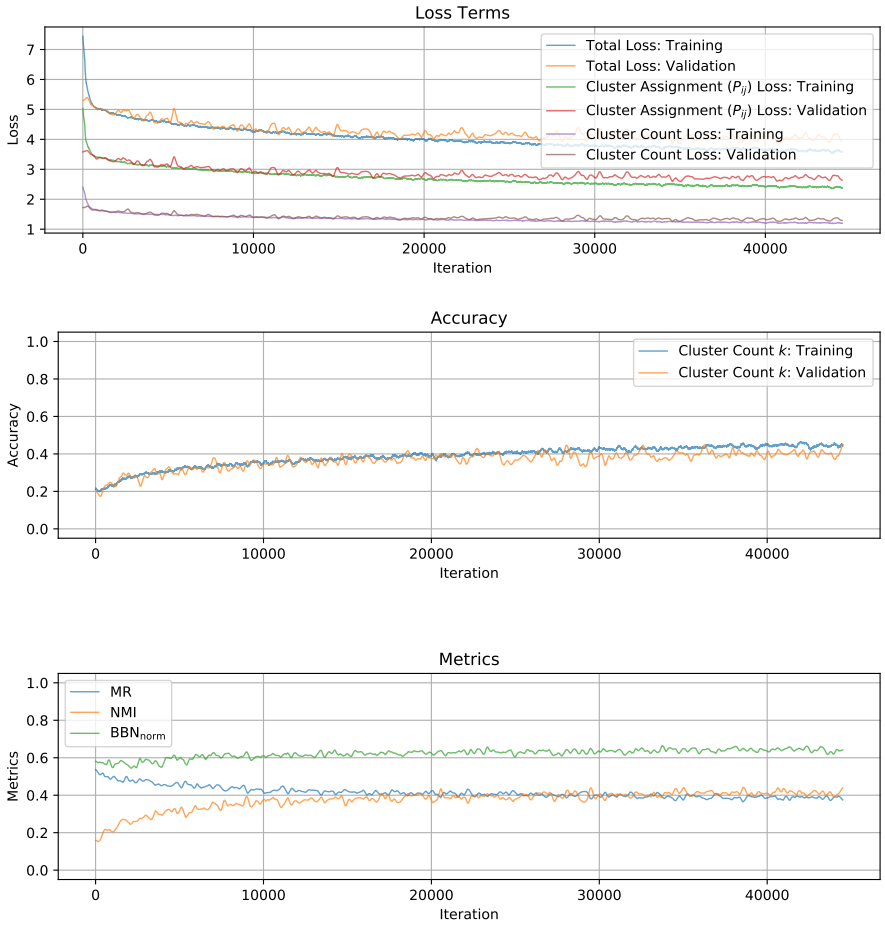


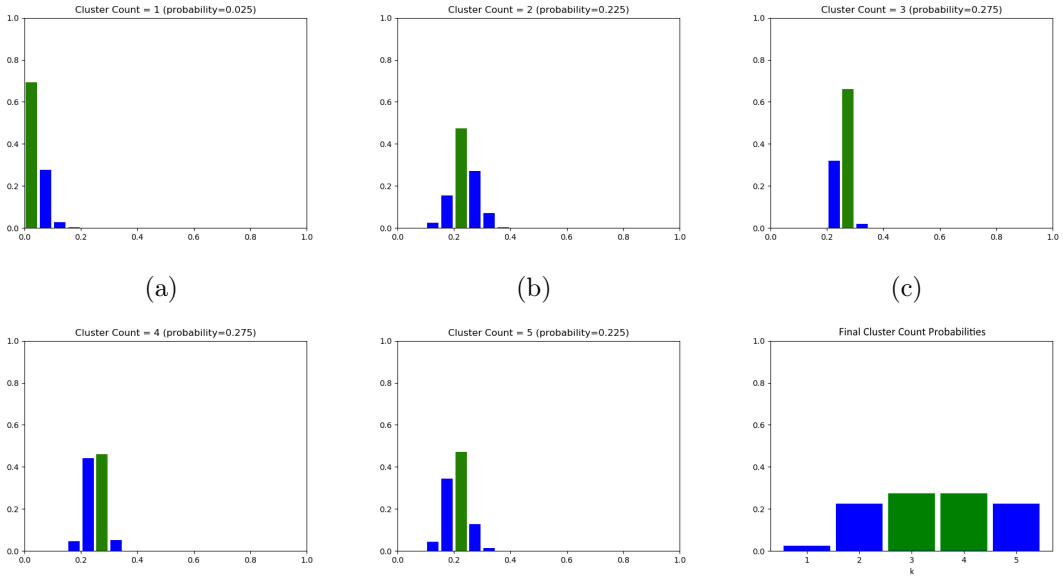
Figure 30.: Tiny ImageNet: Training graphs

The metrics finally reached for the test data by using the best network configuration, are shown in Table 15.

Table 15.: Tiny ImageNet: The metrics achieved by performing 300 clusterings in the test set.

Metric	**Mean**
NMI	0.4097 ± 0.0172
BBN_{norm}	0.6431 ± 0.0126
MR	0.4010 ± 0.0112

By the results achieved conducted with forward dropout are and it could be shown that the network is not very confident in choosing the number of clusters in the data. The confidence is only high if only one cluster is present, which is usually not the case. An example of the bad confidence is shown in Figure 31.



(a) (b) (c)

(d) (e) (f)

Figure 31.: It can be seen that the network is not very confident about the number of clusters in the data. The modus for each cluster count prediction and the final predicted cluster count are marked green. According to the graphs shown, the probability for $k = 3$ and $k = 4$ clusters is the same, but the network is more confident by choosing $k = 3$ which can be seen by the slightly more concentrated distribution in Figure 31c than in Figure 31d.

11.5.4. Example

To complete the results of the described experiment, there is an example clustering given in Table 16. The predicted cluster count distribution $P(k)$ is shown in Figure 32. The reached metrics for the shown example are NMI=0.467, BBN_{norm}=0.675 and MR=0.500.

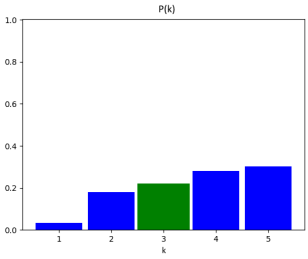
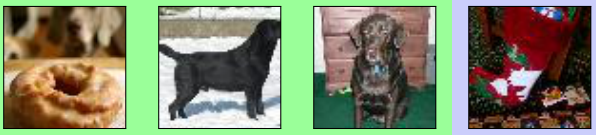


Figure 32.: The predicted cluster count distribution $P(k)$. The true cluster count is marked green. According to the neural networks, $k = 5$ is the most probable choice. On the other side, it can be seen that the network cannot really decide which is the correct cluster count, except for that it is almost sure that it is not $k = 1$.

Table 16.: The network's proposed clusters for a given set of Tiny ImageNet images of the test set. Each row is a cluster proposed by the network, and each class has a different color (=ground truth).

Cluster	Objects
0	
1	
2	
3	
4	

11.5.5. Conclusions

The clustering task on this dataset completely failed: It is even worse than on COIL-100. One reason for this might be that the image quality is too bad and there are too few classes in the dataset. Another test on all 1000 classes of ImageNet [113] with a resolution of 128×128 should produce a more comparable result. This test was not conducted because of the limited time and the huge size of the ImageNet dataset: Much time is required to download and pre-process the dataset. Unfortunately, there was not much time left at the end of the thesis. Another reason that the proposed approach failed might be that already the classification task on the Tiny ImageNet is quite hard[1] [116] [117] and the clustering task, in general, is even harder.

[1] https://www.kaggle.com/c/tiny-imagenet/leaderboard

11.6. FaceScrub

11.6.1. Description

This experiment is conducted on the final model and uses the dataset described in Section 10.8. The CNN-embedding network described in Section 7.2 is used. This test is conducted to see if clustering on a very specific image type (faces) works well.

11.6.2. Training

The minibatch size used is $N = 25$ and the training was done for 42 400 iterations, where each iteration basically contains one sampled minibatch. The input example count is $n = 20$. The training was ended by early stopping after 10 hours.

11.6.3. Results

Some training graphs are shown in Figure 33. They visualize the improvement over time for the loss terms, the cluster-count estimation accuracy and the metrics on the validation set. By using early stopping, not the latest network configuration on the graphs is used for further tests.

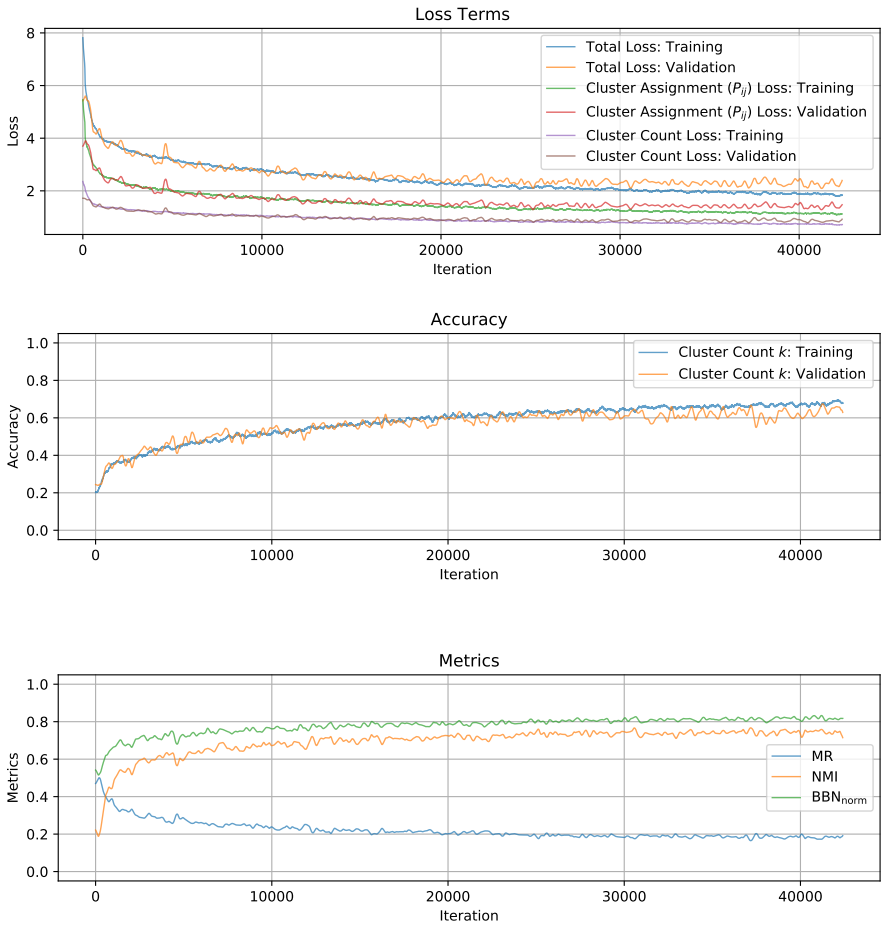


Figure 33.: FaceScrub: Training graphs

The finally reached metrics on the test data by using the best network configuration, are shown in Table 17.

Table 17.: FaceScrub: The reached metrics by conducting 300 clusterings of the test set.

Metric	**Mean**
NMI	0.7372 ± 0.0158
BBN_{norm}	0.8103 ± 0.0111
MR	0.1888 ± 0.0104

Tests with forward dropout are conducted and it could be shown that the network is often not confident for cluster counts larger than $k = 2$. This effect is shown in Figure 34.

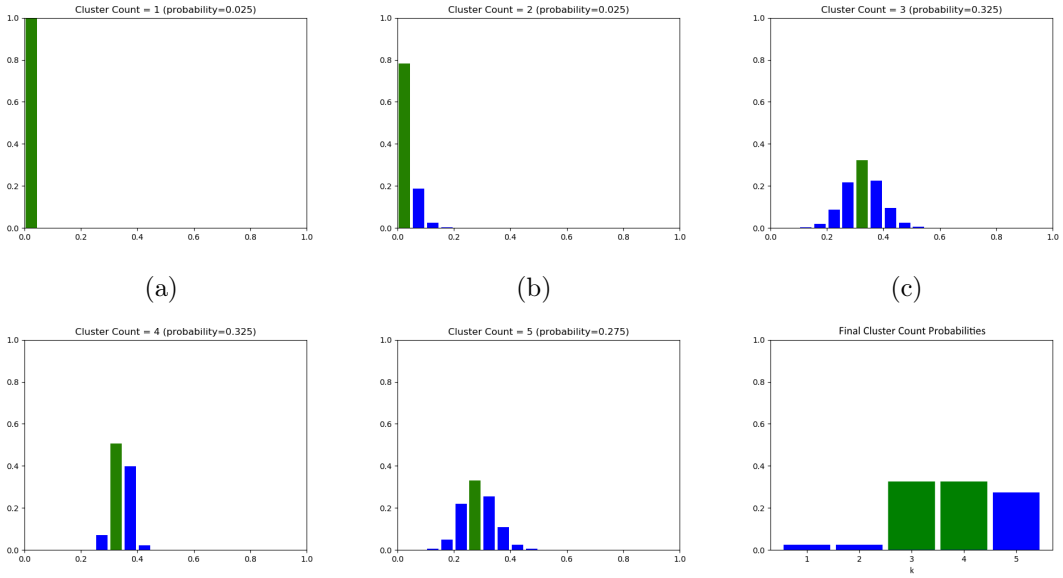


Figure 34.: It can be seen that the network is not sure about if there are $k = 3$, $k = 4$ or $k = 5$ clusters. On the other hand, it can be seen that the network is sure if there are not $k = 1$ or $k = 2$ clusters. The modus for each cluster count prediction and the final predicted cluster count are marked green. According to the graphs shown, the probability for $k = 3$ and $k = 4$ clusters is the same, but the network is more confident by choosing $k = 4$ which can be seen by the more concentrated distribution in Figure 34d than in Figure 34c.

11.6.4. Example

To complete the results of the described experiment, there is an example clustering given in Table 18. The predicted cluster count distribution $P(k)$ is shown in Figure 35. The reached metrics for the shown example are NMI=0.781, BBN_{norm}=0.862 and MR=0.150.

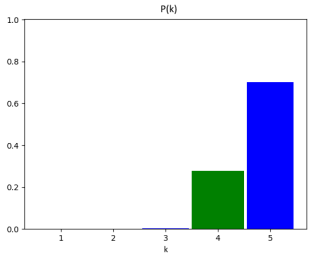


Figure 35.: The predicted cluster count distribution $P(k)$. The true cluster count is marked green.

Table 18.: The network's proposed clusters for a given set of FaceScrub faces of the test set. Each row is a cluster proposed by the network, and each person has another color for his/her face image (=ground truth).

Cluster	Objects
0	
1	
2	
3	
4	 

11.6.5. Conclusions

The quality is not very well as can be seen by the reached scores. The network works a bit, but the quality is not competitive. Finally, it can be said that the network is not able to produce a good clustering quality for the face images used.

11.7. Labeled Faces in the Wild

11.7.1. Description

This experiment is conducted on the final model and uses the dataset described in Section 10.9. The training is not done on this dataset, just an evaluation. Actually, the trained model from the experiment described in Section 11.6 is used. Also, the same input count $n = 20$ is used. This test is conducted to see if clustering trained on one dataset also reaches comparable results on other datasets. Only classes with at least 20 records are used. This reduces the size of the LFW-crop dataset from 5749 classes to 62 classes. Unfortunately, most classes only contain one image, but for good nontrivial clustering tests, many different records per class are required. Because the minimum cluster count is 1 and there are 20 inputs, the minimum required number of images per class is set to 20.

11.7.2. Results

The reached metrics on the given dataset are shown in Table 19. As expected the metrics are a bit worse for LFW-crop than for FaceScrub.

Table 19.: LFW-crop: The reached metrics by conducting 300 clusterings of the test set. It can be seen that the test on FaceScrub reaches higher values.

Metric	**Mean**
NMI	0.6204 ± 0.0191
BBN_{norm}	0.6835 ± 0.0141
MR	0.2737 ± 0.0128

By the results achieved with forward dropout it could be shown that the network is less confident than when the dataset was used on which the model was trained. In general, the network is not very confident.

11.7.3. Example

To complete the results of the experiment described, an example of the clustering is given in Table 20. The predicted cluster count distribution $P(k)$ is shown in Figure 36.

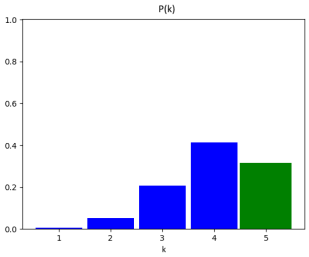
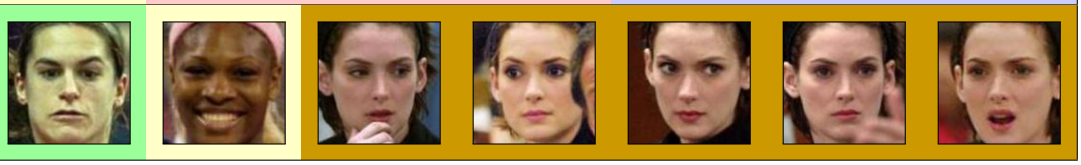


Figure 36.: The predicted cluster count distribution $P(k)$. The true cluster count is marked in green.

Table 20.: The network's proposed clusters for a given set of LFW-crop images. Each row is a cluster proposed by the network, and each class has a different color (=ground truth).

Cluster	Objects
0	
1	
2	
3	

11.7.4. Conclusions

The quality achieved is quite low and not competitive. At least, it works well sometimes. This experiment shows that a model trained on FaceScrub is also able to cluster images from LFW-crop. The metrics are a bit lower but in a similar range. Therefore, the network seems to generalize at least a bit.

11.8. Cluster-Assignment Regularization

11.8.1. Description

This experiment is conducted to test whether the cluster-assignment regularization which is described in Section 6.3 can lead to some improvements. To conduct this test, a sufficiently large dataset has to be used where the model without this regularization does not perform great, but it still works somewhat. For datasets where the network already works great, there is often already some hierarchical clustering present; therefore, it is hard to say if the regularization even has an effect. For this reason, the FaceScrub dataset is chosen for this experiment. The experiment is equal to the experiment described in Section 11.6, except for the additional regularization that is added with a factor of 1.0. If this additional regularization leads to improvements, then it should be possible to measure them with the metrics used.

11.8.2. Training

The minibatch size used is $N = 25$, and the training was done for 71 300 iterations where each iteration basically contains one sampled minibatch. The input example count is $n = 20$. The training was ended by early stopping after 23 hours.

11.8.3. Results

Some of the training graphs are shown in Figure 37. These visualize the improvement over time in the loss terms, the cluster-assignment regularization, the cluster-count estimation accuracy, and the metrics on the validation set. By using early stopping, not the latest network configuration is used for further tests on the graphs. As can be seen on the graphs, the cluster-assignment regularization term L_{CAR} minimizes very well.

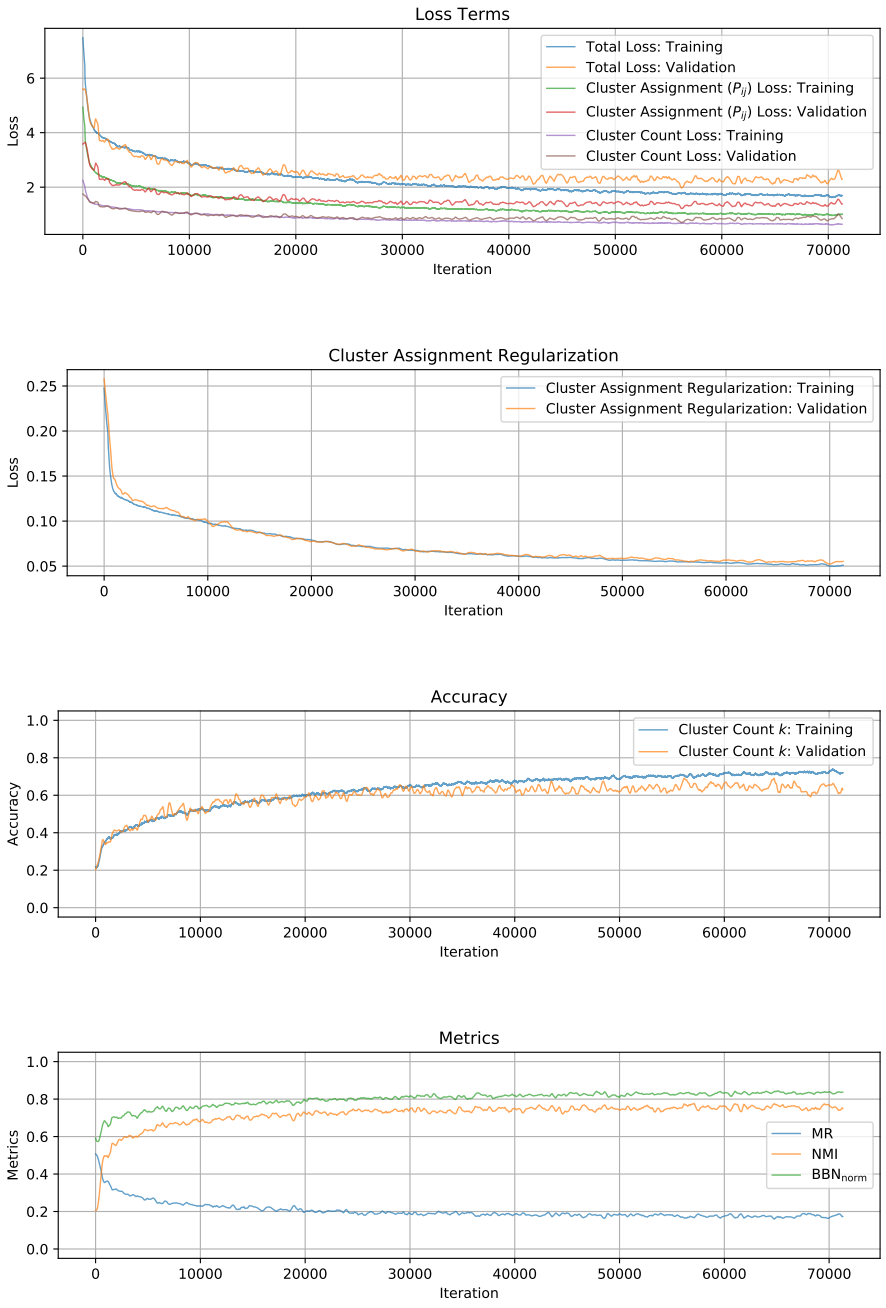


Figure 37.: FaceScrub with cluster-assignment regularization: Training graphs

The metrics finally achieved on the test data by using the best network configuration are shown in Table 21.

Table 21.: FaceScrub with cluster-assignment regularization: The metrics achieved by performing 300 clusterings on the test set.

Metric	**Mean**
NMI	0.7337 ± 0.0135
BBN_{norm}	0.8016 ± 0.0099
MR	0.1928 ± 0.0091

By the results conducted with forward dropout, it could be shown that the network is not confident for cluster counts larger than $k = 2$. The quality is similar to the FaceScrub experiment without the cluster-assignment regularization (see Section 11.6).

11.8.4. Example

The example for this test should not primarily show how well the clustering is done because it is very comparable to the experiment described in Section 11.6 as can be seen by the metrics achieved. This example should rather show how the cluster merging/split process by assuming different values for k works. Therefore, the network outputs are given for all cluster count possibilities. The cluster order is changed in the visualizations because it makes the interpretation of the cluster merging/split process much easier. This, in general, has no effect on the loss functions used or the network because according to the cluster index order, the loss is invariant. The predicted cluster count distribution $P(k)$ is shown in Figure 38.

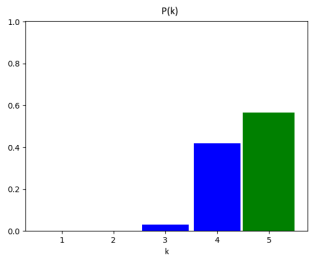
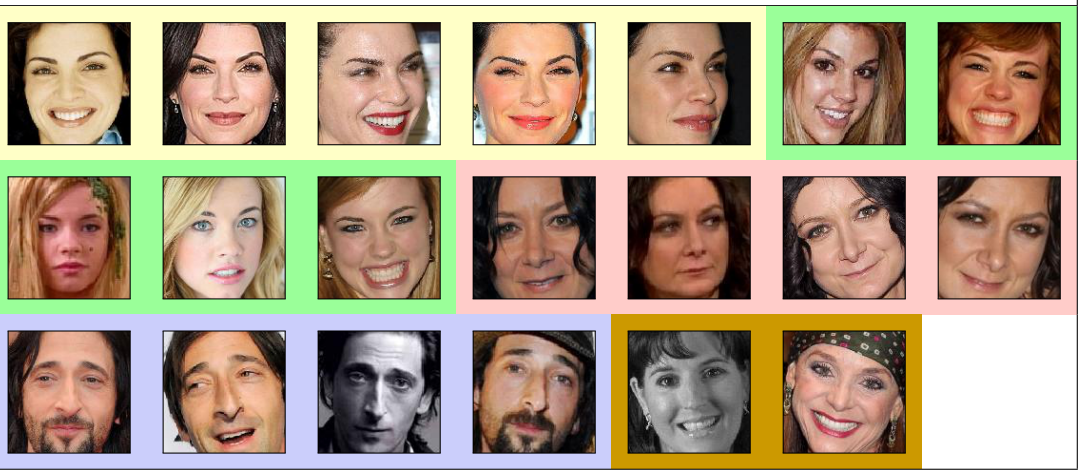


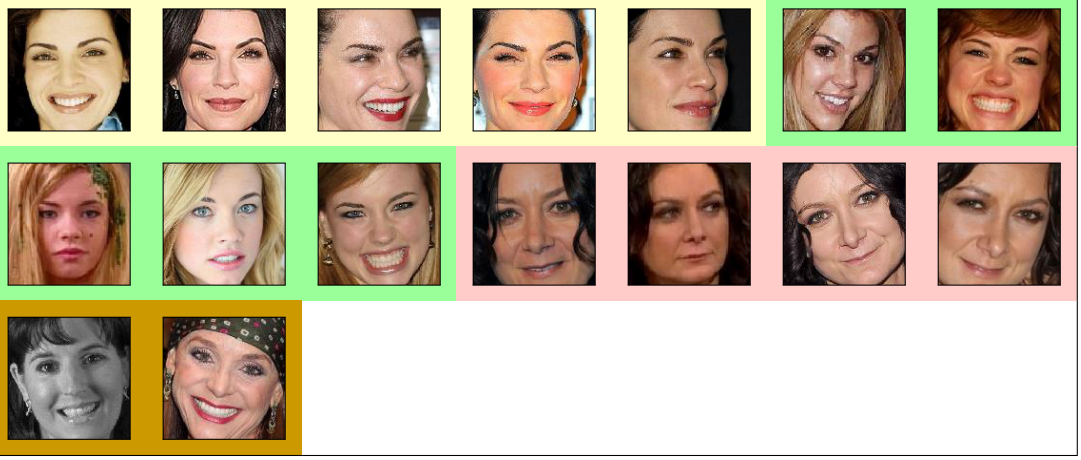
Figure 38.: The predicted cluster count distribution $P(k)$. The true cluster count is marked in green. It can be seen that the networks predict the correct cluster count $k = 5$.

Table 22.: The network's proposed cluster, given that there is $k = 1$ cluster. Each row is a cluster proposed by the network, and each person has another color for his/her face image (=ground truth).

Cluster	Objects
0	

Initially, there is only one cluster available by assuming $k = 1$, as can be seen in Table 22. For this clustering, there is no effect by the used regularization.

Table 23.: The network's proposed clusters, given that there are $k = 2$ clusters. Each row is a cluster proposed by the network, and each person has another color for his/her face image (=ground truth).

Cluster	Objects
0	
1	

If $k = 2$ is assumed, the network should output two clusters, and both clusters should be non-empty. These two clusters should be a split of a previous cluster, which is always true for $k = 2$. In addition, the new cluster should be non-empty, which is also the case. The newly performed clustering is visualized in Table 23. As can be seen there is no class for which elements are in different clusters. The cluster-assignment regularization is therefore fulfilled.

Table 24.: The network's proposed clusters, given that there are $k = 3$ clusters. Each row is a cluster proposed by the network, and each person has another color for his/her face image (=ground truth).

Cluster	Objects
0	
1	
2	 

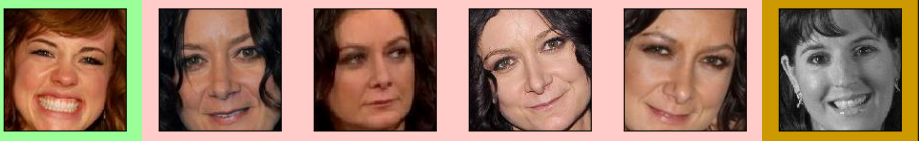
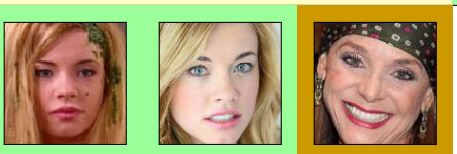
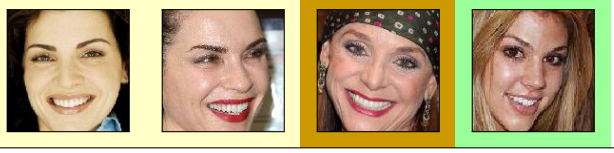
The next step is to assume $k = 3$. The network should now split exactly one of the clusters, which is given by assuming $k = 2$. Exactly this happens, as can be seen in Table 24, but the split is not optimal because it splits two elements of the same class into different clusters. Nevertheless, the cluster-assignment regularization is still fulfilled.

Table 25.: The network's proposed clusters, given that there are $k = 4$ clusters. Each row is a cluster proposed by the network, and each person has another color for his/her face image (=ground truth).

Cluster	Objects
0	
1	
2	
3	

In the next step, also, the network should do a split of a single cluster, but now, multiple splits occur. This suboptimal behavior is visualized in Table 25. The previous clusters 1 and 2 are merged and then split into 3 completely different clusters. The cluster-assignment regularization is therefore only partially fulfilled. The cluster count $k = 4$ has the highest quality (MR=0.150, NMI=0.845, $\text{BBN}_{\text{norm}} = 0.800$), but it is not the preferred choice of the network.

Table 26.: The network's proposed clusters, given that there are $k = 5$ clusters. Each row is a cluster proposed by the network, and each person has another color for his/her face image (=ground truth).

Cluster	Objects
0	
1	
2	
3	
4	

For the final assumption of $k = k_{\max} = 5$, there is also no clean split of a previous cluster available. Things get mixed up. This is shown in Table 26. According to the network, $k = 5$ is the most probable cluster count; which is actually the correct value. The scores are actually worse than for the assumption $k = 4$ (MR=0.350, NMI=0.639, $\text{BBN}_{\text{norm}} = 0.641$).

11.8.5. Conclusions

The soft-constraint for a hierarchical clustering does not change the quality of the model. Unfortunately, there is not always a clean hierarchy visible. Further experiments may be conducted with modified versions of the cluster-assignment regularization. In general, the hierarchical clustering is at least a bit improved by this regularization.

11.9. Summary Including Further Experiments

In order to obtain good results on images, more experiments on other image datasets are conducted. These experiments were not very successful. To summarize the experiments described previously and give an overview of the results of some more experiments, the metrics achieved are listed in Table 27.

Table 27.: Experiments on different datasets: 2DPD=2D-point data, TIN=Tiny ImageNet, Birds-200=Caltech UCSD Birds 200, FS=FaceScrub, FSCR=FaceScrub with the cluster-assignment regularization, FLW-102=Flowers-102, DC=Devanagari-Characters. All experiments that are not described in detail in a previous section use the CNN-embedding network from Section 7.2; with a changed input size when required.

Dataset	**Training Iterations**	**Training Time [h]**	**Metrics**		
			NMI	**BBN$_{\text{norm}}$**	**MR**
2DPD	128 700	123	0.992 ± 0.002	0.993 ± 0.001	0.004 ± 0.001
TIMIT	72 500	18	0.928 ± 0.007	0.935 ± 0.006	0.060 ± 0.006
COIL-100	32 100	8	0.867 ± 0.011	0.904 ± 0.052	0.115 ± 0.008
TIN	44 500	10	0.410 ± 0.017	0.643 ± 0.013	0.401 ± 0.011
Birds-200	41 200	10	0.639 ± 0.016	0.746 ± 0.012	0.255 ± 0.011
Cars-196	31 300	8	0.399 ± 0.018	0.609 ± 0.014	0.417 ± 0.012
SUN-397	69 500	17	0.634 ± 0.016	0.753 ± 0.011	0.257 ± 0.010
CIFAR-100	38 700	6	0.470 ± 0.017	0.684 ± 0.012	0.356 ± 0.011
FLW-102	21 900	9	0.568 ± 0.016	0.710 ± 0.012	0.317 ± 0.011
DC	17 500	6	0.621 ± 0.016	0.749 ± 0.011	0.262 ± 0.010
FS	42 400	10	0.737 ± 0.016	0.810 ± 0.011	0.189 ± 0.010
FSCR	71 300	22	0.734 ± 0.014	0.802 ± 0.010	0.192 ± 0.009
LFW-crop	-	-	0.620 ± 0.019	0.684 ± 0.014	0.274 ± 0.013

Many experiments on image datasets have been conducted, but the quality, in general, is not that good. The most widely used image dataset for evaluating clustering tasks is COIL-100 for which the proposed architecture achieves acceptable results, but unfortunately, much worse than the NMI of 0.985 reached by Yang et al. [49]. The quality is still acceptable. For TIMIT, the quality is even better. The scores achieved are still not state-of-the-art (Lukic et al. [12] reach a perfect MR of 0.0) but are quite good. For 2D-point data, the model proposed is better than the k-means or DBSCAN, for instance. It can be seen that datasets that have only a very few classes (e.g. the Devanagari characters) have poor performance. This is because there is not a sufficient number of classes to train an inter-class distance.

Additional to the experiments described, a modified approach based on the method proposed by Kampffmeyer et al. [10] was tested on TIMIT. It works well (MR$\leq$ 0.10) with a small number of speakers ($k \leq 4$), but the training time required to perform clustering is enormous: The network has to be re-trained to conduct a new clustering, which basically takes about an hour on a GPU. This means it usually requires an hour on a GPU to perform clustering on a set of examples; by knowing a given set of examples contains the same clusters as a previous set, there is no more training or only much less training required. An advantage is that no labels are needed for the training, but on the other hand, the exact number of clusters has to be known.

12. Results

The results may be reduced to a very few facts: It can be seen that simple clustering tasks like 2D-point data clustering or speaker clustering work well, and on the other hand, more complex tasks do not work that well. Even when making the networks deeper and by using more features, it did not work much better. This also may be due to a limit of the datasets used: A test done using the complete ImageNet with 1000 classes and more than 14 000 000 images may lead to better results.

It can also be pointed out that if a dataset based on classes is used, many classes are required. For instance, for TIMIT, the proposed model achieves accurate and good results. The same is true for the 2D-point data: There are no classes used, but a huge number of possible inputs can be generated.

An advantage of the proposed model is the performance: Once the training is done, the model is very fast and even for image inputs, the clustering can be performed in a few milliseconds. On the other hand, other neural network-based clustering methods like [10] [9] require much more time for clustering data. They basically have to train a neural network for each clustering. Classical approaches like k-means, DBSCAN, etc., are often faster than the proposed model, but are generally less powerful.

It can be concluded that the proposed model (see Chapter 3) and the loss function used (see Chapter 5) do work. Some improvements might be required, but the core functionality works. There are limitations for the current model, especially given the huge amount of data that is required for the training. Finally, it can be said that the final trained architecture is powerful, computationally efficient ($\mathcal{O}(n)$), and flexible.

13. Implementation

The entire code, including all experiments and models, is implemented in Python. The deep-learning libraries used are Keras [118] and TensorFlow [119]. The data augmentation is done by using [120]. The complete source code, including all experiments, is available online.[1]

The main focus of this thesis is not to describe the implementation details but rather the idea of the model. Nonetheless, this section describes the coarse architecture of the software implemented. This should allow third parties to not only reproduce the experiments but also to add new experiments easily and test new models and datasets. More detailed insights about how everything is implemented may be obtained by looking at the code and asking questions on Github.

The code uses a high abstraction of the neural networks used. This allows testing new loss functions easily based on the network output described (see Chapter 3) and any new network architectures. The following sections briefly explain the coarse structure of the code and the ideas.

13.1. Data Provider

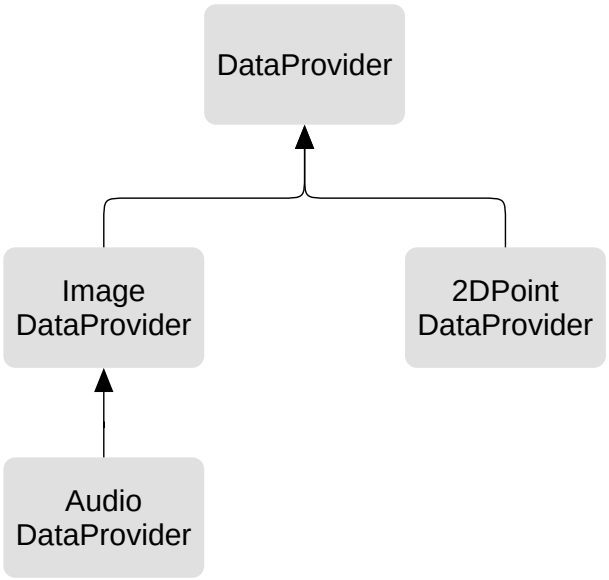


Figure 39.: The architecture used for the data providers.

The implemented architecture contains an abstraction for providing data. This is especially useful because many different datasets are used. The main requirements for data providers are:

- Providing the data in a suitable format: For instance, images as tensors in the format $H \times W \times C$ (H=height, W=width, C=channels).
- Providing metadata about the data, the cluster configuration, etc.
- Generating clusters: This might be very different, depending on the underlying dataset (see Section 8.2).
- Providing the possibility for data augmentation.

[1] https://github.com/kutoga/learning2cluster

– Being able to visualize the data: This is very helpful for getting a feeling of how well the clustering works and if this feature is available, the software is able to generate HTML files that allow navigating through the tests.

The coarse structure of the data providers already implemented is shown in Figure 39. It may be noted that the AudioDataProvider class is a subclass of the ImageDataProvider. The reason for this is that in the described tests, audio is handled as a 2D image (as in [12]). The implemented class also allows handling audio as a 1D-image (like [121]); in this case, for visualization purposes, a 2D representation is still used.

Many of the data providers implemented use the ImageDataProvider class as a base. Some of the data providers implement automatic dataset downloads, and others expect the data to be already available on a disk.

13.2. Neural Networks

The low-level neural network code is based on Keras and TensorFlow. This section only describes higher-level abstractions and does not go into the details about how to use Keras and TensorFlow because it is assumed that the reader is familiar with these libraries if he/she wants to study the code. By knowing the libraries and the model (see Chapters 3 and 5), it is no problem to understand how this lower-level implementation is done.

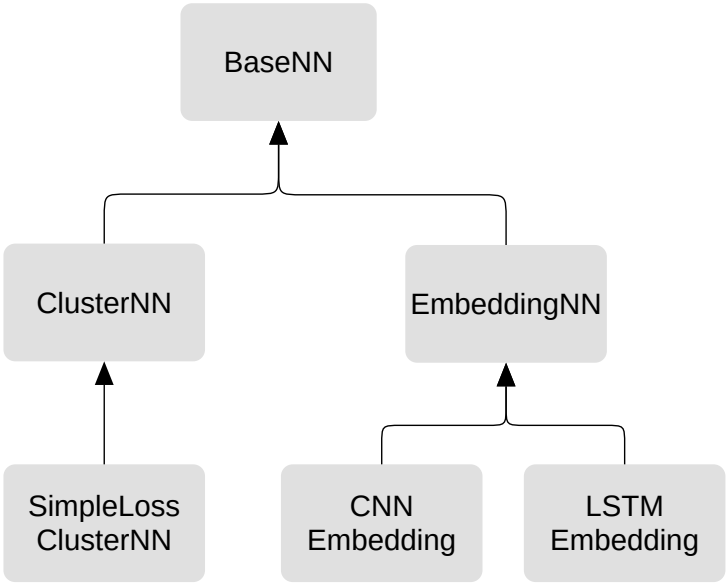


Figure 40.: The class hierarchy for the core of the neural networks.

The class hierarchy used to implement the neural network models is shown in Figure 40. There is a BaseNN class implemented, which holds a history, has the possibility to create graphs, it can save and load the current state (model, optimizer state, history, etc.), and it can be used to register some events. This class has the advantage that it allows handling a hierarchy of neural networks: This is especially required for the embedding networks used at runtime. For the tests, it is important that one can easily change the embedding networks to test an architecture with different data types; therefore, this hierarchy is implemented.

The ClusterNN class provides an interface for handling the clustering task: It has functions for the training and evaluates the underlying network(s). It allows evaluating results automatically with some pre-defined or custom metrics. The SimpleLossClusterNN implements the loss described in Chapter 5: It uses a flexible implementation and allows adding regularizations. The other very important class shown in Figure 40 is EmbeddingNN: This is usually the simplest

neural network (e.g. a CNN). The embedding networks available (see Chapter 7: CNN-based, LSTM-based, CNN-LSTM based, etc.) are already very flexible and are usually sufficient. These embedding networks allow using information from the data provider about the data dimensions, and therefore, one does not have to change any line of code when the data provider is changed from a dataset with $128 \times 128 \times 3$ images to, e.g., one with $32 \times 32 \times 1$ images. The provided embedding networks also have the possibility to configure, e.g., the number of layers, the units per layers, the activations and many other parameters.

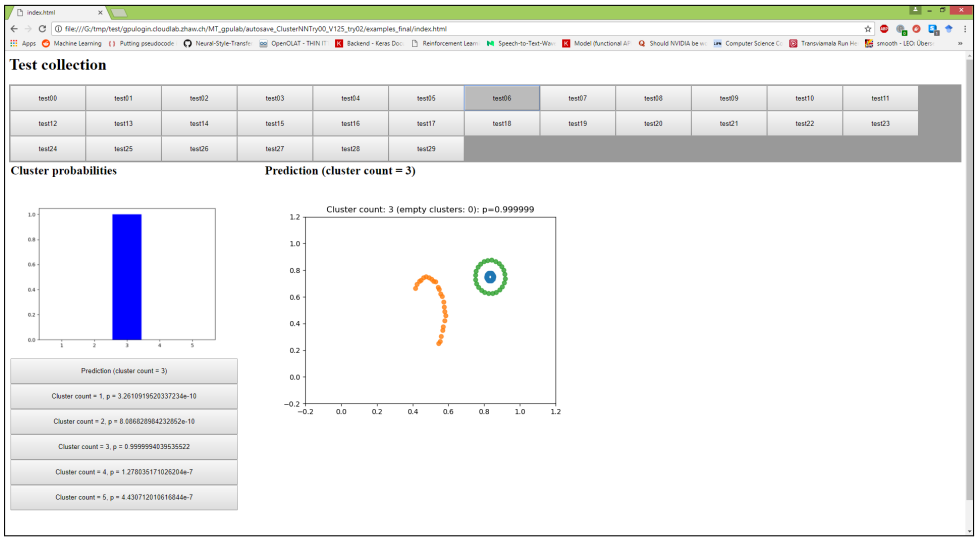
13.3. Workflow

The workflow is simple, which allows conducting many experiments. In general, one has to provide a data provider, an optional embedding network, and a clustering network. These components are independent until the runtime: The data provider and the embedding network are passed to the clustering network which handles everything else. It is possible to configure many training parameters (e.g. early stopping) and to automatically generate reports.

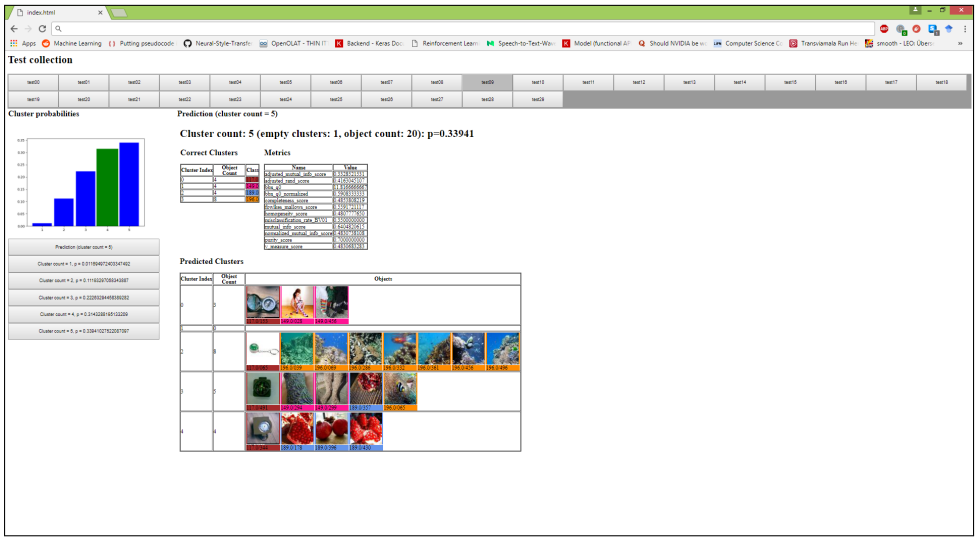
Examples of experiments are available in the source folders `./app*/`. Newer experiments usually use more of the features available and are a better point for starting to learn about how another new experiment can be created. Once an experiment is configured and started, (if it is not disabled) it creates different graphs and test clustering from time to time. The default events for the tests which include reports are:

- A new best validation loss is reached: A detailed test on the validation data is conducted.
- Every n_ith iteration, a detailed test on the training data is conducted (e.g. $n_i = 2000$).
- As soon as the training is finished (by early stopping), a detailed test on the test data is conducted; this includes tests with forward dropout and more accurate estimations of the mean metric values.

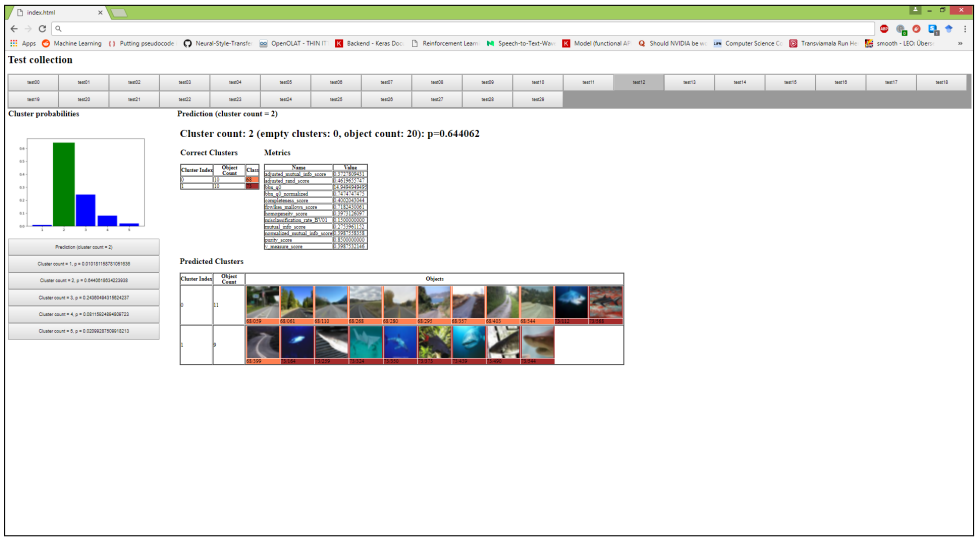
A detailed test contains the complete network input, output (all probabilities), all metrics for all possible cluster counts, and a visualization. Possible visualizations are shown in Figure 41.



(a)



(b)



(c)

Figure 41.: Possible visualizations of tests conducted on different datasets: These visualizations are created during training and finally, after the model is trained. The exact visualization depends on the data type used, the data provider, and its configuration.

14. Conclusions and Future Work

It was possible to show that supervised end-to-end-trained deep-learning models are able to cluster different data types with a promising accuracy. No assumptions about the data and the clustering algorithm have to be made because an embedding and the algorithm itself are trained by the network. State-of-the-art solutions may reach higher accuracies for specific tasks [12] [122], but the proposed architecture is much more general and requires much less data-specific tuning. Almost any data, given an embedding network, can be clustered by the proposed approach, whereas other models and algorithms require much hand-crafted tuning to get these models working. Some algorithms are even data-specific and are not designed to work with other data types [14].

Nevertheless, the proposed model may be improved in terms of accuracy and especially as to the amount of data required for the training. It works well for the 2D-point data, but there, the data can just be generated. It also works well for TIMIT, but in this dataset, there are 630 classes available. The performance on FaceScrub is ok, but here, there are also more than 500 classes available. The only dataset for which the performance is ok and there are only 100 classes available is COIL-100, but this dataset is relatively simple. There are always about $60 - 80\%$ of the classes used for the training. In general, fewer classes also work, but then the quality of the network decreases, and it starts to overfit to the given classes. Therefore, one may make the use of data more efficient.

The model proposed by Kaiser et al. [123] could be used as an inspiration for a more general embedding network. There the architecture would be more general: One would no longer have to tune the embedding network for a specific data type.

The current model uses a fixed number of RBDLSTM layers, independent of the data type used. It may be assumed that different data types require a different number of RBDLSTM layers because the comparison of 2D-points is easier than the comparison of high-dimensional image embeddings. To generalize this, one could use the layer described in [124]. This type of layer perfectly fits the requirements for stacked RBDLSTM layers, but it allows a neural network to train the number of stacked layers via back-propagation. Therefore, the network would have the choice to use fewer RBDLSTM layers for simpler tasks and more RBDLSTM layers for more difficult tasks.

It might be assumed that given the correct embedding network, the clustering network, including the weights, may be very similar or even the same for similar data types. Therefore, one could freeze the entire clustering, cluster assignment, and the cluster-count estimating network and just re-train the embedding network. This could improve the network in terms of data requirements: The clustering network could be trained with a dataset that has many classes. To avoid overfitting of the network, it might be required to train the clustering, cluster assignment, and the cluster-count estimating network at the same time with different datasets in parallel. In this case, the embedding network is not shared. This could highly generalize the internals of the network: It learns to learn the clustering task [125]. Transfer learning, in general, is often used to improve the model quality for small datasets [126] [127] [128]. This could also be helpful for the proposed model if the dataset used is small.

The cluster-assignment regularization described (see Chapter 6) shows promising results in some conducted experiments. The scores are almost equal to the currently used model's scores. Additionally, it can be seen that it includes a more hierarchical-like clustering. The information that the clustering must be in a hierarchical fashion simplifies the solution space because many fewer network outputs are valid; therefore, this could lead to some improvements. Tests showed

that the two loss components provided by this regularization have to be added to a high factor, e.g. 5. More tests have to be conducted on this loss.

No tests on text clustering [17] were conducted, but also this could be included in the current architecture. A special embedding network has to be used, but the network architecture, in general, still would be the same.

One might ask whether or not a higher runtime complexity may lead to better results? The current model has a runtime complexity of $\mathcal{O}(n)$ which is minimal because to cluster data, it can be assumed that every record has to be accessed at least once. Other algorithms like k-means have the same complexity, but there are algorithms like DBSCAN with a higher runtime complexity. The currently implemented model is, therefore, not able to implement this exact algorithm. Allowing the model to have a higher runtime complexity, e.g. $\mathcal{O}(n^2)$, might lead to more powerful neural networks. Some models with this runtime were tested (e.g., see Section 4.2), but no better results were achieved, and the training time increased incredibly. It may still be the case that a model with a higher runtime complexity is able to learn more complex clustering methods. E.g., if a neural network is used for sorting, then implementations based only on stacked LSTM layers[1] are not able to learn a true comparison sorting for an arbitrary large input size n, because comparison sorting has a lower bound computational complexity of $\Omega(n \log(n))$ and stacked LSTM layers only provide a complexity of $\mathcal{O}(n)$. Nevertheless, this does not seem to be one of the main limitations of the currently used model.

Tests on ImageNet could be conducted: There are 1000 classes and in total, more than 14 000 000 images. This dataset is so large that overfitting is almost not possible. This could help to find the true limitations of the architecture described here.

A clustering model usually takes an unordered set of examples as input. In the proposed model the input has an order which may have an influence to the internal state of the network. During training and evaluation it is assumed that the order is random and the network is trained in a way that the input order should not matter. The network still could create different outputs for different input orders. To improve this interface and use real set inputs where the order does not matter, one could use the ideas proposed in [129] [130].

Finally, the model could be extended to an unsupervised model: Once could, e.g., use the proposed method by Kampffmeyer et al. [10] to retrieve the clusters in a dataset and use these clusters to train the architecture described in this thesis. To estimate the correct cluster count, it may be possible to use techniques like forward dropout [97] in combination with the proposed unsupervised trainable model by Kampffmeyer et al. [10]: The right cluster count might deliver more confident results. Or maybe, the (unsupervised) loss might be better minimized by using the correct cluster count.

[1] https://github.com/apache/incubator-mxnet/tree/master/example/bi-lstm-sort

15. Discussions

The proposed model does not reach a quality higher than the state-of-the-art, but it shows the effectiveness of end-to-end solutions, even more specifically for the clustering task. Despite the sub-optimal results for image clustering, it might be seen that, in general, the proposed approach reaches very promising results.

The work demonstrates that neural networks and differentiable programs, in general, are able to learn complex tasks or "algorithms". It could be seen that it is a harder to implement a task like clustering as a end-to-end differentiable and trainable program compared to classical programming, but finally a more flexible solution can be created: Especially, the neural network based solution does not only work for simple data types, but also for high-dimensional data like audio or images.

The contribution of this work is mainly to show that neural networks are able to perform end-to-end clustering and to show how powerful differentiable programs are. Other works may use this information to build more powerful end-to-end clustering solutions (e.g. based on the proposed model) or to implement even more complex "algorithms" by the use of a neural network.

A. Bibliography

[1] S. Nayar, S. Nene, and H. Murase, "Columbia object image library (coil 100)," *Department of Comp. Science, Columbia University, Tech. Rep. CUCS-006-96*, 1996.

[2] M. J. Zaki, W. Meira Jr, and W. Meira, *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.

[3] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA., 1967, pp. 281–297.

[4] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.

[5] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[6] M. Robin, G. Nicolle, and A. Rota, "Automatic sounds clustering approach based on a likelihood measure computation."

[7] C. Vallespi, F. De la Torre, M. Veloso, and T. Kanade, "Automatic clustering of faces in meetings," in *Image Processing, 2006 IEEE International Conference on*. IEEE, 2006, pp. 1841–1844.

[8] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 815–823.

[9] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[10] M. Kampffmeyer, S. Løkse, F. M. Bianchi, L. Livi, A.-B. Salberg, and J. Robert, "Deep divergence-based clustering," in *Machine Learning for Signal Processing (MLSP), 2017 IEEE 27th International Workshop on*. IEEE, 2017, pp. 1–6.

[11] T. Stadelmann and B. Freisleben, "Unfolding speaker clustering potential: a biomimetic approach," in *Proceedings of the 17th ACM international conference on Multimedia*. ACM, 2009, pp. 185–194.

[12] Y. Lukic, C. Vogt, O. Dürr, and T. Stadelmann, "Learning embeddings for speaker clustering based on voice equality," in *Machine Learning for Signal Processing (MLSP), 2017 IEEE 27th International Workshop on*. IEEE, 2017, pp. 1–6.

[13] M. A. Siegler, U. Jain, B. Raj, and R. M. Stern, "Automatic segmentation, classification and clustering of broadcast news audio," in *Proc. DARPA speech recognition workshop*, vol. 1997, 1997.

[14] A. Borji and A. Dundar, "Human-like clustering with deep convolutional neural networks," 2017.

[15] C. C. Aggarwal and C. Zhai, "A survey of text clustering algorithms," in *Mining text data*. Springer, 2012, pp. 77–128.

[16] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[17] J. Xu, B. Xu, P. Wang, S. Zheng, G. Tian, and J. Zhao, "Self-taught convolutional neural networks for short text clustering," *Neural Networks*, vol. 88, pp. 22–31, 2017.

[18] Y. Zhao and G. Karypis, "Evaluation of hierarchical clustering algorithms for document datasets," in *Proceedings of the eleventh international conference on Information and knowledge management*. ACM, 2002, pp. 515–524.

[19] A. Samal, D. Parida, M. R. Satapathy, and M. N. Mohanty, "On the use of mfcc feature vector clustering for efficient text dependent speaker recognition," in *Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2013*. Springer, 2014, pp. 305–312.

[20] K. K. Vasan and B. Surendiran, "Dimensionality reduction using principal component analysis for network intrusion detection," *Perspectives in Science*, vol. 8, pp. 510–512, 2016.

[21] C. Ding and X. He, "K-means clustering via principal component analysis," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 29.

[22] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, vol. 2. IEEE, 2006, pp. 2169–2178.

[23] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[24] B. Hariharan, J. Malik, and D. Ramanan, "Discriminative decorrelation for clustering and classification," *Computer Vision–ECCV 2012*, pp. 459–472, 2012.

[25] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International Conference on Machine Learning*, 2016, pp. 478–487.

[26] I. T. Jolliffe and J. Cadima, "Principal component analysis: a review and recent developments," *Phil. Trans. R. Soc. A*, vol. 374, no. 2065, p. 20150202, 2016.

[27] G. Antipov, S.-A. Berrani, N. Ruchaud, and J.-L. Dugelay, "Learned vs. hand-crafted features for pedestrian gender recognition," in *Proceedings of the 23rd ACM international conference on Multimedia*. ACM, 2015, pp. 1263–1266.

[28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[30] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[31] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," in *European Conference on Computer Vision*. Springer, 2014, pp. 346–361.

[32] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.

[33] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[34] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[35] Y. Lukic, C. Vogt, O. Dürr, and T. Stadelmann, "Speaker identification and clustering using convolutional neural networks," in *Machine Learning for Signal Processing (MLSP), 2016 IEEE 26th International Workshop on*. IEEE, 2016, pp. 1–6.

[36] X. Guo, X. Liu, E. Zhu, and J. Yin, "Deep clustering with convolutional autoencoders," in *International Conference on Neural Information Processing*. Springer, 2017, pp. 373–382.

[37] K. G. Dizaji, A. Herandi, and H. Huang, "Deep clustering via joint convolutional autoencoder embedding and relative entropy minimization," *arXiv preprint arXiv:1704.06327*, 2017.

[38] S. Ide and S. Uchida, "How does a cnn manage different printing types?" in *Document Analysis and Recognition (ICDAR), 2017 IAPR 14th International Conference on*. CPS, 2017, pp. 1–6.

[39] M. Ghodsi, B. Liu, and M. Pop, "Dnaclust: accurate and efficient clustering of phylogenetic marker genes," *BMC bioinformatics*, vol. 12, no. 1, p. 271, 2011.

[40] J. Paparrizos and L. Gravano, "k-shape: Efficient and accurate clustering of time series," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1855–1870.

[41] D. Müllner, "Modern hierarchical, agglomerative clustering algorithms," *arXiv preprint arXiv:1109.2378*, 2011.

[42] K. C. Gowda and G. Krishna, "Agglomerative clustering using the concept of mutual nearest neighbourhood," *Pattern recognition*, vol. 10, no. 2, pp. 105–112, 1978.

[43] T. Kurita, "An efficient agglomerative clustering algorithm using a heap," *Pattern Recognition*, vol. 24, no. 3, pp. 205–209, 1991.

[44] A. Roy and S. Pokutta, "Hierarchical clustering via spreading metrics," in *Advances in Neural Information Processing Systems*, 2016, pp. 2316–2324.

[45] N. Ailon and M. Charikar, "Fitting tree metrics: Hierarchical clustering and phylogeny," in *Foundations of Computer Science, 2005. FOCS 2005. 46th Annual IEEE Symposium on*. IEEE, 2005, pp. 73–82.

[46] L. Lin and J. Li, "Clustering with hidden markov model on variable blocks," *Journal of Machine Learning Research*, vol. 18, no. 110, pp. 1–49, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-342.html

[47] Y. G. Jung, M. S. Kang, and J. Heo, "Clustering performance comparison using k-means and expectation maximization algorithms," *Biotechnology & Biotechnological Equipment*, vol. 28, no. sup1, pp. S44–S48, 2014.

[48] O. A. Abbas, "Comparisons between data clustering algorithms." *International Arab Journal of Information Technology (IAJIT)*, vol. 5, no. 3, 2008.

[49] J. Yang, D. Parikh, and D. Batra, "Joint unsupervised learning of deep representations and image clusters," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 5147–5156.

[50] D. Liu and F. Kubala, "Online speaker clustering," in *Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03). 2003 IEEE International Conference on*, vol. 1. IEEE, 2003, pp. I–I.

[51] T. Stadelmann, "Voice modeling methods for automatic speaker recognition," 2010.

[52] L. Lerato and T. Niesler, "Investigating parameters for unsupervised clustering of speech segments using timit," in *Twenty-Third Annual Symposium of the Pattern Recognition Association of South Africa*, 2012, p. 83.

[53] M. H. Farouk, "On the application of quantum clustering on speech data," *International Journal of Speech Technology*, vol. 20, no. 4, pp. 891–896, 2017.

[54] W. M. Rand, "Objective criteria for the evaluation of clustering methods," *Journal of the American Statistical association*, vol. 66, no. 336, pp. 846–850, 1971.

[55] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure." in *EMNLP-CoNLL*, vol. 7, 2007, pp. 410–420.

[56] E. B. Fowlkes and C. L. Mallows, "A method for comparing two hierarchical clusterings," *Journal of the American statistical association*, vol. 78, no. 383, pp. 553–569, 1983.

[57] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo, "A comparison of extrinsic clustering evaluation metrics based on formal constraints," *Information retrieval*, vol. 12, no. 4, pp. 461–486, 2009.

[58] A. J. Gates and Y.-Y. Ahn, "The impact of random models on clustering similarity," *Journal of Machine Learning Research*, vol. 18, no. 87, pp. 1–28, 2017. [Online]. Available: http://jmlr.org/papers/v18/17-039.html

[59] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[60] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[61] R. J. Williams and D. Zipser, "Gradient-based learning algorithms for recurrent networks and their computational complexity," *Backpropagation: Theory, architectures, and applications*, vol. 1, pp. 433–486, 1995.

[62] S. Kisilevich, F. Mansmann, and D. Keim, "P-dbscan: a density based clustering algorithm for exploration and analysis of attractive areas using collections of geo-tagged photos," in *Proceedings of the 1st international conference and exhibition on computing for geospatial research & application*. ACM, 2010, p. 38.

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

[64] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, D. S. Pallett, and N. L. Dahlgren, "Darpa timit acoustic phonetic continuous speech corpus cdrom," 1993.

[65] Y.-C. Hsu and Z. Kira, "Neural network-based clustering using pairwise constraints," *arXiv preprint arXiv:1511.06321*, 2015.

[66] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[67] X. Peng, S. Xiao, J. Feng, W.-Y. Yau, and Z. Yi, "Deep subspace clustering with sparsity prior." in *IJCAI*, 2016, pp. 1925–1931.

[68] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering." in *AAAI*, 2014, pp. 1293–1299.

[69] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *International Conference on Medical Image Computing and Computer-Assisted Intervention*. Springer, 2015, pp. 234–241.

[70] Z. Wang, S. Chang, J. Zhou, M. Wang, and T. S. Huang, "Learning a task-specific deep architecture for clustering," in *Proceedings of the 2016 SIAM International Conference on Data Mining*. SIAM, 2016, pp. 369–377.

[71] A. Dosovitskiy, P. Fischer, J. T. Springenberg, M. Riedmiller, and T. Brox, "Discriminative unsupervised feature learning with exemplar convolutional neural networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 9, pp. 1734–1747, 2016.

[72] X. Wang and A. Gupta, "Unsupervised learning of visual representations using videos," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 2794–2802.

[73] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1422–1430.

[74] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *International Workshop on Similarity-Based Pattern Recognition*. Springer, 2015, pp. 84–92.

[75] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin, "Deep metric learning with angular loss," *arXiv preprint arXiv:1708.01682*, 2017.

[76] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, "Signature verification using a" siamese" time delay neural network," in *Advances in Neural Information Processing Systems*, 1994, pp. 737–744.

[77] M. Guillaumin, J. Verbeek, and C. Schmid, "Is that you? metric learning approaches for face identification," in *Computer Vision, 2009 IEEE 12th international conference on*. IEEE, 2009, pp. 498–505.

[78] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[79] A. Graves, S. Fernández, and J. Schmidhuber, "Bidirectional lstm networks for improved phoneme classification and recognition," *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005*, pp. 753–753, 2005.

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

[81] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[82] A. Griewank, "Who invented the reverse mode of differentiation?" *Documenta Mathematica, Extra Volume ISMP*, pp. 389–400, 2012.

[83] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659*, 2017.

[84] Y. Wen, K. Zhang, Z. Li, and Y. Qiao, "A discriminative feature learning approach for deep face recognition," in *European Conference on Computer Vision*. Springer, 2016, pp. 499–515.

[85] L. Hubert and P. Arabie, "Comparing partitions," *Journal of classification*, vol. 2, no. 1, pp. 193–218, 1985.

[86] M. Hughes, I. Li, S. Kotoulas, and T. Suzumura, "Medical text classification using convolutional neural networks," *Stud Health Technol Inform*, vol. 235, pp. 246–50, 2017.

[87] K. Kamnitsas, C. Ledig, V. F. Newcombe, J. P. Simpson, A. D. Kane, D. K. Menon, D. Rueckert, and B. Glocker, "Efficient multi-scale 3d cnn with fully connected crf for accurate brain lesion segmentation," *Medical image analysis*, vol. 36, pp. 61–78, 2017.

[88] S. Ji, W. Xu, M. Yang, and K. Yu, "3d convolutional neural networks for human action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 1, pp. 221–231, 2013.

[89] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.

[90] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural Networks*, vol. 18, no. 5-6, pp. 602–610, 2005.

[91] F. A. Gers, D. Eck, and J. Schmidhuber, "Applying lstm to time series predictable through time-window approaches," in *Neural Nets WIRN Vietri-01*. Springer, 2002, pp. 193–200.

[92] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal, "Long short term memory networks for anomaly detection in time series," in *Proceedings*. Presses universitaires de Louvain, 2015, p. 89.

[93] J. Egli and T. Gygax, "Bachelorarbeit informatik frühling 2017: Speaker clustering mit metric embeddings," https://ba-pub.engineering.zhaw.ch/BA_WebPublication/BA_OverviewPage.aspx, 2017.

[94] T. N. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, long short-term memory, fully connected deep neural networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*. IEEE, 2015, pp. 4580–4584.

[95] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[96] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, 2007.

[97] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, 2016, pp. 1050–1059.

[98] M. Kotti, V. Moschou, and C. Kotropoulos, "Speaker segmentation and clustering," *Signal processing*, vol. 88, no. 5, pp. 1091–1124, 2008.

[99] A. F. McDaid, D. Greene, and N. Hurley, "Normalized mutual information to evaluate overlapping community finding algorithms," *arXiv preprint arXiv:1110.2515*, 2011.

[100] Y. Yang, D. Xu, F. Nie, S. Yan, and Y. Zhuang, "Image clustering using local discriminant models and global integration," *IEEE Transactions on Image Processing*, vol. 19, no. 10, pp. 2761–2773, 2010.

[101] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[102] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona, "Caltech-UCSD Birds 200," California Institute of Technology, Tech. Rep. CNS-TR-2010-001, 2010.

[103] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 / cifar-100 (canadian institute for advanced research)." [Online]. Available: http://www.cs.toronto.edu/~kriz/cifar.html

[104] F. Nie, X. Wang, M. I. Jordan, and H. Huang, "The constrained laplacian rank algorithm for graph-based clustering." in *AAAI*, 2016, pp. 1969–1976.

[105] Y. Chen, L. Zhang, and Z. Yi, "Subspace clustering using a low-rank constrained autoencoder," *Information Sciences*, vol. 424, pp. 27–38, 2018.

[106] A. K. Pant, S. P. Panday, and S. R. Joshi, "Off-line nepali handwritten character recognition using multilayer perceptron and radial basis function neural networks," in *2012 Third Asian Himalayas International Conference on Internet*. IEEE, 2012, pp. 1–5.

[107] H.-W. Ng and S. Winkler, "A data-driven approach to cleaning large face datasets," in *Image Processing (ICIP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 343–347.

[108] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," University of Massachusetts, Amherst, Tech. Rep. 07-49, October 2007.

[109] O. M. Parkhi, A. Vedaldi, A. Zisserman *et al.*, "Deep face recognition." in *BMVC*, vol. 1, no. 3, 2015, p. 6.

[110] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 1701–1708.

[111] X. Cao, Y. Wei, F. Wen, and J. Sun, "Face alignment by explicit shape regression," *International Journal of Computer Vision*, vol. 107, no. 2, pp. 177–190, 2014.

[112] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

[113] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[114] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," in *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, Sydney, Australia, 2013.

[115] J. Xiao, K. A. Ehinger, J. Hays, A. Torralba, and A. Oliva, "Sun database: Exploring a large collection of scene categories," *International Journal of Computer Vision*, vol. 119, no. 1, pp. 3–22, 2016.

[116] L. Yao and J. Miller, "Tiny imagenet classification with convolutional neural networks," *CS 231N*, 2015.

[117] H. Kim, "Residual networks for tiny imagenet," *CS-231N Final Project Report*, 2016.

[118] F. Chollet *et al.*, "Keras," https://github.com/keras-team/keras, 2015.

[119] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[120] M. D. Bloice, C. Stocker, and A. Holzinger, "Augmentor: An image augmentation library for machine learning," *arXiv preprint arXiv:1708.04680*, 2017.

[121] S. Dieleman and B. Schrauwen, "End-to-end learning for music audio," in *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*. IEEE, 2014, pp. 6964–6968.

[122] M. Hayat, S. H. Khan, and M. Bennamoun, "Empowering simple binary classifiers for image set based face recognition," *International Journal of Computer Vision*, pp. 1–20, 2017.

[123] L. Kaiser, A. N. Gomez, N. Shazeer, A. Vaswani, N. Parmar, L. Jones, and J. Uszkoreit, "One model to learn them all," *arXiv preprint arXiv:1706.05137*, 2017.

[124] B. Meier, "Going deeper: Infinite deep neural networks," https://github.com/kutoga/going_deeper, 2017.

[125] S. Thrun and L. Pratt, *Learning to learn*. Springer Science & Business Media, 2012.

[126] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on*. IEEE, 2014, pp. 1717–1724.

[127] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *International conference on machine learning*, 2014, pp. 647–655.

[128] G. M. Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, P. Vincent *et al.*, "Unsupervised and transfer learning challenge: a deep learning approach," in *Proceedings of ICML Workshop on Unsupervised and Transfer Learning*, 2012, pp. 97–110.

[129] S. Ravanbakhsh, J. Schneider, and B. Poczos, "Deep learning with sets and point clouds," *arXiv preprint arXiv:1611.04500*, 2016.

[130] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, vol. 1, no. 2, p. 4, 2017.

B. Symbols

Symbol	Description
k	The cluster count.
$k_{\max}$	The maximum cluster count.
d	The dimension of a given data type.
X	The shape of an input example for a specific data type/encoding.
E	The shape of an embedding for a specific embedding network.
T	The size/length of a time dimension.
n	The number of input examples.
m	The number of RBDLSTM layers in the proposed model.
x_i	The ith input example.
$z(x)$	The embedding network: The input is an example, and the output is an embedding.
z_i	The embedding of the ith input example: $z_i = z(x_i)$
ξ_i	The output for the ith element after the RBDLSTM_m layer.
ℓ	The cluster index. By adding an index, e.g. ℓ_i, the cluster index for a specific example x_i can be addressed.
λ	A factor to weight loss terms.
C_i	A set of all input indices of all examples which are in the same cluster as x_i (including i).
$\zeta(\xi_i)$	The mean vector over all ξ_j-vectors for which the input example is in the same cluster cluster as x_i (including ξ_i).
$P(k)$	The model distribution for the estimated cluster count.
$P_r(k)$	The model distribution for the estimated cluster count for a specific run r. Multiple runs are only conducted in combination with forward dropout.
$P_{ij}(k)$	The model probability that the examples x_i and x_j are in the same cluster, given there are k clusters in all examples present. This is equal to $P(\ell_i = \ell_j \mid k)$.
P_{ij}	The model probability that the examples x_i and x_j are in the same cluster. This is equal to $P(\ell_i = \ell_j)$.
$\Omega_k(i,j)$	The model probability that x_i and x_j are in the same cluster, given that there are $k+1$ clusters, and that x_i and x_j are **not** in the same cluster, given that there are k clusters.
y_{ij}	The true label if x_i and y_i belong together (=are in the same cluster).
s_{ij}	The examples x_i and y_i are similar if s_{ij} is 1. On the other hand, if s_{ij} is 0, they are not similar.
$\text{CCE}(y, P(k))$	The categorical cross-entropy, given the true cluster count distribution as the first argument and the network prediction as the second argument.
$\text{BCE}_w(y_{ij}, P_{ij})$	The weighted binary cross-entropy, given the information, if two examples x_i and x_j are in the same cluster as the first argument and the network prediction as the second argument.
φ	The expected value of y_{ij}.
$\widetilde{\varphi}$	The approximated expected value of φ.
φ_0	The error-weight for cluster assignment outputs with the true class 0.
φ_1	The error-weight for cluster assignment outputs with the true class 1.
$\text{KL}(\text{P} \parallel \text{Q})$	The Kullback-Leibler divergence for the two distributions P and Q.
L_{cc}	The cluster-count estimation loss term.
L_{ca}	The cluster-assignment loss term.
L_{tot}	The complete loss expression that is used for the final model. It is equal to $L_{\text{cc}} + \lambda L_{\text{ca}}$.
L_{KL}	A loss term based on the KL-divergence.
L_{CA}	The cluster-assignment regularization loss term.
L_{NE}	A loss term that punishes empty clusters.
L_{CAR}	This is equivalent to $L_{\text{CA}} + \lambda_1 L_{\text{NE}}$; the default value for λ_1 is 1.
L_{RI}	A loss term based on a differentiable version of the RI.
RI	The *rand index*.
BBN_Q	The BBN metric.
BBN_{norm}	The normalized BBN metric.
MR	The *misclassification rate*.
MI	The *mutual information*.
NMI	The *normalized mutual information*.

C. List of Figures

D. List of Tables

E. Documents

Zürcher Hochschule
für Angewandte Wissenschaften

School of Engineering

InIT Institut für
angewandte Informationstechnologie

MSE | MASTER OF SCIENCE IN ENGINEERING

HS 2017

Masterarbeit

Studierender:	Benjamin Meier	Betreuer:	Dr. Thilo Stadelmann
Industriepartner:	-	Korreferent:	Dr. Oliver Dürr
		Experte:	Dr. Andreas Weiler

Ausgabe:	20. September 2017	Umfang:	27 ECTS (810 Ph)
Abgabetermin:	02. März 2018	Unterschrift:	
Präsentation:	22. März 2018		

Thema: Learning to Cluster - Investigations on Deep Learning for End-to-End Clustering

Many high-level algorithms and software products exist that use clustering algorithms to do their tasks. There already exist many approaches to solve the clustering problem. Some of these methods use neural networks to do the clustering, but they do not train this task in an end-to-end fashion. This master thesis investigates on the task of end-to-end clustering with neural networks. The input of the neural network is a set of data records and the output a clustering for the given data. The output of the network should contain the number of clusters and the data points for each cluster. The possible output range for the number of clusters may be fixed to a given range. The idea is that the neural network learns an optimal representation of the data to do clustering. This can be achieved by the end-to-end training.

Initially the state-of-the-art has to be explored to gain knowledge about what methods based on neural networks already exist to do clustering.

In a second step, possible solutions have to be created, implemented and to be evaluated. For the evaluations different datasets should be used. First some simple datasets (e.g. generated 2D point sets) and later more advanced datasets (images, audio, …).

The best solution is then fine-tuned and improved. A more detailed evaluation has to be done for this neural network.

All tasks shall be documented.

Aufgaben

1. Explore the state-of-the-art: What classical clustering algorithms exist? What approaches based on neural networks exist? Is already an end-to-end approach available?
2. Develop new ideas for a neural network architecture to do end-to-end clustering.
3. Evaluate the developed approach(es). Prepare different datasets to do this.
4. Fine-tune the best approach. Evaluate it with a more advanced dataset.
5. Write a documentation about the project, its results and your reflections in a adequate format.

6. Write a conference paper about the most important results of the work.
7. Prepare a 30 min presentation which contains the results of the project, the findings and your reflections.

Leistungsnachweis und Bewertung

- Written report incl. management review.
- Summary of the most important results as a conference paper.
- Presentation and an oral exam (each 30 min).