

# Learning to Cluster

Benjamin Meier, Thilo Stadelmann, and Oliver Dürr  
Zurich University of Applied Sciences  
Winterthur, Switzerland  
meierbe8@students.zhaw.ch, {stdm, dueo}@zhaw.ch

**Abstract**—We propose a novel neural network architecture to learn the task of clustering end-to-end: salient features for any similarity criterion specified through weakly labeled training data are extracted with an embedding network; during evaluation, the network groups similar data of any modality together, by assigning a probabilistic cluster index, and further gives a probabilistic estimate of the number of clusters. The method is evaluated on 2D-point data, speaker data from the TIMIT corpus, and images from the COIL-100 dataset, reaching promising results.

**Keywords**-clustering; deep learning; learning representations

## I. INTRODUCTION

In the last decades, there were many attempts made to solve the clustering problem. Many different algorithms [1] [2] [3] [4] exist which sometimes only work on a specific type of data [5] [6]. The clustering problem is very general and may be applied for almost any data type, e.g., for images [7] [8] [9], audio data [10] [11] [12], point data [13], or even text [14] [15] [16]. Often a distance measure (e.g. for hierarchical clustering [17]) and many other parameters are required to obtain good results. High dimensional data often requires a dimension reduction to be used with these algorithms [11] [18]. 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 [19] [20], handcrafted features [21] [22] [23] or embeddings generated by neural networks [11] [24]. PCA [25] 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 [26]. Hand-crafted features are being increasingly replaced by learned features, e.g. for classification tasks [27] [28] [29] and object detection tasks [30] [31] [32] [33]. Low dimensional representations generated by neural networks are often obtained by training a classification network [34], an auto-encoder [35] [36], or more advanced techniques [11]. It may be asked whether this representation is optimal for the clustering task? As can be shown [37], 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 [38] [39], including neural network-based approaches [11] [13].

Conventional clustering algorithms can be categorized into hierarchical and partitional approaches.  $k$ -means [2] and expectation maximization (EM) [3] are the best-known partitional algorithms, and agglomerative clustering is a well-known hierarchical clustering approach [40] [41] [42], but there are also other hierarchical clustering approaches proposed [43] [44]. Another possibility is the Hidden Markov Model (HMM)-based models, e.g. the one proposed by Lin et al. [45]. It can be shown that the quality of these algorithms for different problem statements may be very different [46] [47]. 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 [11] [13] [48] and unsupervised methods [8] [9]. 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 [11] [49] for a speaker-recognition task [50] and something else according to its content [51] [52] 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

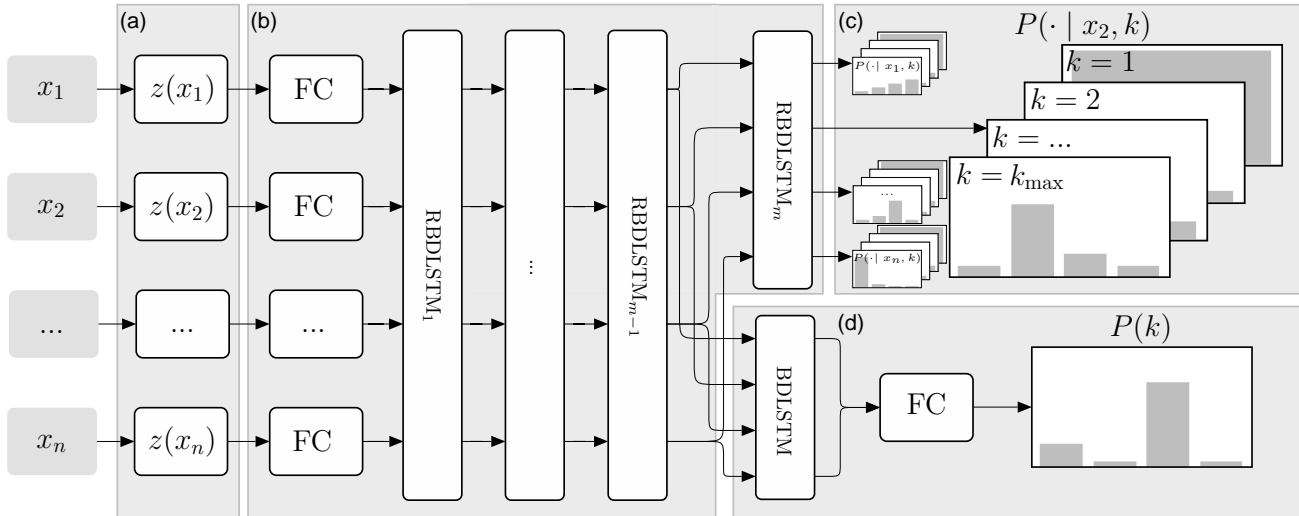


Figure 1: The complete model, consisting of (a) the embedding network, (b) the clustering network, (c) the cluster-assignment network, and (d) the cluster-count estimating network.

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 [49] [53] [54] [55] and which compared to one another [56] [57].

In this paper, we present a supervised end-to-end neural network architecture that uses a set of objects as input and produces directly a set of clusters as its output. Therefore, the embeddings are trained at the same time as the clustering “algorithm” itself.

We outline our approach in Section III and describe the used loss function more detailed in Section IV. The training process is described in Section V before reporting on the experimental evaluation in Section VI, giving all necessary details to make the presented work reproducible. Section VII concludes the paper with discussions and an outlook to future work.

## II. RELATED WORK

Several clustering approaches based on neural networks exist. Lukic et al. [11] train a neural network on a given TIMIT dataset [58] based on the approach described by Hsu et al. [59]. 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. [35] 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 [60] dataset. Peng et al. [61] and Tian et al. [62] 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. [9] use unsupervised learning methods based on a neural network to solve the clustering task. The proposed method works very well for MNIST [60]. 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. 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. Mahzani et al. [8] use a similar architecture to predict the cluster index. On the other hand, our proposed approach does not require that the training data contains all classes, and it does not have to be retrained for new classes.

Yang et al. [48] 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

Table I: 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	-	-	$n \times X$
<b>Embedding-Network (a)</b>				
	en-embedding	input	-	$n \times E$
<b>Clustering-Network (b)</b>				
	cn-fully-connected	en-embedding	288	$n \times 288$
	cn-leaky-relu[ $\alpha = 0.3$ ]	cn-fully-connected	-	$n \times 288$
	rbd lstm <sub>1</sub>	cn-leaky-relu[ $\alpha = 0.3$ ]	288	$n \times 288$
	rbd lstm <sub>2</sub>	rbd lstm <sub>1</sub>	288	$n \times 288$
	...	-	-	$n \times 288$
	rbd lstm <sub>m</sub>	rbd lstm <sub>m-1</sub>	288	$n \times 288$
<b>Cluster-Assignment Network (c): The layers are executed in parallel for each output of cn-rbd lstm<sub>m</sub>. Therefore, the shapes are always "<math>n \times \dots</math>".</b>				
	ca-fully-connected <sub>1</sub>	rbd lstm <sub>m</sub>	1	$n \times 1$
	ca-fully-connected <sub>2</sub>	rbd lstm <sub>m</sub>	2	$n \times 2$
	...	...	...	...
	ca-fully-connected <sub>k<sub>max</sub></sub>	rbd lstm <sub>m</sub>	$k_{\max}$	$n \times k_{\max}$
	ca-softmax <sub>1</sub>	ca-fully-connected <sub>1</sub>	-	$n \times 1$
	ca-softmax <sub>2</sub>	ca-fully-connected <sub>2</sub>	-	$n \times 2$
	...	...	-	...
	ca-softmax <sub>k<sub>max</sub></sub>	ca-fully-connected <sub>k<sub>max</sub></sub>	-	$n \times k_{\max}$
<b>Cluster-Count Estimating Network (d)</b>				
	cc-bd lstm	rbd lstm <sub>m-1</sub>	128	$n \times 128$
	cc-concat[first, last]	cc-bd lstm	-	256
	cc-fully-connected <sub>1</sub>	cc-concat	256	256
	cc-leaky-relu <sub>1</sub> [ $\alpha = 0.3$ ]	cc-fully-connected <sub>1</sub>	-	$n \times 288$
	cc-batch-norm <sub>1</sub>	c-leaky-relu <sub>1</sub>	-	256
	cc-dropout [ $p = 0.5$ ]	cc-batch-norm <sub>1</sub>	-	$n \times 288$
	cc-fully-connected <sub>2</sub>	cc-dropout	$k_{\max}$	$k_{\max}$
	cc-softmax	cc-fully-connected <sub>2</sub>	-	$k_{\max}$

the embedding. Our proposed method does not only work for images, but it is more general. Another big difference to the method proposed in this paper is the clustering algorithm: [48] uses an agglomerative-based clustering method, whereas the method proposed in this paper 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. [13] show that 2D CNNs are able to cluster 2D-point data. The network architecture is based on U-Net [63] 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. Our proposed method is able to cluster point data without any discretization and high-dimensional data-like images.

Our 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

[64] [65] [66] [67]. 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.

### III. PROPOSED MODEL ARCHITECTURE

Our described probabilistic model 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 1 and described more detailed in Table I. The network allows using another count of input examples  $n$  after the training since it is based on a recurrent architecture.

An important layer type of the proposed model is a residual bi-directional LSTM [68] layer (RBDLSTM). This layer is visualized in Figure 2. It is based on a bidirectional LSTM layer (BDLSTM) [69] [70] 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  $k$  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 [71]. This layer is very similar to the residual LSTM layer described in [72], 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 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 [48]: the CNN used varies for different image datasets). For example, for images, a CNN [28] 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. A CNN-based embedding network is described in Section VI.

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

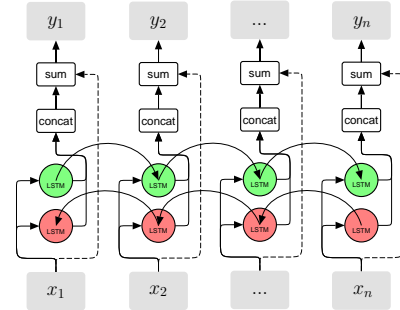


Figure 2: The RBDLSTM Layer: There are two underlying LSTM layers for both possible directions. Their output vectors are concatenated and then summed up element-wise with the input. The residual connections are dashed. All these sublayers together build an RBDLSTM layer.

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 advantages of using stacked RBDLSTM layers compared to a stacked BDLSTM network without residual connections; for instance, the back-propagation works much better [71] 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. The final representation of  $z_i$  after the  $\text{RBDLSTM}_m$  layer (see Figure 1) is called  $\xi_i$ . 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.

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  $P(\ell | x_i, k)$  which is the distribution for the cluster index  $\ell$  given a specific total cluster count and input example  $x_i$ . For instance,  $P(\ell = 2 | x_2, k = 3)$  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  $\text{RBDLSTM}_m$  layer, but this was not the case.

The cluster-count estimating network (d) uses the representation after the  $\text{RBDLSTM}_{m-1}$  layer as input. It could

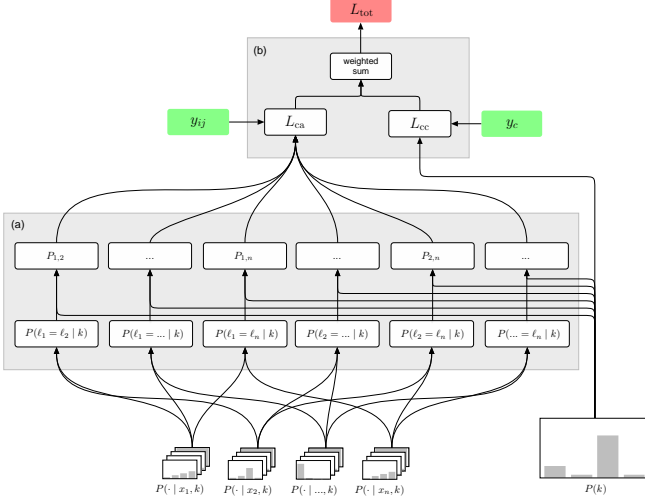


Figure 3: The used loss function which takes the given network outputs as input: The part (a) of the loss functions is used to calculate all  $\frac{n(n-1)}{2}$  combinations of  $P_{ij}$  with  $1 \leq i < j \leq n$ . The sub-sequential part (b) calculates the binary and the categorical cross-entropy 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  $P(k)$ .

be seen that this representation leads to better results. The reason for this is that the final representation  $\xi_i$  after the  $\text{RBDLSTM}_m$  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  $\text{RBDLSTM}$  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  $\xi_i$ ). This list of vectors is processed in a  $\text{BDLSTM}$  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.

#### IV. LOSS-FUNCTION

The different network outputs are described in Section III. 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  $P(\ell | x_i, k)$ .

For the cluster-count estimating output, the categorical

cross-entropy [73]  $\text{CCE}(y_c, P(k))$  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:

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

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 [73] loss is used. It is called  $\text{BCE}_w(y_{ij}, P_{ij})$ . 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 values to compare.

The formulas below show how to derive the probability  $P_{ij}$  given the network outputs described in Section III. The formula also contains 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 3, 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  $P(\ell_i = \ell_j | k)$ .

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

By marginalizing over  $k$ , the term

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

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

$$L_{ca} = \frac{-2}{n(n-1)} \sum_{i < j} \text{BCE}_w(y_{ij}, P_{ij})$$

with  $\text{BCE}_w(y_{ij}, P_{ij})$  being equal to the expression

$$\varphi_1 y_{ij} \log(P_{ij}) + \varphi_2 (1 - y_{ij}) \log(1 - P_{ij}).$$

The  $\text{BCE}_w(y_{ij}, P_{ij})$  loss uses different weights for the available classes. The value  $\varphi$  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 Section V, the cluster count is uniformly distributed in the training: This has a direct effect on  $\varphi$ . For the model proposed,  $\varphi$  is approximated by sampling data until the 95% confidence interval is smaller than 0.005. This approximated probability is called  $\tilde{\varphi}$ . 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  $\varphi$  (and therefore, also  $\tilde{\varphi}$ ) is quite small but still higher than 0.  $\tilde{\varphi}$ , 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:

$$\begin{aligned} \varphi_0 &= c\sqrt{\tilde{\varphi}} \\ \varphi_1 &= c\sqrt{1 - \tilde{\varphi}} \\ \varphi_0 + \varphi_1 &= 2 \\ \Rightarrow \varphi_0 &= 2 \left( 1 + \sqrt{\tilde{\varphi}^{-1} - 1} \right)^{-1} \\ \varphi_1 &= 2 \left( 1 + \sqrt{\frac{\tilde{\varphi}}{1 - \tilde{\varphi}}} \right)^{-1} \end{aligned}$$

The final loss is described by the following formula where the default value of  $\lambda$  is 5.0:

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

## V. 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 [74] 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 [75] 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.

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.

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  $N$

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 and a point count).

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.

## VI. EXPERIMENTAL RESULTS

For all experiments, several evaluation metrics are used, including the BBN metric [76], the *misclassification rate* (MR) [49] and the *normalized mutual information* (NMI) [77]. In the literature there are many different metrics used [49] [53] [54] [55] and there is no single “best” metric. For this reason, our experiments only contain a subset of them. The value range for all used metrics is  $[0, 1]$ , where all metrics are better if the value is higher, except for the MR; there is a lower value better.

The BBN metric [76] 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

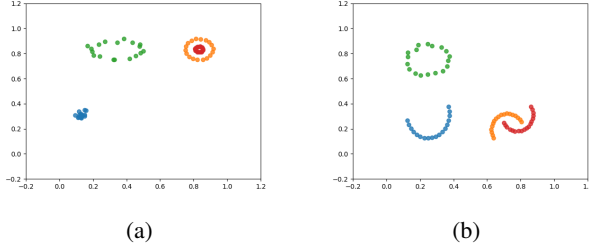


Figure 4: Two examples of possible point sets that are clustered by the network. Different colors indicate different cluster indices chosen by the network.

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:

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

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  $\text{BBN}_{\text{norm}}$  (given  $Q = 0$ ) is described in the formula below and has a value range of  $[0, 1]$ :

$$\text{BBN}_{\text{norm}}(y, \tilde{y}) = \frac{\text{BBN}_{Q=0}(y, \tilde{y})}{\text{BBN}_{Q=0}(y, y)}$$

The normalized BBN metric mainly measures the purity of clusters, i.e. pure clusters increase the score.

The MR [49] is used, because previous works also use it [11] [34] and it is a very intuitive measure: Assuming each cluster represents a class, then this measure is defined as the fraction of wrong assigned objects. Because there are many possibilities to assign a class to a cluster, the MR is defined as the minimum possible value. This is especially very useful if the underlying data set is a classification dataset. Originally, the MR was used in speaker clustering and is defined as

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

where  $N$  defined the total amount of audio pieces to cluster,  $N_s$  is the number of speakers and  $e_j$  the number of wrong assigned audio pieces of speaker  $j$ . We generalize this metric and use instead of audio pieces more general objects and instead of speakers, we use the true object clusters.

The NMI metric [77] is a widely used metric [48] [78] in clustering. The *mutual information* (MI) describes the information of one random variable  $U$  obtained through a second random variable  $V$  (e.g. if  $V$  fully describes  $U$ , then the metric is maximized). The NMI metric normalizes this

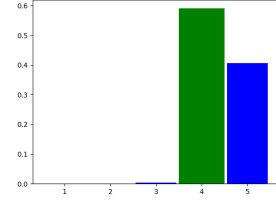


Figure 5: The cluster probabilities given by the neural network: The  $x$ -axis describes the cluster count and the  $y$ -axis the probability. The green marked probability is the true number of clusters.

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:

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

where MI is given by

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

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

All experiments are conducted on a computer with a NVIDIA Titan X Pascal with 12 GiB memory, an Intel Xeon E5-2650 CPU and 48 GiB of RAM. In this section, there are three experiments described that show how the proposed architecture may be used. The first experiment is done on 2D-point data, the second on the TIMIT [58] dataset and the last experiment is done on the COIL-100 [79] dataset.

For all experiments the means of all metrics are calculated on the defined test set with 300 randomly generated clusterings. The code to train the networks and to conduct all tests is online available.<sup>1</sup>

#### A. 2D-Point Clustering

This experiment is conducted on a set of 2D-points. The network input is a set of  $n = 72$  points and the batch size is  $N = 200$ . These points form different non-overlapping cluster types. E.g., there are (concentric) cycles, Gaussian distributed clusters and half moon shaped clusters. They are all mixed together. We use 1-5 clusters. The data for the training and also the test time is generated in real time.

The input points are in the range  $[0, 1] \times [0, 1]$  and are not preprocessed by any embedding network, i.e. the embedding network is just the identity function. The reason for this is, because the data is already very low dimensional and does

<sup>1</sup><https://github.com/kutoga/learning2cluster>

Table II: NMI  $\in [0, 1]$  and MR  $\in [0, 1]$  averaged over 300 evaluations of a trained network.

	2D Points (self generated)		TIMIT		COIL-100	
	MR	NMI	MR	NMI	MR	NMI
Our method	0.004	0.993	0.060	0.928	0.116	0.867
Random cluster assignment	0.485	0.232	0.435	0.346	0.435	0.346
Baselines (related work)	k-Means: MR = 0.178, NMI = 0.796 DBSCAN: MR = 0.265, NMI = 0.676		[11]: MR = 0		[48]: NMI = 0.985	

Table III: 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. All LeakyReLU-activations use the parameter  $\alpha = 0.3$ .

Name/Type	Kernel Size	Stride	Padding	Units	Output Shape
input	-	-	-	-	$128 \times 128 \times X$
conv <sub>1</sub>	$3 \times 3$	$1 \times 1$	1	32	$128 \times 128 \times 32$
leaky-relu <sub>1</sub>	-	-	-	-	$128 \times 128 \times 32$
batch-norm <sub>1</sub>	-	-	-	-	$128 \times 128 \times 32$
max-pool <sub>1</sub>	$2 \times 2$	$2 \times 2$	0	-	$64 \times 64 \times 32$
conv <sub>2</sub>	$3 \times 3$	$1 \times 1$	1	64	$64 \times 64 \times 64$
leaky-relu <sub>2</sub>	-	-	-	-	$64 \times 64 \times 64$
batch-norm <sub>2</sub>	-	-	-	-	$64 \times 64 \times 64$
max-pool <sub>2</sub>	$2 \times 2$	$2 \times 2$	0	-	$32 \times 32 \times 64$
conv <sub>3</sub>	$3 \times 3$	$1 \times 1$	1	128	$32 \times 32 \times 128$
leaky-relu <sub>3</sub>	-	-	-	-	$32 \times 32 \times 128$
batch-norm <sub>3</sub>	-	-	-	-	$32 \times 32 \times 128$
max-pool <sub>3</sub>	$2 \times 2$	$2 \times 2$	0	-	$16 \times 16 \times 128$
fully-connected <sub>1</sub>	-	-	-	256	256
leaky-relu <sub>4</sub>	-	-	-	-	256
batch-norm <sub>4</sub>	-	-	-	-	256
fully-connected <sub>2</sub>	-	-	-	256	256
leaky-relu <sub>5</sub>	-	-	-	-	256
batch-norm <sub>5</sub>	-	-	-	-	256

not contain redundant information. Figure 4 shows some example clusterings by the final trained network.





















Table II shows the final metrics after the training was finished by early stopping after 128 700 iterations (=123 hours). It can be seen that the network reaches a very high accuracy on the given task. This task may be seen as solved with our proposed architecture.

### B. Speaker Clustering

The speaker clustering experiment is conducted on the TIMIT [58] dataset. The network input consists of  $n = 20$  audio snippets with a length of 1.28 seconds. The encoding of the input data is based on [11] and is basically a mel-spectrogram with 128 coefficients and 128 timesteps. This input is handled as an image with one color channel like in [11]. The used embedding network which is described in Table III contains 3 convolutional and max-pooling layer and 2 fully connected layers. The final representation contains 256 values. A batch-size of  $N = 25$  is used.

The TIMIT dataset contains 602 speakers with 10 sentences, where each sentence has a length of 2-4 seconds. We first concatenate all spoken sentences for each speaker, where the audio pieces are ordered by their filename. This results in one long piece of audio per speaker and, therefore,

Table IV: 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	    
1	   
2	     
3	    

in 602 audio records. Finally, there is exactly one audio record per speaker. This dataset is split into a training set with 402 speakers, a validation set with 100 speakers and a test set with 100 speakers. The network expects inputs with a length of 1.28 seconds. These small audio pieces are randomly sampled from the audio sequence of a given speaker. The objective of the network is to decide which audio snippets are generated by the same speaker. The network can detect 1-5 clusters.


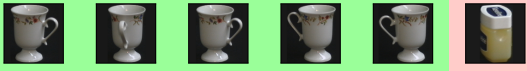


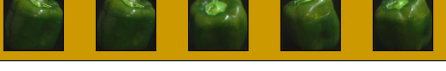
After 72 500 iterations (=18 hours) the training was finished by early stopping. The reached metrics on the test set are shown in Table II. It can be seen that the quality is not as high as in the 2D-point cluster test, but the network is still very accurate.

As described in Section III, the network has an output that estimates the count of clusters. An example of the cluster-count estimation output is given in Figure 5. We noticed that the confidence for a given cluster count is higher if there are only a few clusters and lower if there are many clusters. To obtain this result, we used methods based on [80]. An output of the network is visualized in Table IV.

Lukic et al. [11] reach a perfect MR of 0.0 given 20 speakers/clusters with two samples per speaker. They do



Table V: 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.

Cluster	Objects
0	
1	
2	
3	
4	

combine multiple 1 second snippets of a speaker which we cannot do with our architecture. One could solve this by using a BDLSTM-embedding network, but this makes the training time impractical, as we found out. Therefore, [11] performs in general better, but the advantage of our approach is the generality and the end-to-end way in which we can train the clustering task.

### C. Image Clustering

This experiment is conducted on the COIL-100 [79] dataset. The input of the network is a set of  $n = 20$  images with 3 color channels and a size of  $128 \times 128$  pixels. The same CNN like in the speaker clustering experiment is used. This CNN is described in Table III. The first layer in the CNN differs, because the input channel count for the two experiments is different.

The task of the network is to cluster images with the same object. [81] is used to do data augmentation on the given dataset. We use a random distortion with a grid size of  $4 \times 4$ , a magnitude of 8 and a probability of 1, followed by a left-right flip with the probability 0.5. The network can detect 1-5 clusters. We use 80 classes for the training, 10 classes for validation and 10 classes for the final test. After 32 100 iterations (=8 hours) the training was finished by early stopping. The reached metrics on the test set are shown in Table II.

## VII. DISCUSSION & CONCLUSIONS

We have shown that our novel method is able to cluster different data types with promising results. It is a complete end-to-end approach to clustering, that learns both the relevant features and the “algorithm” by which to produce the clustering itself, as well as the number of clusters in the data. The learning phase only requires pairwise labels between examples, and no explicit similarity measure needs

to be provided. Promising results are achieved on a variety of modalities.

We observe that the final clustering accuracy depends on the availability of a large number of different classes during training. We attribute this to the fact that the network needs to learn intra-class distances, a task inherently more difficult than just to distinguish between objects of a fixed amount of classes like in classification problems.

## REFERENCES

- [1] M. J. Zaki, W. Meira Jr, and W. Meira, *Data mining and analysis: fundamental concepts and algorithms*. Cambridge University Press, 2014.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] M. Robin, G. Nicolle, and A. Rota, “Automatic sounds clustering approach based on a likelihood measure computation.”
- [6] 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.
- [7] 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.
- [8] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, “Adversarial autoencoders,” *arXiv preprint arXiv:1511.05644*, 2015.
- [9] M. Kampffmeyer, S. Lkse, 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] A. Borji and A. Dundar, “Human-like clustering with deep convolutional neural networks,” 2017.

- [14] C. C. Aggarwal and C. Zhai, "A survey of text clustering algorithms," in *Mining text data*. Springer, 2012, pp. 77–128.
- [15] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] K. K. Vasam and B. Surendiran, "Dimensionality reduction using principal component analysis for network intrusion detection," *Perspectives in Science*, vol. 8, pp. 510–512, 2016.
- [20] 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.
- [21] 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.
- [22] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.
- [23] B. Hariharan, J. Malik, and D. Ramanan, "Discriminative decorrelation for clustering and classification," *Computer Vision—ECCV 2012*, pp. 459–472, 2012.
- [24] J. Xie, R. Girshick, and A. Farhadi, "Unsupervised deep embedding for clustering analysis," in *International Conference on Machine Learning*, 2016, pp. 478–487.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
- [30] 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.
- [31] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.
- [32] 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.
- [33] 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.
- [34] 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.
- [35] 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.
- [36] 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.
- [37] 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.
- [38] 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.
- [39] 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.
- [40] D. Müllner, "Modern hierarchical, agglomerative clustering algorithms," *arXiv preprint arXiv:1109.2378*, 2011.
- [41] 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.
- [42] T. Kurita, "An efficient agglomerative clustering algorithm using a heap," *Pattern Recognition*, vol. 24, no. 3, pp. 205–209, 1991.
- [43] A. Roy and S. Pokutta, "Hierarchical clustering via spreading metrics," in *Advances in Neural Information Processing Systems*, 2016, pp. 2316–2324.
- [44] 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.
- [45] 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>
- [46] 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.

- [47] O. A. Abbas, "Comparisons between data clustering algorithms." *International Arab Journal of Information Technology (IAJIT)*, vol. 5, no. 3, 2008.
- [48] 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.
- [49] 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.
- [50] T. Stadelmann, "Voice modeling methods for automatic speaker recognition," 2010.
- [51] 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.
- [52] 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.
- [53] 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.
- [54] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure." in *EMNLP-CoNLL*, vol. 7, 2007, pp. 410–420.
- [55] 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.
- [56] 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.
- [57] 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>
- [58] 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.
- [59] Y.-C. Hsu and Z. Kira, "Neural network-based clustering using pairwise constraints," *arXiv preprint arXiv:1511.06321*, 2015.
- [60] Y. LeCun and C. Cortes, "MNIST handwritten digit database," 2010. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [61] X. Peng, S. Xiao, J. Feng, W.-Y. Yau, and Z. Yi, "Deep subspace clustering with sparsity prior." in *IJCAI*, 2016, pp. 1925–1931.
- [62] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, "Learning deep representations for graph clustering." in *AAAI*, 2014, pp. 1293–1299.
- [63] 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.
- [64] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *International Workshop on Similarity-Based Pattern Recognition*. Springer, 2015, pp. 84–92.
- [65] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin, "Deep metric learning with angular loss," *arXiv preprint arXiv:1708.01682*, 2017.
- [66] 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.
- [67] 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.
- [68] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [69] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.
- [70] 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.
- [71] 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.
- [72] 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.
- [73] K. Janocha and W. M. Czarnecki, "On loss functions for deep neural networks in classification," *arXiv preprint arXiv:1702.05659*, 2017.
- [74] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.
- [75] Y. Yao, L. Rosasco, and A. Caponnetto, "On early stopping in gradient descent learning," *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, 2007.
- [76] M. Kotti, V. Moschou, and C. Kotropoulos, "Speaker segmentation and clustering," *Signal processing*, vol. 88, no. 5, pp. 1091–1124, 2008.
- [77] A. F. McDaid, D. Greene, and N. Hurley, "Normalized mutual information to evaluate overlapping community finding algorithms," *arXiv preprint arXiv:1110.2515*, 2011.
- [78] 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.
- [79] 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.
- [80] 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.
- [81] M. D. Bloice, C. Stocker, and A. Holzinger, "Augmentor: An image augmentation library for machine learning," *arXiv preprint arXiv:1708.04680*, 2017.