

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

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

Abstract—Clustering has been studied for a long time, because it is very central for many data-driven applications. Neural networks are also studied more intensive in the last years, especially end-to-end solutions. We propose a new end-to-end clustering architecture based on neural networks. Our architecture learns embeddings and the clustering task at the same time. The architecture is general and may be used for many different data types.

Keywords—clustering; deep learning

I. INTRODUCTION

In the last decades many attempts to solve the clustering problems are done. There are many different algorithms [1] [2] [3] which sometimes only work on a specific type of data [4] [5]. The clustering problem is very general and may be applied for almost any data type, e.g. for images [6], audio data [7], point data [8] or even text [9]. Often a distance measure (e.g. for hierarchical clustering [10]) and many other parameters are required to obtain good results. High dimensional data often requires a dimension reduction to be used with these algorithms [7] [11]. 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 [12] [13], handcrafted features [14] [15] [16] or embeddings generated by neural networks [7] [17]. PCA [18] can only do linear transformations which is often not sufficient, hand-crafted features are often very hard to obtain and it is not always sure if they are even optimal [19]. Hand-crafted features are more and more replaced by learned features, e.g. for classification tasks [20] [21] [22] or object detection tasks [23] [24] [25] [26]. Low dimensional representations generated by neural networks are often obtained by the training of a classification network [27], an auto-encoder [28] [29] or more advanced techniques [7]. It may be asked if this representation is optimal for the clustering task? As can be shown [30], different objects of the same class, but of different subclasses, may have a very different representation in neural networks until the very last layer. Therefore, the previous representations may be not very well for clustering. It also may 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 [31] [32], including neural network based approaches [7].

Conventional clustering algorithms can be categorized into hierarchical and partitional approaches. K-means [1] and expectation maximization (EM) [2] are the best-known partitional algorithms, and agglomerative clustering is a well-known hierarchical clustering approach [33] [34] [35], but there are also other hierarchical clustering approaches proposed [36] [37]. Another possibility are Hidden Markov Model (HMM) based models, e.g. the one proposed by Lin et al. [38]. It can be shown that the quality of these algorithms for different problem statements may be very different [39] [40]. 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 big categories: Supervised methods [27] [41] [8] and unsupervised methods [6]. Unsupervised methods do not use any label to learn the clustering. This is an advantage, because labels are often expensive to get. On the other hand, one has to trust the neural network to learn a low-dimensional representation that contains the relevant information. If 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 to cluster the data may be different, even for the same dataset. E.g., one may like to cluster audio according to the voice [7] [42] for a speaker recognition task [43] and someone else according to the content [44] [45] for a speech recognition task. This means it is in general not possible to deduce a unique, natural and correct similarity function or clustering just from the data. Therefore, unsupervised clustering may not work in any use-case, because the used method decides the criterion to cluster the data.

Because there are so many clustering algorithms available, it is important to get the best algorithm for a specific task. This requires a metric for the clustering quality. Unfortunately, there is not a single natural metric available, like the accuracy for classification problems. Therefore, many different metrics to measure the quality of a clustering result are proposed [46] [47] [48] [42] and compared to each other [49] [50].

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. [7] train a neural network given the TIMIT dataset [51] based on the approach described by Hsu et al. [52]. The described supervised speaker clustering method reaches state-of-the-art results with the trained embedding and a conventional hierarchical clustering approach for the given data. However, the used embedding is not explicitly trained for the given task and they are used in combination with an external clustering algorithm, therefore the training is not done in an end-to-end fashion.

Guo et al. [28] use an auto-encoder to generate embeddings that 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 [53] and the USPS datasets. Peng et al. [54] and Tian et al. [55] 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 in a way that the content completely can be decompressed again. Often there are only a very few aspects which are relevant for the clustering and they even might not be extremely relevant to reproduce the input. So the embedding may be much smaller and more effective if other training approaches are used.

Kampffmeyer et al. [6] use unsupervised learning methods based on a neural network to solve the clustering task. The proposed method works very well for MNIST [53]. On the other side, it is not possible to give the network hints which aspect should be used for the clustering task. Therefore, the network itself chooses just a set of features for the clustering task. This means the network decides what the meaning of similar inputs is. 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. Additionally, the exact number of clusters has to be known, because the trained network finally takes one input, independent of all other objects, and predicts the cluster index. On the other hand, our approach does not require that the training data contains all classes and it does not have to be re-trained for new classes. The exact number of clusters does not have to be known, but while creating the network, a limited range of possible clusters per input for

the network must be known. Our network is able to cluster data records dependent on other data records, this is, e.g., required if point clouds are clustered. For this task it is not possible to obtain the correct target cluster for each point independent of all other points.

Yang et al. [41] 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 optimization through the complete network and, therefore, can be used to optimize the embedding. Our method does not only work for images, but is more general. Another big difference to our method is the clustering algorithm: [41] uses an agglomerative-based clustering method and we just let the network decide how the data should be clustered. Our network learns a clustering algorithm from scratch and, therefore, can learn more specialized algorithms.

Borji et al. [8] show that 2D CNNs are able to cluster 2D point data. The network architecture is based on U-Net [56] and can predict up to N clusters, where N 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.

Wang et al. [57] 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, we are able to give the network information about what kind of feature we want to cluster. There are more methods that extract a good representation of images with unsupervised training [58] [59] [60], but we do not compare our approach to these methods, because we use supervised training and, therefore, have more information at the training time.

Our proposed method has to learn some kind of similarity function for different input objects, this exact task is explored in the deep learning subfield Deep Metric Learning [61] [62] [63]. However, we do focus on this specific task, but more on the complete clustering task, which we learn end-to-end. It cannot even be shown how exactly our proposed network does compare two objects, because we do not use an explicit function or loss for this similarity on the embedding level. Therefore, we do not compare us against Deep Metric Learning solutions.

III. PROPOSED MODEL ARCHITECTURE

The described model does end-to-end clustering, therefore, the input of the model is a set of objects x_i (for $0 \leq i < n$) and the output a set of clusters. There is an output that describes the cluster count and for each input object x_i the cluster index, assuming there are k clusters.

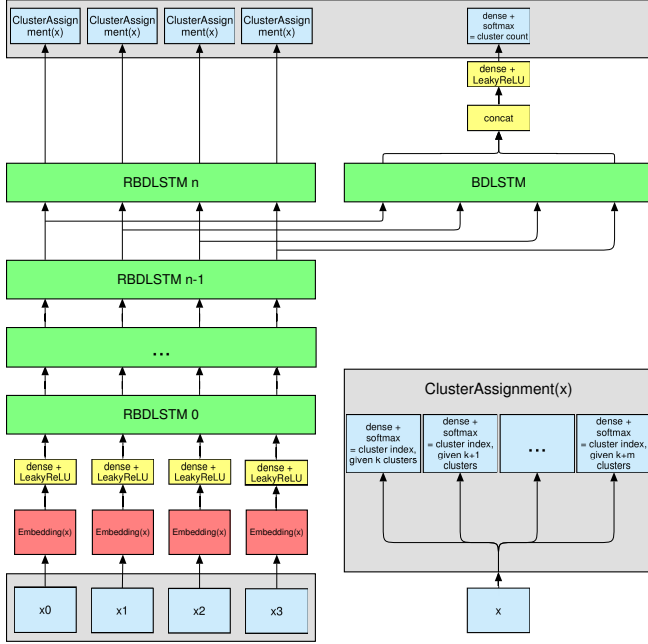


Figure 1: The used network architecture with 4 inputs. In general the number of inputs is not limited and has to be a non-empty set of objects.

Given the most probable cluster count, the most probable cluster index for each object x_i can be calculated easily. There is a limited number k of possible cluster counts, for which the following limits exists: $k_{\min} \leq k \leq k_{\max}$. k_{\min} and k_{\max} have to be defined before the training process. The complete network architecture is visualized on Figure (1) and described more detailed in Table I. The network may use another count of input objects after the training, because it is based on a recurrent architecture.

An important layer type of the proposed model is a residual bi-directional LSTM [64] layer (RBDLSTM). This layer is visualized on Figure (2). It is based on a bidirectional LSTM-layer (BDLSTM) [65] [66] with an additional residual connection that adds a direct connection from the input to the output of the BDLSTM-layer. A bidirectional layer is used instead of a single-directional layer, because the input sequence has not really an order and information has to be exchanged from every object to every object. The underlying BDLSTM-layer concatenates the output of the two LSTM-networks. Residual connections allow in general much deeper architectures and are very helpful for an efficient backpropagation [67]. An important restriction of a RBDLSTM-layer is that the input shape must match the output shape. Therefore, the amount of internal units for each BDLSTM-layer inside the RBDLSTM-layer, must be equal to half of the number of components of the input vectors.

The proposed model architecture contains an initial di-

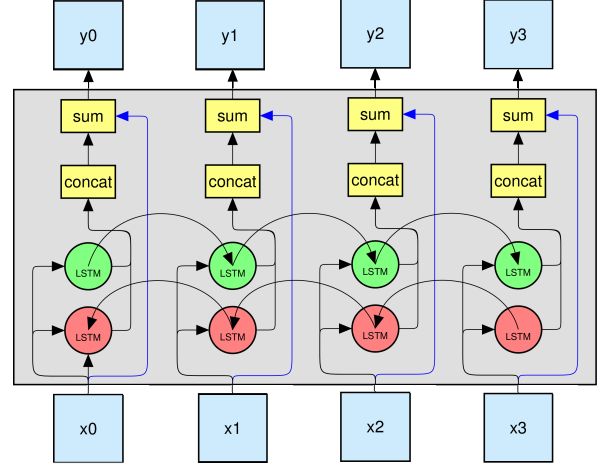


Figure 2: RBDLSTM Layer. The residual connections are blue.

mension reduction network that creates for each input object x_i an embedding e_i . The embedding network may be data specific (like in [41]). E.g., for images a CNN [21] may be used. Other highly compact data types, like n -dimensional points may not use any embedding-network at all, because their representation is already very compact. For 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 Section VI are some experiments described, including an embedding network.

Once the embeddings are available, the representation is scaled to the required size for the RBDLSTM-network. 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 to use exact the same network architecture, except for the embedding network, for many different data types.

These scaled representations are the input for a stacked RBDLSTM network with 14 layers. These layers build the clustering network. Each vector of each input contains after each RBDLSTM-layer less information about the embedding e_i and more information about the target cluster. Each RBDLSTM-layer refines this information, by using all available inputs and combining the information of them. Therefore, the clustering algorithm itself is encoded in these stacked RBDLSTM-layers. There are several advantages by using stacked RBDLSTM-layers compared to an stacked BDLSTM-network without residual connections, e.g., back-propagation works much better [67], 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 where the representations are always modified by a delta. LSTM-layers always would not only modify the input

by a delta, but create really new outputs, so RBDLSTM-layers seem to be a more natural choice. We were able to show that this type of layer performs much better on the given clustering-task. The final representation after all RBDLSTM-layers is called ξ_i . These vectors no longer have to contain specific information about the input object x_i , but only about to which cluster a given input is assigned to.

There are two following networks after the clustering-network. One is the cluster-count-guessing network, and the other the cluster-assignment network. The cluster-assignment network contains for each possible cluster count $k_{\min} \leq k \leq k_{\max}$ in combination with each input-vector x_i a fully connected layer that uses a softmax-activation. $\text{CI}(x_i)$ returns the assigned cluster index of the input x_i . For this reason, $0 \leq \text{CI}(x_i) < k$ is always true. Therefore, network-outputs $P(\text{CI}(x_i) = c | k_{\text{real}} = k)$ for all input indices i , cluster-counts $k_{\min} \leq k \leq k_{\max}$ and cluster-indices $0 \leq c < k$ are available.

The cluster-count-guessing network uses as input the representation before the last RBDLSTM-layer of the clustering-network. It could be seen that this representation leads to better results. The reason for this is that the final representation ξ_i 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 before the last RBDLSTM-layer is a better choice. This layer contains already 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 vector are used for further processing. These two vector are concatenated and then processed in a fully connected layer. Then there is a final fully connected layer that uses a softmax-activation and produces the cluster-count distribution. The outputs describe $P(k_{\text{real}} = k)$ for each $k_{\min} \leq k \leq k_{\max}$.

IV. LOSS-FUNCTION

The different network outputs are described in Section III. They are used to define a loss function for the given network architecture. For the cluster-count output the categorical crossentropy [68] $\text{CCE}(y, \tilde{y})$ is used. We call this term ℓ_{cc} . It is not possible to predefine which elements should be assigned to which cluster indices, because there are multiple valid solutions for this problem. Therefore, the loss function must take care of this property. This fact makes it impossible to just use another categorical crossentropy for the cluster-assignment outputs. Therefore, the probability $P(\text{CI}(x_i) = \text{CI}(x_j))$ is calculated, because this probability makes no more assumptions about a fixed cluster index for a given element. For this probability, a weighted version of the binary crossentropy [68] loss is used. It is called $\text{BCE}_w(y, \tilde{y})$. For a given clustering it is simple to get the required labels, because it is known whether two objects are in the same cluster. If they are in the same cluster, the target

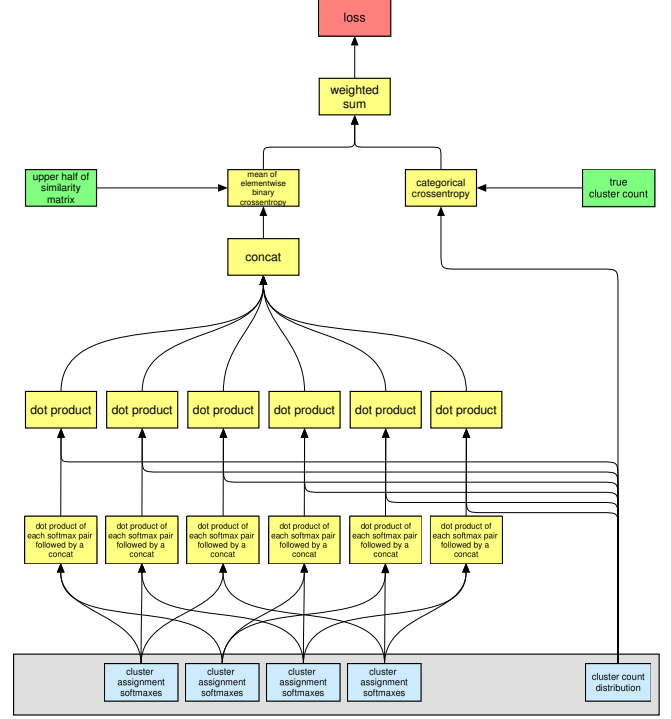


Figure 3: The used loss function. The upper half of the similarity matrix contains all labels for $P(\text{CI}(x_i) = \text{CI}(x_j))$ for all $i, j \in \{0, 1, \dots, n-1\}$ where $i < j$.

is 1, otherwise it is 0. This loss requires that every element is compared with every other element, and therefore, there are $\frac{n(n-1)}{2}$ resulting values to compare.

The formula below shows how to derive the probability $P(\text{CI}(x_i) = \text{CI}(x_j))$, given the network outputs described in Section III. The formula also contains the loss term ℓ_{ca} that describes the average error on these calculated probabilities compared to the ground truth. The complete loss function is visualized on Figure (3), where the input values are at the bottom and the loss, based on these values is at the top.

$$\begin{aligned}
 P(\text{CI}(x_i) = \text{CI}(x_j) | k_{\text{real}} = k) &= \\
 \sum_{c=0}^{k-1} P(\text{CI}(x_i) = c | k_{\text{real}} = k) P(\text{CI}(x_j) = c | k_{\text{real}} = k) &= \\
 P(\text{CI}(x_i) = \text{CI}(x_j)) &= \\
 \sum_{k=k_{\min}}^{k_{\max}} P(\text{CI}(x_i) = \text{CI}(x_j) | k_{\text{real}} = k) P(k_{\text{real}} = k) &= \\
 \ell_{\text{ca}} = &= \\
 \frac{2}{n(n-1)} \sum_{0 \leq i < j < n} \text{BCE}_w(y_{\text{true}}, P(\text{CI}(x_i) = \text{CI}(x_j))) &
 \end{aligned}$$

Table I: All layers of the network are described in this table. The input layer is green and all output layer 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 depends on the used embedding. This value is fixed during the training and testing time.

Name / Type	Input Layer	Units	Output Size
input	-	-	$N \times X$
Clustering-Network			
cn-embedding	input	-	$N \times E$
cn-dense	cn-embedding	288	$N \times 288$
cn-leaky-relu[$\alpha = 0.3$]	cn-dense	-	$N \times 288$
cn-rbd lstm ₀	cn-leaky-relu[$\alpha = 0.3$]	288	$N \times 288$
...	-	-	$N \times 288$
cn-rbd lstm _n	cn-rbd lstm _{n-1}	288	$N \times 288$
Cluster-Count Network			
cc-bd lstm	cn-rbd lstm _{n-1}	128	$N \times 128$
cc-concat[first, last]	cc-bd lstm	-	256
cc-dense ₀	cc-concat	256	256
cc-batch-norm ₀	cc-dense ₀	-	256
cc-leaky-relu ₀ [$\alpha = 0.3$]	cc-batch-norm ₀	-	$N \times 288$
cc-dropout[$p = 0.5$]	cc-leaky-relu ₀	-	$N \times 288$
cc-dense ₁	cc-dropout ₀	$k_{\max} - k_{\min} + 1$	$k_{\max} - k_{\min} + 1$
cc-softmax	cc-dense ₁	-	$k_{\max} - k_{\min} + 1$
Cluster-Assignment Network			
ca-dense _{k_{min}}	cn-rbd lstm _n	k_{\min}	$N \times k_{\min}$
ca-dense _{k_{min}+1}	cn-rbd lstm _n	$k_{\min} + 1$	$N \times k_{\min} + 1$
...	cn-rbd lstm _n
ca-dense _{k_{max}}	cn-rbd lstm _n	k_{\max}	$N \times k_{\max}$
ca-softmax _{k_{min}}	ca-dense _{k_{min}}	-	$N \times k_{\min}$
ca-softmax _{k_{min}+1}	ca-dense _{k_{min}+1}	-	$N \times k_{\min} + 1$
...	ca-dense _n	-	...
ca-softmax _{k_{max}}	ca-dense _{k_{max}}	-	$N \times k_{\max}$

The $\text{BCE}_w(y, \tilde{y})$ -loss uses different weights for the available classes. We define φ as the probability that a label y of our $\text{BCE}_w(y, \tilde{y})$ -function is 1. This probability is taken over all possible cluster counts for a fixed input object count. We approximate φ 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 φ (and, therefore also $\tilde{\varphi}$) is quite small, but still higher than 0. $\tilde{\varphi}$ in general makes only sense if the value is larger than 0 and smaller than 1, otherwise the task is trivial. If $\varphi = 0$, then every element is always in an own cluster. On the other side if $\varphi = 1$, then all elements are always in the same cluster. To soften the proportion between the weights, the \sqrt{x} -function is used. 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 formula below shows how these error weights are derived.

$$\begin{aligned}
w_0 &= c\sqrt{\tilde{\varphi}} \\
w_1 &= c\sqrt{1 - \tilde{\varphi}} \\
w_0 + w_1 &= 2 \\
\Rightarrow c &= \frac{2}{\sqrt{\tilde{\varphi}} + \sqrt{1 - \tilde{\varphi}}} \\
\Rightarrow w_0 &= 2 \left(1 + \sqrt{\tilde{\varphi}^{-1} - 1} \right)^{-1} \\
w_1 &= 2 \left(1 + \sqrt{\frac{\tilde{\varphi}}{1 - \tilde{\varphi}}} \right)^{-1}
\end{aligned}$$

The final loss is described by the following formula:

$$\ell_{\text{tot}} = \alpha \ell_{\text{cc}} + \beta \ell_{\text{ca}}$$

The used hyperparameter in the described architecture are $\alpha = 1.0$ and $\beta = 5.0$. This configuration is used for any input data.

V. TRAINING

For different data types, the training may require much more or less time, e.g. point data is much easier to train than images. In general, the Adadelta [69] optimizer is used with a learning rate of 5.0. Early stopping [70] is used: If there is no new best valid loss for more than 15'000 iterations, the training is stopped. A test on the validation data is only done every 100th iteration.

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 and training set, where each class is present in both sets. The given amount of objects 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 objects per class are also important. Many classes are required to learn some kind of class distance function, therefore, the limitation is often the given amount of classes. The network not only has to learn there are some classes which contain a given feature, but the difference between classes. This also has some implications to the test and training set. It is split in a way that the training set contains some classes and all objects of these classes and the test set contains a disjunct set of classes with all objects according to them. This is required to test if the network really learns the true class distance-function, because then this function is 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 and then the training and the tests are done on the same classes, but disjunct object sets, the task is much easier. This basically let the network overfit to some classes and the network might not even learn a real distance function.

In general the network does not require class labelled data for the training, but cluster labelled data. The difference is that each object in general is exactly in one class, but it may appear in different clusters. Given the 2D-point (1, 2): Depending on neighbouring 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 labelled data as cluster labelled data, because it just can be defined that one class with all objects is equal to a cluster. 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 labelled data types are used, then an object can be assigned to a fixed cluster without having a look to all other input objects. E.g., audio snippets with different voices are such objects. Each voice is a class. The class could be extracted independent of all other objects and then it could be compared to them. For other data types, this property is not given. E.g., for n -dimensional point sets all

objects / points must be observed to know where a cluster is and what points are contained in it. The cluster structure highly depends on the neighbouring data points. Therefore, approaches like [6] that assume a cluster index may be determined for an object independent of all other objects do not work on data like n -dimensional point sets. Our approach can handle these cases very well.

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 allow lower minibatch sizes than low-dimensional point data.

In each iteration, input data is generated: First the number of target clusters is generated. This number is sampled from an uniform distribution over all possible cluster counts. Then for each cluster a distinct random class is selected. In the experiments a minimal number n_m of objects per cluster is defined (usually $n_m = 2$), therefore, for each class are n_m objects sampled and added to the set of input objects. Then, until there are enough input objects, a random class of the defined clusters is sampled and the a random object of this class is sampled. 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 objects are shuffled before they are used as input for the network. This method allows it to generate extremely many distinct training records for a classification dataset. Already the input permutation generates $n!$ possibilities. The task is in general quite difficult to train and the limitation is often the amount of classes.

VI. EXPERIMENTAL RESULTS

It is quite important to evaluate the results: We used several evaluation metrics, including the BBN-metric [71], the Misclassification Rate (MR) [42], the Normalized Mutual Information (NMI) [72], the Completeness Score (CS) [47] and many more. The BBN-metric is not normalized, therefore, a normalized version is introduced, because otherwise it may be hard to interpret the score given different cluster counts. The normalized version BBN_{norm} (given $Q = 0$) is described in the formula below. In the literature there are many different metrics used [46] [47] [48] [42]. For this reason, our experiments only contain a subset of them.

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

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.

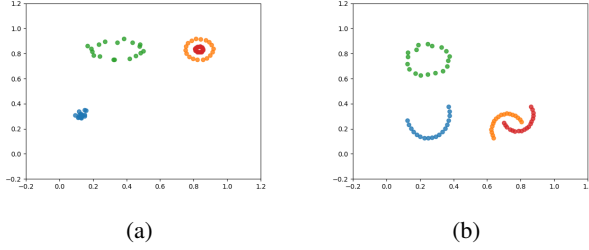


Figure 4: Two examples of possible point sets that are clustered by the network. Different colors indicate different cluster indices chosen by the network.

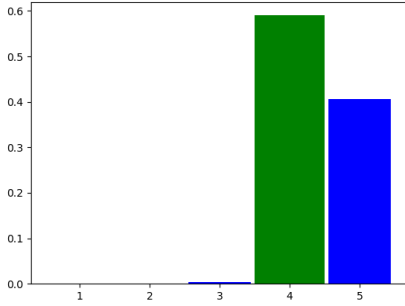


Figure 5: The cluster probabilities given by the neural network: The green marked probability is the true number of clusters. The x-axis describes the cluster count and the y-axis the probability.

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 dataset [51] and the last experiment is done on the FaceScrub [73] dataset.

All metrics are calculated on the defined test set with 300 randomly generated clusterings. For each metric, the mean (μ) and the standard deviation (σ) is given.

The code to train the networks and to conduct all tests is online available¹.

A. 2D Point Clustering

This experiment is conducted on a set of 2D points. The network input is a set of 72 points and the batch size is 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, therefore, we do not depend on a dataset with a limited number of records.

Table II: Different metric values for the 2D point clustering network. μ describes the mean and σ the standard deviation for 300 conducted clusterings on the test set.

Metric	μ	σ
MR	0.0076	0.0297
BBN _{norm}	0.9889	0.0359
NMI	0.9876	0.0382
CS	0.9879	0.0363

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

Table II shows the final metrics after the training finished after XYZ iterations. 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 [51] dataset. The network input consists of 20 audio snippets with a length of 1.28 seconds. The encoding of the input data is based on [7] and is basically a mel-cepstral with 128 coefficients and 128 timesteps. This input is handled as an image with one color channel like in [7]. The used embedding network which is described in Table V contains 3 convolutional and max-pooling layer and 2 fully connected layers. The final representation contains 256 values. A batch-size of 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, 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 XYZ iterations the early stopping finished the training. The reached metrics on the test set are shown in Table III. 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 guesses the count of clusters and for each input record an output that describes in which cluster the element is, given

¹<https://github.engineering.zhaw.ch/meierbe8/ClusterNN>

Table III: Different metric values for the speaker clustering network. μ describes the mean and σ the standard deviation for 300 conducted clusterings on the test set.

Metric	μ	σ
MR	0.0603	0.0977
BBN _{norm}	0.9350	0.1037
NMI	0.9283	0.1251
CS	0.9377	0.1252

there are k clusters. An example of the cluster guessing output is given on 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 [74]. An output of the network is visualized in Table IV. The rows visualize the different clusters generated by the neural network.

Another fact is that the networks accuracy in general increases if the input length per snippet is increased, e.g. to 2 seconds. As we found out, 1.28 seconds is a good compromise in the input length and the reached quality. The quality decreases if the possible count of clusters increases. As we found out a maximum value of 5 clusters still produces good results. Lukic et al. [7] reach a MR of 0.1 given 20 speakers / clusters with two samples per speaker. They do 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, [7] 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. Face Clustering

This experiment is done on the FaceScrub [73] dataset. The input of the network is a set of 20 images with 3 color channels and a size of 128×128 pixels. The same CNN like in the speaker clustering experiment is used. This CNN is described in Table V. The first layer in the CNN differs, because the input channel count for the two experiments is different, but the final representation has the same size.

The task of the network is to cluster images with the same person / face. [75] is used to do data augmentation on the given dataset. We use a random distortion with a grid size of 4×4 , 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 424 classes for the training, 53 classes for validation and 53 classes for the final test. After XYZ iterations the early stopping finished the training. The reached metrics on the test set are shown in Table VI.

Hayat et al. [76] solve the face recognition problem on the FaceScrub dataset. They compare different methods and reach average identification rates over 95% on FaceScrub. We do not do directly solve face recognition, but face

Table IV: The networks proposed clusters for a given set of TIMIT audio snippets of the test set. Each row is a proposed cluster by the network and each speaker has another color for his / her snippets. For each snippet the TIMIT speaker name and the audio offset in milliseconds is given.




















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
3	 FSXA0 1569-1697	 FSXA0 1645-1773	 FSXA0 1955-2083	 FSXA0 286-414	 FSXA0 3204-3332

Table V: This table describes a general CNN-based embedding network for images with the shape $128 \times 128 \times X$. It is used for the speaker clustering task and the face clustering task. The used network 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.

Name / Type	Kernel Size	Stride	Padding	Units	Output Shape
input	-	-	-	-	$128 \times 128 \times X$
conv ₀	3×3	1×1	1	32	$128 \times 128 \times 32$
LeakyReLU ₀	-	-	-	-	$128 \times 128 \times 32$
batchNorm ₀	-	-	-	-	$128 \times 128 \times 32$
maxpool ₀	2×2	2×2	0	-	$64 \times 64 \times 32$
conv ₁	3×3	1×1	1	64	$64 \times 64 \times 64$
LeakyReLU ₁	-	-	-	-	$64 \times 64 \times 64$
batchNorm ₁	-	-	-	-	$64 \times 64 \times 64$
maxpool ₁	2×2	2×2	0	-	$32 \times 32 \times 64$
conv ₂	3×3	1×1	1	128	$32 \times 32 \times 128$
LeakyReLU ₂	-	-	-	-	$32 \times 32 \times 128$
batchNorm ₂	-	-	-	-	$32 \times 32 \times 128$
maxpool ₂	2×2	2×2	0	-	$16 \times 16 \times 128$
dense ₀	-	-	-	256	256
LeakyReLU ₃	-	-	-	-	256
batchNorm ₃	-	-	-	-	256
dense ₁	-	-	-	256	256
LeakyReLU ₄	-	-	-	-	256
batchNorm ₄	-	-	-	-	256

clustering. Our approach still can be used to identify faces, but we do not reach as high accuracies, as already can be seen by the reached MR.

VII. CONCLUSIONS AND FUTURE WORK

We are able to show that supervised end-to-end trained deep learning models are able to cluster different data types with a high accuracy. No assumptions about the data and the clustering algorithm have to be done, because an embedding and the algorithm itself are trained by the network. State-

Table VI: Different metric values for the face clustering network. μ describes the mean and σ the standard deviation for 300 conducted clusterings on the test set.

Metric	μ	σ
MR	0.1682	0.1478
BBN _{norm}	0.8307	0.1557
NMI	0.7703	0.2184
CS	0.7864	0.2142

of-the-art solutions may reach higher accuracies for specific tasks [7] [76], but our architecture is much more general and requires much less data specific tuning. Almost any data, given a embedding network, can be clustered by our 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 [8].

Nevertheless, the proposed model may be improved in terms of accuracy and especially in the amount of required data for the training. Our used datasets always contain about 500-600 classes, except for the 2D point data, there we have no classes and just can generate real clusters. We use about 65-80% of the classes for the training. Less 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.

REFERENCES

- [1] 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.
- [2] 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.
- [3] 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.
- [4] M. ROBIN, G. NICOLLE, and A. ROTA, “Automatic sounds clustering approach based on a likelihood measure computation.”
- [5] 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.
- [6] 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.
- [7] 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.
- [8] A. Borji and A. Dundar, “Human-like clustering with deep convolutional neural networks,” 2017.
- [9] 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.
- [10] 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.
- [11] 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.
- [12] 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.
- [13] 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.
- [14] 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.
- [15] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.
- [16] B. Hariharan, J. Malik, and D. Ramanan, “Discriminative decorrelation for clustering and classification,” *Computer Vision—ECCV 2012*, pp. 459–472, 2012.
- [17] J. Xie, R. Girshick, and A. Farhadi, “Unsupervised deep embedding for clustering analysis,” in *International Conference on Machine Learning*, 2016, pp. 478–487.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] 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.
- [22] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [23] 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.

- [24] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1440–1448.
- [25] 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.
- [26] 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.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] J. Paparizos 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.
- [33] D. Müllner, “Modern hierarchical, agglomerative clustering algorithms,” *arXiv preprint arXiv:1109.2378*, 2011.
- [34] 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.
- [35] T. Kurita, “An efficient agglomerative clustering algorithm using a heap,” *Pattern Recognition*, vol. 24, no. 3, pp. 205–209, 1991.
- [36] A. Roy and S. Pokutta, “Hierarchical clustering via spreading metrics,” in *Advances in Neural Information Processing Systems*, 2016, pp. 2316–2324.
- [37] 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.
- [38] 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>
- [39] 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. suppl, pp. S44–S48, 2014.
- [40] O. A. Abbas, “Comparisons between data clustering algorithms,” *International Arab Journal of Information Technology (IAJIT)*, vol. 5, no. 3, 2008.
- [41] 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.
- [42] 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.
- [43] T. Stadelmann, “Voice modeling methods for automatic speaker recognition,” 2010.
- [44] 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.
- [45] 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.
- [46] 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.
- [47] A. Rosenberg and J. Hirschberg, “V-measure: A conditional entropy-based external cluster evaluation measure,” in *EMNLP-CoNLL*, vol. 7, 2007, pp. 410–420.
- [48] 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.
- [49] 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.
- [50] 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>
- [51] 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.
- [52] Y.-C. Hsu and Z. Kira, “Neural network-based clustering using pairwise constraints,” *arXiv preprint arXiv:1511.06321*, 2015.
- [53] Y. LeCun and C. Cortes, “MNIST handwritten digit database,” 2010. [Online]. Available: <http://yann.lecun.com/exdb/mnist/>
- [54] X. Peng, S. Xiao, J. Feng, W.-Y. Yau, and Z. Yi, “Deep subspace clustering with sparsity prior,” in *IJCAI*, 2016, pp. 1925–1931.
- [55] F. Tian, B. Gao, Q. Cui, E. Chen, and T.-Y. Liu, “Learning deep representations for graph clustering,” in *AAAI*, 2014, pp. 1293–1299.
- [56] 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.
- [57] 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.

- [58] 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.
- [59] 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.
- [60] 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.
- [61] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin, “Deep metric learning with angular loss,” *arXiv preprint arXiv:1708.01682*, 2017.
- [62] 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.
- [63] 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.
- [64] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [65] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.
- [66] 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.
- [67] 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.
- [68] K. Janocha and W. M. Czarnecki, “On loss functions for deep neural networks in classification,” *arXiv preprint arXiv:1702.05659*, 2017.
- [69] M. D. Zeiler, “Adadelta: an adaptive learning rate method,” *arXiv preprint arXiv:1212.5701*, 2012.
- [70] Y. Yao, L. Rosasco, and A. Caponnetto, “On early stopping in gradient descent learning,” *Constructive Approximation*, vol. 26, no. 2, pp. 289–315, 2007.
- [71] M. Kotti, V. Moschou, and C. Kotropoulos, “Speaker segmentation and clustering,” *Signal processing*, vol. 88, no. 5, pp. 1091–1124, 2008.
- [72] A. F. McDaid, D. Greene, and N. Hurley, “Normalized mutual information to evaluate overlapping community finding algorithms,” *arXiv preprint arXiv:1110.2515*, 2011.
- [73] 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.
- [74] 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.
- [75] M. D. Bloice, C. Stocker, and A. Holzinger, “Augmentor: An image augmentation library for machine learning,” *arXiv preprint arXiv:1708.04680*, 2017.
- [76] 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.