LEARNING TOPOLOGY PRESERVING EMBEDDINGS FOR SPEEDING UP NEAREST NEIGHBOR RETRIEVAL

by

MASON LARY

THESIS

Submitted in partial fulfillment of the requirements

for the degree of Master of Science in Computer Science at

The University of Texas at Arlington

May 2022,

Arlington, Texas

Supervising Committee

Vassilis Athitsos, Supervising Professor

Chris Conly

Alex Dillhoff

ABSTRACT

LEARNING TOPOLOGY PRESERVING EMBEDDINGS FOR SPEEDING UP NEAREST NEIGHBOR RETRIEVAL

MASON LARY, M.S.

The University of Texas at Arlington, 2022

Supervising Professor: Vassilis Athitsos

Given a database of objects and a query object, it's possible to gather a number of the closest neighbors to the query object. This operation is important to a number of diverse fields such as computer vision, content-based information retrieval, and chemistry. However, distance measures used to determine neighbors can cause queries to be computationally expensive, either because the distance measure is complex or because it is nonmetric and prevents efficient indexing methods.

This work presents novel methods of triplet mining that enable neural networks using triplet loss to learn the manifold that data resides in. These neural networks can learn to embed arbitrary data with arbitrary distance measures between them. Experiments are performed on an offline digit dataset, speech commands, and offline and online sign language data. Results demonstrate effectiveness over a baseline when a network architecture

suited for a particular dataset is trained. When compared to other methods of topology preserving embeddings, the neural network based method outperforms in all but one dataset. Results show there is not a particular method of triplet mining that vastly outperforms the others, and the best method likely depends on the problem being addressed.

ACKNOWLEDGEMENTS

I thank my advisor, Vassilis Athitsos, for the introduction to this problem, as well as his guidance over the past few years. He has helped me grow from zero research experience to the point where I can present original ideas. I would also like to thank professors Chris Conly and Alex Dillhoff for taking their time to serve on my committee.

I would also like to thank professors Pedro Maia and Yuan Peng for allowing me to work on interesting problems in cross departmental research. They've helped me bring an original project from start to finish, and have given me ample opportunity to learn how to present my work to audiences.

Thank you to Vicram and Marcus for keeping me sane with your conversations. You've helped me from developing tunnel vision as I complete this work. I would likely have become burned out without the welcome distractions you both provided. Best of luck in your respective works.

Finally, thank you to my friends and family for supporting me during this process. My mother for her undying support in whatever I strive to do, Hadi, Syed, Kosi, Neil, and Kabir for keeping me from becoming a recluse, and Mageida, for forcing me to stop and smell the roses. This work would not have been possible without the small army of support provided by all these people.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we demonstrate that networks trained with triplet loss are capable of topology preserving encodings when triplets are chosen intelligently. This encoding is capable of speeding up nearest neighbor queries in a variety of situations. This chapter serves as an introduction to nearest neighbors, topology preserving encodings, and some of the applications that can benefit from these encodings. It concludes with the contributions produced by this work.

1.1 Nearest Neighbors Retrieval

K nearest neighbors is a simple algorithm where a query object is presented to a database of other objects, and the k nearest database objects to the query object are returned. These objects can then be used in further processing,

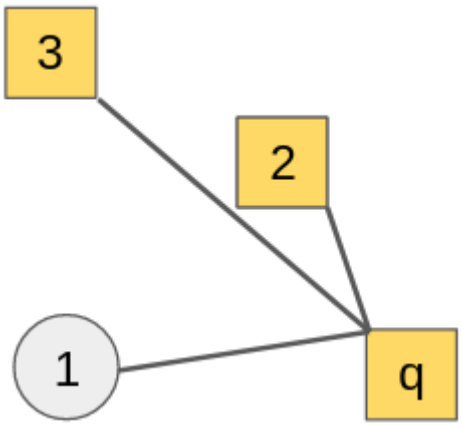

Figure 1.1: An example of the KNN algorithm. When asked for the 2 nearest neighbors of query object q, objects 1 and 2 would be returned if L_2 is used as a distance measure.

such as when trying to classify the object, or can be used as-is in the case of a retrieval problem.

To determine the nearest neighbors, a distance function d must be used. Typically, d is a Minkowski distance such as L_2 between query objects and database objects. However, as seen in Figure 1.2, a Minkowski distance may perform poorly for a particular space. Fortunately, specialized distance functions exist in several domains that can provide superior classification accuracy compared to Euclidean distances [1–4]. Unfortunately, these customized distance functions are generally much less efficient to compute than Minkowski distances. With large datasets of objects, using these distances with query points can be too slow for applications.

In addition, many of these domain specific distances are nonmetric and do not follow the triangle inequality. Unfortunately, many of the data structures

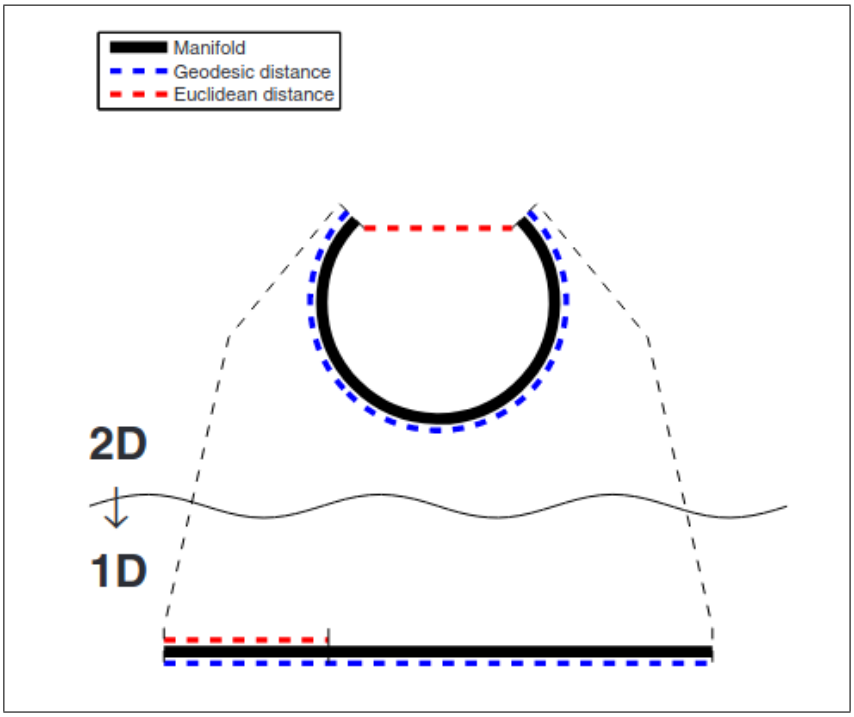

Figure 1.2: A demonstration from [5] of how Euclidean distance measures may be inappropriate in certain spaces. In the C curve, data points at the ends of the manifold would be considered close using Euclidean distance while being far apart if the manifold is followed. However, the geodesic distance is likely more expensive to calculate. By embedding this curve into a 1D shape, the efficient Euclidean distance can be taken in place of the geodesic distance and still provide the same results.

that are used to speed up nearest neighbor searches rely on the triangle inequality holding in the space [6–9]. Without this guarantee, brute-force searches must be performed, further slowing down queries. As represented in Figure 1.2, this work presents a method of training neural networks to embed data points from these nonmetric spaces into lower dimensional metric spaces where efficient retrieval methods can be performed.

1.2 Topology Preservation

An embedding that attempts to speed up nearest neighbor computations is useless unless it maintains the relative distances between objects. In other words, between objects A, B, and C in a space, if A and B are closer than A and C, this relationship should hold in the embedded space. Perfectly preserving the topology of the original space would allow nearest neighbor queries to return the exact same results as in the original space.

It's usually impossible to perfectly preserve these relationships. However, for use with k nearest neighbors, relationships need to be preserved only for k neighbors of an object, meaning only local relationships need to be focused on. The methods of triplet mining presented in this work attempt to focus on these local relationships, ignoring the extraneous global structure of the data.

1.3 Applications

Retrieval problems based on nearest neighbors can be found in many fields, all of which could benefit from this work. This section explores a few of these applications.

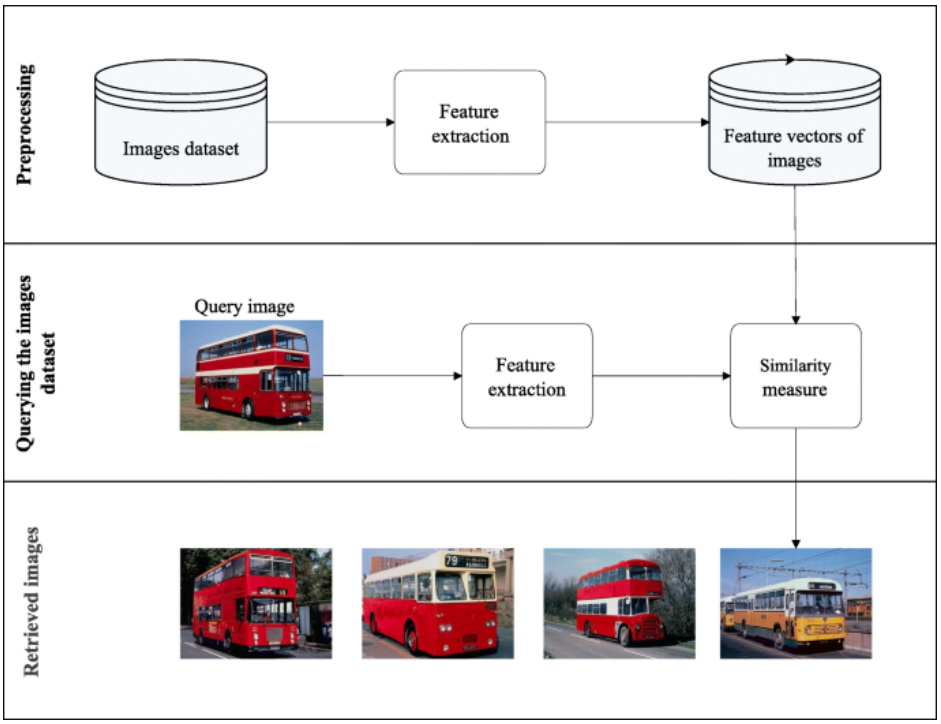

Figure 1.3: A diagram from [10] demonstrating the mechanism of a CBIR system for image retrieval. Images have features extracted and stored in the database. When a query is presented, features are calculated for it and a similarity measure is used to compare it to database feature vectors. The most similar images are returned.

Content-Based Information Retrieval

Content-based information retrieval, or CBIR, refers to a number of systems that are able to retrieve objects from a dataset using the content of a query. Perhaps the most well known of these systems are reverse image services, although similar systems exist for audio [11, 12] and document lookup [13].

For image lookup systems, we find several algorithms at play. They generally act by extracting features from images and using them in a distance

function between images, as seen in 1.3. For instance, the QBIC system [14] uses color, shape, and texture features to compare images. WebSeek [15] is another of these systems that includes color and wavelet transform features for comparison. Besides being useful for web searches, these systems can be employed for more serious instances, such as medical diagnoses and adult content detection [16].

Document retrieval systems also exist. As mentioned in [17, 18], signature files or inverted index files act as compressed vectors representing entire documents, allowing for quick queries of a document database using a distance function. It's even possible to store and retrieve audio this way, using features such as loudness, bandwidth, and pitch [19, 20].

Chemoinformatics

The structure of a molecule helps determine its properties. Therefore, when designing drugs or determining the properties of a new molecule, knowing the properties of similar molecules is incredibly useful. Databases of molecular structure have been built up, and queries can be made to determine the closest structures. Efficient distance measure design is therefore important as the size of these databases grow [21, 22].

Robotics

Robotic systems can employ nearest neighbor-based systems for a number of tasks. For instance, in human-robot interaction [23], the exploration of complex environments [24], or even approximating inverse kinematics [25].

For all of these systems, speeding up the nearest neighbor search while retaining accuracy is a prime goal. Specialized distance functions are created that work on object representations to find nearest neighbors when given a query object. The work presented here can obviously have an impact in these fields, as it attempts to solve the same problem, but in a general way. Instead of defining specific methods for the data, a generic method can be applied to new data types to reduce their dimensions and speed up database retrieval.

1.4 Thesis Contributions

As has been shown, methods for speeding up nearest neighbor search are heavily researched in both academic and industrial applications. Accurate distance functions are effectively useless in a world of large datasets if they are not efficient. In order to reach the speed required by a production system, either distance functions need to be simpler or need to be metric, which may come at the cost of decreased accuracy.

The work presented here demonstrates Siamese neural networks trained using triplet loss. While this is not a novel approach, a large part of training

these networks comes from choosing training examples. This work presents novel methods for triplet mining with the express purpose of speeding up these nearest neighbor queries by weighting objects based on their distance from the anchor point.

These methods are agnostic to the type of the network and the type of data, as long as it can be passed into a network. This allows for a general solution to this problem rather than specialized solutions for particular applications, providing a significant speedup to queries in various domains.

Chapter 2

Background

This chapter introduces some background information needed while going forward in this thesis. Section 2.1 introduces the problem formally, along with definitions and notation. Section 2.2 introduces some of the distance functions that would be considered complex. Section 2.3 concludes the chapter with an explanation of the filter and refine method for computing nearest neighbors from embedded points.

2.1 Formal Problem Definition

We are given a space of objects $\mathcal{A}$ with a distance measure D between objects in the space. A database of objects $\mathcal{O}$ is drawn from this space. To find the k nearest neighbors of an object x $\in \mathcal{A}$ - $\mathcal{O}$, we can compute D(x, y) for $\forall y \in \mathcal{O}$ and retrieve the k objects with the lowest distances.

Unfortunately, D may be an expensive distance function, and may therefore be prohibitive in a realistic application. What we require is a mapping F: $\mathcal{A} \rightarrow \mathcal{R}^d$ which can map an object into a real d-dimensional space. From here, an efficient Minkowski distance such as L_2 can be applied to the mapped objects to obtain a distance. This new distance function is represented by θ. In other words, to obtain a distance in this new space, we can compute $\theta(F(x), F(y))$.

For this mapping to be useful, it should produce embedded points that are topologically equivalent to the original space. Topological equivalence is defined as follows: For all x, y, z $\in \mathcal{A}$

$$\theta(F(x), F(y)) < \theta(F(x), F(z)) \leftrightarrow D(x, y) < D(x, z)$$

Unfortunately, perfectly preserving the topology of a higher dimensional space through a lower dimensional mapping is incredibly difficult. Our goal then is to create a mapping that tries to preserve as many of the relationships between points as possible.

2.2 Computationally Expensive Distances

Throughout the introduction of this thesis, allusions have been made about computationally expensive distance functions without a definition of what they are. In this section, a few of these functions are presented.

The concept of computationally expensive distance functions is subjective based on the system being implemented. For the purposes of this thesis, the view taken in [26] that distances that are superlinear to compute in the dimension of the objects are considered expensive. Since Minkowski distances are linear to compute in the size of the vector, they are considered computationally efficient.

Chamfer Distance

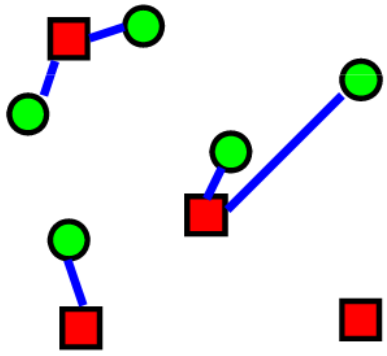

Figure 2.1: An example of the directed chamfer distance, where the distance from each circle object to its nearest square neighbor is taken.

The chamfer distance is a distance measure between point clouds. Given point clouds A and B, the directed chamfer distance is calculated as

$$D(A,B) = \frac{\sum_{a \in A} \min_{b \in B} ||b - a||_2}{|A|}$$

In other words, distances are calculated as the average distance from a

Figure 2.2: By converting images into binary images, chamfer distance can be taken between two images. Here, the two stars will obviously show a lower distance than the star and moon image

point in cloud 1 to its nearest point in cloud 2 using Euclidean distance, as can be seen in Figure 2.1. The direct form of chamfer distance is obviously not symmetrical, but an undirected form can be created as

$$D_{sym}(A, B) = \frac{D(A, B) + D(B, A)}{2}$$

By converting images into binary images, they effectively act as point clouds. They can then be compared with each other using chamfer distance, a technique seen in Figure 2.2. This is a useful measure in problems such as image matching [27, 28].

As mentioned in [26], chamfer distance is O(dlog d) where d is the number of possible white pixels in an image. Finding the closest white pixel takes O(log d) using a 2D binary search, and this process is done for O(d) pixels. This makes chamfer distance superlinear and therefore inefficient for nearest neighbors.

Also mentioned in [26], chamfer distance can be calculated in O(d) time if the distance transform is calculated for each image. However, this would

require the storage of a distance transform for each database image. This would effectively double the amount of storage needed for the database, which is less than ideal.

Dynamic Time Warping (DTW)

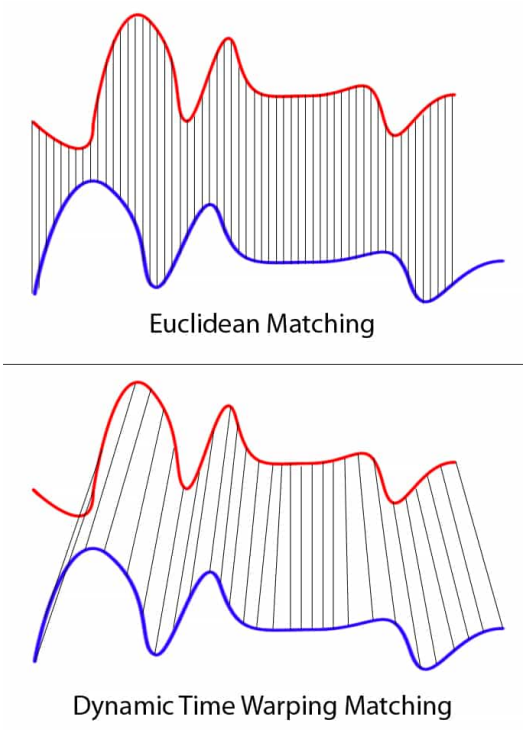

Figure 2.3: Demonstrates an example of dynamic time warping and how it maps points between two time series, even if they are on different temporal scales.

Dynamic time warping is a distance measure employed on time series that may have different lengths from each other. The approach is based on dynamic programming and aligns the two time series in such a way that they

minimize a cost function.

Dynamic time warping and variants such as dynamic time-space warping have been applied in many situations involving time series, such as speech recognition and handwriting recognition [29–32].

In a naive form, DTW is O(NM) where N and M are lengths of sequences. Recently, [33] showed that O($N^2/loglog(N)$) time and space can be achieved for two inputs of length N, but this still doesn't meet the criteria laid out in the beginning of the section for an efficient distance measure. As described in [34], DTW can also be generalized to time series with multiple dimensions.

Edit Distance

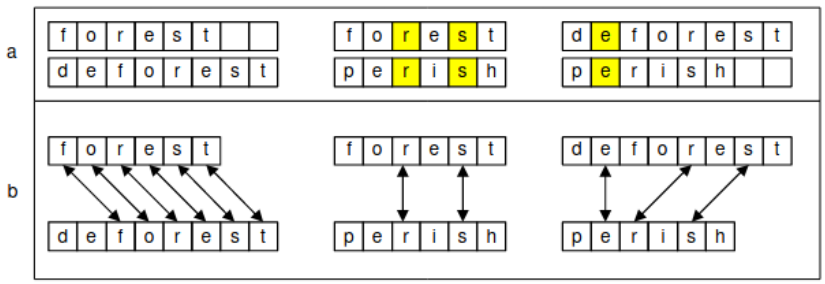

Figure 2.4: A visualization of the Levenshtein edit distance. This method uses dynamic programming to find the alignment between two strings and looking at the number of transformations to turn one string into another.

Various distance measures known as an edit distance can be defined on the space of strings. These measures quantify dissimilarity between strings by looking at the number of transforms to bring one string to the other. The various measures differ in the types of operations they allow between the

strings.

For instance, the Levenshtein distance [35], the most common edit distance, uses the number of deletions, insertions, and substitutions between two strings to compute a distance. Using these properties, dynamic programming can be used to align the strings and find the distance value. The dynamic programming algorithm has a runtime of O(mn) between strings of length m and n. A more advanced edit distance called Smith-Waterson can be employed to align sequences of DNA, which also uses an O(mn) dynamic programming algorithm.

2.3 Filter and Refine

At retrieval time, we would normally embed the query object and find however many nearest neighbors are required by the application in the embedded space. However, because our embedding process is likely far from perfect, these neighbors are not likely to be the true neighbors of the query point.

However, if our embedding has been trained with fairly high accuracy, then the nearest neighbors from the original space are likely still close to the query. In that case, we can perform a filter and refine step to retrieve the nearest neighbors to our query object

In filter and refine, we define a number of neighbors, p, in the embedding space that we want to consider. This number p is greater than the number of nearest neighbors k we would like to retrieve from the database. In the filter

step, we compute the distance between the query object and all database objects using some efficient Minkowski distance. In the refine step, we utilize the more expensive, but likely more accurate, distance function to recalculate the order of the p nearest neighbors. From these p objects, we can retrieve the k nearest neighbors.

A filter and refine strategy allows for an approximation of the nearest neighbors, skipping the need to calculate the expensive distance function for each database object. If the embedding performs well, this approximation may not be far off from the ground truth nearest neighbors. This strategy is quite common in nearest neighbor retrieval using dimensionality reduction [3, 26, 36].

Chapter 3

Related Work

Topology preserving encodings are useful for numerous purposes. Therefore, there is much literature focusing on them. Presented is a small sample of the works focused on topology preserving embeddings for use in dimensionality reduction, data visualization, and the speedup of k nearest neighbor queries. A survey of many of these methods can be found in [3, 5].

3.1 Topology Preserving Reductions

Nonlinear reduction is a broad topic consisting of methods that can reduce the amount of dimensions in data that follow nonlinear relationships. Of particular importance to this work are methods that preserve the topology of the original space, although other reduction techniques exist, such as those that attempt to preserve distance between objects, a much stricter restriction

than topology preservation. A reference for these methods as well as others not focused on topology can be found at [5].

Many of these methods utilize a lattice to in order to describe the manifold that the data exists on. A lattice is essentially a graph, where points in the space are vertices and weighted edges represent distances between the points. These methods can be categorized based on whether the lattice is fixed or dynamic during the reduction process.

Self-Organizing Maps

Self-Organizing maps are a method of dimensionality reduction based on the concept of vector quantization [5, 37]. Vector quantization is the representation of data points using a smaller number of prototype points. As a simple example, k means represents a cluster using a centroid point.

An SOM starts out as a P dimensional lattice, where P is the dimension of the embedding space. However, this lattice also has coordinates in the data space. The goal of the learning algorithm is to learn these data space coordinates.

In simple terms, points in the lattice are moved toward data points, similar to the centroids moving in k means. However, rather than each lattice point moving independently, neighboring lattice points are moved at the same time. To determine the lower dimensional coordinates of a point in the data, a nearest neighboring lattice point is chosen to represent it. Since the lattice point has a lower dimensional coordinate, this can be used as the embedded

coordinates for the point. Because neighboring points in the lattice stay together as the lattice is morphed to the data, similar points in the data get mapped to similar points in the lattice.

Self-Organizing maps have found success in data visualization and time series forecasting [38, 39]. In their base form, they are algorithmically simple to implement and can be quite robust when used for visualizations. However, being a vector quantization method, data is not actually embedded in the lower dimensional space. In addition, lattices above 2 dimensions are very tough to learn, limiting the ability of this method for creating higher dimensional embeddings.

Generative Topographic Mappings

Proposed in [40], GTMs are essentially a probabilistic SOM. Points on the lower dimensional manifold can be sampled and transformed through a nonlinear function to produce a point in the data space. Noise is added to the point, turning the model into a mixture of Gaussians that can then be trained through EM.

GTMs incur the same disadvantages as an SOM. Higher dimensional latent spaces can not be used for the manifold, meaning embeddings of arbitrary dimensions are not possible.

While both of these methods have been used successfully in several applications, neither can be used for embedding in arbitrary dimensions. The work presented in this paper rectifies this issue and allows a user defined dimension to map points to.

3.2 Data Visualization

t-SNE

t-SNE, or t-Distributed Stochastic Neighbor Embedding [41], is a popular method for high dimensional data visualization. It attempts to view the global structure of a dataset in a lower dimensional space by creating a lower dimensional map with a similar distribution of points to the original space. This is accomplished by using the distances between objects to generate probabilities of points picking each other as neighbors. By minimizing the KL divergence between the probability distributions in each space, similar relative distances between objects are created in the lower space.

However, t-SNE suffers from a few issues that this paper attempts to rectify. For instance, the main parameter for performing t-SNE mappings is perplexity, which can be thought of as a continuous number of nearest neighbors. This parameter is infamously confusing with works attempting to remove it from the equation [42]. This issue is further compounded by

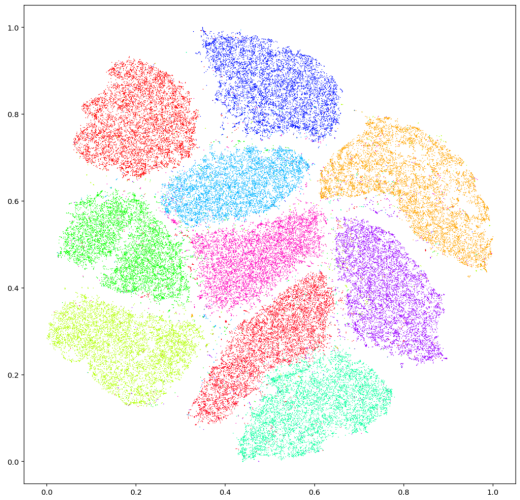

Figure 3.1: A visualization in 2 dimensions of the MNIST dataset using t-SNE. Notice the clusters that form between items in the same class.

the fact that t-SNE is not deterministic, so viewing changes caused by this parameter can be tricky.

Unfortunately, as a tool for pure visualization, t-SNE can not embed a test set, preventing it from being used in the speedup of nearest neighbor queries. In addition, since t-SNE is only useful for visualization, only low dimensional maps can be used by it.

It should be noted that parametric versions of t-SNE have been created which allow the embedding of new data points [43]. However, this work has looked at the ability of the mapping to preserve classification accuracy, a slightly less restrictive problem than preserving retrieval accuracy.

UMAP

UMAP, or Uniform Manifold Approximation and Projection [44], is another dimensionality reduction technique that is generally used for data visualizations. Using principles of Riemannian geometry and algebraic topology, UMAP is able to embed a manifold into an arbitrary dimension.

What sets UMAP apart from t-SNE is its ability to embed new objects into the learned space. This makes it useful as both a data visualization technique and dimensionality reduction technique.

3.3 Efficient Nearest Neighbor Embeddings

The other methods in this section perform some form of dimensionality reduction that attempts to preserve relative distances between objects. However, not all of these methods are built to embed new data points or result in a space in which Minkowski distances can be taken. The following methods attempt to perform both of these tasks.

SparseMap

SparseMap, discussed in [45], is a variant of a method of embedding known as Lipschitz embeddings. The idea behind Lipschitz embeddings is to define a coordinate space using subsets of elements in the dataset. A distance function can be defined between objects and these reference sets. These distances can then act as coordinates in this coordinate system. SparseMap

presents heuristics that allow for efficient Lipschitz calculations.

MetricMap

MetricMap, discussed in [46], projects data from the original space into a pseudo-Euclidean space, meaning some coordinate axes produced contribute negatively to "distances" between objects [3]. Projections of the data are performed using pairs of reference objects, where the number of pairs used determines the dimension of the embedded space.

FastMap

FastMap is described in [47] and is one of the methods of comparison used in experiments. FastMap is similar to MetricMap in that each dimension of the embedding is generated from a pair of objects in the space. In FastMap, these projects are into a Euclidean space, rather than a pseudo-Euclidean space. Unfortunately, the base form of FastMap can not handle nonmetric distances and must be altered to embed nonmetric spaces [3].

BoostMap

BoostMap has several versions described in multiple papers [26, 48, 49] and is the method most heavily focused on for comparisons. The version from 2004 has been chosen as a comparison with the method described in this work.

The idea behind BoostMap is that embeddings can be seen as equivalent to combinations of weak classifiers. SparseMap, MetricMap, and FastMap use geometric principles to find embeddings, but the view taken by BoostMap allows for machine learning techniques to be applied to learn an embedding. For each dimension, embeddings are defined as either the distance from a reference object, or the projection onto the line between two reference objects as seen in FastMap. These simple embeddings can be used on triplets of objects (q, a, b) to act as weak classifiers, determining if a or b is closer to q.

The objects that act as reference objects belong to $\mathcal{C}$ and the objects that can be used for triplets belong to $\mathcal{T}$. The number of triplets that can be created is also capped to a user specified value to control the runtime of the algorithm.

Finding the best combination and weights of a group of weak classifiers can be done using AdaBoost. At the end of training, the final embedding can be found by applying each learned 1-dimensional embedding to the data. BoostMap has been shown to perform well across under a variety of situations [26, 48–50].

Chapter 4

Triplet Mining Methods

This chapter presents the techniques employed and invented for this work. Section 4.1 explains triplet loss and its importance in topology preservation. Section 4.2 covers the triplet mining technique used by our networks.

4.1 Triplet Loss

When training neural networks, the loss function is of utmost importance. It determines what the network learns by penalizing outputs that don't conform to the expected. For instance, a neural network set to classify objects can have a cross entropy loss function, which penalizes outputs which classify the object as something different from its actual label.

These neural networks can embed objects into any dimension. However, to obtain topology preservation, we need a loss function that penalizes en-

codings that have different topologies in the embedded space. This is the goal of triplet loss.

Triplet loss is based on the notion of an anchor object, positive object, and negative object. An anchor object is any object that we would like to use as a reference point. The positive object is another object, separate from the anchor, that should be embedded closer to the anchor than the negative object. The negative object is separate from the anchor and the positive object, and should be farther from the anchor object than the positive object.

Our neural network can be seen as an encoding $F : X \to R^d$, where $X \in \mathcal{A}$, the space of all objects, and d is the dimension of the space we'd like to encode it into. Given anchor $x \in \mathcal{A}$, positive object $y \neq x$, and negative object z where $z \neq x$, $z \neq y$, and D(x, y) < D(x, z), we can define triplet loss for a single triple as

$$L(x, y, z) = max\left\{0, ||F(x) - F(y)||_2 - ||F(x) - F(z)||_2 + m\right\}$$

m represents the margin of the loss function, which makes sure the encodings of positive and negative examples are not too close to each other in the embedded space. Given a batch of triplets, we can average the loss for each triplet to form a total loss for the batch. Also note that we don't have to choose L_2 as our metric in the function. Euclidean distance was chosen for its speed in the embedded space.

As stated in [51], triplet loss works well when we want semantically similar

points in the original space to map to close points in R^d and semantically different points in the original space to map to distant points in R^d. However, we don't just want similar points to be close together, but to have their relative distances to other points be the same. Therefore, we have to be careful about what triplets we give to the loss function.

4.2 Triplet Mining

As our dataset grows, the number of triplets that can be formed grows cubically. The number of valid triplets to choose from is a subset of all possible triples, but the size of this set makes it infeasible to train with. Triplet mining describes the technique employed in generating triplets with which to train our networks.

When mining triplets, we need to take care to not get triplets that are too easy or too hard for the network. As shown in [51], using triplets that are of a low quality can extend training times or even cause networks to not learn the necessary mappings for a majority of the data. Therefore, we will need a definition of important triplets.

To further limit the number of triplets that need to be considered, a note should be made that not all points around the anchor are of the same importance. In other words, if we expect queries to look for n neighbors, then we can focus on perfecting the embedding for n neighbors of each anchor. This saves us from spending too much time training the network to preserve

the topology for points that are too far apart and whose distances cause little change in nearest neighbor retrieval. A similar stipulation was made in [26] as a means of focusing on important training triplets.

It should also be noted that the process of generating a positive object and a negative object are separated from each other. This means that a combination of the positive generating techniques and negative generating techniques is possible. For all the examples given, each training object in a batch is used as the anchor object of a single triple.

Random Positive

The simplest method of generating triplets is randomly. Using every training object as an anchor, the n nearest neighbors can be calculated using the original distance measure. From these nearest neighbors, a random neighbor can be chosen as a positive object using equal weighting for each object.

Since the nearest neighbors have been calculated for the anchor object, this method assumes that each nearest neighbor object is equally important to focus on.

Random Negative

By choosing a positive object, we have chosen the set of objects we may choose as a negative object. Any object that is farther from the anchor object in the original space than the positive object can be chosen as the

negative object with equal weighting.

However, choosing an object at random, we run the risk of choosing an object that is embedded farther from the anchor than the positive object by at least the margin distance. This means that the loss for this triplet is 0, and it would not contribute to the overall learning for the epoch. We ideally want to pick an object that has been embedded closer to the anchor object than the positive object, a mistaken embedding, in order to learn the most from an example. However, if there are no embedded objects that fit this description, we can't choose a negative object. For simplicity, we stick to choosing one at random, despite the possible inefficiency.

Gaussian Positive

Choosing a positive object uniformly may not be the optimal choice as it may allow the learning of topology far from the anchor point to take place rather than a closer object. For instance, if n is 50, choosing to calculate loss using the 50th nearest neighbor to an anchor point may not be as helpful as using the first 10 nearest neighbors.

We therefore want to weight our positive pick so that closer objects are picked more often than further ones. This can be accomplished using a technique from [41]. We can define $p_{j|i}$ as the probability that object x_i would choose object x_j as its neighbor, assuming that neighbors are picked according to their probability density under a Gaussian centered at object

x_i. Mathematically, the density of object x_j given x_i is

$$f_{j|i} = e^{\frac{-D(x_i,x_j)^2}{2\sigma_i^2}}$$

The probability of choosing j as a neighbor given i as an anchor is then

$$p_{j|i} = \frac{f_{j|i}}{\sum_{j \neq i} f_{j|i}}$$

An object closer to the anchor has a higher chance of being chosen as the positive object, while objects that are too far away have a close to 0 chance of being chosen. Of course, this raises the question of the value of σ. In choosing our positive object, we would still like to focus on the k nearest objects to our anchor point. Assuming the distance between anchor x_i and its kth nearest neighbor is 3σ from the center, we can easily calculate sigma. This prioritizes choosing objects within the k nearest neighbors while giving more attention to objects that should be closer to the anchor point.

Gaussian Negative

Of course, if we can choose our positive object using a more complicated weighting scheme, we can do the same with our negative object. Once we've chosen our positive object, the set of objects from which a negative object can be drawn has also been fixed. As discussed in uniform choice, we only want to consider objects that have been mistakenly embedded closer to the

anchor than the positive object.

To do this, we need to carefully choose our σ value. The distance from the anchor point to the positive point in the embedded space is chosen to be 3σ. This heavily weights these points with incorrect embeddings to be chosen, while still allowing correctly embedded points to be chosen in the case that no negative points are encoded closer to the anchor point. This alleviates the issue of never being able to choose a negative object.

Closest Negative

The Gaussian weighting of negative objects leads to a simpler idea. When choosing a negative object, we can choose the negative closest to the anchor point. This object would have the highest probability in the Gaussian selection process, so this may serve as a faster alternative when selecting negative objects.

Similar to Gaussian weighting, this would focus on the most incorrect negative objects, hopefully learning to embed these objects quickly.

Chapter 5

Experiments

In order to demonstrate the efficacy of these techniques, experiments are performed on several datasets and distance functions. This section describes the datasets chosen as well as their associated distance measures. It then goes on to describe the setup of the experiments, including parameters for different models and training phases. Finally, results of experiments are shown.

5.1 Datasets

Australian Sign Language Dataset (Tctodd)

The Tctodd dataset was created using 95 Australian signs for work found in [52]. A single signer produced 27 examples of each sign while wearing gloves built for the tracking of signing.

Recording was done for both hands, with 11 data points per hand. These data points were

- x, y, z position in meters
- Roll, pitch, yaw
- Bend angle of each finger

22 values were gathered for the duration of the sign, producing for each sign a 22 dimension time series. These time series were filled with 0 values in order to match the length of the longest sequence. The distance between each time series was taken using multi-dimensional dynamic time warping [34].

1795 of the time series were taken as a training set. The other 770 were chosen as a test set, giving a 70/30 split of the data.

Using 3-NN, classification accuracy is 20.3% versus a random chance guess of 1.05%. The distance between a query object and all database objects can be calculated in an average of .097 seconds.

MNIST

The second dataset is the very well known MNIST dataset. 70000 28x28 images are given representing handwritten digits.

A random subset of 15000 images were used for training and 5000 for testing. This gives this experiment a 75/25 split between training and testing.

The chamfer distance was chosen for this dataset. Edge images were created using a Canny edge detector. Using 3-NN, classification accuracy is 93.3% versus a random chance guess of 10%. The distance between a query object and all database objects can be calculated in an average of 4.435 seconds.

Speech Commands

The third dataset is from [53] and contains 105,829 audio samples of 35 words, spoken by 2618 speakers. Each sample is approximately 1 second long at 16 KHz.

To preprocess, samples were resampled to 1 KHz and filled with 0 values to have the same length. A subset of the data was also taken to reduce runtime. 8484 samples were drawn to represent the training set, and 1100 samples were drawn to represent the test set.

Using 3-NN, classification accuracy is 9.5% versus a random chance guess of 2.9%. The distance between a query object and all database objects can be calculated in an average of 39.572 seconds.

ASL Finger Spelling (ASL5)

The final dataset comes from [54]. It consists of 48000 images from 5 signers signing each letter of the English alphabet, except for 'j' and 'z' due to the need for motion data. From videos of these signs, frames containing the signs

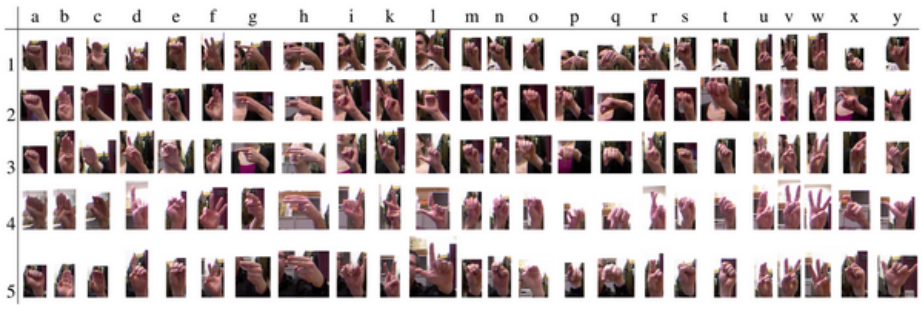

Figure 5.1: An overview of the ASL5 dataset. As can be seen, 5 signers signed the letters of the alphabet except for 'j' and 'z'.

are taken.

A subset of 16000 and 6000 images were randomly chosen for training and test sets, respectively. Preprocessing consisted of resizing each image to 80 by 80 pixels. The chamfer distance was chosen as a distance measure between images. Using 3-NN, classification accuracy is 16.3% versus a random chance guess of 4.2%. The distance between a query object and all database objects can be calculated in an average of 16.62 seconds.

The distance measures chosen for each dataset are capable of performing better than random chance in a k nearest neighbors classification. While the accuracies are not particularly high, the goal of this work is not to increase classification accuracy, but to embed data points in a topologically similar space. Therefore, classification accuracy should not be the main focus of attention when viewing experiments.

5.2 Network Architecture

While the focus of this work is on the mining of triplets and the use of triplet loss in networks, the architecture used in experiments should be mentioned. Triplet loss and triplet mining is agnostic of the type of the network, but 2 different architectures are presented depending on the type of data.

For experiments using the ASL5, MNIST, and Speech Commands dataset, a simple feed forward network is used. The design of the architecture was inspired by [55] due to its success in dimensionality reduction. It consists of 4 linear layers with 200 nodes, 100 nodes, 50 nodes, and d nodes, where d is the dimension of the embedding, respectively. Each layer except the last has a ReLU activation function, while the last contains a sigmoid activation function, which places embeddings on an n-dimensional hypercube to prevent large distances in the embedded space.

Time series data has a different format from regular data vectors, so a different network architecture is needed to work with it. A network based on [56] is used, which is a deep architecture for working with raw audio signals. Since the Speech Commands dataset consists of raw audio signals, another experiment is performed on this dataset using this architecture to compare how the network design affects the results of the embedding. This network is also used for the Tctodd dataset as it consists of time series data.

The network contains a 1-dimensional convolutional filter, increasing channels to 256, followed by a batch normalization and max pool layer. This is

followed by another 1-dimensional convolutional filter, batch normalization, and max pool layer that results in a d channel output, where d is the dimension of the embedding space. An average pooling layer is used to reduce the size of the output vectors. A sigmoid activation layer is applied at the last step of the network.

5.3 Retrieval Experiments

To compare the efficacy of these neural networks, BoostMap and FastMap models are created for each dataset and distance measure. The specific version of BoostMap used comes from [48], which requires a set $\mathcal{C}$ of candidate objects, a set $\mathcal{T}$ of training objects, and a number of objects used in validation calculations. In addition, a number of random triples to draw is also specified for each experiment. Explanations of these parameters can be found in Section 3.3. For FastMap, only a dimension of the embedded space has to be specified. For all experiments, a FastMap model was trained to embed data into a 256 dimension space. BoostMap models were built to embed into a 128 dimension space, but if validation loss was unchanging over a long period of time, the process was stopped and the dimension taken to be whatever the training process stopped at. For neural networks, for each dataset, 7 networks of dimensions 1, 2, 4, 8, 16, 32, 64, and 128 are trained. Each network is trained for 40 epochs with a k nearest neighbors parameter of 10. Fully connected networks are trained with a learning rate of .001 using the

Adagrad optimizer while convolutional networks are trained with a learning rate of .01 using the Adam optimizer.

As mentioned in section 4.2, we have 2 ways of picking positive objects and 3 ways of for negative objects. In the following figures, the different methods are represented with the following identifiers.

GC Positive items are chosen using Gaussian weighting. Negative objects are chosen as the closest negative object to the anchor.

GG Positive items are chosen using Gaussian weighting. Negative objects are chosen using Gaussian weighting.

GR Positive items are chosen using Gaussian weighting. Negative objects are chosen randomly.

RC Positive objects are chosen randomly. Negative objects are chosen as the closest negative object to the anchor.

RG Positive objects are chosen randomly. Negative objects are chosen using Gaussian weighting.

RR Positive objects are chosen randomly. Negative objects are chosen randomly.

In addition, networks are trained with completely random triplets as a baseline to demonstrate the effectiveness of proper triplet mining.

To measure how well the system performs, metrics from [26] are used. A set of test vectors can be embedded in each dimension that a model exists for. Of course, the test vectors have ground truth nearest neighbors in the original space and different nearest neighbors in the embedded space. We can define a percentage p and a number of nearest neighbors k. Given k, we can report the minimum number of exact distance calculations across all dimensions that are needed in a filter-and-refine approach to correctly retrieve the k nearest neighbors for p percent of test vectors. For all experiments, we report results for 90%, 95%, and 99%.

MNIST

To train BoostMap, 7000 instances were used for $\mathcal{C}$ and $\mathcal{T}$. The remaining 1000 objects were used as a validation set. The training process created a 47 dimensional embedding before validation loss stopped changing, well below the requested 128 dimensions.

Looking at Figure 5.2, we see that triplet mining based methods outperform both BoostMap and FastMap by a large margin. The performance of BoostMap is quite surprising. As shown in [26], BoostMap is capable of performing well when approximating chamfer distance and has a tendency to outperform FastMap. This is true for advanced versions of BoostMap [50] and earlier iterations [49].

It should be noted that when training BoostMap on the MNIST dataset, [49] uses a distance measure called shape context matching rather than the

chamfer distance. The difference in distance measures may be causing these results for BoostMap. The maximum dimension of the embedding is also concerning. It's likely a greater amount of parameter tuning is needed for BoostMap to achieve its greatest results.

Looking at the methods relying on triplet mining, we find that GR method has outperformed other methods, especially at 90% and 95% accuracy. Surprisingly, as the accuracy level increases, the differences in speedup between RR and GR become more marginal, as shown in Table 5.1. It can also be seen that each data mining method outperformed the baseline method using completely random triplets.

It should also be noted here that except for the 99% accuracy level, the Gaussian positive methods outperform the corresponding random methods, demonstrating the slight superiority of the Gaussian weighting on positive objects, at least for this dataset. In addition, it can be seen as the number of neighbors being retrieved increases, the margins between the methods decreases, meaning the GR method sees most of its benefit at lower k values.

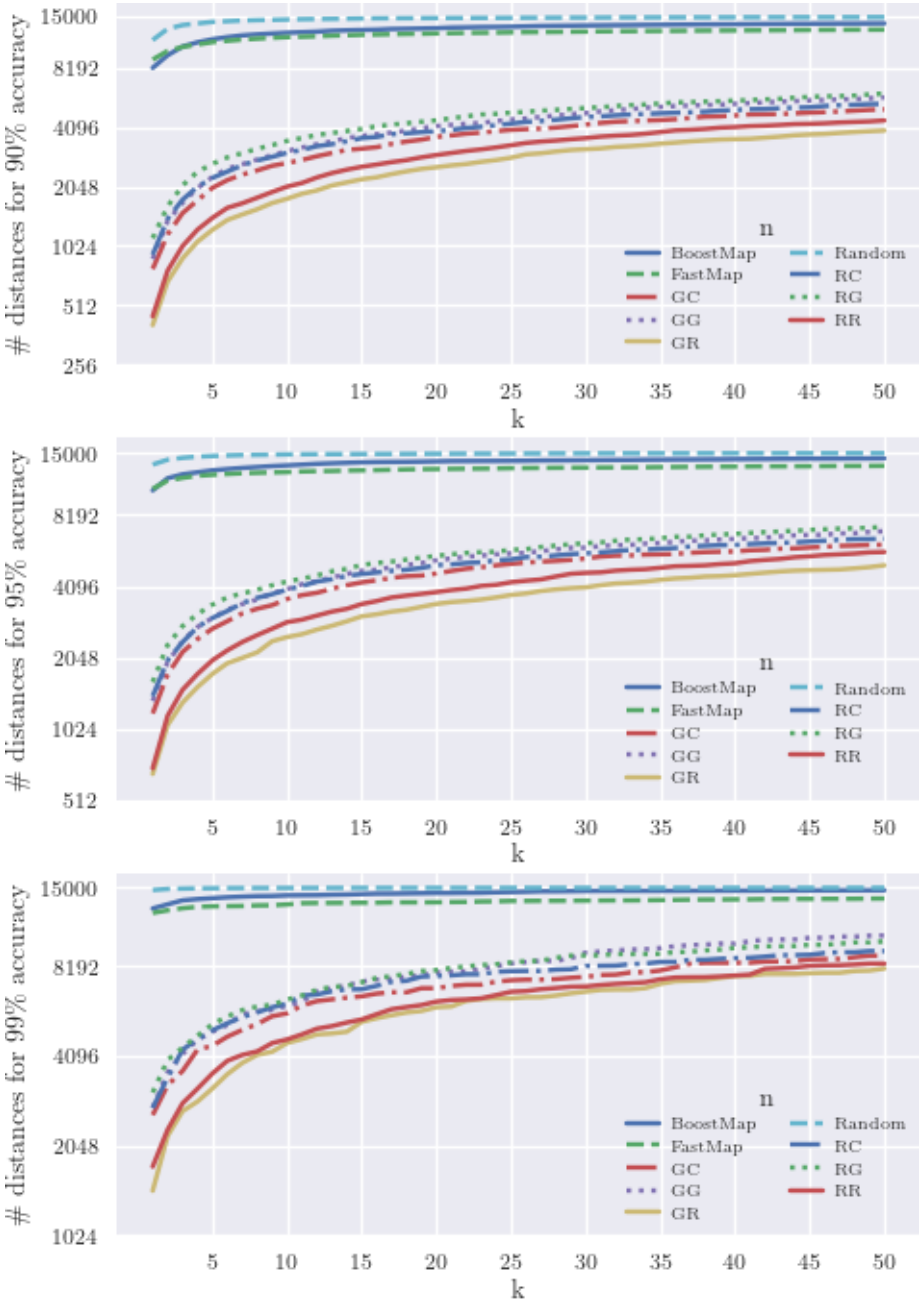

Figure 5.2: Results on the MNIST dataset using chamfer distance for comparisons. 5000 test objects were used.

MNIST Dataset with Chamfer Distance										
		90%			95%			99%		
		k=1	k=10	k=50	k=1	k=10	k=50	k=1	k=10	k=50
R	# Distances	11336	14403	14815	13381	14758	14939	14726	14963	14990
	Speedup	1.32	1.04	1.01	1.12	1.02	1.00	1.02	1.00	1.00
BM	# Distances	8190	12342	13775	10404	13268	14208	12771	14135	14699
	Speedup	1.83	1.22	1.09	1.44	1.13	1.06	1.17	1.06	1.02
FM	# Distances	9026	11725	12799	10568	12428	13217	12343	13172	13788
	Speedup	1.66	1.28	1.17	1.42	1.21	1.13	1.22	1.14	1.09
GC	# Distances	782	2696	5042	1188	3614	6132	2594	5685	8874
	Speedup	19.18	5.56	2.98	12.63	4.15	2.45	5.78	2.64	1.69
GG	# Distances	878	3099	5798	1341	3980	6993	2741	6023	10361
	Speedup	17.08	4.84	2.59	11.19	3.77	2.15	5.47	2.49	1.45
GR	# Distances	**406**	**1776**	**3940**	**660**	**2498**	**5022**	**1446**	**4523**	**8034**
	Speedup	**36.95**	**8.45**	**3.81**	**22.73**	**6.00**	**2.99**	**10.37**	**3.32**	**1.87**
RC	# Distances	925	3037	5381	1406	3962	6506	2740	6121	9191
	Speedup	16.22	4.94	2.79	10.67	3.79	2.31	5.47	2.45	1.63
RG	# Distances	1120	3489	6066	1612	4290	7291	3087	6321	9908
	Speedup	13.39	4.30	2.47	9.31	3.50	2.06	4.86	2.37	1.51
RR	# Distances	450	2046	4432	697	2888	5714	1742	4648	8346
	Speedup	33.33	7.33	3.38	21.52	5.19	2.63	8.61	3.23	1.80

Table 5.1: A table summarizing the number of exact distances needed by each method to return the k nearest neighbors with 90%, 95%, and 99% accuracy. Also includes the speedup from brute force search in the original space, which takes 4.435 s on 15000 database objects. The best method for a particular percent and k is bolded. BoostMap and FastMap have been abbreviated to BM and FM, respectively. R represents the baseline network trained with completely random triplets.

ASL5

BoostMap was trained with 7500 objects $\in \mathcal{C}$ and 7500 objects $\in \mathcal{T}$ with the remaining 1000 objects used for verification. The embedding that was learned consists of 109 dimensions.

The same pattern as the MNIST dataset emerges in Figure 5.3, with BoostMap and FastMap performing fairly poorly at learning this chamfer distance. We also find that the GR method of data mining is once again the best performing, but only by a small margin when compared to methods like RG and GG. As shown in Table 5.2, these other methods have very similar speedups, and therefore using the GR method may provide only marginal benefits over other methods, at least when learning the chamfer distance. However, all three methods GR, GG, and RG were able to beat the baseline at different accuracy level, demonstrating improvement over random triplet mining despite there not being a visibly superior method.

Unlike results from the MNIST dataset, it can be seen in Table 5.2 that besides the GR method, the Gaussian positive method does not always outperform its corresponding random method. This lends credibility to the idea that different methods are best for different datasets, rather than a particular method being superior. However, with both chamfer distance datasets, GR was the top performer, possibly demonstrating superiority at learning the chamfer distance.

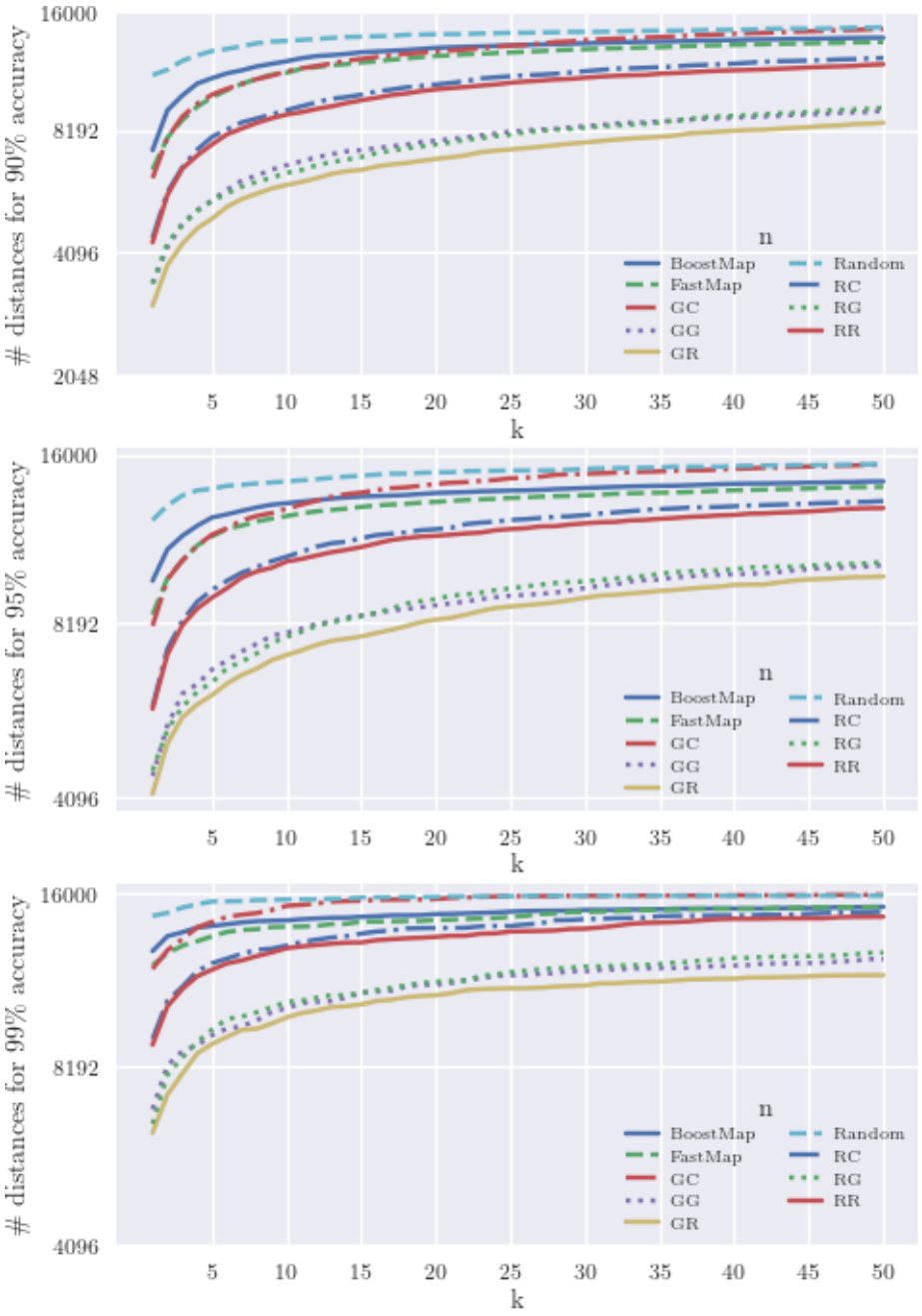

Figure 5.3: Results on the ASL5 dataset using chamfer distance for comparisons. 6000 test objects were used.

ASL5 Dataset with Chamfer Distance										
		90%			95%			99%		
		k=1	k=10	k=50	k=1	k=10	k=50	k=1	k=10	k=50
R	# Distances	11216	13615	14711	12346	14398	15423	14683	15645	15871
	Speedup	1.43	1.18	1.09	1.30	1.11	1.04	1.09	1.02	1.01
BM	# Distances	7328	12132	13858	9703	13204	14398	12811	14425	15205
	Speedup	2.18	1.32	1.15	1.65	1.21	1.11	1.25	1.11	1.05
FM	# Distances	6559	11371	13524	8484	12547	14087	12125	14074	15192
	Speedup	2.44	1.41	1.18	1.89	1.28	1.14	1.32	1.14	1.05
GC	# Distances	6255	11458	14551	8113	12916	15385	11956	15267	15936
	Speedup	2.56	1.40	1.10	1.97	1.24	1.04	1.34	1.05	1.00
GG	# Distances	3442	6732	9163	4470	7902	10323	6940	10273	12418
	Speedup	4.65	2.38	1.75	3.58	2.02	1.55	2.31	1.56	1.29
GR	# Distances	**3038**	**6019**	**8562**	**4164**	**7229**	**9870**	**6337**	**9908**	**11671**
	Speedup	**5.27**	**2.66**	**1.87**	**3.84**	**2.21**	**1.62**	**2.52**	**1.61**	**1.37**
RC	# Distances	4457	9196	12356	5882	10680	13322	9115	13084	14934
	Speedup	3.59	1.74	1.29	2.72	1.50	1.20	1.76	1.22	1.07
RG	# Distances	3437	6448	9318	4569	7768	10462	6561	10511	12737
	Speedup	4.66	2.48	1.72	3.50	2.06	1.53	2.44	1.52	1.26
RR	# Distances	4345	8974	11921	5836	10478	12950	8911	12954	14630
	Speedup	3.68	1.78	1.34	2.74	1.53	1.24	1.80	1.24	1.09

Table 5.2: A table summarizing the number of exact distances needed by each method to return the k nearest neighbors with 90%, 95%, and 99% accuracy. Also includes the speedup from brute force search in the original space, which takes 16.62 s on 16000 database objects. The best method for a particular percent and k is bolded. BoostMap and FastMap have been abbreviated to BM and FM, respectively. R represents the baseline network trained with completely random triplets.

Speech Commands

For the Speech Commands dataset, BoostMap was trained with 4000 objects $\in \mathcal{C}$ and 4000 objects $\in \mathcal{T}$. The remaining 484 objects were used for verification. The model was able to train embeddings up to 53 dimensions, below the 128 expected dimensions.

According to Figure 5.4, BoostMap performs the best at the 90% accuracy level. However, at the 95% and 99% levels, the best performance switches between BoostMap and the RR method. As shown in Table 5.3, the margin of performance between RR and other methods widens, suggesting it as the best method to use for applications. Interestingly, the relationship between Gaussian and random methods are flipped from previous datasets. Here, we see that the random methods of generating positive objects has a larger speedup than their Gaussian counterparts, suggesting that the random weighting method of producing positive objects is better suited to this dataset. We also see that each data mining method outperforms the baseline random triplets, although at the highest accuracy level, various methods approach the same performance as it.

However, the architecture of the network likely has a large impact on the results of training. Filling a temporal sequence with zeros and using a fully connected network is unlikely to find the best representations of each object as relationships between neighboring data in the time series are ignored. Results generated from a convolutional architecture are shown in Figure 5.5.

Here, we see a drastic difference in the results of the neural networks.

Previously, BoostMap was a top performer at all three accuracy levels, but is now one of the worst performers at the 95% and 99% accuracy levels. Even at the 90% level, BoostMap performs about as well as the RR level.

According to Table 5.4, RR shows the highest speedup in most situations. However, the difference between it and other random methods is typically very small, especially at the 99% level. We also see that once again, the random methods outperform the Gaussian methods, providing further evidence that the random weighted method is more suited for this particular dataset. Surprisingly, the random baseline method of training is able to perform better than FastMap at the 90% and 95% level, well above its performance using the fully connected network. This could point to equal importance in the network design and the method of triplet mining for manifold learning.

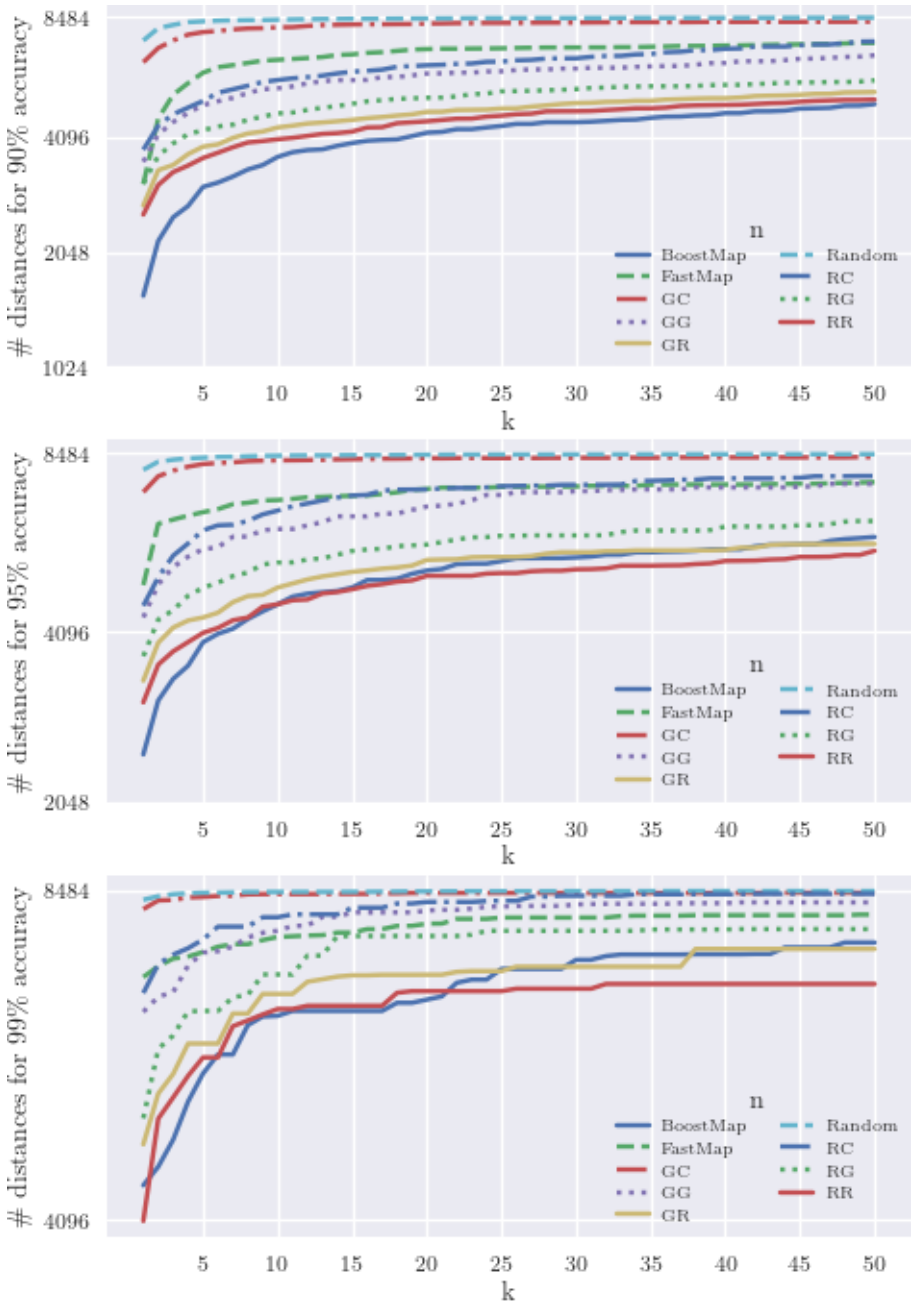

Figure 5.4: Results on the Speech Commands dataset using dynamic time warping for comparisons. 1100 test objects were used. A fully connected network was trained to embed the test points.

Speech Commands Dataset with DTW (FC Architecture)										
		90%			95%			99%		
		k=1	k=10	k=50	k=1	k=10	k=50	k=1	k=10	k=50
R	# Distances	7356	8343	8448	7943	8414	8467	8316	8462	8478
	Speedup	1.15	1.02	1.00	1.07	1.01	1.00	1.02	1.00	1.00
BM	# Distances	**1572**	**3639**	**5008**	**2487**	**4588**	6039	4423	**6434**	7566
	Speedup	**5.40**	**2.33**	**1.69**	**3.41**	**1.85**	1.40	1.92	**1.32**	1.12
FM	# Distances	3085	6545	7249	4963	7021	7553	7010	7660	8048
	Speedup	2.75	1.30	1.17	1.71	1.21	1.12	1.21	1.11	1.05
GC	# Distances	6450	7966	8246	7254	8256	8369	8146	8422	8446
	Speedup	1.32	1.07	1.03	1.17	1.03	1.01	1.04	1.01	1.00
GG	# Distances	3533	5510	6726	4351	6238	7501	6491	7765	8269
	Speedup	2.40	1.54	1.26	1.95	1.36	1.13	1.31	1.09	1.03
GR	# Distances	2711	4348	5387	3361	4913	5871	4843	6753	7460
	Speedup	3.13	1.95	1.57	2.52	1.73	1.45	1.75	1.26	1.14
RC	# Distances	3801	5792	7314	4562	6726	7744	6767	8001	8426
	Speedup	2.23	1.46	1.16	1.86	1.26	1.10	1.25	1.06	1.01
RG	# Distances	3101	4732	5781	3714	5440	6445	5129	7050	7794
	Speedup	2.74	1.79	1.47	2.28	1.56	1.32	1.65	1.20	1.09
RR	# Distances	2568	4048	5148	3076	**4589**	**5708**	**4088**	6532	**6904**
	Speedup	3.30	2.10	1.65	2.76	**1.85**	**1.49**	**2.08**	1.30	**1.23**

Table 5.3: A table summarizing the number of exact distances needed by each method to return the k nearest neighbors with 90%, 95%, and 99% accuracy. Also includes the speedup from brute force search in the original space, which takes 39.57 s on 8484 database objects. The best method for a particular percent and k is bolded. BoostMap and FastMap have been abbreviated to BM and FM, respectively. A fully connected architecture was used to obtain these results. R represents the baseline network trained with completely random triplets.

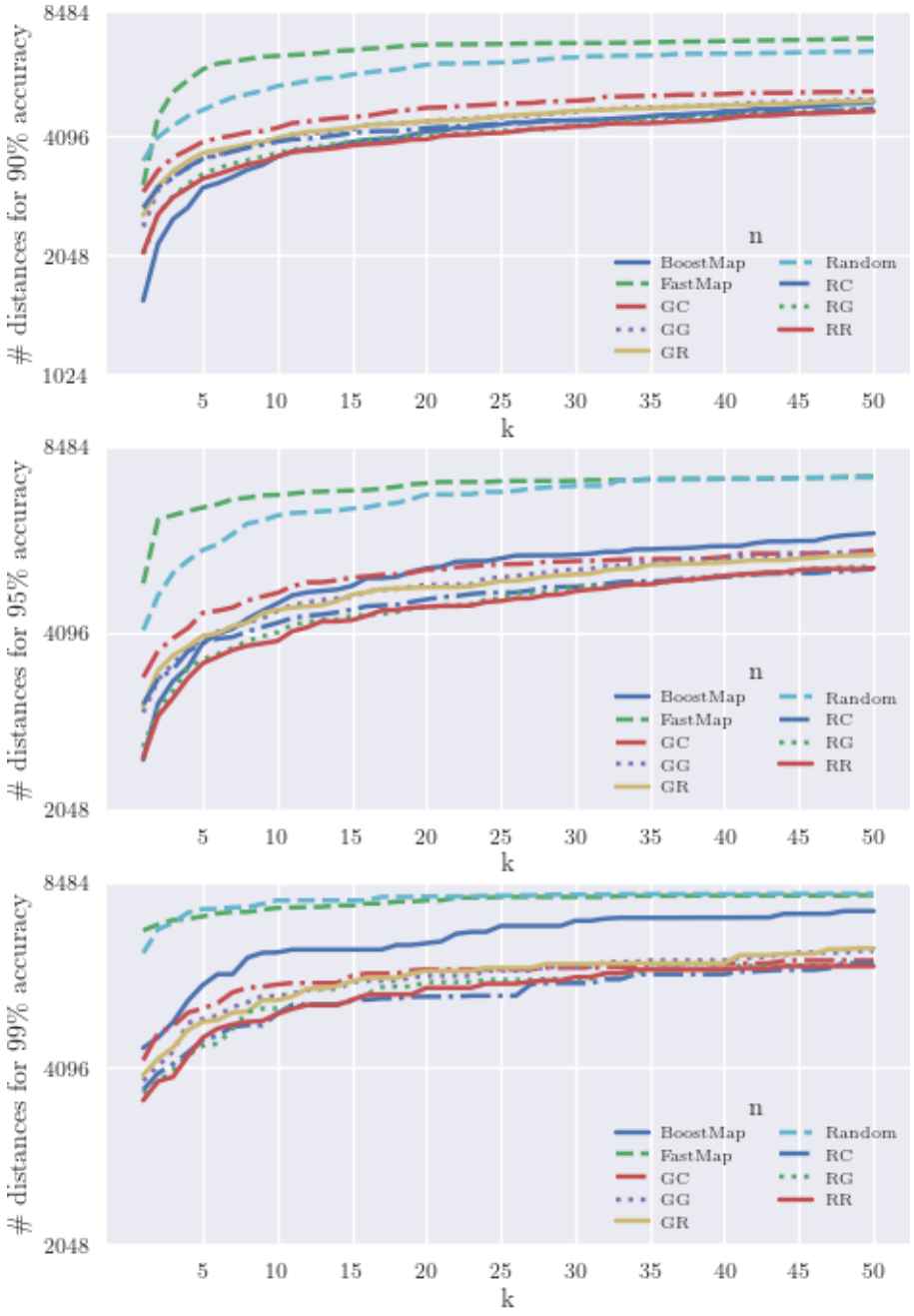

Figure 5.5: Results on the Speech Commands dataset using dynamic time warping for comparisons. 1100 test objects were used. A convolutional network architecture was used to embed points

Speech Commands Dataset with DTW (CNN Architecture)										
		90%			95%			99%		
		k=1	k=10	k=50	k=1	k=10	k=50	k=1	k=10	k=50
R	# Distances	3549	5491	6727	4134	6477	7515	6419	7887	8094
	Speedup	2.39	1.55	1.26	2.05	1.31	1.13	1.32	1.08	1.05
BM	# Distances	**1572**	**3639**	5008	**2487**	4588	6039	4423	6434	7566
	Speedup	**5.40**	**2.33**	1.69	**3.41**	1.85	1.40	1.92	1.32	1.12
FM	# Distances	3085	6545	7249	4963	7021	7553	7010	7660	8048
	Speedup	2.75	1.30	1.17	1.71	1.21	1.12	1.21	1.11	1.05
GC	# Distances	2963	4304	5321	3435	4781	5656	4215	5674	6235
	Speedup	2.86	1.97	1.59	2.47	1.77	1.50	2.01	1.50	1.36
GG	# Distances	2417	4043	5079	2992	4454	5625	3888	5420	6470
	Speedup	3.51	2.10	1.67	2.84	1.90	1.51	2.18	1.57	1.31
GR	# Distances	2588	4061	5050	3065	4475	5551	3976	5335	6537
	Speedup	3.28	2.09	1.68	2.77	1.90	1.53	2.13	1.59	1.30
RC	# Distances	2699	3970	4822	3088	4252	**5253**	3754	**5059**	6167
	Speedup	3.14	2.14	1.76	2.75	2.00	**1.62**	2.26	**1.68**	1.38
RG	# Distances	2057	3710	4792	2612	4097	5291	3703	5162	6114
	Speedup	4.12	2.29	1.77	3.25	2.07	1.60	2.29	1.64	1.39
RR	# Distances	2082	**3638**	**4736**	2505	**3960**	5274	**3603**	**5056**	**6095**
	Speedup	4.07	**2.33**	**1.79**	3.39	**2.14**	1.61	**2.35**	**1.68**	**1.39**

Table 5.4: A table summarizing the number of exact distances needed by each method to return the k nearest neighbors with 90%, 95%, and 99% accuracy. Also includes the speedup from brute force search in the original space, which takes 39.57 s on 8484 database objects. The best method for a particular percent and k is bolded. BoostMap and FastMap have been abbreviated to BM and FM, respectively. R represents the baseline network trained with completely random triplets.

Tctodd

For the Tctodd dataset, the BoostMap model used 850 objects for $\mathcal{C}$ and 850 objects for $\mathcal{T}$. The other 95 objects were used for validation. The model trained a 33 dimensional embedding.

Immediately, it's apparent from Figure 5.6 that the neural network models were not able to learn a great embedding. Both BoostMap and FastMap provide significantly higher speedup at all data points contained in Table 5.5. What speedup neural networks provide is seen for very low k values while providing virtually no benefit at higher k values, essentially mimicking the performance of the baseline training method.

The network architecture used on the Tctodd dataset comes from [56] and was developed for working with raw audio data. While layer sizes were changed to decrease the receptive window for the data, the nature of the network may be inappropriate for this dataset. This is a deep neural network expecting large audio signals. The small dataset size and the small size of each signal may make this network unsuitable for this application. Perhaps, there are networks more suited to learning representations of this data.

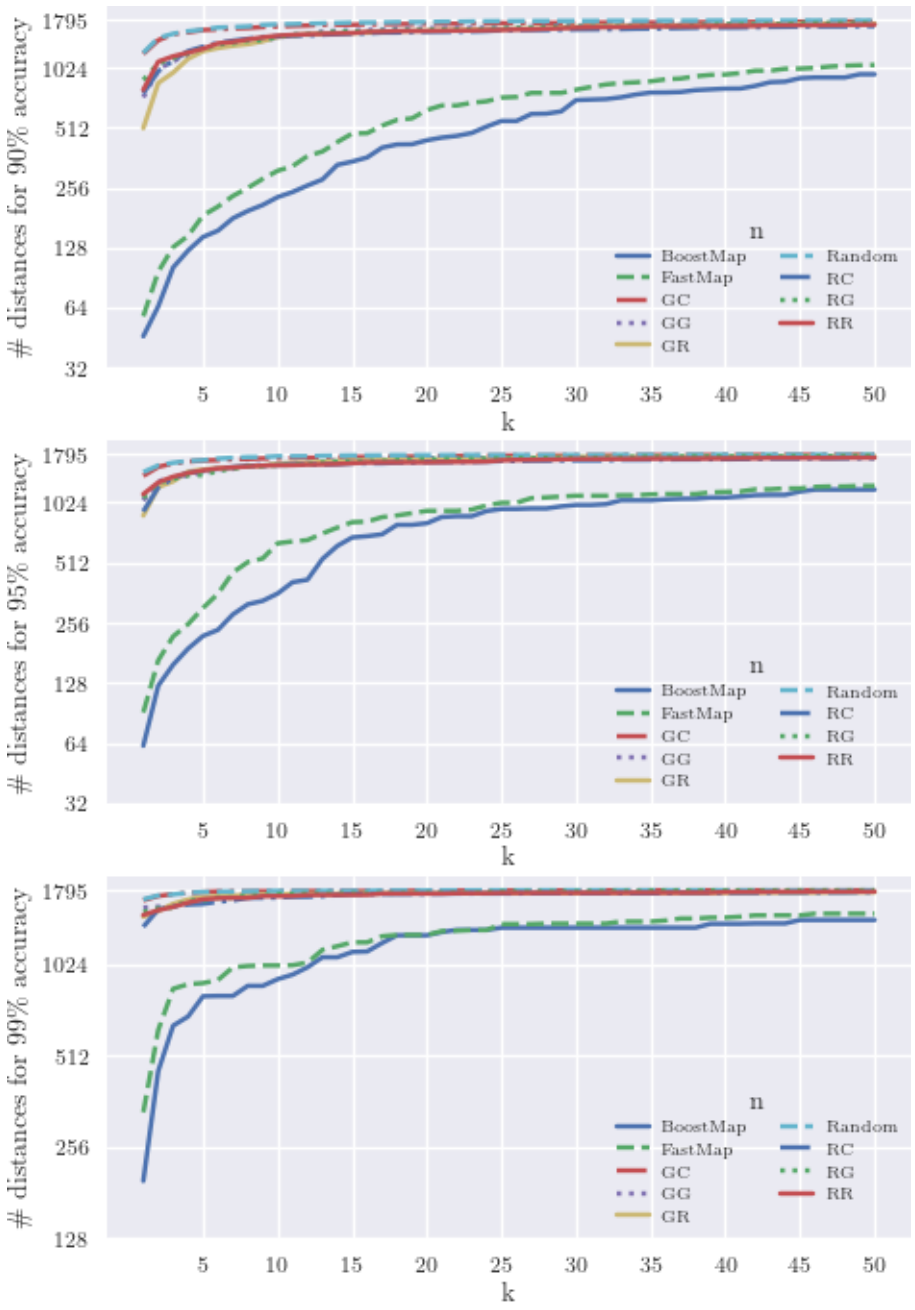

Figure 5.6: Results on the Tctodd dataset using multidimensional dynamic time warping for comparisons. 770 test objects were used.

Tctodd Dataset with Multidimensional DTW										
		90%			95%			99%		
		k=1	k=10	k=50	k=1	k=10	k=50	k=1	k=10	k=50
R	# Distances	1220	1700	1775	1457	1751	1782	1672	1782	1793
	Speedup	1.47	1.06	1.01	1.23	1.03	1.01	1.07	1.01	1.00
BM	# Distances	**46**	**230**	**953**	**62**	**359**	**1191**	**198**	**913**	**1430**
	Speedup	**39.02**	**7.80**	**1.88**	**28.95**	**5.00**	**1.51**	**9.07**	**1.97**	**1.26**
FM	# Distances	58	312	1061	91	641	1244	333	1013	1500
	Speedup	30.95	5.75	1.69	19.73	2.80	1.44	5.39	1.77	1.20
GC	# Distances	1208	1655	1756	1396	1710	1779	1659	1777	1791
	Speedup	1.49	1.08	1.02	1.29	1.05	1.01	1.08	1.01	1.00
GG	# Distances	730	1457	1718	1050	1593	1745	1570	1747	1777
	Speedup	2.46	1.23	1.04	1.71	1.13	1.03	1.14	1.03	1.01
GR	# Distances	511	1467	1715	883	1607	1746	1465	1737	1766
	Speedup	3.51	1.22	1.05	2.03	1.12	1.03	1.23	1.03	1.02
RC	# Distances	766	1473	1664	926	1580	1713	1358	1695	1767
	Speedup	2.34	1.22	1.08	1.94	1.14	1.05	1.32	1.06	1.02
RG	# Distances	892	1467	1722	1082	1580	1766	1520	1740	1791
	Speedup	2.01	1.22	1.04	1.66	1.14	1.02	1.18	1.03	1.00
RR	# Distances	796	1489	1693	1130	1577	1722	1485	1713	1773
	Speedup	2.26	1.21	1.06	1.59	1.14	1.04	1.21	1.05	1.01

Table 5.5: A table summarizing the number of exact distances needed by each method to return the k nearest neighbors with 90%, 95%, and 99% accuracy. Also includes the speedup from brute force search in the original space, which takes .097 s on 1795 database objects. The best method for a particular percent and k is bolded. BoostMap and FastMap have been abbreviated to BM and FM, respectively. R represents the baseline network trained with completely random triplets.

5.4 Nearest Neighbor Parameter

As explained in Section 4.2, a parameter n was defined that determined which objects to focus on when choosing triplets. In Section 5.3, it was mentioned that this parameter was set to 10 for all networks trained. Experiments were performed by getting the nearest 50 neighbors, so it may have been better to choose 50 as a parameter to optimize the networks for this number of neighbors. This section demonstrates the effect on experiment results from changing this parameter to focus on larger numbers of neighbors. We explore the effects of the n parameters having values 10, 20, 30, 40, and 50 neighbors on the MNIST dataset.

The experiment is identical to that in Section 5.3, where the number of exact distances needed to find k nearest neighbors is taken at different accuracy levels. In this case, a single method of triplet mining is used in each figure, rather than all 6, to reduce graph clutter.

The nearest neighbors parameter has less of an effect on results than expected. As shown in Figures 5.7 and 5.10, changing the parameter for the GC and RC method has little effect on the outcome of the experiment. As seen in Figures 5.8 and 5.11, changing the parameter has more of an influence on the output. However, differences appear to be marginal, and as k grows, the difference in results becomes negligible. GR, shown in Figure 5.9, shows a preference for n = 10, but this benefit is lost as k increases. RR, shown in Figure 5.12, shows a benefit for n = 30 as long as k is low.

These results match [50], which has a similar parameter that provides limited benefit during training.

Similar results were found on other datasets, but were left out for brevity. Overall, choosing n seems to have a low effect on the training of the model. Therefore, an arbitrary low value of 10 was chosen for n in the experiments in Section 5.3.

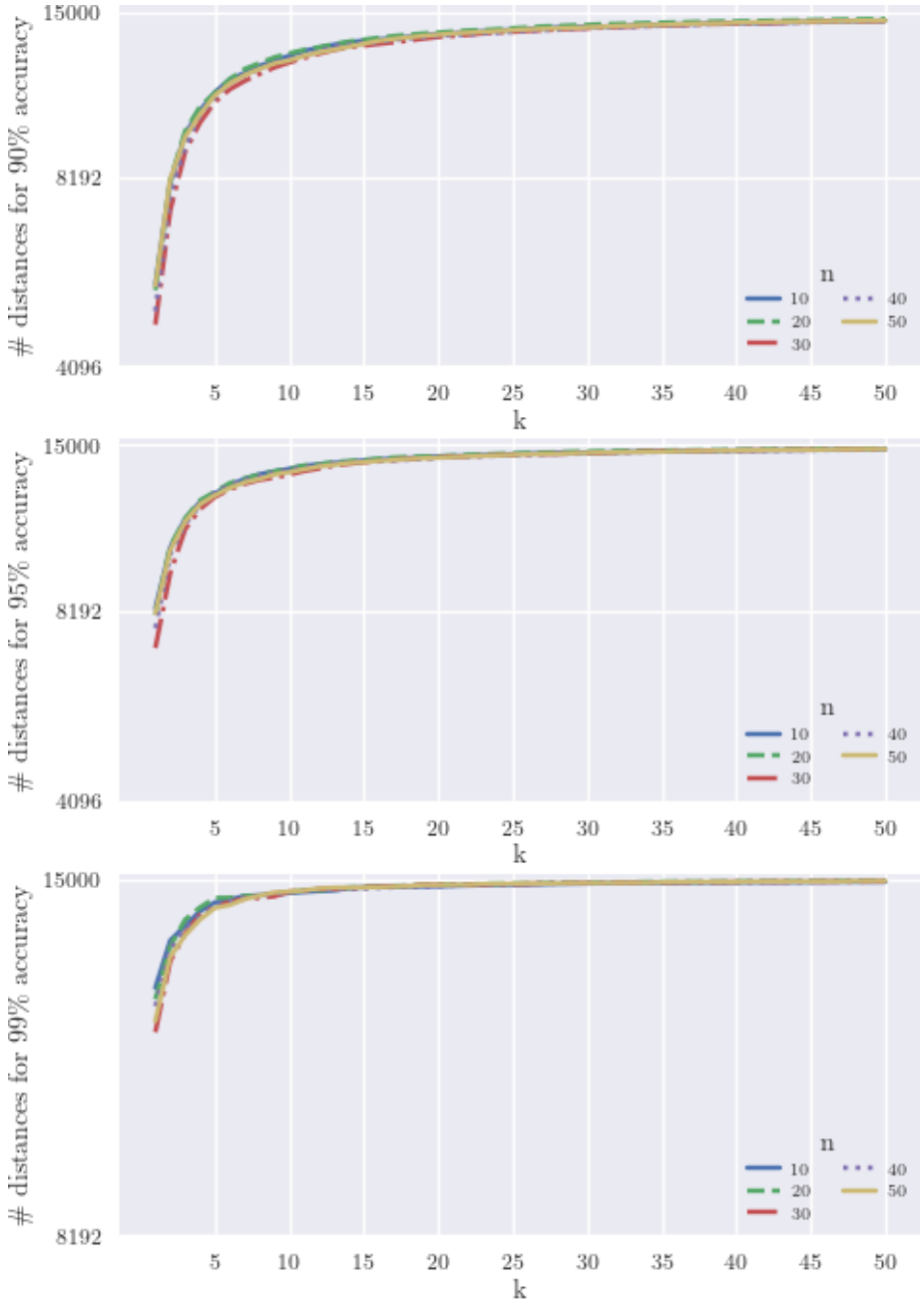

Figure 5.7: Results on the MNIST dataset using chamfer distance for comparisons. 6000 test objects were used. Values of 10, 20, 30, 40, and 50 were used for the n parameter when training a neural network using the GC triplet strategy.

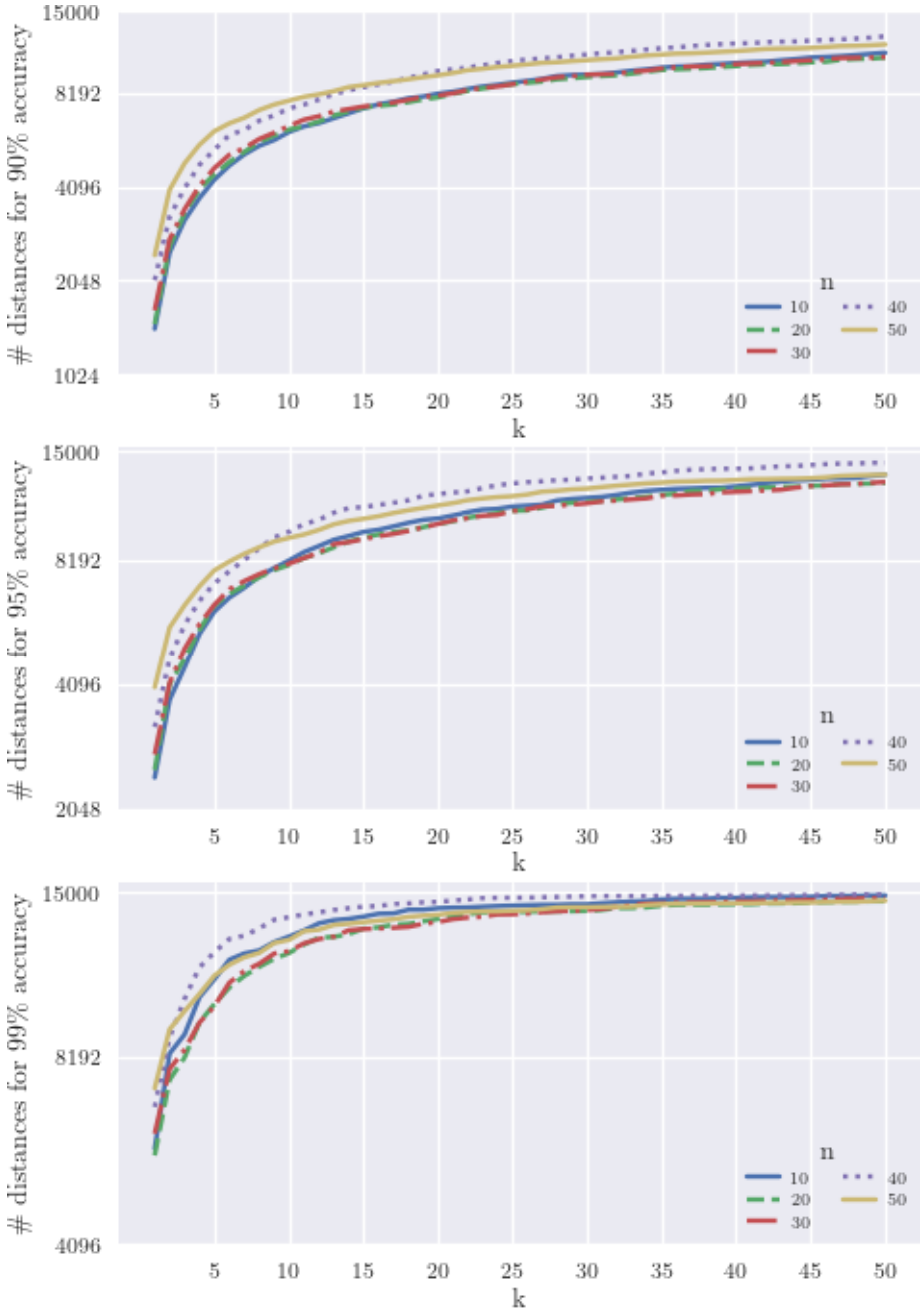

Figure 5.8: Results on the MNIST dataset using chamfer distance for comparisons. 6000 test objects were used. Values of 10, 20, 30, 40, and 50 were used for the n parameter when training a neural network using the GG triplet strategy.

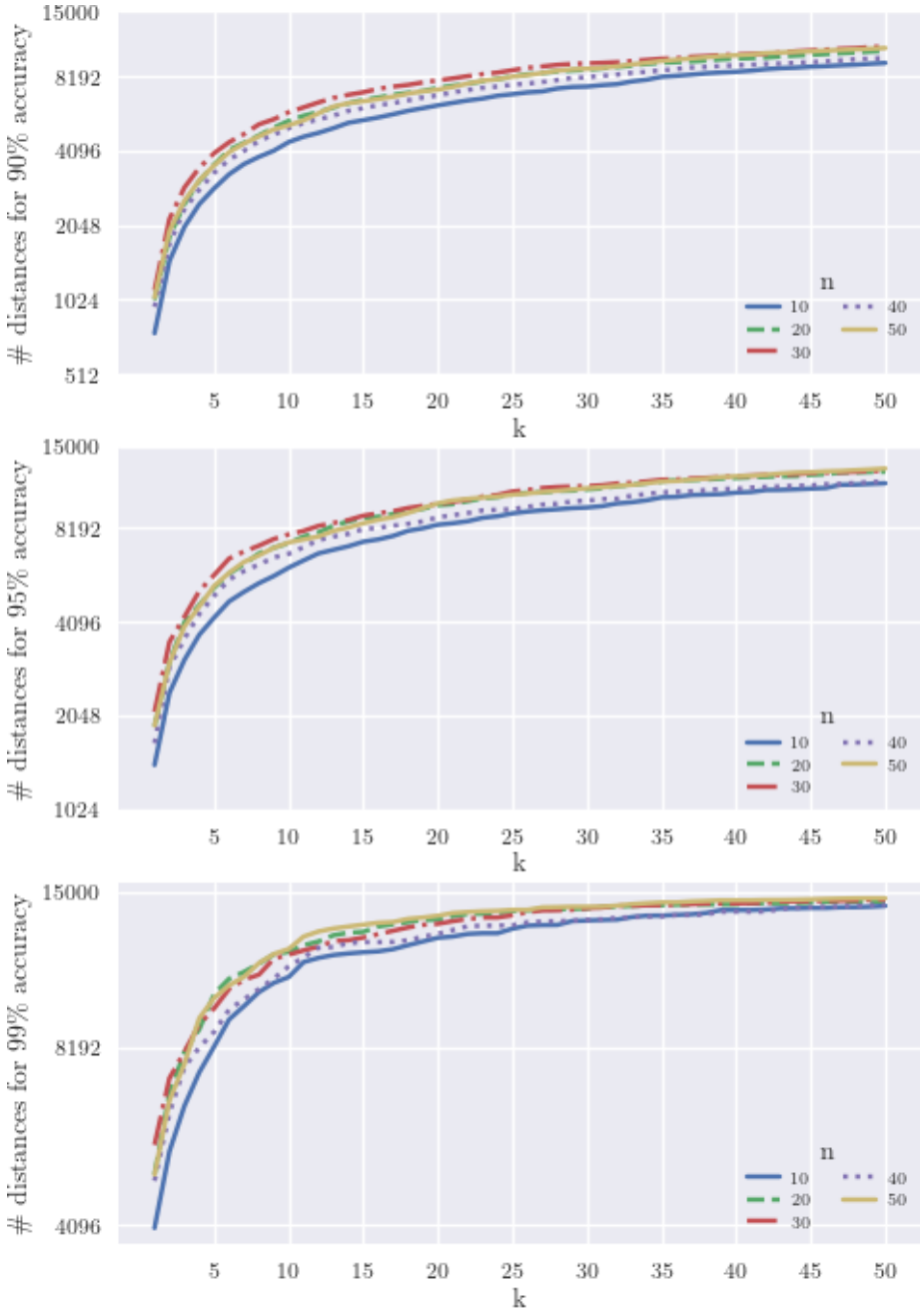

Figure 5.9: Results on the MNIST dataset using chamfer distance for comparisons. 6000 test objects were used. Values of 10, 20, 30, 40, and 50 were used for the n parameter when training a neural network using the GR triplet strategy.

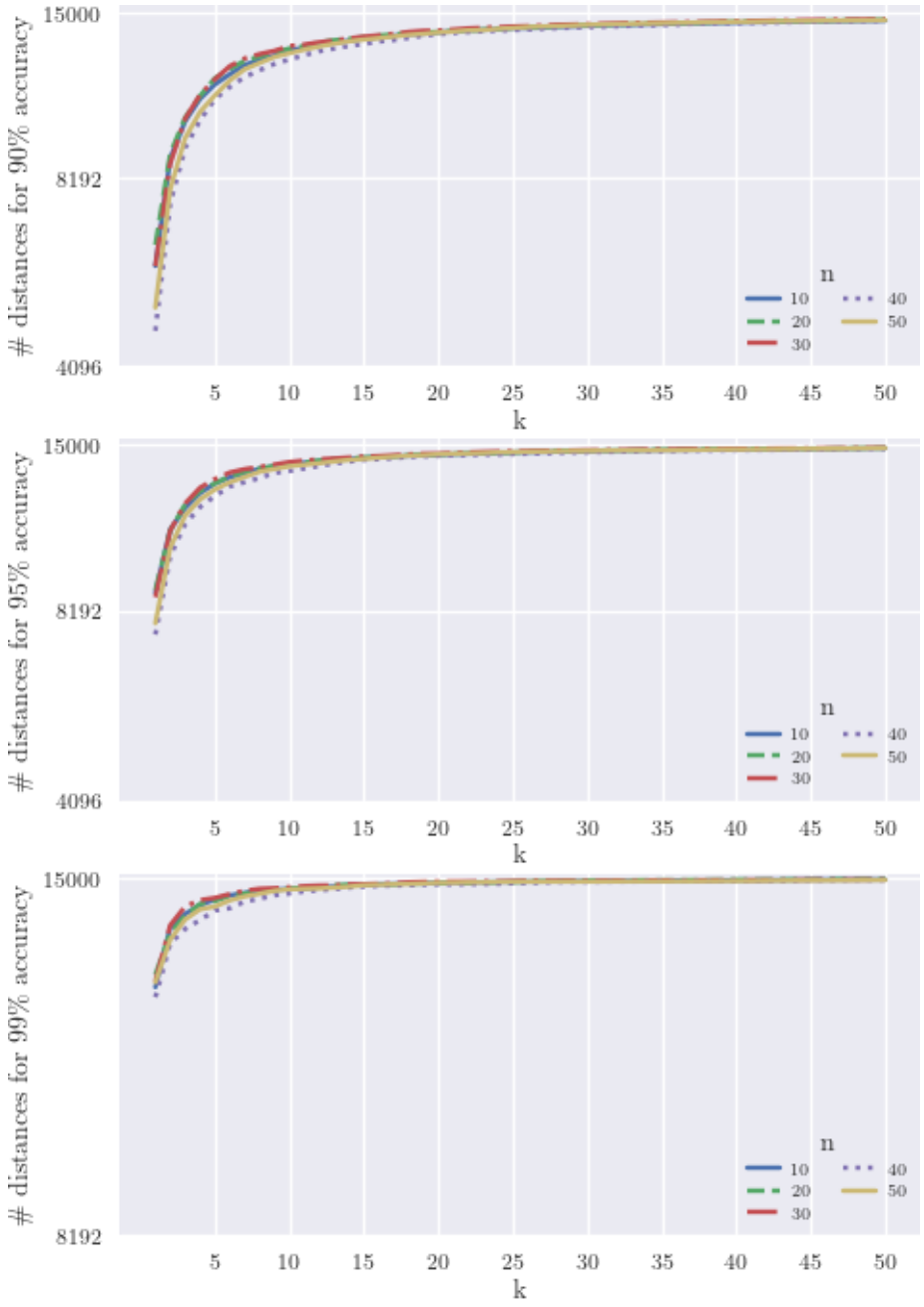

Figure 5.10: Results on the MNIST dataset using chamfer distance for comparisons. 6000 test objects were used. Values of 10, 20, 30, 40, and 50 were used for the n parameter when training a neural network using the RC triplet strategy.

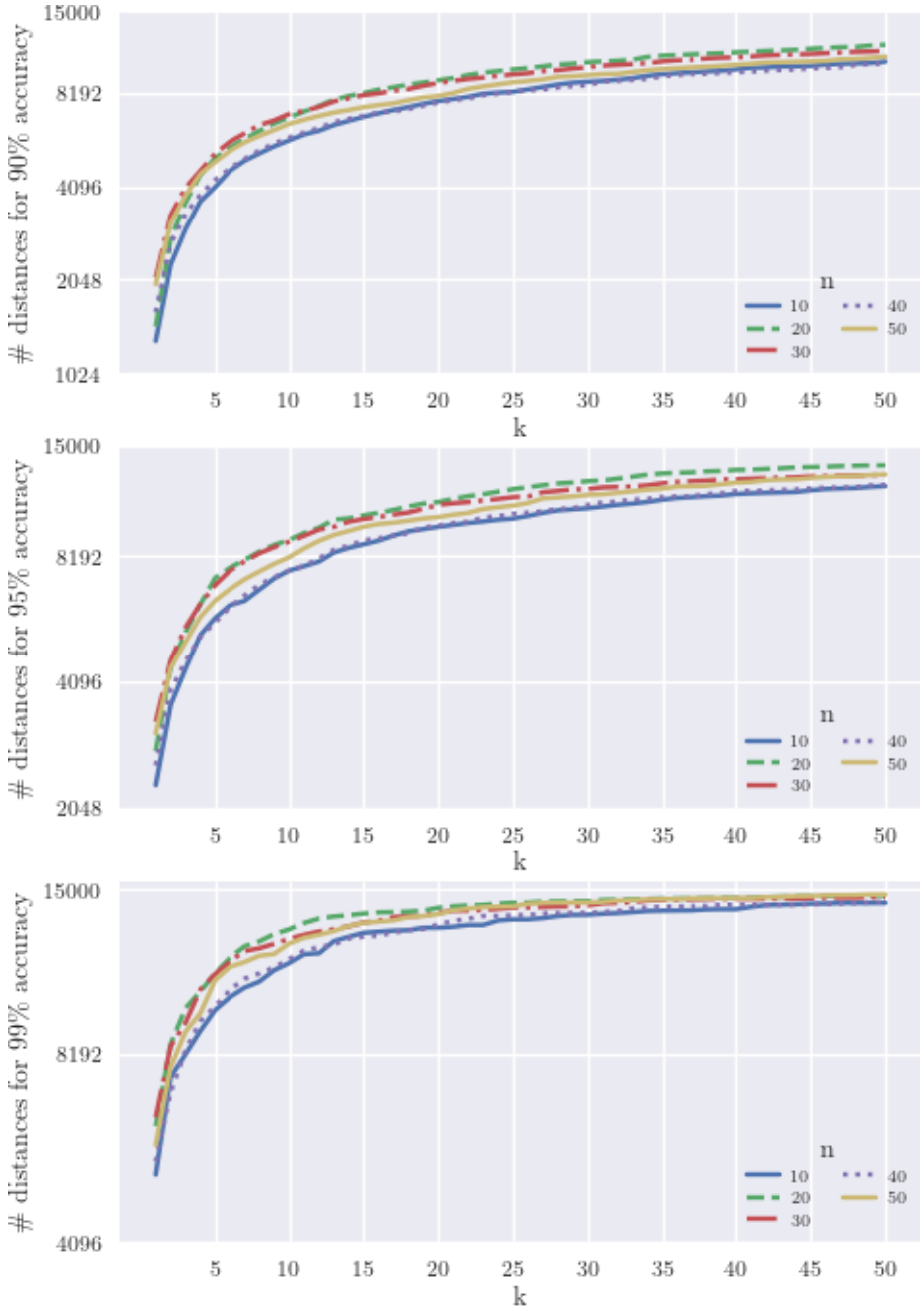

Figure 5.11: Results on the MNIST dataset using chamfer distance for comparisons. 6000 test objects were used. Values of 10, 20, 30, 40, and 50 were used for the n parameter when training a neural network using the RG triplet strategy.

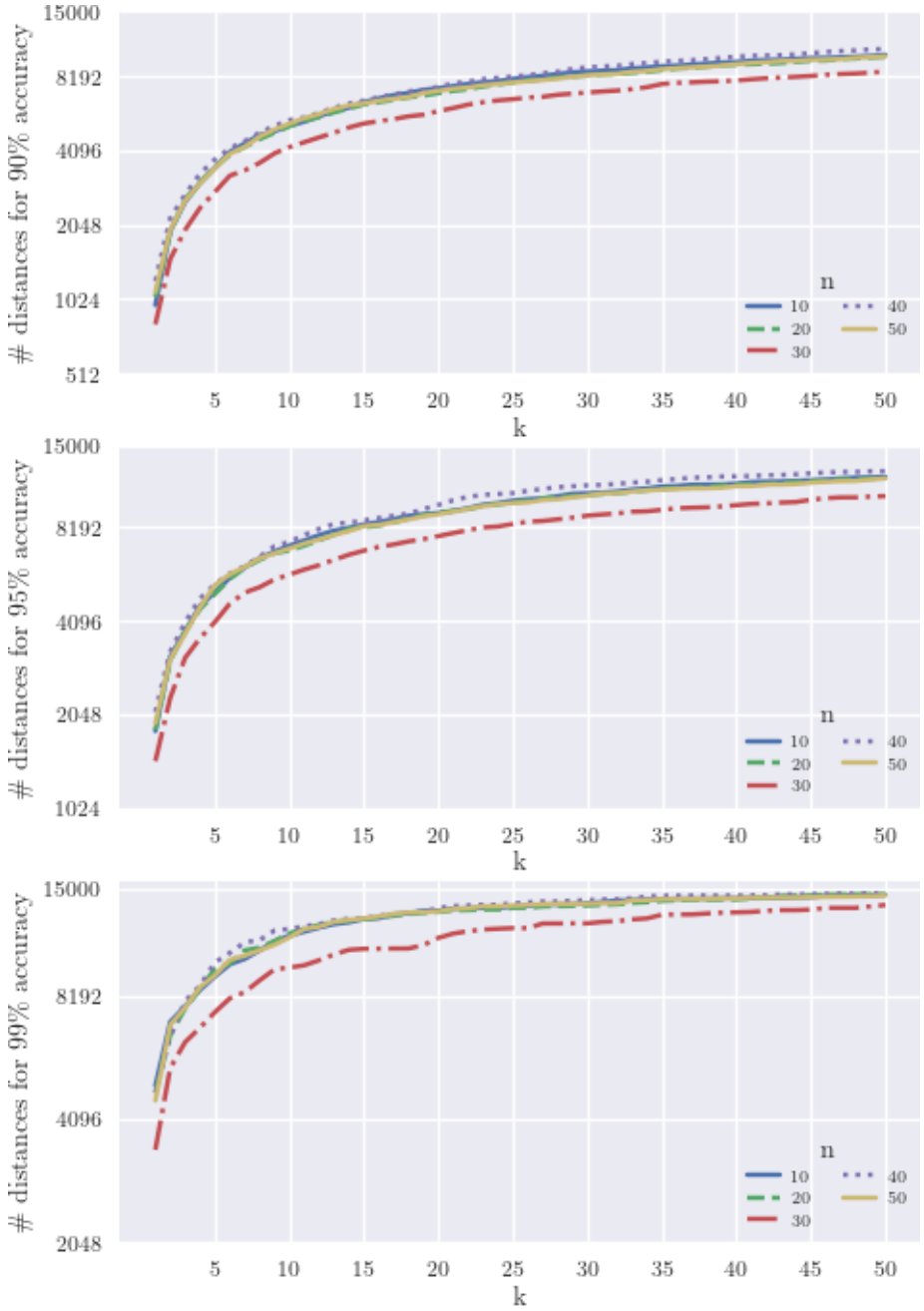

Figure 5.12: Results on the MNIST dataset using chamfer distance for comparisons. 6000 test objects were used. Values of 10, 20, 30, 40, and 50 were used for the n parameter when training a neural network using the RR triplet strategy.

Chapter 6

Conclusion

Due to its simplicity, accuracy, and ability to allow domain specific distances, k nearest neighbors is a staple in many fields. The ability to speed up queries is therefore worthy of looking into, especially for queries in difficult, non-metric spaces of objects. This work has demonstrated a method for solving this problem using neural networks.

Through experiments, we have shown the effectiveness of training neural networks using triplet loss to speed up nearest neighbor searches, beating or coming even with previous methods such as BoostMap and FastMap in a number of domains using diverse distance functions. In addition, these methods of data mining perform better than a baseline method of random triplets, demonstrating their usefulness in manifold learning.

Neural network based model also provide hidden speedups not seen in these experiments. The time to train a BoostMap model, and the time to

embed multiple objects, can take a long time. Neural network models are quick to train, and multiple objects can be embedded at the same time using matrices, providing even more of a speed increase over other models.

The only dataset that proved a failure was the Tctodd dataset. The results from this and using a fully connected network on the Speech Commands dataset demonstrates the need for good network design when designing an embedding system. More research is needed to determine if an architecture exists that performs well on the Tctodd dataset or if the size and nature of the dataset prevented positive results from a neural network-based model.

This work has demonstrated potential methods of triplet mining for the express purpose of training neural networks to embed objects such that nearest neighbor queries are sped up. With the popularity of neural network tools, these methods are also simple and efficient to implement in a number of workflows. With the presented evidence, these methods have been shown to be effective for various objects and situations, demonstrating the potential of a neural network-based approach to topology preservation.

Bibliography

[1] Vivek Mahato, Martin O'Reilly, and Padraig Cunningham. "A Comparison of k-NN Methods for Time Series Classification and Regression". en. In: (), p. 12.

[2] Mihael Ankerst et al. "3D Shape Histograms for Similarity Search and Classification in Spatial Databases". en. In: *Advances in Spatial Databases*. Ed. by Gerhard Goos et al. Vol. 1651. Series Title: Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 207–226. ISBN: 978-3-540-66247-1 978-3-540-48482-0. DOI: 10.1007/3-540-48482-5_14. URL: http://link.springer.com/10.1007/3-540-48482-5_14 (visited on 04/11/2022).

[3] G.R. Hjaltason and H. Samet. "Properties of embedding methods for similarity searching in metric spaces". en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.5 (May 2003), pp. 530–549. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2003.1195989. URL: http://ieeexplore.ieee.org/document/1195989/ (visited on 12/28/2021).

[4] Flip Korn, Nikolaos Sidiropoulos, and Christos Faloutsos. “Fast Nearest Neighbor Search in Medical Image Databases”. en. In: (), p. 27.

[5] John A. Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Information science and statistics. OCLC: ocm77256575. New York: Springer, 2007. ISBN: 978-0-387-39350-6.

[6] Sergey Brin. “Near Neighbor Search in Large Metric Spaces”. en. In: (), p. 11.

[7] Peter N. Yianilos. “Data structures and algorithms for nearest neighbor search in general metric spaces”. In: *Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms*. SODA ’93. USA: Society for Industrial and Applied Mathematics, Jan. 1993, pp. 311–321. ISBN: 978-0-89871-313-8. (Visited on 04/17/2022).

[8] Alina Beygelzimer, Sham Kakade, and John Langford. “Cover trees for nearest neighbor”. en. In: *Proceedings of the 23rd international conference on Machine learning - ICML ’06*. Pittsburgh, Pennsylvania: ACM Press, 2006, pp. 97–104. ISBN: 978-1-59593-383-6. DOI: 10.1145/1143844.1143857. URL: http://portal.acm.org/citation.cfm?doid=1143844.1143857 (visited on 04/17/2022).

[9] P. Ciaccia, M. Patella, and P. Zezula. “M-tree: An Efficient Access Method for Similarity Search in Metric Spaces”. In: *VLDB*. 1997.

[10] Sachendra Singh and Shalini Batra. “An efficient bi-layer content based image retrieval system”. en. In: *Multimedia Tools and Applications*

79.25 (July 2020), pp. 17731–17759. ISSN: 1573-7721. DOI: 10.1007/s11042-019-08401-7. URL: https://doi.org/10.1007/s11042-019-08401-7 (visited on 04/18/2022).

[11] Matthew L Miller, Ingemar J Cox, and Manuel Acevedo Rodriguez. "Audio Fingerprinting: Nearest Neighbor Search in High Dimensional Binary Spaces". en. In: (), p. 5.

[12] Michael Casey and Malcolm Slaney. "Song Intersection by Approximate Nearest Neighbor Search". en. In: (), p. 6.

[13] Dario Lucarella. "A document retrieval system based on nearest neighbour searching". en. In: *Journal of Information Science* 14.1 (Feb. 1988), pp. 25–33. ISSN: 0165-5515, 1741-6485. DOI: 10.1177/016555158801400104. URL: http://journals.sagepub.com/doi/10.1177/016555158801400104 (visited on 04/21/2022).

[14] M. Flickner et al. "Query by image and video content: the QBIC system". In: *Computer* 28.9 (Sept. 1995). Conference Name: Computer, pp. 23–32. ISSN: 1558-0814. DOI: 10.1109/2.410146.

[15] Joshua R. Smith and Shih-Fu Chang. "Image and video search engine for the World Wide Web". In: *Electronic Imaging*. 1997. DOI: 10.1117/12.263446.

[16] James Ze Wang et al. "System for screening objectionable images". en. In: *Computer Communications* 21.15 (Oct. 1998), pp. 1355–1360. ISSN: 01403664. DOI: 10.1016/S0140-3664(98)00203-5. URL: https:

//linkinghub.elsevier.com/retrieve/pii/S0140366498002035 (visited on 04/18/2022).

[17] Ben Carterette and Fazli Can. "Comparing inverted files and signature files for searching a large lexicon". en. In: *Information Processing & Management* 41.3 (May 2005), pp. 613–633. ISSN: 03064573. DOI: 10.1016/j.ipm.2003.12.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S0306457303001158 (visited on 04/18/2022).

[18] Justin Zobel, Alistair Moffat, and Kotagiri Ramamohanarao. "Inverted files versus signature files for text indexing". In: *ACM Transactions on Database Systems* 23.4 (Dec. 1998), pp. 453–490. ISSN: 0362-5915. DOI: 10.1145/296854.277632. URL: https://doi.org/10.1145/296854.277632 (visited on 04/18/2022).

[19] Philippe Aigrain. "Introduction to Audio Retrieval and Navigation Interfaces". en. In: *Readings in Multimedia Computing and Networking*. Ed. by Kevin Jeffay and Hongjiang Zhang. The Morgan Kaufmann Series in Multimedia Information and Systems. San Francisco: Morgan Kaufmann, Jan. 2002, pp. 199–203. ISBN: 978-1-55860-651-7. DOI: 10.1016/B978-155860651-7/50101-7. URL: https://www.sciencedirect.com/science/article/pii/B9781558606517501017 (visited on 04/18/2022).

[20] Erling Wold et al. "Content-Based Classification, Search, and Retrieval of Audio". en. In: *Readings in Multimedia Computing and Network-*

ing. Ed. by Kevin Jeffay and Hongjiang Zhang. The Morgan Kaufmann Series in Multimedia Information and Systems. San Francisco: Morgan Kaufmann, Jan. 2002, pp. 222–231. ISBN: 978-1-55860-651-7. DOI: 10.1016/B978-155860651-7/50104-2. URL: https://www.sciencedirect.com/science/article/pii/B9781558606517501042 (visited on 04/18/2022).

[21] Jérôme Hert et al. "Enhancing the effectiveness of similarity-based virtual screening using nearest-neighbor information". eng. In: *Journal of Medicinal Chemistry* 48.22 (Nov. 2005), pp. 7049–7054. ISSN: 0022-2623. DOI: 10.1021/jm050316n.

[22] David C. Anastasiu and George Karypis. "Efficient identification of Tanimoto nearest neighbors". en. In: *International Journal of Data Science and Analytics* 4.3 (Nov. 2017), pp. 153–172. ISSN: 2364-4168. DOI: 10.1007/s41060-017-0064-z. URL: https://doi.org/10.1007/s41060-017-0064-z (visited on 04/21/2022).

[23] David V Pynadath et al. "A Nearest-Neighbor Approach to Recognizing Subjective Beliefs in Human-Robot Interaction". en. In: (), p. 8.

[24] Phillip Quin et al. "Nearest Neighbour Exploration with Backtracking for Robotic Exploration of Complex 3D Environments". en. In: (2013), p. 8.

[25] Hannes Schulz et al. "Learning Kinematics from Direct Self-Observation Using Nearest-Neighbor Methods". In: *Advances in Robotics Research:*

Theory, Implementation, Application. Journal Abbreviation: Advances in Robotics Research: Theory, Implementation, Application. Jan. 2009, pp. 11–20. ISBN: 978-3-642-01212-9. DOI: 10.1007/978-3-642-01213-6_2.

[26] Vassilis Athitsos. "LEARNING EMBEDDINGS FOR INDEXING, RETRIEVAL, AND CLASSIFICATION, WITH APPLICATIONS TO OBJECT AND SHAPE RECOGNITION IN IMAGE DATABASES". en. In: (), p. 172.

[27] H G Barrow. "PARAMETRIC CORRESPONDENCE AND CHAMFER MATCHING: TWO NEW TECHNIQUES FOR IMAGE MATCHING". en. In: (), p. 10.

[28] G. Borgefors. "Hierarchical chamfer matching: a parametric edge matching algorithm". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 10.6 (Nov. 1988). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 849–865. ISSN: 1939-3539. DOI: 10.1109/34.9107.

[29] C. Myers, L. Rabiner, and A. Rosenberg. "Performance tradeoffs in dynamic time warping algorithms for isolated word recognition". In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 28.6 (Dec. 1980). Conference Name: IEEE Transactions on Acoustics, Speech, and Signal Processing, pp. 623–635. ISSN: 0096-3518. DOI: 10.1109/TASSP.1980.1163491.

[30] H. Sakoe and S. Chiba. “Dynamic programming algorithm optimization for spoken word recognition”. In: *IEEE Transactions on Acoustics, Speech, and Signal Processing* 26.1 (Feb. 1978). Conference Name: IEEE Transactions on Acoustics, Speech, and Signal Processing, pp. 43–49. ISSN: 0096-3518. DOI: 10.1109/TASSP.1978.1163055.

[31] C.C. Tappert, C.Y. Suen, and T. Wakahara. “The state of the art in online handwriting recognition”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.8 (Aug. 1990). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 787–808. ISSN: 1939-3539. DOI: 10.1109/34.57669.

[32] Alon Efrat, Quanfu Fan, and Suresh Venkatasubramanian. “Curve Matching, Time Warping, and Light Fields: New Algorithms for Computing Similarity between Curves”. en. In: *Journal of Mathematical Imaging and Vision* 27.3 (Apr. 2007), pp. 203–216. ISSN: 1573-7683. DOI: 10.1007/s10851-006-0647-0. URL: https://doi.org/10.1007/s10851-006-0647-0 (visited on 04/12/2022).

[33] Omer Gold and Micha Sharir. “Dynamic Time Warping and Geometric Edit Distance: Breaking the Quadratic Barrier”. en. In: (July 2016). DOI: 10.48550/arXiv.1607.05994. URL: https://arxiv.org/abs/1607.05994v4 (visited on 04/12/2022).

[34] Mohammad Shokoohi-Yekta et al. “Generalizing DTW to the multi-dimensional case requires an adaptive approach”. In: *Data mining and*

knowledge discovery 31.1 (Jan. 2017), pp. 1–31. ISSN: 1384-5810. DOI: 10.1007/s10618-016-0455-0. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5668684/ (visited on 04/11/2022).

[35] Vladimir Iosifovich Levenshtein. "Binary codes capable of correcting deletions, insertions and reversals." In: *Soviet Physics Doklady* 10.8 (Feb. 1966), pp. 707–710.

[36] Dominik Schnitzer, Arthur Flexer, and Gerhard Widmer. "A FILTER-AND-REFINE INDEXING METHOD FOR FAST SIMILARITY SEARCH IN MILLIONS OF MUSIC TRACKS". en. In: *Oral Session* (2009), p. 6.

[37] Teuvo Kohonen. "Self-organized formation of topologically correct feature maps". en. In: *Biological Cybernetics* 43.1 (1982), pp. 59–69. ISSN: 0340-1200, 1432-0770. DOI: 10.1007/BF00337288. URL: http://link.springer.com/10.1007/BF00337288 (visited on 04/15/2022).

[38] A. Lendasse et al. "Forecasting electricity consumption using nonlinear projection and self-organizing maps". en. In: *Neurocomputing* 48.1 (Oct. 2002), pp. 299–311. ISSN: 0925-2312. DOI: 10.1016/S0925-2312(01)00646-4. URL: https://www.sciencedirect.com/science/article/pii/S0925231201006464 (visited on 04/15/2022).

[39] Amaury Lendasse et al. "Time series forecasting using CCA and Kohonen maps - application to electricity consumption." In: Jan. 2000, pp. 329–334.

[40] Christopher M. Bishop, Markus Svensén, and Christopher K. I. Williams. "GTM: The Generative Topographic Mapping". en. In: *Neural Computation* 10.1 (Jan. 1998), pp. 215–234. ISSN: 0899-7667, 1530-888X. DOI: 10.1162/089976698300017953. URL: https://direct.mit.edu/neco/article/10/1/215-234/6127 (visited on 01/24/2022).

[41] L. V. D. Maaten and Geoffrey E. Hinton. "Visualizing Data using t-SNE". en. In: *undefined* (2008). URL: https://www.semanticscholar.org/paper/Visualizing-Data-using-t-SNE-Maaten-Hinton/1c46943103bd7b7a2c7be86859995a4144d1938b (visited on 04/13/2022).

[42] Francesco Crecchi et al. "Perplexity-free Parametric t-SNE". In: *arXiv:2010.01359 [cs, stat]* (Oct. 2020). arXiv: 2010.01359. URL: http://arxiv.org/abs/2010.01359 (visited on 04/17/2022).

[43] Laurens van der Maaten. "Learning a Parametric Embedding by Preserving Local Structure". en. In: *Proceedings of the Twelth International Conference on Artificial Intelligence and Statistics*. ISSN: 1938-7228. PMLR, Apr. 2009, pp. 384–391. URL: https://proceedings.mlr.press/v5/maaten09a.html (visited on 04/10/2022).

[44] Leland McInnes, John Healy, and James Melville. "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction". In: *arXiv:1802.03426 [cs, stat]* (Sept. 2020). arXiv: 1802.03426. URL: http://arxiv.org/abs/1802.03426 (visited on 04/17/2022).

[45] Gabriela Hristescu and Martin Farach-Colton. "Cluster-Preserving Embedding of Proteins". In: *Tech rep* (July 2001).

[46] Ella Bingham and Heikki Mannila. "Random projection in dimensionality reduction: applications to image and text data". In: *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. KDD '01. New York, NY, USA: Association for Computing Machinery, Aug. 2001, pp. 245–250. ISBN: 978-1-58113-391-2. DOI: 10.1145/502512.502546. URL: https://doi.org/10.1145/502512.502546 (visited on 04/15/2022).

[47] Christos Faloutsos and King-Ip Lin. "FastMap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets". In: *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*. SIGMOD '95. New York, NY, USA: Association for Computing Machinery, May 1995, pp. 163–174. ISBN: 978-0-89791-731-5. DOI: 10.1145/223784.223812. URL: https://doi.org/10.1145/223784.223812 (visited on 12/27/2021).

[48] V. Athitsos et al. "Boostmap: a method for efficient approximate similarity rankings". en. In: *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.* Vol. 2. Washington, DC, USA: IEEE, 2004, pp. 268–275. ISBN: 978-0-7695-2158-9. DOI: 10.1109/CVPR.2004.1315173.

URL: http://ieeexplore.ieee.org/document/1315173/ (visited on 12/24/2021).

[49] Vassilis Athitsos et al. *Learning Euclidean Embeddings for Indexing and Classification:* en. Tech. rep. Fort Belvoir, VA: Defense Technical Information Center, Apr. 2004. DOI: 10.21236/ADA461760. URL: http://www.dtic.mil/docs/citations/ADA461760 (visited on 12/24/2021).

[50] V. Athitsos et al. "BoostMap: An Embedding Method for Efficient Nearest Neighbor Retrieval". en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 30.1 (Jan. 2008), pp. 89–104. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2007.1140. URL: http://ieeexplore.ieee.org/document/4359304/ (visited on 02/18/2022).

[51] Alexander Hermans, Lucas Beyer, and Bastian Leibe. "In Defense of the Triplet Loss for Person Re-Identification". en. In: *arXiv:1703.07737 [cs]* (Nov. 2017). arXiv: 1703.07737. URL: http://arxiv.org/abs/1703.07737 (visited on 12/24/2021).

[52] Mohammed Waleed Kaduos. *High-quality recordings of Australian sign language signs.* URL: https://kdd.ics.uci.edu/databases/auslan2/auslan.data.html (visited on 04/11/2022).

[53] Pete Warden. "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition". In: *arXiv:1804.03209 [cs]* (Apr. 2018). arXiv:

1804.03209. URL: http://arxiv.org/abs/1804.03209 (visited on 04/11/2022).

[54] Nicolas Pugeault. *ASL Finger Spelling Dataset*. URL: https://empslocal.ex.ac.uk/people/staff/np331/index.php?section=FingerSpellingDataset (visited on 04/11/2022).

[55] L. V. D. Maaten, E. Postma, and J. Herik. "Dimensionality Reduction: A Comparative Review". en. In: *Journal of Machine Learning Research* (2009). URL: https://www.semanticscholar.org/paper/Dimensionality-Reduction%3A-A-Comparative-Review-Maaten-Postma/2309f7c5dad934f2adc2c5a066eba8fc2d8071ec (visited on 04/10/2022).

[56] Wei Dai et al. "Very Deep Convolutional Neural Networks for Raw Waveforms". en. In: *arXiv:1610.00087 [cs]* (Oct. 2016). arXiv: 1610.00087. URL: http://arxiv.org/abs/1610.00087 (visited on 04/18/2022).