CSC 874 Big Data Term Project

Visual learning and hand sign recognition using a fast nearest neighbor algorithm

December 16, 2022

Maxence Cabiddu

Table of Contents

Abstract

The project we are going to talk about in this paper is on the topic of hand sign detection using a specific nearest neighbors algorithm. The National Center for Health Statistics estimates that 28 million Americans have some degree of hearing loss [1]. It is estimated that 500,000 people use sign language as a natural language in the United States [1].
This represent a significant part of the population. That's why I told myself that this project corresponded to what I wanted to do and that it was feasible using the Nene and Nayar algorithm [2] which we will discuss.

In this article, I present the Fast Nearest Neighbor (FNN) algorithm to attempt a transition to computer-aided sign language understanding. Although the performance of FNN in sign detection is sufficient to provide context in decision making, image preprocessing steps need to be improved to be used in real time, such as feature detection to detect hand and analyze it.
The algorithm proposed in this article only classifies images that have already been pre-processed to match the model's expectations. Which is the recognition of signs. We will see the future possibilities that we can associate with this algorithm.

Note that all the tests have been made on an Apple MacBook Pro with the Apple silicon M1 Pro chip.

Introduction

The world of technology and communication has seen a number of advancements in recent years, particularly in the realm of hand sign recognition.

Hand sign recognition is a technology used to identify and interpret hand gestures made by a person.

Recognizing sign language using artificial intelligence has the potential to make communication more accessible to those who are deaf or hard of hearing, as well as bridge the language barrier between sign language users and non-sign language users.
By enabling people who communicate using sign language to communicate more easily with people who do not understand sign language, recognizing sign language using AI can open up many more opportunities for employment for those who use sign language, and create applications and services that can help deaf and hard of hearing people better understand and interact with the world around them. In short, recognizing sign language using AI can bring more inclusivity and accessibility, benefiting not only those who are deaf or hard of hearing, but society as a whole.

It is a challenging task due to the complexity and variety of objects, actions, and gestures that must be recognized. The main challenge lies in accurately classifying the input image and associating it with the correct class.
Additionally, the data used in training models must be varied and comprehensive, making it difficult to achieve a high degree of accuracy.
The algorithms used must be robust enough to handle different lighting and environmental conditions encountered in real-world scenarios. Finally, the task of hand sign recognition is challenging due to the wide range of shapes, sizes, and orientations of the hand signs.

Traditional hand sign recognition methods rely heavily on feature extraction techniques, which are computationally expensive and may not yield the desired accuracy.

In this paper, we will discuss the merits of using a fast nearest neighbor search technique for hand sign recognition.
We will also discuss its potential applications and limitations.

Hand sign recognition is an important area of research, and existing methods have been developed to address this problem. In this paper, we provide an overview of the existing hand sign recognition methods, followed by a discussion on the principles of fast nearest neighbor search and its application to hand sign recognition.

We then compare the performance of fast nearest neighbor search to other existing methods, and provide a detailed discussion on the potential applications of fast nearest neighbor search in hand sign recognition, as well as its current limitations.

Proposed problem formulation

Deaf people face difficulty in communication due to the lack of auditory signals to understand spoken language. This can be problematic as it limits their ability to interact with the hearing world. Hand sign recognition technology could provide a solution to this problem by providing a way for deaf people to interact with the hearing world through visual hand signs.

The objectives on this project is to develop a fast nearest neighbor algorithm for visual learning and hand sign recognition. The algorithm should be able to accurately recognize hand signs in a variety of conditions, including varying lighting conditions, different angles, and different hand sizes. Furthermore, the algorithm should be able to process the input quickly, such that it can provide an accurate result within a few seconds.

The dataset used in this paper come from Kaggle [3]. It is a simple MNIST which allows to test the algorithm on hand signs detections. The model should be able to accurately classify a labeled image into one of the possible classes. The goal of this research is to develop a fast nearest neighbor algorithm for visual learning and hand sign recognition that is both efficient and accurate. The algorithm should be able to reduce the dimensionality of the input data and use a fast nearest neighbor search to find the closest point in the reduced space.

High-Level description

The idea of using a fast nearest neighbor algorithm to classify hand signs was developed by combining two existing technologies: image preprocessing and nearest neighbor algorithms. Image preprocessing is the process of preparing an image for computer vision algorithms. This involves a variety of techniques such as image resizing, color space conversion, noise reduction, contrast enhancement, and more. The goal of preprocessing is to make the image easier to interpret by the computer vision algorithms.
Here, the technique used is to reduce the dimensionality of data, while nearest neighbor algorithms are used to classify data points based on their similarity to other data points.
The first step in developing this idea was to identify a suitable preprocessing technique. The algorithm develop in this project is a fast nearest neighbor (FNN) based on Nayar and Nene high dimensionality nearest neighbor algorithm [2]. FNN is a simple powerful algorithm that classifies data points based on their similarity to other data points. It works by searching nearest neighbors of a given data point inside a reduced hyper-cube among every data points and then assigning the data point to the class that is most common among its neighbors. Once the nearest neighbor algorithms were identified, I began to develop it for hand sign recognition. The algorithm was designed to be more efficient than other lazy algorithms in high dimensionality. To achieve this, I was searching a solution to improve the accuracy. In their research, Nene and Nayar have used a technique called eigenspace to reduce dimensionality. I decided to follow their by reducing the number of dimensionalities. The dataset images was 28*28 which mean 784 pixels (dimensions). I was looking for a reduction algorithm and I have decide to use Principal Component Analysis (PCA) from the Sklearn library. PCA is a powerful technique for reducing the dimensionality of data by transforming it into a new set of variables that capture the most important features of the data. This transformation allows the data to be represented in fewer dimensions, which can then be used to classify the data more efficiently. This allowed the algorithm to classify the data more quickly and accurately.
The algorithm was then tested on this MNIST dataset. The results showed that the algorithm was able to accurately classify the hand signs with a high degree of accuracy. This demonstrated that the algorithm was able to effectively classify hand signs using the fast nearest neighbor algorithm.

The construction of this algorithm was not an easy thing. Especially because of the age of the paper and the complex data structure. On the other hand, the techniques used remain very useful and interesting to study, in particular for their simplicity compared to other recent algorithms such as neural networks.
We will see in detail how this algorithm is interesting and what can be improved so that we can use it in real time.

Previous Work

The K-Nearest Neighbors

The first thing to understand was the K-Nearest Neighbors (KNN) algorithm. The K-Nearest Neighbors is an algorithm used for supervised learning. It is a non-parametric, lazy learning algorithm that stores all available cases and classifies new cases based on a similarity measure (e.g., distance functions). KNN has been used in statistical estimation and pattern recognition already in the beginning of 1970's as a non-parametric technique. The algorithm is used in many applications such as economic forecasting, data mining, and image recognition.

KNN works by finding the k-nearest neighbors of a given data point. The number of neighbors (k) is a user-defined parameter. The algorithm then uses the data points of the k-nearest neighbors to determine the class of the given data point. This is done by taking the majority vote of the k-nearest neighbors. For example, if k = 3 and two of the three nearest neighbors are of class A and one is of class B, then the given data point is classified as class A.

One of the main problems with KNN is that it can be computationally expensive. This is because the algorithm must calculate the distance between the given data point and all other data points in the dataset. This can be especially problematic when dealing with large datasets. Another problem with KNN is that it can be sensitive to outliers. This is because the algorithm takes the majority vote of the k-nearest neighbors, so if there are outliers in the dataset, they can have a large influence on the classification of the given data point. Finally, KNN can be prone to overfitting. This is because the algorithm takes the majority vote of the k-nearest neighbors, so if the dataset is not representative of the population, the algorithm can be biased towards the training data.

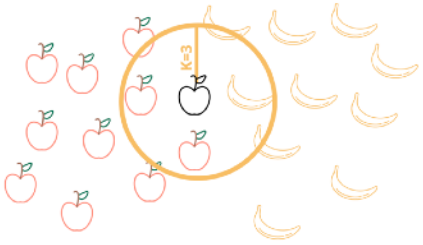

Figure 1: A simple example of a KNN using a size of k=3. Here we are trying to determine the class of the new fruit. 2 of the nearest neighbors are apple and 1 is a banana. The new fruit is labeled as an apple.

These problems lead us to think that a simple KNN is not enough to answer our problem. The development of the FNN therefore remains the best option.

Now that the decision of the algorithm is made, we must decide how to choose the dataset, should we create a tailor-made one or even which images to choose.

Choose a dataset

The decision of the dataset for a machine learning problem is important because it directly affects the accuracy and performance of the model. The dataset should be chosen carefully to ensure that it contains enough data points to accurately represent the problem and that it is of high quality. Poorly chosen datasets can lead to inaccurate models and poor performance. Additionally, the dataset should be chosen to ensure that it is representative of the problem and that it contains enough data points to accurately represent the problem.

To answer our problem, I hesitated between 2 data sets. The first, the one that was finally chosen, comes from the Kaggle platform. This is a mnist designed for hand sign recognition. That is to say that all the images have the same format, the same number of pixels and the same color gradient, here in black and white. It contains 34,627 images with a resolution of 28*28 or 784 pixels.
The second dataset comes from the Arxiv platform [4]. This dataset named HaGRID is a 716 GB dataset. It contains 552,992 images with a resolution of 1920*1080. In addition, each image must be preprocessed so as to crop the correct hand. This preprocessing requires an algorithm in its own right and therefore does not correspond to the objectives of this research. On the other hand, this dataset could be the objective of other research in order to be used with the algorithm developed here.

The Algorithm

Searching by Slicing

The figure 2 from Nayar and Nene paper [2] represent the working process of the algorithm. It simply slice the dataset by the size of 2 epsilon in each dimensions. The remaining points in the hypercube are then used to find the closet one from our new point. A 3 dimensional space have been used because it is easier to represent but any number of dimensionality can be used with this algorithm.

The FNN algorithm is a class that implements a fast nearest neighbor search algorithm. The constructor takes as input the training and testing data, the number of components for the PCA, and whether the data needs to be normalized. It initializes the attributes of the class, such as the sorted data in orderedSet, the mapping from the pointSet to the orderedSet in fmap, and the mapping from the orderedSet to the pointSet in bmap. The closet function finds the closest point to the given point p within 2 epsilon by first trimming the list of points and then performing an exhaustive search on the remaining points. Finally, it returns the label of the closest point. The score function is used to return the accuracy of the model.

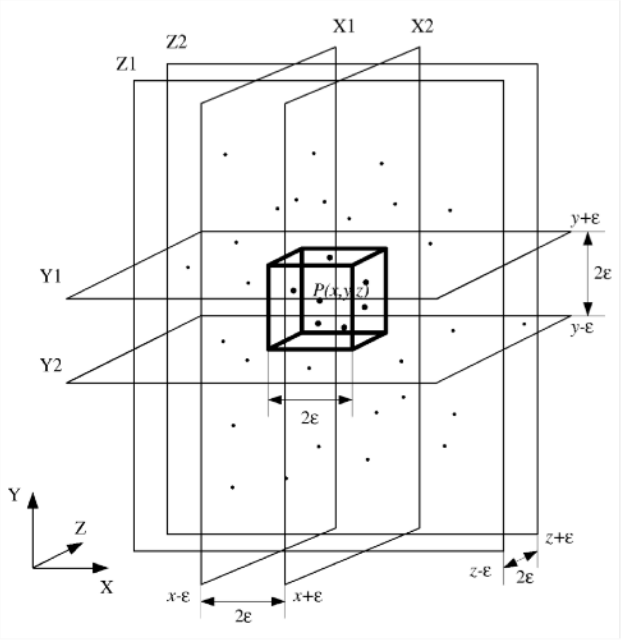

Figure 2: The proposed algorithm efficiently finds points inside a cube of size 2e around the novel query point **Q**. The closest point is then found by performing an exhaustive search within the cube using the Euclidean distance metric.

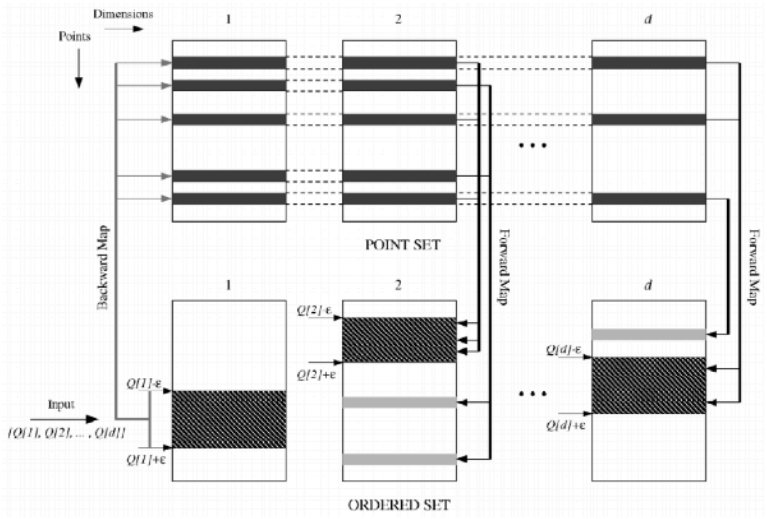

Figure 3: Data structures used for constructing and trimming the candidate list. The point set corresponds to the raw list of data points, while in the ordered set each coordinate is sorted. The forward and back- ward maps enable efficient correspondence between the point and ordered sets.

Data Structure

The data structure is the most important part of the algorithm.
The ordered set is simply a point set copy sorted. The forward make the link from the ordered set to the point set, and the backward map, make the link from the point set to the ordered set.

Example:
O => ordered set
P => point set
F => forward map
B => backward map
O[i][F[i][j]] = P[i][j]
P[i][B[i][j]] = O[i][j]

The construction is made by sorting each dimension while keeping the indexes in forward and backward map. The example show you how to retrieve the point set value from the ordered set value by using the forward map and how to retrieve the ordered set value from the point set value by using the backward map.

Figures 2 and 3 are taken from the paper of Nayar and Nene [2]

Experimental evaluations

Dataset

The figure 4 represent an image from the dataset. We can see that the picture is really clean, there is no specific background noise or any other objects who can create noise. The dataset can be used directly.

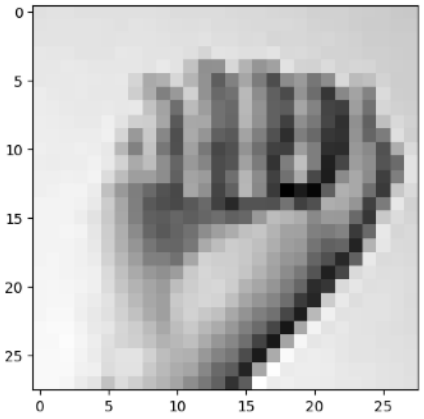

Figure 4: Sample of the letter A from the dataset.

The dataset used is clean and well balanced, meaning that it does not contain any missing values and no preprocessing is needed. This is beneficial as it allows us to quickly and accurately analyze the data without having to spend time and resources on preprocessing. Additionally, a well-balanced dataset ensures that the results of the analysis are not skewed by any outliers or irregularities in the data.
Moreover, since the dataset only contains pixels, exploratory data analysis is not necessary here.

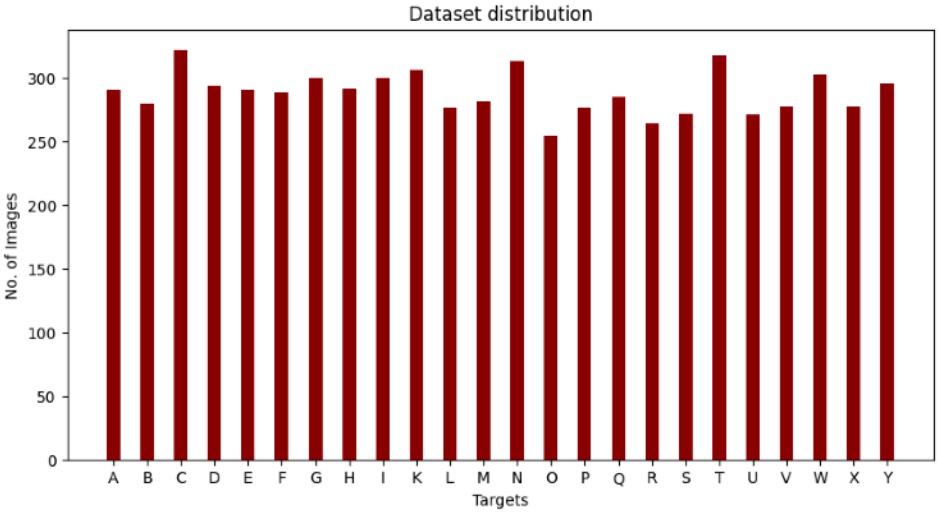

Figure 5: Dataset distribution by number of images.

Hand sign recognitions

The first test was done using the 784 dimensionality of the images and the calculation time was very long but despite that, the result was not too bad. To improve the calculation time, I used the principal component analysis technique to reduce the number of dimensions to 30 while keeping a correct image.
When I retrieved the dataset it was already split into training and test set. Because of that, the results I got weren't very good. So I decided to mix the 2 datasets to make one big and mix it to separate again.
I tried to normalize the data and to do this I divided the pixels of my image by 255 and from there I thought the predictions were going to be better but ultimately normalization doesn't seem necessary for this dataset.

Therefore, the use of this algorithm on this dataset has been pretty easy. The test size was set to 20%, which means that 6926 images were used for the test set. This allowed the algorithm to quickly find the closest neighbors to each image in the test set. The algorithm was able to quickly identify similar images in the dataset, which can be used for various applications such as image recognition and classification.

The algorithm work in 2 part. The first part is the « training », or the « preprocessing » part, it mean that the algorithm is indexing all the training images by sorting and creating lists. The execution time of this part is rather long since it lasts 18 min and 49 sec for 27701 images. The second part is the prediction which is

really fast. The 6926 images are processed in 1 min 14 sec which means that a little more than 51 images are processed per second.
This result is obtained with an epsilon of size 1 and the precision depends on the shuffling of the dataset but it is always close to 0.99.

The results obtained here are very interesting for real-time use, the algorithm could easily process 30 frames per second.

The advantage of this algorithm is that once the data structure is created, the model can be used again and again to quickly classify new images. This is a big advantage over other lazy algorithms like KNN which need to recalculate all distances for each new point.

Figure 6: Sample of the dataset used

Results

Confusion Matrix

This confusion matrix represents the classification result of the model. We can see some minor errors in the classification like 6 errors for the M while P contains none. There is no specific problem represented by this matrix that we can analyze to improve performance.

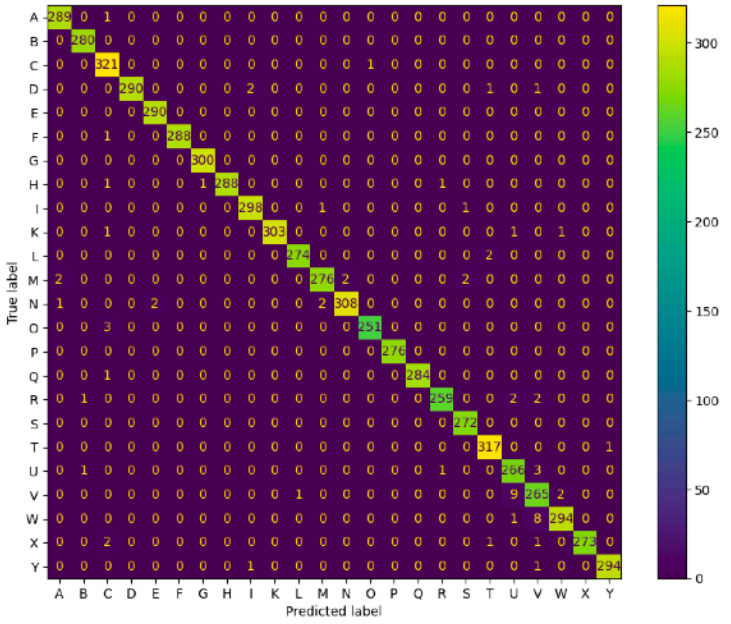

Figure 7: Confusion matrix of the FNN obtained during the tests.

Classification Report

This classification report provides a summary of the performance of a model on a set of test data. It includes the precision, recall, and F1-score for each class, as well as the overall accuracy and macro-averaged scores. The precision score indicates the proportion of true positives out of all predicted positives, while the recall score indicates the proportion of true positives out of all actual positives. The F1-score is a combination of the precision and recall scores, and is a measure of the model's overall accuracy. The macro-averaged scores provide an overall measure of the model's performance across all classes.
We can see that the f1 score is quite close to 1 in all cases, demonstrating that all classes are well ranked.

	precision	recall	f1-score	support
A	0.9897260273972601	0.996551724137931	0.993127147766322	290.0
B	0.992907801418439	1.0	0.996441281138789	280.0
C	0.969788519637462	0.996894409937888	0.9831546707503831	322.0
D	1.0	0.9863945578231291	0.993150684931506	294.0
E	0.993150684931506	1.0	0.9965635738831611	290.0
F	1.0	0.9965397923875431	0.9982668977469671	289.0
G	0.9966777408637871	1.0	0.998336106489184	300.0
H	1.0	0.9896907216494841	0.99481865284974	291.0
I	0.9900332225913621	0.9933333333333331	0.991680532445923	300.0
K	1.0	0.990196078431372	0.9950738916256151	306.0
L	0.9963636363636361	0.9927536231884051	0.9945553539019961	276.0
M	0.9892473118279571	0.9787234042553191	0.9839572192513371	282.0
N	0.9935483870967741	0.9840255591054311	0.9887640449438201	313.0
O	0.996031746031746	0.9881889763779521	0.9920948616600791	254.0
P	1	1	1	276
Q	1.0	0.9964912280701751	0.9982425307557111	285.0
R	0.9923371647509571	0.9810606060606061	0.986666666666666	264.0
S	0.9890909090909091	1.0	0.9945155393053011	272.0
T	0.9875389408099681	0.9968553459119491	0.992175273865414	318.0
U	0.9534050179211471	0.9815498154981551	0.9672727272727271	271.0
V	0.9430604982206401	0.9566787003610101	0.949820788530465	277.0
W	0.98989898989899	0.97029702970297	0.98	303.0
X	1.0	0.9855595667870031	0.9927272727272721	277.0
Y	0.996610169491525	0.9932432432432431	0.9949238578680201	296.0
accuracy			0.9898931562229281	6926.0
macro avg	0.989975698681003	0.9897928215109544	0.9898470656823504	6926.0
weighted avg	0.9900164714806091	0.9898931562229281	0.9899172192843747	6926.0

Figure 8: Classification report obtained during the tests.

Conclusions

This dataset is not enough general to be used for any project, but it does give us a good overview of what is possible to do with this algorithm. It shows us that the algorithm is capable of accurately classifying images into different categories, and that it can be used to identify objects in images. It also gives us a good idea of how the algorithm works and how it can be used to improve image classification tasks. Overall, this dataset provides us with a good starting point for further exploration and experimentation with this algorithm.
Moreover, this experiment showed the efficiency of the algorithm and its speed. This is something that was expected but it is a success to have seen it. The next step will be to effectively introduce real-time recognition.
I started working on it using Mediapipe [5] and OpenCV [6] and the results are promising. I would base my work on the video of @murtazasworkshop [7] to integrate the FNN. The goal should be to use the media pipe algorithm to detect hand, crop the image and classify it using the FNN.

References

(1) Startasl: https://www.startasl.com/american-sign-language/
(2) S.K. Nayar and S.A. Nene « A Simple Algorithm for Nearest Neighbor Search in High Dimensions »
(3) Kaggle dataset: https://www.kaggle.com/datasets/datamunge/sign-language-mnist
(4) ARXIV dataset: https://arxiv.org/abs/2206.08219
(5) Mediapipe: https://mediapipe.dev/
(6) OpenCV: https://opencv.org/
(7) @murtazasworkshop: https://youtu.be/wa2ARoUUdU8