# KNN Optimization for ASL: Variable K Values and Distance Metrics Analysis

Matt Pettit PTTMAT005
Holly Lewis LWSHOL001

May 16, 2024

## 1 Problem Statement

Effective communication is essential for bridging the gap between the hearing and deaf communities. While lipreading allows many deaf people to understand spoken language, expressing themselves to someone unfamiliar with sign language can be more challenging. This project aims to design and implement a machine-learning system capable of accurately classifying individual hand signs from static images representing the 24 letters of the ASL alphabet to try bridge this gap. The letters i and j will not be included as they require movement. This system has the potential to serve as a foundation for applications like real-time sign language translation or educational tools.

This report aims to first detail the implementation of a K Nearest Neighbour image classification algorithm and then to determine the best hyperparameter K as well as distance formula for this application. This report defines ”best” as producing the highest result in accuracy, F1 score, and Jaccard index.

This report hypothesised that for the K Nearest Neighbours, hereafter KNN, algorithms used, a chosen value of K between 4 and 10 will produce the best performance in terms of accuracy, F1-score and Jaccard index. These values are hypothesised as they are likey to balance the trade off between underfitting (large K) and overfitting (small K) given the moderate size of the dataset. Furthermore, this report hypothesised that the Euclidean distance formula will produce the best results for this classification due to its widespread use and proven success in similar applications.

## 2 Literature Review

### 2.1 Background

Hearing loss and speaking impairments rank among the most common disabilities worldwide [1]. Currently, sign language interpreters play a critical role in bridging the communication gap between the hearing impaired and the hearing community, however this is often unrealistic because of financial concerns or simply a lack of translators [1]. Sign Language recognition technology should be playing a large role in bridging the divide however it is nearly 30 years behind speech recognition [2]. One of the reasons for this is the complexity associated with processing video signals in comparison to audio signals [2]

### 2.2 Sensor Based Approaches

A common approach to sign language recognition is to use sensors to detect the movement of a person’s hands and fingers. This implementation is used by Zinah Raad Saeed et al.[3] who developed smart gloves that analyses the movement. However, these gloves cost around $65 which makes them inaccessible to a large amount of people who need them. Furthermore, this method may feel invasive, uncomfortable or isolating due to the requirement of constantly wearing the gloves.

## 2.3 Machine Learning Approaches

A variation on this approach was taken by Wang and Popovic [4] who also used gloves, not for detecting movement with sensors, but to act as markers to improve the machine learning's image analysis. Specifically, they used KNN to recognise coloured markers on the gloves thus improving recognition of the hand's position. However, since this approach also uses gloves it still has the disadvantage of being invasive and uncomfortable. Furthermore, rolling this out on a large scale would require mass distribution and manufacturing of marked gloves. This means that machine learning algorithms that are based on bare-hand use have the advantage of being more cost-effective, more easily distributed and more normal for sign language users [2]. KNN is not the only machine learning implementation that has been developed, there are various methods using Neural networks. The earliest of which was developed by Kouichi Murakami and Hitomi Taguchi[5] which used the neural network algorithm to recognise static hand signals in 1991 [2]. A more recent implementation of this has been done by Akmeliawati et al. [6] who managed to achieve a 96% accuracy for 13 different gestures. Their implementation only requires a single camera and a device to run the software which makes it extremely financially viable to distribute[6]. Another notable implementation used the Harr Cascade Classifier to classify Indian sign language [7]. It functioned by extracting frames from webcam video input and could correctly identify 5 different phrases with different lighting and multiple hands with an accuracy of 92.68% [7].

## 2.4 Isolated Vs Continuous Sign Recognition

A big limitation of numerous implementations is the dependence on still frames or isolated signs. For example, Pugeault and Bowden [8]'s implementation was able to recognise all 26 ALS alphabet characters with an accuracy of 75% but their system required the signs to be isolated with pauses in between. Successful implementations have been developed like Dominio et al. [9] whose system correctly recognised signs with a 97.6% accuracy without clear pauses.

## 2.5 KNN Distance Metrics

KNN has been used to varying success in numerous applications. For example, Kai Sun et al. [10] utilised this algorithm to classify facial images. D S Guru et al. [11], utilised KNN to classify flower images. Ferid Bajramovic et al. utilized the algorithm for object recognition [12] A review by Surya Prasath et al. [13] compared the use of different distance metrics when working with KNN, their results show the effects on accuracy, precision, and recall with different distance metrics. Their analysis of numerous works found that there is no overall optimal metric, however, one will achieve similar results when using related equations [13]. Furthermore, they found that when using any realistic distance formula (top 10) the KNN model only degraded around 20% with a 90% noise level. Punam and Nitin [14] found that the Manhattan distance equation performed better in terms of accuracy, sensitivity and specificity in comparison to Chebychev and Euclidian. Similarly, Lopes and Ribeiro [15] found that Euclidean and Manhattan performed significantly better than Canberra, Chebychev and Minkowsky.

## 2.6 KNN Hyperparameter Choice

Selecting the optimal value for K is essential to maximising the model's performance. Large K values can result in underfitting and an overall increase in approximation error however small values of K can result in overfitting due to noisy data [16]. A solution to this issue was proposed by Zhang et al. [17], in which the KNN was enhanced by learning different K values for each test data point based on prior knowledge. This proved to be more robust against noisy datasets. A similar implementation proposed by Han et al. [18] used an adaptive K value, which takes into account both the majority and the second majority classes among the K Nearest Neighbours. A different approach suggested by Liu et al. [19] uses a weighted KNN algorithm to determine local K values for each test data point from the training data and assigns a weight to the neighbours based on the distances between them and using a majority rule to decide on a final class. Finally one can simply set the K value to a constant value before running the tests. As mentioned in a report by Zhang et al. [20], Wettschereck suggested that setting the K value to 1 is useful in 3 situations: if the training set is small, if the task is noise free with little overlap between decision boundaries or if the data is distributed in a highly non-Gaussian manner. However, no ideal K value exists that guarantees the best performance across all tests [20].

# 3 Design and Theory

## 3.1 Theory of K Nearest Neighbour

KNN is a simple and widely used method of classification. In fact, it is ranked in the top 10 models in data mining [13]. Furthermore, it requires no assumptions about the underlying data distribution and therefore becomes a great choice since the project involves no previous knowledge about the distribution of the data [13]. It is a type of supervised algorithm meaning that it has access to the ground truth during its operation. The formalised notation for the labelled training set that the algorithm has access to is as follows:

$$D = (x^{[i]}, y^{[i]}), ..., (x^{[n]}, y[n])$$

Where x and y represent the data and ground truths respectively.It is however described as a " lazy algorithm": as rather than producing a model that can accurately predict the output it stores the training data and compares it with the inputted unseen data to produce a result. It is therefore technically not "learning" anything as without the stored training data it is unable to make a prediction. This does however make the implementation easier and, therefore, it is still a popular choice. KNN operates by scanning the training data of N values to find the K most similar examples to its input. It achieves this by calculating a difference metric (usually euclidean distance) between the input and the different training examples. For an image, this would be done by converting the image into vectors on a multidimensional plane and then calculating a distance metric between the pixel of the training image and the pixel of the input image for every pixel. This operation can be intensely computationally expensive, it is therefore important to use lower-resolution images if speed is required as adding more resolution will exponentially increase the processing needed and the time taken.

From the set of K training results similar to the input, it finds the modal class and predicts this as the classification.

$$h(x^{[t]}) = mode(y^{[i]}, ..., y^{[k]})$$

## 3.2 Distance metrics

KNN functions by finding the training data closest to the input. However "closest" can be calculated in various ways. The most common is Euclidean distance and is calculated as follows:

$$euclidian : d\left(x,y\right) = \sqrt{\sum_{i=1}^{n}\left(x_i - y_i\right)^2}$$

However, there are other distance metrics that can be used, for example, Manhattan distance, which is the y distance and the x distance added.

$$manhattan : d\left(x,y\right) = \sum_{i=1}^{n}\left|x_i - y_i\right|$$

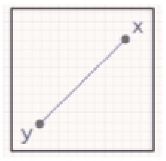

Figure 1: Euclidean Distance diagram

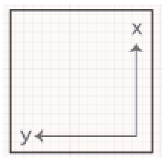

Figure 2: Manhattan Distance diagram

Minkowski is a generalisation of both of these formulas [21]. It can be thought of as the distance between 2 points in a p dimensional space. When p = 2 Minkowski becomes the Euclidian distance and when p = 1 it becomes the Manhattan distance, however, there are an infinite number of p's that can be chosen.

$$Minkowski : d\left(x,y\right) = (\sum_{i=1}^{n} \mid (x_i - y_i) \mid^{p})^{1/p}$$

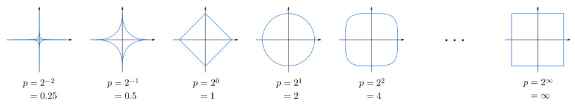

Figure 3: Visulisation of Minkowski with varying P

## 3.3 Data Set

The dataset chosen contains 24 letters of ASL. Letters i and j are excluded as they require movement and therefore would not be able to be identified from static images. In total, the dataset contains 27455 greyscale images with a resolution of 28x28 pixels (784 in total). The dataset can be found on Kaggle. This dataset is an adoption and edited version of another dataset also available on Kaggle. The original has a higher resolution and is in colour. This also offers up interesting expansions of this project, for example finding the optimal resolution of the data or trying the data with different editing like higher contrast or low pass filtering before downsizing. This is all part of image pre-processing that has been found to reduce the error rates [10].

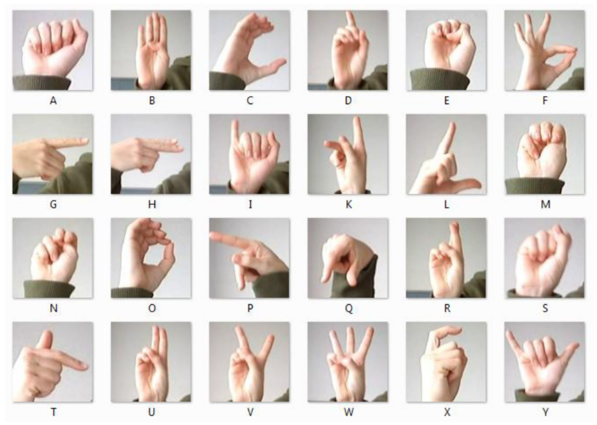

Figure 4: Original Dataset

## 3.4 Design

For this project, we aim to use the Minkowski distance metric with a standard K Nearest Neighbours implementation. By varying the value of p we aim to determine which value will give the best result. For each p value different values for the K hyper parameter will be evaluated in order to determine the best value. We decided to test p values of 1, 2 and 3. These values were chosen as a p value of 1 corresponds to using the Manhattan Distance equation and a p value of 1 corresponds to the Euclidean Distance equation as discussed earlier.

## 3.5 Accuracy Metrics

It is critical to use various accuracy and correctness metrics to evaluate a model to get a holistic view of its performance. Furthermore, this aspect is essential to guide the refinement of the hyper parameters of the model. In our case we aim to first use the standard accuracy formula:

$$\text{Accuracy} = \frac{\text{Number of correct classifications}}{\text{Total number of test samples}}$$

Furthermore, we aim to use the Jaccard index which is a coefficient that measures the similarity between the predicted set and the actual set.

$$J(y_i, \hat{y}_i) = \frac{|y_i \cap \hat{y}_i|}{|y_i \cup \hat{y}_i|}$$

This produces a value between 0 and 1 where 0 is no intersection and 1 is a complete intersection, ie. perfect.

Following this we plan to use the F1 score, which is another common metric used in KNN. It aims to try to balance precision and recall.

$$Precision = \frac{TP}{TP + FP}$$

Where TP is true positive ie. it predicts x and it is x, and FP is false positive, ie. predits x when its not x.

$$Recall = \frac{TP}{TP + FN}$$

Where FN is false negative, it predicts another letter when it is x.

The F1 score is then calculated as follows:

$$F1Score = 2\left(\frac{Precision \cdot Recall}{Precision + Recall}\right)$$

This produces a value between 0 and 1 where 1 is perfect.

Finally, we aim to use a confusion matrix to identify where the algorithm is struggling. A confusion matrix is useful as it not only shows incorrect guesses but also what the incorrect guess was.

### 3.6 Data Split

When testing any machine learning algorithm the test data and the training data must be kept separate. With KNN this is even more important, as it does not have a training stage and therefore directly checks the "training" data each time. For example, if K had a value of 1 it would find the exact image and it would be its nearest neighbour as the distance would be 0. It would then choose this as the modal class. Although this would get 100% accuracy as soon as the algorithm was tested on real data it would perform significantly worse. This is called an information leak. In our case, the downloaded dataset has been pre-split to ensure no information leaks

## 4 Implementation

All code discussed further is available in the Github Repository. The first two functions defined for the KNN implementation were *most_common()* and *Minkowksi()* which are shown below. The *minkowksi()* function is responsible for finding the distance between two corresponding points in the image. One can think of the "Distance" in this case as the difference in the values of each pixel. The *most_common()* function takes in a list of data points and returns the most common result, in our case this would be a list of labels associated with the K Nearest Neighbours.

```python
def most_common(lst):
    # Return the mode of the set (ie the most common)
    return max(set(lst), key=lst.count)

def minkowski(point, data,dimension):
    # Minkowski distance between points a & data
    return np.sum((np.absolute(point - data))**dimension, axis=1)**(1/dimension)
```

These functions are in turn called by the *knn_predict()* function, shown below. This function takes in the Training Set and Training Set Labels, the Test Set and the value K representing how many neighbours it should use in its processing. The function displays a progress bar if the length of the Test Set is longer than 1, ie when benchmarks of the system are being performed, this is a quality of life improvement that allows the user to see the progress of long sets of predictions.

```
|████████                          | ▄█ 386/1200 [32%] in 3s (~7s, 125.6/
```

The function begins calculating the distances for each sample in the Test set compared to the Training Set, from this set the K Nearest Neighbours are selected, these are sent to the *most_common()* which returns the mode of the set, representing the final prediction of the model, which is finally returned to the user.

```python
def knn_predict(X_train, y_train, X_test, k,p):
    neighbors = []
    # If length of test data is greater than 1, show a progres bar for the user
    if len(X_test) > 1:
        with alive_bar(len(X_test)) as bar:
            for x in X_test:
                # Calculate Distances
                distances = minkowski(x, X_train,p)
                # Append K Nearest Neighbours Based On Lowest Distances
                y_sorted = [y for _, y in sorted(zip(distances, y_train))]
                neighbors.append(y_sorted[:k])
                # Update Progress Bar
                bar()
    else:
        for x in X_test:
            # Calculate Distances
            distances = minkowski(x, X_train,p)
            # Append K Nearest Neighbours Based On Lowest Distances
            y_sorted = [y for _, y in sorted(zip(distances, y_train))]
            neighbors.append(y_sorted[:k])

    return list(map(most_common, neighbors))
```

In order to test and evaluate the system two further functions were written, these are the *knn_evaluate()* and *knn_random_examplesPic()* functions that are shown below. The *knn_evaluate()* is responsible for determining the accuracy of the model, it does this by predicting all the data in the Test set and calculating the percentage of predictions that accurately matched the labels of the Test set. It also returns a list of the predictions for further analysis.

```python
def knn_evaluate(X_train, y_train, X_test, y_test, k,p):
    y_pred = knn_predict(X_train, y_train, X_test, k,p)
    accuracy = sum(y_pred == y_test) / len(y_test)
    return accuracy, y_pred
```

The other function *knn_random_examplesPic()* was designed for the users sake. It chooses a number of random samples from the Test set and predicts labels for these, then it uses Python's matplotlib to plot these images with the actual and predicted labels. This allows the user to get a sense of how the model is performing. This is no substitute for actual testing but was helpful to use during the development process. Finally the the system was given a training dataset of size 12000, approximately 500 images per class used for the classification, and a test dataset of size 7172. An example of the output of the *knn_random_examplesPic()* function is shown after the code.

```python
def knn_random_examplesPic(X_train, y_train, X_test, y_test, k, numExamples,p):
    num_rows = math.ceil(numExamples / 3)
    fig, axes = plt.subplots(nrows=num_rows, ncols=3, figsize=(15, 3 * num_rows))

    for i, idx in enumerate(random.sample(range(5, len(X_test)), numExamples)):
        row, col = i // 3, i % 3
        print(f"Actual: {label_dictionary[y_test[idx]]}")
        x_test_example = X_test[idx].reshape(1, -1)  # Reshape to have a single row
        y_pred = knn_predict(X_train, y_train, x_test_example, k,p)
        print(f"Predicted: {label_dictionary[y_pred[0]]}")

        # Visualize the image
        image = X_test[idx].reshape(28,28)  # Reshape to the correct shape for visualization
        axes[row, col].imshow(image, cmap='gray')
        axes[row, col].set_title(f'Actual: {label_dictionary[y_test[idx]]}\nPredicted: {label_diction
        axes[row, col].axis('off')

    # Visualize the image
    image = X_test[idx].reshape(imagesize,imagesize)  # Reshape to the correct shape for visualizatic
    axes[row, col].imshow(image, cmap='gray')
    axes[row, col].set_title(f'Actual: {label_dictionary[y_test[idx]]}\nPredicted: {label_dictionary[
    axes[row, col].axis('off')

    plt.tight_layout()
    plt.show()
```

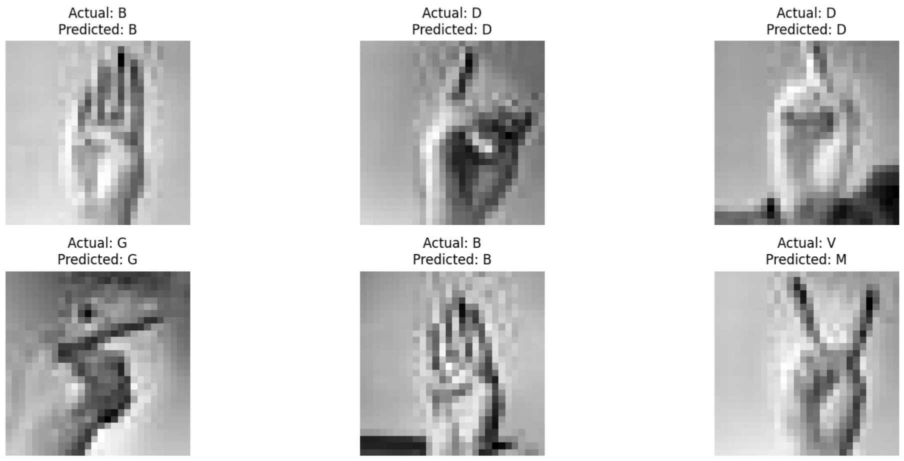

Figure 5: Example run with numExamples = 6

The SciKit Learn Python module was utilized in order to calculate various metrics in order to analyse the performance as discussed in the Section 3.5. These included the precision of the model, the recall, the F1 score and the Jaccard index, 3 of which are shown below in the *calculate_metric()* function:

```
def calculate_metrics(y_test, y_pred):
    precision = precision_score(y_test, y_pred, average='macro')
    recall = recall_score(y_test, y_pred, average='macro')
    f1 = f1_score(y_test, y_pred, average='macro')
    print(f"Precision: {precision}")
    print(f"Recall: {recall}")
    print(f"F1-Score: {f1}")
    return f1
```

## 5 Results

Before the final testing of the system several different values of K were tested in order to determine whether the initial hypothesised range (4 to 10) was useful to test. Values for K = 5, 10, 20, 30, 40 and 50 were tested, which yielded the following accuracy results.

| K Value | Accuracy |
|---|---|
| 5 | 0.781 |
| 10 | 0.752 |
| 20 | 0.716 |
| 30 | 0.687 |
| 40 | 0.666 |
| 50 | 0.646 |

Figure 6: Results for Initial Test

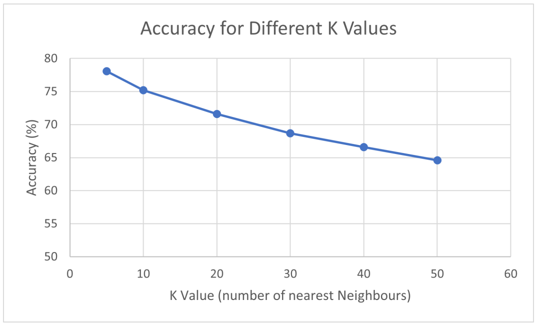

Figure 7: Accuracy vs K Value Plotted

It was clear from the results shown in Figure 6 and 7 that there was an obvious downward trend, therefore the range of K values to be tested was adjusted from 1 to 5. While it would have been preferable to test a large range of K values, for example 1 to 100 this was not practical due to the time required to perform the benchmarks.

For the final testing the system was benchmarked by changing the p variable, representing the number of dimensions used in the Minkowski distance formula from 1 to 3. For each different p value the hyper parameter K was varied from 1 to 5. For each distance formula, ie. p value and each k value, the accuracy, Jaccard, precision, recall and F1 scores were recorded. A confusion matrix was generated for each k value, graphs showing the K value versus accuracy were plotted and lastly graphs showing the Jaccard coefficient for each class with each K value were plotted.

### 5.1 Manhattan Distance, (p = 1)

Below is the table of results for the Manhattan Distance equation. It is clear that the best performing K value was 1, with the highest metrics across the board.

| p = 1 | | | | | |
|---|---|---|---|---|---|
| **K Value** | **Accuracy** | **Jaccard Score** | **Precision** | **Recall** | **F1 Score** |
| 1 | 0.798 | 0.677 | 0.8 | 0.787 | 0.786 |
| 2 | 0.766 | 0.635 | 0.783 | 0.756 | 0.752 |
| 3 | 0.771 | 0.642 | 0.784 | 0.76 | 0.758 |
| 4 | 0.757 | 0.623 | 0.767 | 0.745 | 0.743 |
| 5 | 0.746 | 0.608 | 0.754 | 0.731 | 0.73 |

Table 1: Table of Results for p = 1

Shown below in Figure 8 are the Jaccard Similarity Coefficient Scores per Class. Afterwards in Figure 9 the Confusion Matrix for the best performing K value, 1 is displayed. This is a helpful way of visualising the performance of the algorithm as it allows one to identify where the model is struggling. From both one can observe that the model particularly struggles with over identifying the Letter S, as noted with its poor Jaccard scores, this is also clear from the Confusion Matrix. The model also struggled with actually identifying V, where it often predicted it was a W, once again this is reflected in the poor Jaccard Scores for V and the confusion matrix.

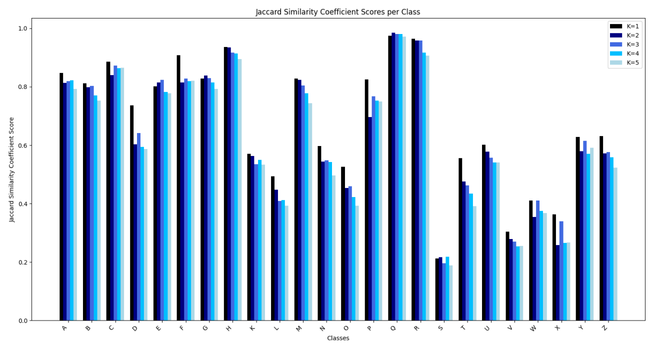

Figure 8: Jaccard Score For Different Classes for p = 1

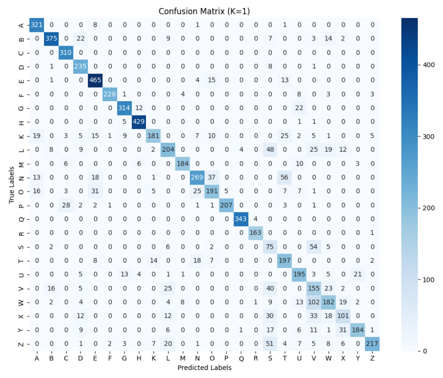

Figure 9: Confusion Matrix for p = 1 and K = 1

## 5.2 Euclidean Distance, (p = 2)

Below is the table of results for the Euclidean Distance equation. Once again the K value of 1 performed the best, with an accuracy of 80.9%

| p = 2 | | | | | |
|---|---|---|---|---|---|
| **K Value** | **Accuracy** | **Jaccard Score** | **Precision** | **Recall** | **F1 Score** |
| 1 | 0.809 | 0.689 | 0.81 | 0.799 | 0.798 |
| 2 | 0.786 | 0.661 | 0.798 | 0.775 | 0.774 |
| 3 | 0.788 | 0.663 | 0.794 | 0.778 | 0.776 |
| 4 | 0.785 | 0.657 | 0.793 | 0.774 | 0.772 |
| 5 | 0.781 | 0.652 | 0.787 | 0.767 | 0.766 |

Table 2: Table of Results for p = 2

Shown below in Figure 10 are the Jaccard Similarity Coefficient Scores per Class. Afterwards in Figure 11 the Confusion Matrix for the best performing K value, once again 1 is displayed. From both one can observe that the model is again struggling particularly with S and V. However it can be observed that X did perform noticeably better, this is apparent in both the confusion matrix and the Jaccard Scores.

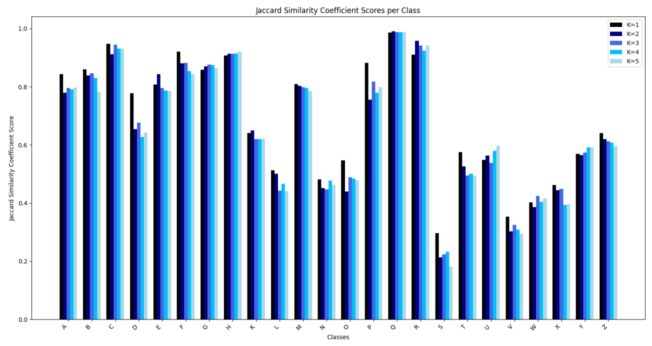

Figure 10: Jaccard Score For Different Classes for p = 2

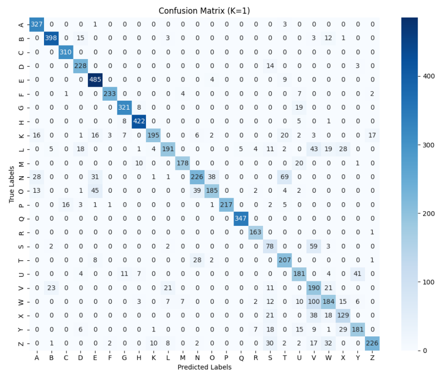

Figure 11: Confusion Matrix for p = 2 and K = 1

## 5.3 Minkowski Distance Equation, (p = 3)

Below is the table of results when P was set to 3.

| p = 3 | | | | | |
|---|---|---|---|---|---|
| **K Value** | **Accuracy** | **Jaccard Score** | **Precision** | **Recall** | **F1-Score** |
| 1 | 0.797 | 0.673 | 0.797 | 0.789 | 0.785 |
| 2 | 0.777 | 0.645 | 0.782 | 0.766 | 0.763 |
| 3 | 0.783 | 0.653 | 0.785 | 0.772 | 0.768 |
| 4 | 0.783 | 0.652 | 0.786 | 0.772 | 0.768 |
| 5 | 0.777 | 0.646 | 0.778 | 0.764 | 0.761 |

Table 3: Table of Results for P = 3

As before Figure 12 shows the Jaccard Scores for the different K values for each class and Figure 13 shows the Confusion matrix for the best performing K value, which was once again K = 1.

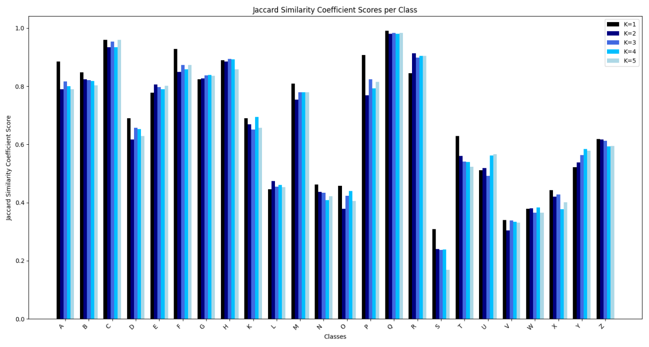

Figure 12: Jaccard Score For Different Classes for p=3

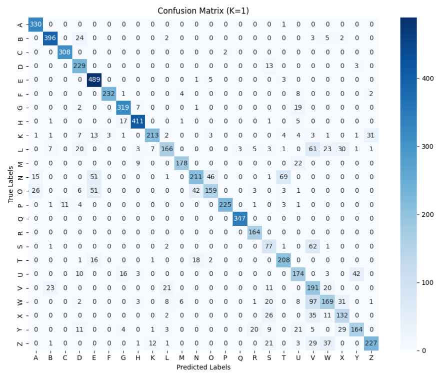

Figure 13: Confusion Matrix for p = 3 and K = 1

## 5.4 Discussion

Overall the the system performed best with p = 2 and K = 1, meaning that the best distance equation in our case was the Euclidean Distance Equation. These hyper parameters also resulted in the best Jaccard, precision, recall and F1 scores. This implementation also had more consistent Jaccard Scores across classes when compared to p=1, ie the Manhattan Distance Equation.

Interestingly the results shown in Figures 12 and 13 are very similar to those in Figures 10 and 11, this tracks with what one would expect based on the theory, the distance algorithms for p=2 and p=3 are more similar than p=2 and p=1 .

All versions of the model seem to struggle on very similar classes, this is somewhat expected as while the distance equations are changing they still offer very similar results in most cases. Based on the Jaccard Scores for the Classes as shown in Figures 8, 10 and 12 the model seemed to struggle on S and V. Evident from the Confusion Matrices shown in Figures 9, 11 and 13 is that it mostly struggles with over classifying classes as S and V. In the case of S, the model seems to often misidentify X, Y and Z as S. In the case of V the model seems to frequently misidentify W, X, S and L as V.

## 6 Future Improvements

There are many limitations of this current implementation that could be addressed in future iterations. Firstly increasing the dataset used would be an obvious start. The dataset was limited to 500 images per class (12000 in total) due to the lengthy time associated with bench marking, but given a more powerful system and possibly utilizing multiprocessing a system with more images per class could be developed. Secondly a comparison of other methods of machine learning such as Convolutional Neural Networks and Recurrent Neural Networks could prove useful as these algorithms may be more capable of achieving a deeper level of nuance than the KNN algorithm used. Thirdly training the system on traditional sign language words may be more useful than on individual letters, however the large number of words makes this very computationally expensive for KNN models.

## 7 Conclusion

In summary it was determined that the KNN implementation performed best at identifying ASL from static pictures when using the Euclidean Distance equation and relying on 1 Nearest Neighbour for classification. This result was contrary to the prediction in the hypothesis in which the ideal value was estimated to be between 4 and 10, however it confirmed the prediction about which distance equation would perform best. The system achieved an overall accuracy of 80.9%, a precision of 0.810, a recall score of 0.799 and an F1 Score of 0.798, the Jaccard Index Score however was lower with a score 0.689. The system was fairly consistent with its ability to predict classes however as demonstrated with the Jaccard Scores for each Class and the Confusion Matrices, some particular classes proved more challenging than others.

One important thing to note was that while the Euclidean Distance equation performed best in our tests, it was by a marginal amount. Therefore while one could conclude that Euclidean is the best, it seems more prudent to conclude that the Euclidean Distance Equation performed similarly to Manhattan, this is in line with what was found in the study by Lopes and Ribeiro [15] as well as the study by Surya Prasath et al. [13].

In conclusion this report achieved the aim of determining the best distance equation and K value for a KNN static image ASL detector as well as creating a successful initial implementation that demonstrates the capability of accurately classifying individual hand signs from static images. This project hopes to serve as a basis for future projects that can further bridge the communication gap between the hearing and deaf communities as well as those who wish to experiment with varying K values and distance equations.

## References

[1] I. Adeyanju, O. Bello, and M. Adegboye, "Machine learning methods for sign language recognition: A critical review and analysis," *Intelligent Systems with Applications*, vol. 12, p. 200 056, 2021, ISSN: 2667-3053. DOI: https://doi.org/10.1016/j.iswa.2021.200056. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2667305321000454.

[2] R. Elakkiya, "Retracted article: Machine learning based sign language recognition: A review and its research frontier," *Journal of Ambient Intelligence and Humanized Computing*, vol. 12, no. 7, pp. 7205–7224, 2021.

[3] Z. R. Saeed, Z. B. Zainol, B. B. Zaidan, and A. H. Alamoodi, "A systematic review on systems-based sensory gloves for sign language pattern recognition: An update from 2017 to 2022," *IEEE Access*, vol. 10, pp. 123 358–123 377, 2022. DOI: 10.1109/ACCESS.2022.3219430.

[4] R. Y. Wang and J. Popović, "Real-time hand-tracking with a color glove," *ACM transactions on graphics (TOG)*, vol. 28, no. 3, pp. 1–8, 2009.

[5] K. Murakami and H. Taguchi, "Gesture recognition using recurrent neural networks," in *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1991, pp. 237–242.

[6] R. Akmeliawati, F. Dadgostar, S. Demidenko, *et al.*, "Towards real-time sign language analysis via markerless gesture tracking," in *2009 IEEE Instrumentation and Measurement Technology Conference*, 2009, pp. 1200–1204. DOI: 10.1109/IMTC.2009.5168637.

[7] K. Dabre and S. Dholay, "Machine learning model for sign language interpretation using webcam images," in *2014 International Conference on Circuits, Systems, Communication and Information Technology Applications (CSCITA)*, IEEE, 2014, pp. 317–321.

[8] N. Pugeault and R. Bowden, "Spelling it out: Real-time asl fingerspelling recognition," in *2011 IEEE International conference on computer vision workshops (ICCV workshops)*, IEEE, 2011, pp. 1114–1119.

[9] G. Marin, F. Dominio, and P. Zanuttigh, "Hand gesture recognition with leap motion and kinect devices," in *2014 IEEE International conference on image processing (ICIP)*, IEEE, 2014, pp. 1565–1569.

[10] K. Sun, H. Kang, and H.-H. Park, "Tagging and classifying facial images in cloud environments based on knn using mapreduce," *Optik*, vol. 126, no. 21, pp. 3227–3233, 2015, ISSN: 0030-4026. DOI: https://doi.org/10.1016/j.ijleo.2015.07.080. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0030402615006324.

[11] D. Guru, Y. Sharath, and S. Manjunath, "Texture features and knn in classification of flower images," *IJCA, Special Issue on RTIPPR (1)*, pp. 21–29, 2010.

[12] F. Bajramovic, F. Mattern, N. Butko, and J. Denzler, "A comparison of nearest neighbor search algorithms for generic object recognition," in *Advanced Concepts for Intelligent Vision Systems: 8th International Conference, ACIVS 2006, Antwerp, Belgium, September 18-21, 2006. Proceedings 8*, Springer, 2006, pp. 1186–1197.

[13] V. Prasath, H. A. A. Alfeilat, A. Hassanat, *et al.*, "Distance and similarity measures effect on the performance of k-nearest neighbor classifier–a review," *arXiv preprint arXiv:1708.04321*, 2017.

[14] M. Punam and T. Nitin, "Analysis of distance measures using k-nearest," *International Journal of Science and Research*, vol. 7, no. 4, pp. 2101–2104, 2015.

[15] N. Lopes and B. Ribeiro, "On the impact of distance metrics in instance-based learning algorithms," in *Pattern Recognition and Image Analysis: 7th Iberian Conference, IbPRIA 2015, Santiago de Compostela, Spain, June 17-19, 2015, Proceedings 7*, Springer, 2015, pp. 48–56.

[16] S. Zhang and J. Li, "Knn classification with one-step computation," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 3, pp. 2711–2723, 2021.

[17] S. Zhang, X. Li, M. Zong, X. Zhu, and D. Cheng, "Learning k for knn classification," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 8, no. 3, pp. 1–19, 2017.

[18] Z. Pan, Y. Wang, and Y. Pan, "A new locally adaptive k-nearest neighbor algorithm based on discrimination class," *Knowledge-Based Systems*, vol. 204, p. 106 185, 2020.

[19] H. Liu, J. Wu, T. Liu, D. Tao, and Y. Fu, "Spectral ensemble clustering via weighted k-means: Theoretical and practical evidence," *IEEE transactions on knowledge and data engineering*, vol. 29, no. 5, pp. 1129–1143, 2017.

[20] X. Zhang and Q. Song, "Predicting the number of nearest neighbors for the k-nn classification algorithm," *Intelligent Data Analysis*, vol. 18, no. 3, pp. 449–464, 2014.

[21] Turing, *How to decide the perfect distance metric for your machine learning model*, https://www.turing.com/kb/how-to-decide-perfect-distance-metric-for-machine-learning-model, Accessed: 11/05/2024, 2022.