UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

COMPUTER SCIENCE HONOURS
FINAL PAPER
2016

Title: **Investigating Machine Learning Classifiers for Sign Language Recognition using the Microsoft Kinect**

Author: **Shaheel Kooverjee**

Project Abbreviation: **HANDGR**

Supervisor(s): **Assoc. Prof. James Gain**
Assoc. Prof. Deshendran Moodley

Category	Min	Max	Chosen
Requirement Analysis and Design	0	20	0
Theoretical Analysis	0	25	0
Experiment Design and Execution	0	20	20
System Development and Implementation	0	15	5
Results, Findings and Conclusion	10	20	20
Aim Formulation and Background Work	10	15	15
Quality of Paper Writing and Presentation	10		10
Quality of Deliverables	10		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	0	10	0
Total marks		**80**	**80**

Investigating Machine Learning Classifiers for Sign Language Recognition using the Microsoft Kinect

Final CSC4000W Project Paper*

Shaheel Kooverjee
University of Cape Town
skooverjee@gmail.com

ABSTRACT

Various motion sensing devices have been used to investigate gesture recognition for sign language. This research investigates the use of certain machine learning methods and their effectiveness in recognition of South African Sign Language alphabet gestures. This is based on data obtained from the Microsoft Kinect V2, a motion sensor primarily intended for gaming, that allows for interaction through gestures or spoken commands. A direct pixel-value method of feature extraction was investigated, along with processing and classification using the OpenCV library. It was found, in terms of machine learning classifiers, that the Support Vector Machine (SVM) was the most accurate classifier, with 75.96% accuracy, when used with a Polynomial kernel of degree 3. The SVM with a Linear kernel, as well as a *k*-Nearest Neighbour algorithm, performed nearly as well. The Multi-Layer Perceptron performed poorly in comparison, with accuracy of less than 50%. In terms of efficiency of the classifiers, prediction times were noted to be quick enough for use in an ideal real-world classification scenario.

KEYWORDS

Classification, hand detection, gesture recognition, Kinect, machine learning, sign language, OpenCV, image processing

1 INTRODUCTION

Gesture recognition deals with the mathematical interpretation of human gestures by a computing device. Typically, gestures originate from the face or hands, and allow for interaction with devices without touching them. This area of research commonly involves machine learning, which allows computers to learn patterns from data and use the learned information to make predictions on new data. Machine learning makes our lives easier in today's data-rich world, and it has been applied in Virtual Assistants and even self-driving cars [2].

One linked area of research is the use of cameras and motion sensors to read and correctly interpret sign language gestures [10]. Various techniques have been applied, primarily based on machine learning, in order to train a system to recognize and classify such gestures correctly. Three of the more frequently used devices for which these algorithms have been implemented are the Myo armband (a gesture recognition device worn on the forearem), the Leap Motion controller (a hand-tracking device used in virtual reality) and the Microsoft Kinect sensor (the focus of this report).

Sign languages typically makes use of different parts and positions of the body, such as the face, arms, hands and body posture. The South African Sign Language (SASL) alphabet, however, is recognised and gestured by only the use of the hand in various positions, as depicted in Figure 1. 24 of the gestures are static, meaning that no hand movement is needed to generate them, while 2 of the gestures (**j** and **z**) are dynamic, requiring the hand to be moved during gesture performance.

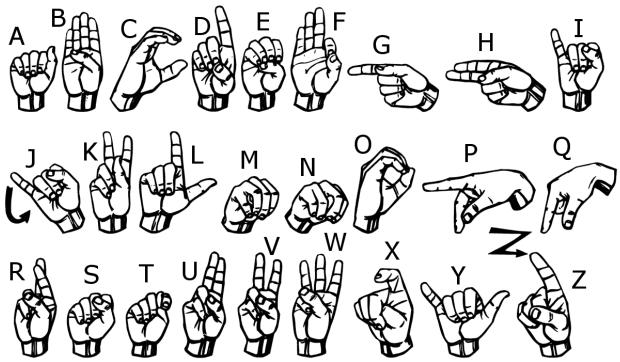

Figure 1: Gestures of the SASL alphabet

The overall project is based on evaluating selected algorithms (which have been reported to work efficiently and accurately in previous sign language gesture recognition research) for each of the three motion-sensing devices, in terms of gesture recognition. Investigating this in terms of the SASL alphabet is intended as a small step towards development of a system to assist in breaking down the communication barrier between the Deaf and the hearing. This paper focuses on the use of the Microsoft Kinect V2, in particular, together with the Open Source Computer Vision Library (OpenCV) for C++.

This study thus aims to find, for the Microsoft Kinect (V2), which machine learning classifier is best suited to gesture recognition in the context of the SASL alphabet. Here we define 'best' mostly in terms of accuracy of the classifier *i.e.* the percentage of gestures correctly categorised by the classifier, as well as the times taken for training and prediction by the classifiers.

2 MICROSOFT KINECT BACKGROUND

The Kinect, pictured in Figure 2, is a motion sensing device mainly used in conjunction with Microsoft Xbox consoles. However, it is also compatible with Microsoft Windows computers, and this, along with the available online software development kit (SDK), has made research possible in other non-gaming contexts that require sensors for gestures and/or spoken commands.

A special feature of the Kinect is its depth sensor, which is particularly useful when it comes to gesture recognition and distinguishing the human body from a busy background. The Kinect effectively provides a relatively cheap and easily available depth sensor [13, 17, 21] that projects an infrared light pattern to provide a depth map (an image which visualizes distances of scene objects from a viewpoint) as output [21]. Additional advantages include the

*All accompanying programs developed, datasets gathered and full result logs are available at http://pubs.cs.uct.ac.za under project HANGR.

Figure 2: Microsoft Kinect V2

Kinect not requiring background image calibration, or special markers or gloves for tracking [5], and the fact that the depth sensors are not affected by environmental conditions, such as low lighting [1, 5]. These features make it highly suitable for this project.

However, due to the low resolution of the depth sensing camera (only 512x424 for Kinect V2, which is lower than the 640x480 resolution of the Kinect V1), it can be a challenge to find and separate specific objects in the image [13]. In our case, the hands of the person using the device are most important, and a hand will occupy a relatively small portion of the already low-quality image extracted from the Kinect's depth sensor.

3 RELATED WORK

In the Kinect context, various methods of both feature extraction (the process of selecting informative values from the initially measured data, to facilitate learning) and classification (learning whereby training data has known category membership and the problem lies in identifying category membership of new data) have been experimented with and tested. Commonly used machine learning techniques for this kind of classification include neural networks [12], hidden Markov models [5, 10], and more recently, support vector machines [1, 17]. Recorded accuracies of the classifiers (where accuracy refers to the percentage of gestures correctly categorised by a particiular classifier), according to the related literature, range between 70% and 100%. However, it is important to bear in mind that the vast majority of research in this field focuses on methods which use the Kinect V1.

Table 1 provides a summary of the explored related work, in terms of classifiers used and their results. These are explained further in the subsections below.

3.1 Artificial Neural Networks

An artificial neural network (ANN) generally consists of layers of nodes, which are connected by links of various weightings. Information is accepted at the input nodes of the network and gets processed together with the links/weightings (the 'strength' of the input nodes in calculating the output to the next layer of nodes) and inner nodes of the network. It then reaches the final layer, where a classification result is outputted.

Pizzolato et al. [12] explores the use of an artificial neural network in the context of static gesture recognition for Brazilian Sign Language. In particular, a multi-layer perceptron (MLP), an ANN that is an evolution of the standard perceptron (a type of binary classifier which predicts according to a linear predictor function), is involved, and helps to distinguish between non-linearly separable data.

The process for segmentation entails using middleware, NITE (specifically for the Kinect V1), developed by PrimeSense, in order to find the user's center of mass. Using this, along with a depth threshold, a 'Virtual Wall' is obtained, and blobs in front of this are obtained through a linear time-component analysis. The relevant blobs are then sorted in terms of area and classified as hands (left, right, or dominant if only one hand is present). The region of interest (ROI) is then defined as the minimum rectangle enclosing the blobs. Pizzolato et al. further implement a heuristic method to adjust the ROI, in order to attempt to reduce, if not completely eliminate, the visible arm from the blob.

In particular, the final ROI size chosen was 25x25, which meant an MLP containing 625 different inputs (one node for each pixel of the ROI) was used. The MLP also contained 100 hidden/middle layer nodes, and 5 output nodes, for the 5 differing outputs/gestures.

The network was tested on 5000 samples, containing two groups of five different letters. The use of the heuristic hand cropping increased accuracy from between 60% and 75% up to 100%.

Although this study produced very promising results, it is not without flaws. Firstly, the hand cropping algorithm only accounts for the hand pointing upwards, and thus limits the types of gestures to which it can be applied. Secondly, only two gesture sets were tested, and each contained five very distinct hand positions. This makes it easier to obtain higher accuracy rates. Another questionable aspect of the paper is that there is no mention of how many users recorded data in the dataset; Thus, there is no indication of how robust the recognition system would be when used by different people (a real-world condition).

Another study [16] regarding recognition of Indian Sign Language numerals (0-9) used an ANN, together with a direct pixel-value method, as done in the Brazilian Sign Language study [12], and produced an accuracy rate of 97%.

3.2 Support Vector Machines

A support vector machine (SVM) is a supervised machine learning technique that is mostly used for classification problems. Each data item is plotted as a point in an n-dimensional space, and a hyperplane (a subspace of the space in n-1 dimensions) is found to differentiate the data. A clear division in the training instances allows the new data to then be classified. Basic SVMs are usually binary classifiers, however, they can be extended to support multi-class classification.

A relatively recent experiment [17] uses a combination of 'Ordinary' features (histogram of oriented gradients, appearance and motion information), together with special 'Kinect' features (body pose, hand shape and hand motion features). A histogram of oriented gradients (HOG), for part of the 'Ordinary' features, is extracted according to a specific algorithm [3]. Very briefly, after dividing the image into small pixel regions, each pixel's gradient orientation is approximated (with respect to surrounding pixels) according to one of nine orientation bins. 1D histograms of gradients in that image portion are then accumulated, in order to record local shape properties. In terms of hand information, a 48x48 pixel region around the hand point is cropped, after which HOG features are extracted on each patch of the region. The remaining feature extraction focuses on aspects such as body pose and is not relevant in our context.

For classification in this case [17], a multi-class latent SVM was used, in order to classify videos of sign language gestures describing words/sentences. Desired state values of each word/sentence are treated as the latent variables (inferred from variables that are directly measured) in the model.

This method was implemented with two different datasets (one with words and one with sentences) and a total of 1971 phrase videos and 1890 sentence videos were collected. Test results were

Table 1: Comparison of highlighted previous research on sign language recognition using Kinect

Reference	Classification technique	Feature extraction	Dataset	Accuracy
Sun et al. [17], American Sign Language	Latent Support Vector Machine	'Ordinary' features: HOG and motion/appearance; and 'Kinect' features: body pose, hand shape and hand motion	Part A: 73 unique ASL signs; 1971 phrases by 9 participants	86.0%
			Part B: 63 unique ASL sentences; 1890 sentences by 10 participants	82.9%
Agarwal et al. [1], Chinese Sign Language	Multi-class Support Vector Machine	Depth histograms and motion characteristics between frames	10 unique numeral gestures; 2 datasets of 47 video sequences each	81.48% (Linear Kernel); 87.67% (RBF Kernel)
Huang et al. [5], American Sign Language	Support Vector Machine	Hand, wrist, arm, shoulder positions and velocities; Distance separation between left and right body parts	10 unique signs; 100 samples by 2 participants	97%
Pizzolato et al. [12], Brazilian Sign Language	Multi-layer Perceptron (ANN)	Direct pixel-value, with and without hand cropping algorithm	Group 1: 5 unique signs; 1250 samples with hand cropping, 1250 without	100% with hand cropping; 67.4% without hand cropping
			Group 2: 5 unique signs; 1250 samples with hand cropping, 1250 without	100% with hand cropping; 75.4% without hand cropping
Trigueiros et al. [19], Various hand gestures/poses	*k*-nearest Neighbours	Convexity defects, orientation histogram, etc.	2 different datasets of 10 different gestures each	95.5% on dataset 1; 88.5% on dataset 2
Sharma et al. [16], Indian Sign Language	*k*-nearest Neighbours	Direct pixel-value	10 unique numeral gestures, 5000 samples by 100 different participants	78.6%
		Hierarchical centroid features		70.9%
Zafrulla et al. [21], American Sign Language	4-state Hidden Markov Models	Body pose and hand features	555 seated samples, including 207 corrupt samples	95.16% before factoring in corrupt samples; 58.86% after factoring in corrupt samples
			155 standing samples, including 9 corrupt samples	94.49% before factoring in corrupt samples; 88.02% after factoring in corrupt samples
Sarhan et al. [15], Arabic Sign Language	3-state Hidden Markov Models	Location, orientation, axes, shape roundness, convexity/concavity, rectangularity and trajectory of hand	16 unique words, 215 samples signed by 4 different participants	73.06%
	4-state Hidden Markov Models			78%
	5-state Hidden Markov Models			80.47%
Jangyodsuk et al. [6], American Sign Language	Dynamic Time Warping	Hand trajectory and hand shape mapped as HOG features	2226 samples signed by 2 participants	82.09% at top 10 rank; 92.54% at top 30 rank
Santos et al. [14], American Sign Language	Dynamic Time Warping combined with Hidden Markov Models	Hand contour shapes	12 dynamic hand gestures	97.49%

favorable: For words, the model reported a high accuracy rate of 86%, and the SVM outperformed other algorithms tested.

A very similar study was undertaken by Agarwal et al. [1], involving both HOG features and an SVM classifier. Simpler gestures were used (only those representing the numbers 0-9) and the dataset was also significantly smaller. The paper also 'safely concludes' (more likely to be an unsafe conclusion as this is casually mentioned without evidence) that the recognition system created is faster than other techniques in hand tracking or hand shape analysis. With these issues aside, very positive results were obtained from the SVM, with accuracy rates of around 80%.

3.3 k-Nearest Neighbours

The k-nearest neighbours (kNN) algorithm is widely used in Computer Vision. It is a lazy algorithm, meaning that it does not do any generalization once given the data points for training. Rather, these training data points are 'remembered', and classification of queried data is done according to which class has the most data points in the nearest k neighbours of the queried data point.

A comparative study of machine learning algorithms [19] uses kNN for the recognition of hand gestures. After segmenting the hand, various hand features are extracted, such as convexity defects and the orientation histogram, amongst others. After testing two different datasets of 10 gestures each, using different combinations of features in each dataset, it was found that the kNN compared favourably when compared to the other tested classifiers, which included an ANN and an SVM. For the first dataset, the 95.5% accuracy rate of the kNN was only beaten by the ANN, by about 1.5%. The kNN was the best classifier with 88.5% accuracy for the second dataset. The study, however, fails to mention or show which exact hand gestures were classified, which makes the validity questionable (even though results are reportedly good).

An Indian Sign Language study [16], which focused on the 10 isolated numeral gestures (1-10), also used a kNN classifier together with two different feature extraction methods, namely the hierarchical centroid and direct pixel-value methods. The direct pixel-value method is similar to the feature extraction method used by Pizzolato et al. [12], in which the cropped image containing the hand is directly fed into the classifiers, pixel by pixel. The result of classification by the kNN was around 78.6% for the direct pixel-value method, and 70.9% for the hierarchical centroid features. Although the ANN produced higher accuracy rates, the kNN results are still worth exploring.

3.4 Other previously explored techniques

Hidden Markov Models: A hidden Markov model (HMM) is a stochastic model of a system, containing hidden or unobservable states, and in which future states depend only on the current state.

An investigation [21] of recognition and verification for American Sign Language phrases was conducted using the Kinect. This deals with both standing and sitting positions, and thus both body pose and hand features are extracted. Body pose features are extracted using the OpenNI framework (for Kinect V1). For training and testing of the system, 4-state HMMs were trained for each of the 19 signs investigated. Recognition results were very high without tracking errors, with both seated and standing data giving around 95% accuracy rates. However, after factoring in the tracking errors of the data and considering them as failures during testing, accuracy rates dropped to 88% for the standing data and 59% for the seated data. The drop in accuracy is attributed to the large amount of corrupted samples in the dataset. Tracking errors are a result of OpenNI's skeletal tracking, and thus the framework is not the most ideal for use in a gesture recognition system where the user would be sitting, and especially if the body pose is not important (as in the SASL alphabet).

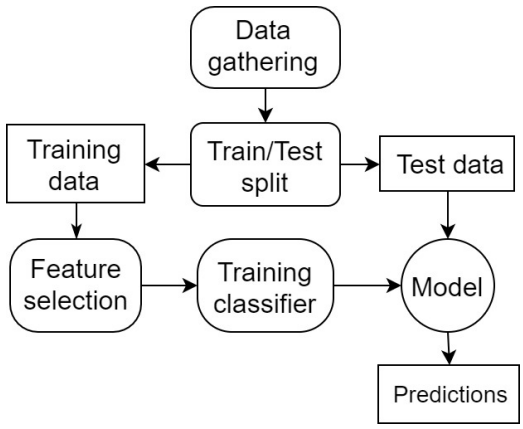

Figure 3: Pipeline representation of the use of machine learning classifiers

A more recent paper [15] explores HMMs for recognition of Arabic Sign Language. As seen in Table 1, the results vary between 73% and 80%, depending on the number of states used in the model.

Dynamic Time Warping: This is an algorithm for classification that is not a machine learning technique, but is still used for gesture recognition. This algorithm is used to find an optimal alignment between two given time series (time-dependent) sequences under certain restrictions. As detailed further in Table 1, two studies [6, 14] explore the use of the method for gesture recognition and achieve relatively high accuracy rates.

4 METHOD OF INVESTIGATION

Figure 3 depicts a typical set of stages in gesture recognition. This study entailed three major stages, namely:

Data gathering: Obtaining data representing the gestures of the SASL alphabet;

Feature extraction: Extracting relevant data from the previously obtained data; and

Classification: Training the classifiers using some data and testing the newly trained classifiers using other data.

These stages of this study are fully explained below.

4.1 Data Gathering

The data gathering setup catered for recording information provided by all of the Kinect V2, Myo armband, and Leap Motion Controller, simultaneously. For this, a program available online[1] for the Kinect V2 to easily track hands and fingertips was modified to allow the additional devices to record their respective data.

The available Kinect V2 hand tracking program was used as it facilitated the feature extraction process. The program was slightly altered to account for the tracking of a single hand at a time (as the SASL alphabet is signed with only one hand). Fingertip tracking was also removed, because it was found not to be entirely accurate: Even when the hand appeared clearly outlined, and thus tracked, on

[1] http://pterneas.com/2016/01/24/kinect-finger-tracking/

screen, not all 5 fingertips were necessarily picked up and visualised in their true locations.

Assuming a hand is found by the program, pressing the Enter key acts as an easy data recording trigger, and the Kinect captures the contour outlines of the hand, as image frames, for a duration of 3 seconds. As the Kinect has a frame rate of 30 frames per second, each gesture thus contains approximately 90 frames. On fusing this program with data gathering components required for the Myo and Leap devices, some blank frames were recorded. However, each gesture contained at least a few usable frames, which showed hand contour outlines.

36 different right-handed participants were recruited for the data gathering process, and each participant recorded 10 instances of each gesture letter. The high number of participants, together with the reasonable number of gestures recorded per letter, was intended to account for the natural variation between hand shapes and hand movements among different people. Due to unavoidable hardware failure during the data gathering process, some letters were recorded more than necessary and are thus oversampled. In very few cases, some participants were not able to finish recording their entire gesture set, and so these sets contain less than 260 gestures. Regardless of this minor issue, the end result is that a very large gesture dataset has been acquired. With almost 10 000 gestures, the dataset is significantly bigger than datasets in previous literature, and can possibly be used in future research for gesture recognition with regard to the Kinect, Leap and Myo devices and the SASL alphabet.

4.2 Feature Extraction

As mentioned above, the modified program for the Kinect V2 assists in feature extraction, as it does not record the entire image as seen by the Kinect, but rather just the contour of the hand. This is done by first detecting the body of the user, and then tracking the relevant hand joint, whether right or left. This, together with depth information given by the Kinect's depth sensor, allows the contour outline of the hand to be drawn and recorded in PNG images.

We then process these images in OpenCV[2], an open source programming library widely used for computer image and video processing. The stages of this processing are depicted in Figure 4.

As the majority of the gestures of the SASL alphabet are static, only 1 frame per gesture is required for training/testing purposes. Thus, per gesture folder, the first valid frame is chosen, so that it may be further processed and classified. A frame, in this context, is considered 'valid' if it is non-empty. Furthermore, after finding and filling the largest contour present in the image (assuming that it is the contour of the hand), the frame is still considered 'valid' if the smallest rectangle around this contour has dimensions of at least 30x30. Assuming validity, the image frame is then cropped to this smallest rectangle, and is then rescaled to a 30x30 image through an algorithm described by Pizzolato et al. [12]. This algorithm entails resizing the cropped image proportionally to obtain 30 pixels in at least one dimension, and then centering the result in a 30x30 image.

The final image is then converted to greyscale (using cv::cvtColor together with cv::BGR2GRAY), resulting in a 900-element (30x30) cv::Mat frame. This is the feature vector which is fed into classifiers, and is essentially the previously described direct pixel-value method of feature extraction [12, 16]. The 30x30 resolution is chosen so as to improve on the 25x25 and 20x30 resolutions used in previous studies, but is not so large as to drastically reduce the efficiency of the classifiers when trained and tested with these feature vectors. Due to the image being inputted to a classifier pixel by pixel, an increase in the size of either (or both) image dimension(s) means an increase in the number of features that the classifier would be required to learn, and hence a decrease in the efficiency of the classifier.

Unfortunately, this study does not account well for the dynamic gestures (**j** and **z**). This is due to the nature of the feature extraction process, as well as the classifiers explored (being best suited to static gestures). However, for completeness sake, in terms of the alphabet, the dynamic gestures are included in the training and testing.

4.3 Classification

Once all gestures have been cropped and stored in cv::Mat frames, these are divided into test and train sets. For each classifier, a variable is initialised along with the required classifier parameters. The classifier can then be trained on the given training set, which contains the data points along with their respective classes/labels.

Testing of a classifier involves inputting a selected portion of data into an already trained classifier. A prediction method is then called on these data points, and predicted classes are then outputted. To obtain the accuracy of a classifier, the predicted classes are compared to the expected/actual classes of the tested data, and the percentage of the data points which have correct predictions is calculated.

In this case, a k-fold cross-validation method of testing [8] was used, where the folds are on each participant's data; Thus k=36. This entails leaving out one participant's data (out of the available 36 participant datasets) for testing, while training on the data of the remaining 35 participants. This process is then repeated k times, where the testing participant is changed each time. Finally, the k accuracy results are then averaged to produce a final approximation of the accuracy of the classifier. This testing method, in which the test participant is different from the training participants (and hence, hand shapes of the test participant are completely new to the trained classifiers), subjects the classifiers to real-world conditions, making the accuracies more viable in this regard.

The following classifiers, together with their respective parameters, were tested. More specifically, these classifiers were used as available in the OpenCV machine learning namespace.

k-Nearest Neighbours: This is one of the simplest classification algorithms for supervised learning problems [11]. The training phase of this classifier is minimal, as training data points are merely 'remembered' by the classifier. Upon providing a new data point for classification, the k neighbours (or training data points) that are closest to the given data point are then found. The class of the given data point is then determined based on the majority vote of its neighbours *i.e.* the class that the most neighbours belong to will be outputted as the predicted class of the data point.

In this case, using OpenCV's kNN implementation, the only parameter which was varied was the k-value, the number of nearest neighbours to be considered during class prediction. The type of k-Nearest implementation was set to brute force (a naive neighbour search, computing distances between all pairs of points). The KDTree implementation (a k-d tree-like structure created during training to reduce the number of distance calculations) was not

[2]http://opencv.org/

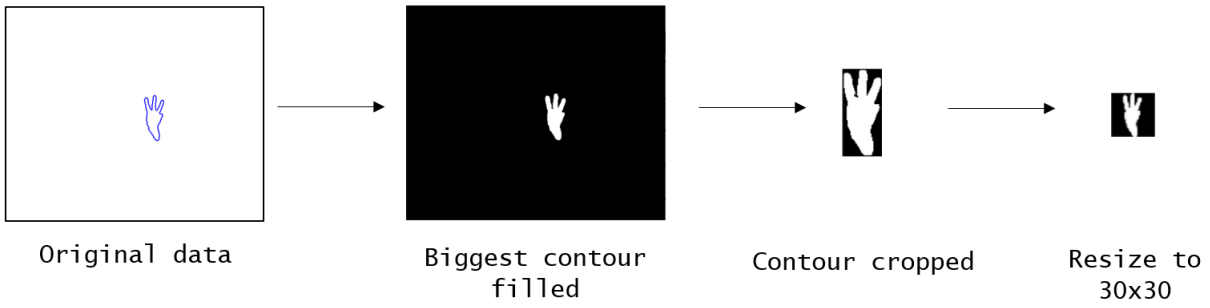

Figure 4: Visual summary of the stages of feature extraction

tested due to the high dimensionality of the feature vector, which would make training highly inefficient. The kNN was also set to be a classifier, rather than a regressor.

Support Vector Machine: The support vector machine is commonly used in classification problems, and is applied in this case as a multi-class classifier. This requires the OpenCV SVM type to be set to C-Support Vector Classification (C-SVC), where the number of classes is at least 2. A parameter, C, acts as a trade-off between the training error and decision boundary. This means that the higher C is, the less the training error will be. However, should C be too high, the classifier could lose its generalization properties and risk over-fitting the training data *i.e.* learning noise of the data.

An SVM kernel specifies the decision functions (or hyperplanes) of the classifier. Following previous literature, three different types of SVM kernels, were investigated, in the OpenCV environment [7], as follows:

Radial Basis Function (RBF) Kernel: This Gaussian function is defined as $e^{-\gamma||x_i - x_j||^2}$, where x_i and x_j are data points, and γ is the parameter Gamma, which defines the function's variance. By the equation, more variance would occur if Gamma is smaller, while a larger value for Gamma would result in less variance. Thus, if two points are far from each other and Gamma is low, the points are still considered similar; On the other hand, if two points are close together and Gamma is high, the points are considered similar. Hence, in terms of the classifier, this parameter dictates how far in the space a single training data point would have an influence.

Linear Kernel: This is the simplest of the three considered kernels, with the function simply being $x_i^T x_j$.

Polynomial Kernel: As the name implies, this function is a polynomial: $(\gamma x_i^T x_j + coef0)^{degree}$. The Gamma parameter here acts as a coefficient for the non-constant term, whereas $coef0$ is a constant, usually kept at the default value of zero. The degree is the most important parameter of this kernel function.

Multi-Layer Perceptron: A multi-layer perceptron (MLP) (previously explained in Section 3.1) can be visualised as shown in the example in Figure 5 below, which consists of 3 neuron layers - 3 inputs, 2 outputs, and 5 neurons in the hidden (middle) layer.

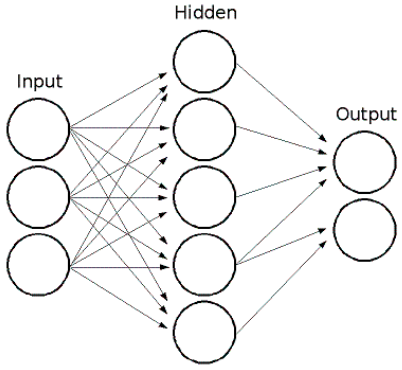

Figure 5: Example of an MLP

Due to the nature of the feature vector used in this study, the input layer is set at 900 (30x30) neurons. The output layer is set to 26 neurons, as there are 26 different possible outputs/classes (26 unique letters/gestures in the dataset). This leaves the number of input layers, along with their sizes, to be determined.

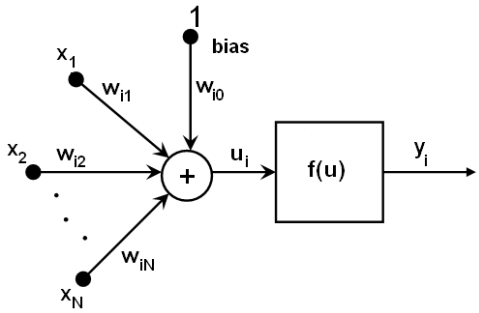

Figure 6: Example neuron with input links and activation function

Each neuron of the network takes in at least one input from the previous layer of neurons. These inputs are summed, along with their respective weightings (individual for each neuron), and a bias term is added to the sum. An activation function is then applied on the sum, to obtain the value which will be passed to some neurons in the next layer of the network. This process is visualised for a single neuron in Figure 6, where x represents some output of the previous layer, w_i a weight value for its respective neuron, u_i the sum of the weighted inputs and bias term, $f(u)$ the activation function applied

on a given value u, and y_i the value passed to the next layer. Thus, given outputs x_j of the layer n, outputs y_i of the layer $n+1$ can be calculated as follows [20]:

$$u_i = \sum_j (w_{i,j}^{n+1} * x_j) + w_{i,bias}^{n+1}$$

$$y_i = f(u_i)$$

The weights $w_{i,j}^{n+1}$ are automatically computed by the training algorithm. This is done through iterative adjustment of the weights, to give the allow the network to give the desired responses (classes) to the training inputs.

The activation function used in this MLP is the sigmoid function. This is the symmetrical sigmoid function shown below, as in OpenCV [7], with the parameters set at default to $\beta = 1, \alpha = 1$.

$$f(x) = \beta * (1 - e^{-\alpha x})/(1 + e^{-\alpha x})$$

Other functions available in OpenCV, but not used, are the Linear (which would be ineffective given our high-dimension feature vector) and the Gaussian (which is not completely supported, at the moment, in OpenCV) functions.

Voting Classifier: The implemented voting classifier acts as a combination of three of the best above-mentioned classifiers, each with their own optimal parameters. This voting classifier does a 'hard'/majority [18] vote, meaning that the most occurring class between the three classifiers becomes the decided class. Should the three inputted classifiers all differ in their decision, the most accurate classifier is then chosen. See Section 5.5 for the chosen classifiers in this case.

5 RESULTS

All experiments were executed on a machine running a 64-bit Windows 10 Home operating system, with an Intel(R) Core(TM) i7-6560U processor (2.20GHz) and 8GB RAM.

5.1 Data Gathering and Feature Extraction

Table 2 summarises the results of these stages. From the 36 participants, 9491 gestures were obtained in total. During the processing and feature extraction of this data, 7 gestures were considered invalid and were thus disregarded, leaving 9484 gestures to be used in the training and testing of the various classifiers. This process (for both training and testing data) generally took between 3 and 4 minutes to complete. Assuming a process of around 3.5 minutes to process 9491 gestures, it would thus take approximately 0.022 seconds to process a single gesture.

Table 2: Results of data gathering and feature extraction stages

Number of participants	36
Total recorded gestures	9491
Total 'valid' gestures	9484
Total 'invalid' gestures	7
Time taken to process all gestures	≈ 3.5 mins
Time taken to process a single gesture	≈ 0.022 secs

Accuracy results, obtained after cross-validation on each parameter choice, per investigated classifier, are discussed below. Unless otherwise specified, training times shown in tables refer to the average time taken, in minutes, to train on the data of 35 participants, and the prediction times refer to the average time taken to predict on one participant's set of data (approximately 260 gestures, which are not included in the training data) on a trained classifier.

5.2 *k*-Nearest Neighbours

For this classifier, the k-values tested ranged from 1 to 25 (inclusive). The results of a pertinent subset of k-values are reported below in Table 3, along with the standard deviation (Std. Dev.) of the accuracies.

Table 3: Accuracies of kNN classifiers, along with times taken for training and prediction (in minutes)

k-value	Accuracy	Training Time	Prediction Time
2	67.01%	0.00018	0.01017
6	69.81%	0.00019	0.01147
10	***70.04%***	0.00021	0.01133
14	69.63%	0.00021	0.01205
18	69.48%	0.00018	0.01121
22	68.99%	0.00018	0.01014
Std. Dev.	1.11		

These values are all within a very small range of each other, with the biggest difference being for k=2. The highest accuracy rate, as highlighted in the table, is just over 70%, when k=10. The trend, within the range of k=[1, 25], is seemingly that accuracy peaks slowly as k increases towards 10, and drops slowly as k increases thereafter.

5.3 Support Vector Machine

Under all kernels, C was varied from very small to very large values (from 0 to 10^7). Changing this parameter, however, seemingly had absolutely no effect on the produced accuracies.

Linear Kernel: After cross-validation testing, the classifier produced an accuracy of 72.62%. On average, it took approximately 0.178 minutes to train this SVM on 35 participants, and 0.0003 minutes to predict on one participant's dataset.

Polynomial Kernel: The parameter investigated here was the degree of the polynomial function, which was varied between 2 and 7. Table 4 reports these performances, as well as the standard deviation of the different accuracies. The $coef0$ parameter was kept at 0 and the Gamma parameter was kept at its default value of 1.

Table 4: Accuracies of the SVM classifier using a Polynomial Kernel, along with times taken for training and prediction (in minutes)

Degree	Accuracy	Training Time	Prediction Time
2	75.22%	0.1704	0.0056
3	***75.96%***	0.1753	0.0053
4	75.36%	0.1873	0.0054
5	5.95%	0.5341	0.0084
6	3.71%	1.0926	0.009
7	3.94%	1.0882	0.0087
Std. Dev.	38.89		

The kernels on degrees 3, 4 and 5 produced highly similar results, with degree 3 having the highest average accuracy, 75.96%. A confusion matrix showing results of all classifications through the cross-validation testing with degree 3, is depicted in Figure 7.

RBF Kernel*:* The OpenCV library contains a special method to automatically train an RBF Kernel in order to find the most optimal parameters. This was used with one randomly selected participant's dataset (not cross-validation) for testing.

The result was that this automatic training took 462.24 minutes to complete and produced an accuracy of 3.85%, classifying all gestures under one class only, proving the method to be completely ineffective.

5.4 Multi-Layer Perceptron

The MLP was set to have one middle layer [4] of neurons, tested on two different sizes. In both cases, the input layer size was set to 900 (number of pixels in an input) and the output layer to 26 neurons (number of different possible classifications/gestures). The middle layer neuron counts follow rules-of-thumb set out by Heaton [4]. One rule suggests that the number of neurons in the middle layers should be between the number of neurons in the input layer and the number of neurons in the output layer. Based on this, the middle layer of one MLP was set to 463 (average of neurons in both input and output layers).

Another rule [4] is that the number of hidden neurons should be two thirds of the input neurons, plus the number of output neurons. Hence, the number of middle layer neurons of the other MLP was set to be 626 ((2/3)*(900 input neurons) + 26 output neurons).

Each MLP produced accuracies of less than 50%, and training times of above 40 minutes. These results are reported in Table 5.

Table 5: Accuracies of MLP classifiers based on middle-layer neuron count, along with times taken for training and prediction (in minutes)

Neurons	Accuracy	Training Time	Prediction Time
463	44.5%	41.28	0.00128
626	41.06%	58.44	0.00179
Std. Dev.	2.43		

5.5 Voting Classifier

The three best performing classifiers, in terms of accuracy alone, and thereafter in terms of efficiency, were the SVM on a Polynomial Kernel (degree 3), the SVM on a Linear Kernel, and the kNN (k=10), respectively. Based on these results, these three classifiers were chosen to be investigated in combination, using a 'hard' voting system. The preferred classifier, in the case of all three classifiers producing differing results, was set to be the SVM with Polynomial Kernel, due to having the highest accuracy rate.

Training of this classifier involves training each of the SVM (Polynomial), SVM (Linear) and kNN separately, while prediction involves prediction on each of the classifiers, followed by a naive method of selecting the most upvoted class.

The training took 0.368 minutes to complete, and prediction completed in 0.0175 minutes. The final averaged accuracy of the classifier, after cross-validation, was 76.2%.

6 DISCUSSION

6.1 Efficiency

In considering the time taken to train the classifiers, there is a stark difference between training times of the MLP and the other classifiers. While the other classifiers usually took much less than a minute to train, the smaller sized MLP took over 40 minutes on average. The larger MLP network took almost one hour. These durations can be attributed to the fact that the network comprises a large number of neurons (900 alone in the first layer) and through training, has its weights automatically adjusted each time a new data point is included. Due to these excessive times, only two setups for the neural network were tested, both with only one hidden layer of neurons. Nevertheless, the classifier can still be regarded as highly inefficient in terms of training when compared to the other classifiers. The two results also clearly show an increase in training time, and a very small increase in prediction time, with an increase in the size of the middle layer. In addition, the classifier generally performed poorly, producing accuracy rates of less than 50%.

Keeping in mind that prediction times recorded account for times taken to predict a single participant's set of gestures *i.e.* ≈260 data points, all predictions (for all classifiers) can be considered rapid in a real-world sense. Furthermore, the times differed by milliseconds, which would go unnoticed in a real-world system should a prediction tool be made using one of these classifiers. The kNN took the longest on average to predict a single set of gestures (most likely due to the brute-force method of calculating and comparing distances for each prediction), at roughly 0.6798 seconds when k=10. Given 260 gestures in a single set, a naive estimate would imply that each gesture takes 0.0026 seconds to be classified. Combining this result with the approximate 0.022 seconds to process a given gesture image (Section 5.1), classifying a gesture end-to-end, on a trained classifier, would take around 0.0246 seconds. With this including the longest prediction time among the classifiers, real-world suitability of these classifiers, in terms of efficiency, is clear.

6.2 Accuracy and Error

In increasing order of accuracy, the classifiers were the MLP, the kNN (with k=10), the SVM on a Linear Kernel, and the SVM on a Polynomial Kernel (with degree = 3). This result led to the investigation of a combined voting classifier, although this barely improved on the accuracy of the Polynomial Kernel SVM, with an

gesture	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	total actual
a	305	0	0	0	2	0	9	0	0	0	0	0	6	5	0	5	0	0	17	1	0	0	0	1	8	2	361
b	0	349	0	0	0	1	0	0	0	0	2	0	0	1	2	0	0	0	0	1	1	0	1	0	1	0	359
c	2	0	356	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	7	0	367
d	0	0	0	244	0	0	0	0	0	0	0	1	0	0	0	0	0	8	0	5	2	1	0	16	4	81	362
e	1	0	0	0	215	0	0	0	0	0	0	1	14	4	89	0	2	0	31	0	0	0	0	0	0	0	357
f	0	1	0	0	0	337	0	0	0	1	1	4	0	0	0	0	0	0	0	0	0	1	3	3	5	1	357
g	3	0	2	0	0	0	342	9	0	0	0	0	0	0	0	1	0	0	2	0	0	0	0	0	10	1	370
h	5	0	0	0	0	0	7	352	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	4	0	369
i	0	0	0	0	1	0	0	0	206	150	0	2	0	0	0	1	0	0	1	1	0	0	0	0	4	0	366
j	0	0	0	0	3	2	0	0	170	184	0	0	0	0	0	2	2	0	1	1	0	0	1	0	6	0	372
k	0	0	0	6	0	2	0	0	0	0	293	0	0	0	0	0	0	4	0	1	4	46	1	1	2	6	366
l	0	0	0	0	0	0	0	0	1	1	0	354	0	0	1	0	0	0	0	1	0	0	0	1	6	5	370
m	4	0	0	0	12	0	0	0	0	0	0	0	246	28	23	6	0	0	47	0	0	1	0	0	2	0	369
n	3	0	0	0	4	0	0	0	0	0	0	0	43	242	15	5	0	0	55	0	0	0	0	1	3	0	371
o	1	0	0	0	97	1	0	0	0	0	0	1	33	23	181	3	1	0	23	0	0	0	1	1	1	0	367
p	16	0	0	0	0	0	6	1	0	1	0	0	8	4	3	265	41	0	6	0	0	0	0	0	17	0	368
q	0	0	1	0	13	0	0	0	0	0	0	0	1	1	2	32	308	0	1	0	0	0	0	0	4	0	363
r	1	2	0	22	0	2	0	0	0	1	0	0	1	0	0	1	0	295	0	0	15	0	0	8	3	11	362
s	19	0	1	0	53	0	0	0	0	0	0	0	54	61	26	3	1	0	140	1	0	0	1	0	4	0	364
t	5	2	1	4	0	1	1	0	0	0	0	0	0	2	0	1	0	0	2	268	0	0	0	67	2	3	359
u	0	3	0	3	0	0	0	0	0	0	1	1	0	0	1	0	0	16	0	0	328	1	0	0	4	0	358
v	0	0	0	0	0	0	0	0	1	0	44	2	1	0	2	0	0	2	1	0	2	302	0	0	2	0	359
w	0	6	0	0	0	2	0	0	0	0	3	0	2	0	1	0	0	0	1	1	0	7	347	0	0	0	370
x	1	0	0	28	1	0	0	0	0	0	0	2	0	3	1	0	0	6	0	80	0	0	0	215	3	24	364
y	5	0	0	0	0	1	3	0	4	6	0	1	0	0	0	1	0	0	0	0	0	1	0	0	340	0	362
z	0	0	1	124	0	4	0	0	0	0	3	4	0	1	0	0	0	9	0	10	0	1	0	30	5	180	372
total predicted	371	363	362	431	401	353	368	362	382	344	347	373	409	375	347	328	355	340	329	371	352	361	355	344	447	314	9484

Figure 7: Confusion matrix of total cross-validation testing on SVM with Polynomial Kernel, degree = 3

Table 6: Accuracies of the best performing classifiers

Classifier	Accuracy rate
kNN (k=10)	70.04%
SVM (Linear kernel)	72.62%
SVM (Polynomial kernel)	75.96%
Combined voting classifier	76.2%

increase of less than 1%. This also comes with additional time taken for training and prediction. Considering the small improvement, the trade-off does not seem worthwhile.

Table 6 summarises the best performing classifiers in terms of accuracy. In previous studies, the accuracy results were generally much higher, ranging from low 80s to high 90s for Support Vector Machines [1, 5, 17]. These studies did, however, use more complex features for classification. Trigueiros et al. [19] also extracted more intricate gesture features, but explored kNNs, achieving accuracies between 88% and 96%. Sharma et al. [16], on the other hand, used kNNs with direct pixel-value feature extraction, and obtained 78.6% accuracy on classification of 10 unique gestures. Remembering that our dataset is based on 26 unique gestures, the SVM (polynomial kernel) result of 75.96% accuracy is hopeful under this feature extraction method.

Setting the voting classifier aside, the best performing classifier was the SVM on the Polynomial kernel (with degree 3). The entire set of results of this cross-validation test can be seen in the confusion matrix shown in Figure 7. The confusion matrix depicts the exact performance of classification, with each row representing actual classes, and each column representing predicted classes. The diagonal of this matrix therefore indicates the numbers of gestures correctly identified by the classifier (and hence, accuracy is calculated by summing the values along the diagonal and finding this as a percentage of the total number of gestures).

Accuracy rates for the investigated classifiers, clearly, were not perfect. Remembering that the data classified is directly based on the visual capture by the Kinect, it is not difficult to surmise how errors can occur in this gesture recognition problem.

As greyscale images (the results of feature extraction) are fed into the classifiers, pixel by pixel, it is easy for the classifiers to learn some noisy pixels, and this can result in overfitted models after the classifiers are trained. Also important to keep in mind is that a contour of the hand alone (with no inner nuances, such as finger positions within the contour) is recorded. A few of the SASL alphabet gestures are very similar in terms of the outer contour of the hand, and this ambiguity is evident in the confusion matrix. Much confusion in this regard arises with the group of fist-like gestures, namely **a**, **e**, **m**, **n**, **o** and **s** (Figure 8).

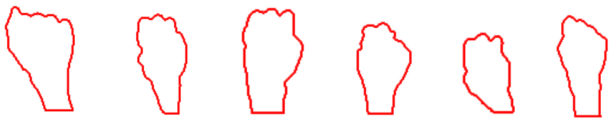

Figure 8: Random samples, from the acquired dataset, of the fist-like gestures: *a, e, m, n, o, s* respectively

Similarly, there is another group of gestures, which have the index finger raised out of a fist-like hand in certain, slightly differing ways, and these result in more error: **d**, **t**, **x** and **z** (Figure 9). Out of this group, the dynamic gesture **z** has a starting position that is basically identical to the static gesture of **d**, and so misclassification between these two gestures is expected.

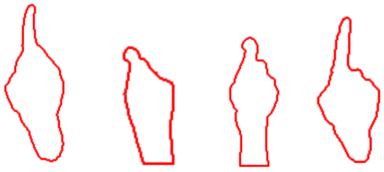

Figure 9: Random samples, from the acquired dataset, of the raised-index gestures: *d, t, x, z* respectively

Similarly, the dynamic gesture **j** is identical in terms of finger positions, though with a slight tilt of the hand, to the static gesture **i** (Figure 10). Unsurprisingly, this causes misclassification between these two classes: Over 40% of the total number of gestures in each class is classified in the other. The last major group of gestures which are most likely to result in error are gestures **k** and **v**, which have both the index and middle finger extended, but the thumb in differing positions (Figure 11). In this case particularly, the thumb may not be visible in different positions on visual inspection of the contour alone, based on a person's hand and their ability to extend the thumb between the index and middle finger.

Figure 10: Random samples, from the acquired dataset, of the gestures *i, j* respectively

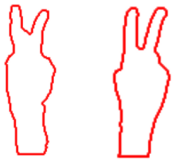

Figure 11: Random samples, from the acquired dataset, of the gestures *k, v* respectively

Beyond these clearly explainable errors, a few random errors do occur throughout the confusion matrix. This can be attributed to multiple factors. First, a test participant's hand may be shaped quite differently to those of the participants in the training set, resulting in more misclassification than usual. A bigger dataset, with more participants of varying hand shapes and sizes, would help in this regard.

Second, and more likely, is that during recording, gestures contained some form of error themselves, and the hand and its contour were not captured in its entirety. The program used for data gathering with the Kinect V2 sometimes fails when picking up the hand, and does not always accurately track the extension of digits. Another shortcoming is that the contour can become noisy, and be slightly extended and fused with other objects in the near background of the hand, during tracking. Contours of hands with defects in certain positions can easily lead to misclassification, based on how closely related some gesture are, as discussed above. These flaws in the hand tracking program are also discussed and pointed out in a finger and hand gesture recognition study based on the Kinect V2 [9].

The program also occasionally picks up part of the arm, below the wrist, depending on the positioning of the hand and arm. In this case, a hand cropping algorithm similar to that explored by Pizzolato et al. [12] would be beneficial to reduce the visible arm from the contour extracted.

7 CONCLUSIONS AND FUTURE WORK

Overall, the SVM running on a Polynomial kernel can be considered the best classifier to solve the gesture recognition problem of the SASL alphabet, through the use of the Microsoft Kinect V2, along with the direct pixel-value method of data extraction. After this, the SVM with Linear kernel and kNN algorithms are not far behind in terms of accuracy. The OpenCV library effectively facilitates the use of these algorithms.

The MLP does not seem well-suited to this problem, perhaps due to the method of feature extraction, which causes training to take significantly longer than any other classifier investigated. A combination classifier may be worth investigating using other classifiers, and also using pre-calculated probabilities of class memberships, based on training prior to testing, together with a 'soft' voting system, which takes into account these probabilities before new predictions.

To help solve the issue of highly similar gestures of the alphabet being confused, other feature extraction methods, which at least pay more attention to the relative positions of fingers, would be worth investigating to combine with the direct pixel method, or to use as a standalone. Finding more intricate features to aid a more informative feature extraction, however, could potentially be more computationally expensive, and slow down data processing. Tying into this is the software used for recording data, which definitely has room for improvement, as previously noted [9].

Lastly, to account dynamic gestures, classifiers that are better suited to dynamic gestures could be investigated further. As found in the literature, certain classifiers are more suited to static gestures, while some classifiers, such as the HMM, deal well with dynamic gestures. In such a case, however, a different library which utilises the HMM classifier, would have to be found, since OpenCV does not support this as of yet.

8 ACKNOWLEDGEMENTS

The author would like to thank Associate Professors James Gain and Deshendran Moodley for their guidance and supervision throughout the course of the project, as well as Anna Borysova and Erin Versfeld for their contributions, particularly to the data gathering stage, and support as members of the project HANDGR.

The author also gratefully acknowledges the partial support for this research, as provided by the National Research Foundation, Trevor Winer Scholarship Trustees and Investec.

REFERENCES

[1] Anant Agarwal and Manish K Thakur. 2013. Sign Language Recognition using Microsoft Kinect. In *Sixth International Conference on Contemporary Computing (IC3)*. IEEE, Noida, India, 181–185.

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, and others. 2016. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316* (2016).

[3] Navneet Dalal and Bill Triggs. 2005. Histograms of Oriented Gradients for Human Detection. In *International Conference on Computer Vision & Pattern Recognition*. IEEE, San Diego, United States, 886–893.

[4] J. Heaton. 2008. *Introduction to Neural Networks with Java.* Heaton Research. https://books.google.co.za/books?id=Swlcw7M4uD8C

[5] Frank Huang and Sandy Huang. 2011. Interpreting American Sign Language with Kinect. *Stanford University term paper for CS 299* (2011).

[6] Pat Jangyodsuk, Christopher Conly, and Vassilis Athitsos. 2014. Sign Language Recognition using Dynamic Time Warping and Hand Shape Distance Based on Histogram of Oriented Gradient Features. In *Proceedings of the 7th International Conference on Pervasive Technologies Related to Assistive Environments.* ACM, Island of Rhodes, Greece, Article No. 50.

[7] Adrian Kaehler and Gary Bradski. 2016. *Learning OpenCV 3: Computer Vision in C++ with the OpenCV Library* (1st ed.). O'Reilly Media, Inc.

[8] Ron Kohavi and others. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, Vol. 14. Stanford, CA, 1137–1145.

[9] Anna Lekova, D Ryan, and Reggie Davidrajuh. 2016. Fingers and Gesture Recognition with Kinect V2 Sensor. *International Conference Automatics and Informatics 2016* (10 2016).

[10] Becky Sue Parton. 2006. Sign Language Recognition and Translation: A Multidisciplined Approach From the Field of Artificial Intelligence. *Journal of Deaf Studies and Deaf Education* (2006), Vol. 11, No. 1, pp. 94–101.

[11] Leif E Peterson. 2009. K-nearest neighbor. *Scholarpedia* 4, 2 (2009), 1883.

[12] Ednaldo Brigante Pizzolato, Mauro dos Santos Anjo, and Sebastian Feuerstack. 2012. A Real-Time System to Recognize Static Gestures of Brazilian Sign Language (Libras) alphabet using Kinect. In *IHC Í2 Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems.* SBC, CuiabÃą, MT, Brazil, 259–268.

[13] Zhou Ren, Jingjing Meng, Junsong Yuan, and Zhengyou Zhang. 2011. Robust hand gesture recognition with kinect sensor. In *19th ACM international conference on Multimedia.* ACM, Scottsdale, Arizona, USA, 759–760.

[14] Diego G. Santos, Bruno J. T. Fernandes, and Byron L. D. Bezerra. 2015. HAGR-D: A Novel Approach for Gesture Recognition with Depth Maps. *Sensors 15(11)* (2015), 28646–28664.

[15] Noha A. Sarhan, Yasser El-Sonbaty, and Sherine M. Youssef. 2015. HMM-based Arabic Sign Language Recognition using Kinect. In *Tenth International Conference on Digital Information Management (ICDIM).* IEEE, Jeju, 169 – 174.

[16] M Sharma, R Pal, and Ashok Sahoo. 2014. Indian sign language recognition using neural networks and kNN classifiers. 9 (jan 2014), 1255–1259.

[17] Chao Sun, Tianzhu Zhang, and Changsheng Xu. 2015. Latent Support Vector Machine Modeling for Sign Language Recognition with Kinect. *ACM Transactions on Intelligent Systems and Technology (TIST)* (2015), Special Section on Visual Understanding with RGB–D Sensors.

[18] Muchenxuan Tong, Kun-Hong Liu, Chungui Xu, and Wenbin Ju. 2013. An ensemble of SVM classifiers based on gene pairs. *Computers in biology and medicine* 43, 6 (2013), 729–737.

[19] Paulo Trigueiros, Fernando Ribeiro, and LuÃŋs Paulo Reis. 2012. A comparison of machine learning algorithms applied to hand gesture recognition. In *7th Iberian Conference on Information Systems and Technologies (CISTI).* IEEE, Madrid, Spain.

[20] B Yegnanarayana. 2009. *Artificial neural networks.* PHI Learning Pvt. Ltd.

[21] Zahoor Zafrulla, Helene Brashear, Thad Starner, Harley Hamilton, and Peter Presti. 2011. American sign language recognition with the Kinect. In *13th international conference on multimodal interfaces.* ACM, Alicante, Spain, 279–286.