UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

COMPUTER SCIENCE HONOURS
FINAL PAPER
2016

Title: Leap Motion Controller for South African Sign Language Recognition

Author: Anna Borysova

Project Abbreviation: HANDGR

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

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**

Leap Motion Controller for South African Sign Language alphabet recognition

Anna Borysova
University of Cape Town
Cape Town, South Africa
annasborysova@gmail.com

ABSTRACT

Gesture recognition has become more feasible due to the emergence of depth based devices, such as the Leap Motion Controller (LMC). This paper investigates the use of the LMC for sign language recognition using various machine learning techniques, in a real-world setup. The 26 static and dynamic alphabet gestures of South African Sign Language provide an interesting challenge with its many similar gestures, while being limited enough not to require facial recognition or body movement. The classification algorithms explored include: Support Vector Machine (SVM), Artificial Neural Networks (ANN), k-Nearest Neighbour (kNN), and a soft voting classifier to combine all three. Hyper-parameter tuning and feature selection improved classification accuracy, but gave diminishing returns. Using the Leap frame with the highest confidence, instead of the first frame improved accuracy further. The classifiers were evaluated using a leave-one-out approach on 35 participants. The best performing classifier, at 52% accuracy, is the SVM, the second, at 50%, is the ANN, and finally the kNN at 49%.

KEYWORDS

Machine Learning, Leap, sign language, hand gesture recognition

1 INTRODUCTION

Over the last several years, the scene of the hand gesture recognition field has changed due to the emergence of several depth based devices, for example, the Microsoft Kinect and the Leap Motion Controller (LMC). Depth based gesture recognition devices use infra-red light to detect the distances between the device's camera and the object in from of it to create a depth map. These devices have been successful in their intended use cases, but sign language recognition provides more of a challenge due to its complexity.

The LMC has been used to recognise several sign languages, including Greek Sign Language (GSL) [25], Arabic Sign Language (ArSL) [11, 19, 20], Chinese Sign Languages [29], American Sign Language (ASL) [8, 12, 16–18], and South African Sign Language (SASL) [24]. More information on previous work is given in Section 3. The SASL alphabet has been chosen for this classification task for its relevance to South Africa, and the machine learning challenge its several similar looking gestures provide (see Figure 1).

The data will be recorded using the Kinect and the Myo in conjunction with the LMC, to allow a comparison between the three devices, as they record different aspects of the gesture. The Kinect is a depth-based camera similar to the Leap, but positioned differently. The Myo reads electromyographic data from the forearm when a gesture is made. This data will be recorded in a real-world scenario in order to evaluate the devices more accurately for real use.

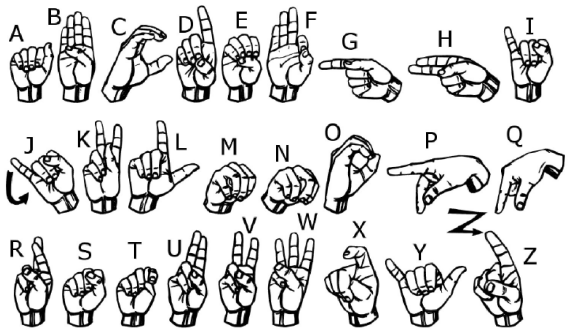

Figure 1: SASL alphabet gestures

This paper will attempt to answer the following research question: *Which machine learning techniques and which classifier out of the kNN, the SVM, the ANN and a Voting classifier result in the highest accuracy score and precision score, when classifying SASL alphabet gestures recorded by the LMC in a real-world scenario?* The chosen methods followed in order to answer this question may be found in Section 4. Section 5 will follow with the results, and Section 6 will conclude.

2 OVERVIEW

2.1 Leap Motion

The LMC is a depth based hand recognition tool with two main use cases: standalone gesture recognition, or gesture recognition when using a virtual reality headset. Two separate modes are provided for these two use cases. There are two main software versions for the LMC: V2 and Orion. The V2 version is supported for more platforms, while Orion focuses on the VR use case.

The Leap software interprets the depth based images from its two cameras and constructs a model of the hand. Its API provides access to the features of this model hand and the raw images, but no depth map data [22]. The features of the hand model include the location of finger joints, the directions of the fingers, and the confidence value, which is how accurately the hand model seems to match the images of the real hand. These predefined features simplify testing, but LMC's own recognition software struggles with representing the hand accurately when some fingers are obscured[22]. It is not clear whether Orion addresses these issues. Occlusion may be reduced either by tilting the device or by facing the gestures towards the camera. Mohandes et al. [20] used two perpendicular LMC devices, however the performance was not increased drastically when compared to other approaches.

A broader problem with using the LMC for sign language recognition (SLR) is that it does not recognise facial expressions which is extremely important for interpretation of sign language [3, 15, 27]. This is not a problem for the current research, as it is focused on

the alphabet, but any extension of this research into broader SASL must also include a device capable of recognising facial and body movements, such as the Kinect.

2.2 Machine Learning

The use of machine learning helps overcome several difficulties. For example, it can help with some shortfalls of the LMC, it can handle the complex and numerous sign language gestures [9], and it can handle with the different ways people repeat a particular sign [19].

The variables affecting the results of the final results can be categorised into the following stages: data gathering set-up, data, feature selection, classification, and the train-test split. All of these interact in order to produce the final results (see Figure 2). The feature selection stage selects a subset of the data features in order to increase the accuracy of the classifier. The training data is used to train a classifier to recognise new instances of gestures, and place them into classes, which in this case are the 26 letters of the SASL alphabet. The variables for this paper have been chosen by applying combinations of previously explored variables to a realistic data set.

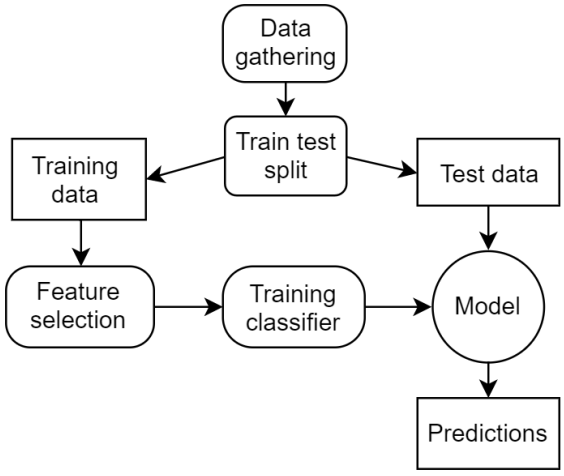

Figure 2: Machine learning pipeline

3 RELATED WORK

Comparison between papers is often not straightforward as they use different sizes of data, different evaluation methods, and most importantly, different sets of gestures. Those with more data per participant can be expected to perform better, as there is more data and less variation. However, papers with more participants are more likely to have results that are generalisable to the wider population.

Several papers on alphabet gesture recognition are compared in Table 1. The research done by Seymour et al. on SASL[24] was not focused on alphabet gestures and so it is left out of this comparison. However, the ASL alphabet is similar to the SASL alphabet, so in the absence of SASL alphabet studies, ASL alphabet results are considered most relevant.

The gestures analysed in the literature tend to be quite different from one another. ASL was the most frequently studied language, but even within that, the chosen gestures varied significantly. Most notably, Marin et al. chose only ten gestures, corresponding to the numbers 1-10. This means that the gestures chosen are even less representative of sign language than just the fingerspelling alphabet. Indeed, the gestures most often confused in the rest of the literature (M, N, and T)[8] were all missing from this data set.

3.1 Data gathering set-up

Quesada et al. explicitly compared two set ups of LMC experimentation [23]. One was the user-sensor set up, where the LMC lies flat on a surface, and the user tilts their gestures down towards the camera. The other was the user-user set up, which positioned the LMC underneath the gestures, but the gestures were facing the horizon. The user-sensor set up did perform better, but surprisingly, the user-user set up also recognised a fair number of gestures, despite the problems with occlusion. Marin et al. [17, 18] tilted the hand forward towards the LMC and achieved a maximum accuracy of 95.8% (see Table 1), and Mapari et al. [16] tilted the LMC towards the palm and achieved 90% accuracy.

For data gathering involving both the Kinect and the LMC, Marin et al. [17] placed the Kinect on a table and the leap directly under the hand. To prevent occlusion, the gestures were tilted slightly towards the LMC. They achieved slightly over 80% accuracy.

The gestures are mostly recorded one at a time [17, 18], however, Quesada et al. and Simos et al. implemented a continuous data gathering process [23, 25]. Quesada et al. interspersed every gesture with the gesture for the number 5 to ensure separation of gestures, while Simos et al. used prolonged pauses to indicate a new gestures, and movement to indicate the beginning of a new (static) gesture.

The continuous process, while it seems more complicated, may actually improve results, as the Leap software needs to track the hand, and so in the continuous process, it has more opportunity to tailor its internal model of the hand to the input it receives.

3.2 Feature selection

Marin et al. [18] had interesting results when comparing three different feature selection algorithms: f-score (measure of how discriminative a factor is) feature selection; sequential feature selection(the feature whose addition achieves the greatest improvement in accuracy is added to the set, until the required number of features is reached); and Random Forests feature selection. The best results were generally found with the sequential algorithm and the SVM classifier (reaching 95.8% with only 16 features). However, because a linear SVM was used for the sequential algorithm, it may have bias towards the SVM over the Random Forests classifier. This result can be compared to Marin's earlier paper [17] which finds a somewhat lower accuracy of 91.28% (joint Leap and Kinect data) using 6 features. This difference may be either due to better selected features, or simply due to more features available.

Simos's feature sets control for hand size (boneTranslation) and hand location (palmTranslation) and both of these get very good results (about 99% accuracy). However, previous papers [17, 18] also adjust for these variables and do not get accuracies as high as Simos et al, even when keeping the classifier constant. This suggests that another variable is responsible for Simos's success, possibly the GSL gestures.

3.3 Classification

Classification involves deciding to which gesture class a new gesture belongs. Classifier suitability is very much dependent on the problem, for example some of the classifiers explored have a bias towards either static or dynamic gestures: the ANN performs best for static gestures while Random Forests performs best for dynamic

Table 1: A summary of various papers using LMC for SLR

Authors	Gestures	Data set	Testing	Features	Classifier	Accuracy
Chuan et al. [8]	26 ASL letters	2 people x 2 sets	four-fold cross validation (3 train : 1 test)	pinch strength, grab strength, average distance, average spread, average tri-spread, extended distance, dip-tip projection, OrderX, and angle	kNN	72.78%
					SVM	79.83%
Elons et al. [11]	50 ArSL gestures	4 people x 1 set	four-fold cross validation (2 train : 2 test)	finger positions distances	MLP	88%
				finger position		82%
Funasaka et al. [12]	24 static ASL letters	unclear	unclear	palm normal vector, fingertips position, arm direction and fingertip direction	Decision tree	82.71%
Marin et al. [17]	10 ASL numbers	14 people x 10 sets	training set of M users	position of the fingertips, palm center, hand orientation, fingertips angle, fingertips distance, and fingertips elevation.	SVM	80.86% (Leap only)
Marin et al. [18]	10 ASL numbers	14 people x 10 sets	leave-one-person-out	F-Score	SVM	94.5% (128 features) 60.1% (16 features)
					Random Forests	92.6% (128 features) 57.5% (16 features)
				Sequential	SVM	95.9% (128 features) 95.8% (16 features)
					Random Forests	94.1% (128 features) 90.7% (16 features)
				Random forests	SVM	95.8% (128 features) 93.7% (16 features)
					Random Forests	94.2% (128 features) 90.8% (16 features)
Mohandes et al. [20]	28 ArSL fingerspelling gestures	10 samples per letter (M, N unknown)	leave-one-out cross validation	finger length, finger width, average fingertip position, hand sphere radius, palm position, hand pitch, roll and yaw	DS	97.1%
					LDA	97.7%
Mohandes et al. [19]	28 ArSL fingerspelling gestures	10 samples per letter (M, N unknown)	five-fold cross validation	finger length, finger width, average fingertip position, hand sphere radius, palm position, hand pitch, roll and yaw	Naive Bayes	98.3%
					MLP	99.1%
Simos et al. [25]	24 GSL fingerspelling gestures	6 people x 10 sets	6-fold leave one person out cross validation	boneTranslation	SVM	99.028%
				palmTranslation		98.96%
Mapari et al. [16]	32 ASL fingerspelling and number gestures	146 people x 1 set	cross validation (90% training, 10% test)	finger position, palm position, distance between positions, angle between positions	MLP	90%

gestures [7]. The SVM and the kNN perform well for both types of gestures [7].

Support Vector Machine (SVM). This is the most popular algorithm in the literature for classification of the gestures [7]. The SVM algorithm finds a hyperplane that separates two classes cleanly, and with as much margin as possible [1]. The hyperplane is a separating plane of dimension $n-1$ where n is the number of features defining a data point. This method was used by Chuan, Marin, and Simos, and they all found accuracies ranging from 79% - 99%.

Neural Networks (ANN). The multi layer processor (MLP) neural network is a type of neural network. It is the most explored type in the literature, so this is the only type of ANN that will be discussed. The MLP takes features as input, processes them in a hidden layer, and outputs decisions [10]. The MLP usually consists of three layers: the input layer, with the same number of nodes as the number of features; the output layer with the same number of nodes as the number of classes; and the hidden layer. The hidden layer allows a complex interaction between the input features and the output classes based on the node connection weights. The weights are modified during the classifier training. This method was used by Mohandes, Elons, and Mapari, and accuracies ranging from 82% - 99% were found.

k-Nearest Neighbour (kNN). The kNN algorithm finds the k nearest neighbours (from the training set) to a given instance, and classifies it according to the classification of the neighbours [26]. For kNNs 'nearest neighbour' means the data point in the training set that is closest to the given data point, when the distance is measured between the two feature vectors (sets of features). This was used by Chuan et al. and Clark et al. who achieved accuracies of 72.78% and 82.5% respectively.

Random Forests (RF). The RF algorithm uses a set of trees for prediction, where each tree depends on a vector sampled randomly

from the same distribution [5]. Marin et al. got accuracies from 57% to 94% using Random Forests, depending on what features were selected.

Naive Bayes Classifier (NBC). The NBC algorithm uses the Bayesian formula $P(A|C_1)P(A|C_2)...P(A|C_n) = 1$ to predict the probability of an event happening, given other events and their probabilities [19]. Mohandes et al. achieved about 98% accuracy using this method.

Dempster-Shafer (DS). The DS algorithm generalises the Bayes algorithm by adding an uncertainty term, θ so the equation is as follows: $P(A|C_1)...P(A|C_n) + \theta = 1$ [20]. Mohandes et al. achieved a 97.1% accuracy when combining the data from two LMCs at the classifier level using DS.

Linear Discriminant Analysis (LDA). To ensure maximum class discrimination, the LDA algorithm decreases data dimensionality by using linear combinations of factors obtained from a projection matrix [20]. Mohandes et al. achieved a 97.7% accuracy when combining the data from two LMCs before classification, and classifying using LDA.

3.4 Evaluation

A typical approach to classifier evaluation is to split the data randomly into a train and a test set, but a couple of papers [17, 18] choose the 'leave one out approach': the test set is one person's data, and the rest of the data is the training set. These two approaches may have different effects on the accuracy. The random data split allows for the classifiers to be more 'prepared' for the test data, as it has already seen similar gestures in the training set performed by the same person. However, while the 'leave one person out' approach allows for a more accurate representation of how the classifier may be used in its applications, it does benefit from having more training data if there are more than 4 participants, as in the research of Marin et al. [17, 18]

4 METHODS

The python scikit-learn library is used for all machine learning processes. [21] An overview of the experimental variables and the machine learning techniques used can be found in Table 2 and Figure 3, and the motivations follow.

Table 2: Table showing experimental variables

Gestures	Data set	Testing	Features	Classifier	Classifier params
26 SASL alphabet gestures	35 people x 10 sets	leave one person out	F-score & RFE (extra trees)	SVM	Randomised grid search
					Default params
				MLP	Randomised grid search
					Default params
				kNN	Randomised grid search
					Default params

4.1 Data type and source

Gestures. As part of the specification of the project, the SASL alphabet gestures were chosen (see Figure 1). This includes some similar-looking gestures (such as A, E, M, N, O, and S) and two dynamic gestures (J and z). These features of the data ensure some sort of relevance of the results to the rest of SASL, and present a challenge suitable for machine learning. The reference images used for recording the data are slightly different to Figure 1. The set used was obtained from June Bothma[1], as it was deemed less ambiguous. Unfortunately the reference used is not publicly available.

Participants. The participants recruited consisted of students of the University of Cape Town who had no previous experience with SASL. Some demographics have been recorded, such as handedness, age, gender, and race. The majority of participants were right-handed white men in their 20s. More detail on the demographics can be found on the project website.

4.2 Data gathering set-up

Figure 4 shows how the three devices are positioned relative to each other: the leap is positioned under the hand, the Kinect faces the participant about half a meter away, and the Myo is worn on the arm which performs the gestures. The physical set-up is very similar to the one used by Marin et al. [17], with the gestures tilted slightly towards the Leap. This tilting does not compromise the recordings taken by the Kinect, as the Kinect is also set up lower than the gestures are performed, and so also benefits from the more direct angle. However, even this tilting is suboptimal for the kinds of gestures the LMC is designed. The gestures are not tilted down any more because priority is given to the gesture being correct from a person's perspective in order to best simulate how the device would normally be used.

Each participant performs 10 instances of every gesture, resulting in 260 data points for every participant. Each of these data points consists of a Myo recording, a Kinect recording, and a Leap recording. For this paper, only the Leap data is considered. The program runs for only 10 gestures at a time due to complications with the Myo, so every 10 gestures the Leap is forced to recreate the hand model. During the 10 gestures, the Leap software is free to track the hand, however only once the participant is in position is the data recorded. The order of these gestures is randomised, for several reasons, the most important of which is that the first gestures will be more prone to misidentification due to the limited time allowed for improving the Leap's internal model of the hand. If the interpretation is so bad that no hand is detected, then it does not record. To ensure recognition, the participant must move into the gesture slowly from an easily recognisable position. Because of this, there is a chance that the gestures that are moved into slowly are more accurately represented by the Leap software.

4.3 Data processing

The hand model which the Leap software constructs is stored in a `frame` object which corresponds to a particular moment in time. `Frame` objects are serialised to text files using the Leap's own serialisation method. Due to the extremely high frame rate of the LMC, only every fifth frame is deserialised and processed. This reduces the effective frame rate from about 100 frames per second to about 20.

[1] SASL teacher at UCT

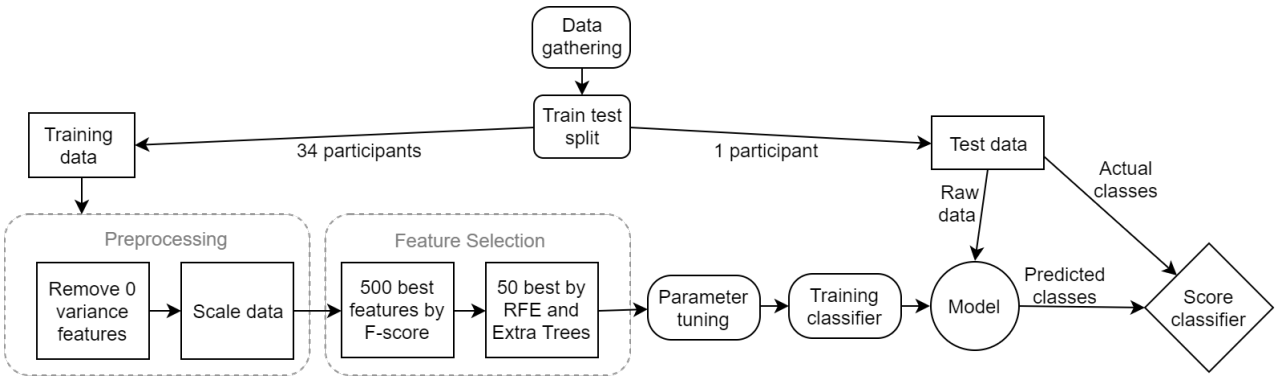

Figure 3: Diagram representing one test

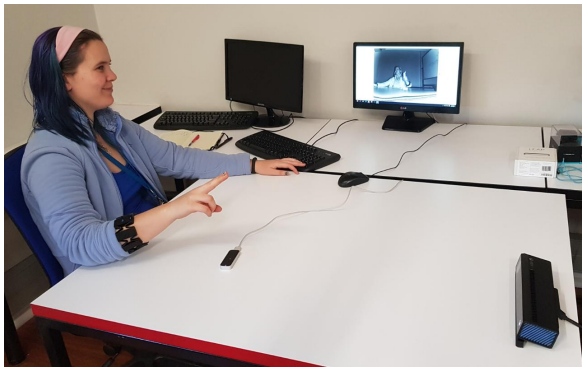

Figure 4: Data gathering set-up

Preprocessing. One performance of a gesture corresponds to one data point, which is defined as a vector of features extracted from the hand model.

All relevant features are extracted recursively from the `frame` object, with the feature label being built as the hand structure is traversed. For example, the x-coordinate of the tip position of the index finger is given by the feature labelled `hand_finger_1_tip_position_x`. The finger positions are transformed to be relative to the palm center and are also included in the potential features. This processing results in 1504 features per frame. It is important to note that depending on the method of handling the frames per gesture, the effective feature number is affected. For example, using two frames to define a gesture would result in 3008 features.

For processing time reasons, only one frame is used per gesture, and treated as representative of the entire gesture. Thus, the dynamic gestures are reduced to a static image. The selected frame is the one with the highest confidence value that the Leap motion software assigns to each hand model. Features which do not vary between any of the data points are removed, however low variance features are not removed as all remaining features are reasonably high variance. At this stage, the data is scaled, because many of the methods explored are designed for scaled data. Scaled data consists of feature vectors where the distribution for each individual feature has zero mean and unit variance. This ensures that features which change on a larger scale in comparison to other features do not dominate the learning process of a classifier. [2]

[2]http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-scaler

Feature selection. The check-list provided by Guyon et al. [13] was loosely used as a guideline for feature selection. Adhoc feature selection was used only in the sense that meta features such as "id" and "timestamp" were not included. Additionally, features conceptually unrelated to the gestures, such as finger width, were excluded as much as possible to ensure generalisability. A problem with the data extracted from the Leap API is that the features do not directly describe the shape of the hand, but rather just give disjointed values. For this reason, it seems prudent to combine features into ones more suitable for gesture interpretation. This is not done, due to time constraints.

As recommended by Guyon et al. [13], a variable ranking method is used first to cull the features to a more reasonable number. The SelectKBest method from sklearn is used to select the top 500 features, sorted by their f-score. The f-score is calculated using the f-test, which computes the ratio of inter-class variance to intra-class variance for a particular feature. If a feature has low variance within a particular class, but a high variance outside it, then it is given a better score, as it is more discriminative.

The features are then recursively eliminated to 50 of the most important features, as determined by an Extra Trees Classifier. This classifier was chosen as it has similar accuracies as the three main classifiers chosen, when comparing the classifiers with the default parameters. The SVM, kNN, and ANN were not chosen for this so as not to unduly bias the features to a particular classifier.

4.4 Classification

Classification involves taking a given data point and labelling it with a class, using knowledge granted by training data. So in the case of gesture recognition, a classifier would be given data representing a gesture, and it would output which letter that gesture represents. Four classifiers were experimented with: the SVM, the kNN, and the MLP, as they are the most successful classifiers in the literature. Additionally, a voting classifier, which uses the output of the previous three classifiers to make a decision, was tested to allow the three classifiers to cover each other's weaknesses.

These classifiers all have hyper-parameters which can be tuned in order to make the classifier perform better for a particular problem. The literature explores only some basic hyper-parameters for these classifiers, however this research experiments with a wider range of hyper-parameters. To run the experiments in a reasonable time limit, a randomised cross validation search across the defined parameter space is used. This involves taking a subset of the given

training data, training a classifier with certain parameters, and testing it on a different subset of the training data. This is repeated for various combinations of parameters until an optimal solution is found. A randomised search is used over a grid search due to its comparable accuracy but much greater efficiency [2].

An interesting trap mentioned by Cawley et al. [6] is the trap of using all the training data for feature selection, and then using a cross validation parameter search on the same training data, using the "pre-selected" features. The correct way of doing this is to do feature selection within every cross validation fold for the model evaluation. However, this would add a lot of extra computational time, so this is not done.

A summary of the parameters explored for each of the classifiers can be found in Table 3, and the motivations for the parameters follows.

Table 3: Parameters explored using randomised grid search

Classifier	Parameter	Values explored
SVM	kernel	linear, rbf, polynomial
	C	2^{-5} to 2^{15}
	gamma	2^{-15} to 2^{3}
	degree	1 to 5
	decision function	one vs one, one vs rest
kNN	k (n_neighbours)	1 to 50
	weights	uniform, distance
	algorithm	auto, ball tree
	metric	1 to 5
ANN	hidden layer nodes	26 to 200
	activation function	identity, logistic, tanh, relu
	alpha	0.00001, 0.001, 0.1, 10, 1000
	solver	lbfgs, sgd, adam
	learning rate	constant, inverse scaling, adaptive
	learning rate init	10^{-6} to 1

SVM. For the SVM, the literature mostly explores linear and Radial Basis Function (RBF) kernels, finding that RBF tends to perform better. However, there are two other options for the kernel, the polynomial and the sigmoid kernel. The sigmoid kernel is invalid for many parameter combinations, and so it is not considered. In general, an SVM forms some sort of decision boundary using the training samples. This decision boundary determines where a data point (or gesture, in this case) is classified as a particular class(or letter). An illustration of the SVM is provided in Figure 5.[3]

The linear kernel separates the training samples using linear hyperplanes, while the polynomial kernel uses polynomial hyperplanes. The RBF kernel (the default choice) uses $e^{-\gamma||x-x'||^2}$ to form the decision boundaries, where x and x' are two data points, and γ is a scaling factor. Because $||x - x'||^2$ is the Euclidean distance squared, the RBF can also be considered a similarity function.

The `C` parameter, used by all three explored kernels, allows the SVM to ignore some misclassified data in order to have a simpler decision boundary. [4] A higher value of `C` means the model prioritises not misclassifying samples, while a lower value means the boundary simplicity and the width of the boundary is prioritised. A higher `C` value may result in over-fitting if the data is noisy, however this effect may be offset by a good feature selection strategy. The range

[3]http://www.coxdocs.org/doku.php?id=perseus:user:activities:matrixprocessing:learning:classificationparameteroptimization

[4]http://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html

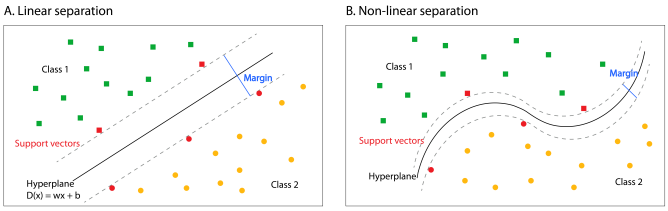

Figure 5: SVM illustration

explored is 2^{-5} to 2^{15} following the advice of Hsu et al. [14] The default value for `C` is 1.

The `gamma` parameter, used by the polynomial and RBF kernels, affects the area of effect of one data point. A low value means one data point has a large area of effect, and a high value means the area of effect is low. This parameter also helps prevent over-fitting, by ensuring that outliers do not affect the decision boundary too strongly, but `gamma` must not be so large as to have only the data point in its area of effect. Thus `gamma` can balance out a lower `C` value, which uses less memory and results in a model that predicts faster. The range of values chosen for this parameter is 2^{-15} to 2^{3} [14]. The default is `auto` which determines the best `gamma` value automatically, and is also explored in the parameter search.

The `degree` parameter is the maximum degree of the terms in the polynomial function of the polynomial kernel. A high value risks over-fitting, and a low value risks not capturing the complexity of the data. The default value is 3, and the range explored is 1-5.

Finally, the decision function shape can be either one vs one (ovo) or one vs rest (ovr). Both are explored. The ovo algorithm finds an optimal boundary between each pair of classes, while the ovr algorithm compares one class to the rest. Thus, ovr (default) results in fewer sub boundaries and thus quicker computation, but also in an imbalanced dataset. [4]

kNN. The k Nearest Neighbour algorithm classifies a data point based on the classification of its k nearest neighbours. The distance between two data points is the distance between the two feature vectors. The main parameter is the number of neighbours to consider, k (called `n_neighbors` in scikit-learn). The range explored for this parameter is 1-50 neighbours. With more data points, and more variation there is a chance a greater upper bound may be needed. The default value for this parameter is 5.

The `weights` parameter determines the relative weighting of the k neighbours. The weights `uniform` (default) and `distance` are explored. Uniform weighting means all k neighbours are given equal consideration, no matter how far they are from the point to be classified. The distance weighting prioritises those points closest to point to be classified. This parameter allows a greater k to make the algorithm more stable, without unduly valuing the further points within the k neighbours.

The `algorithm` parameter allows `auto` (default), `brute`, `kd_tree` and `ball_tree` options. These are the algorithms used to calculate the distances between data points, and thus, potential neighbours. The `brute` algorithm is infeasible for large volumes of data, and the KD tree algorithm is inefficient for high dimensional data. There is a chance that with a lower number of dimensions (more aggressive feature selection) the KD tree algorithm may become viable once again. For this reason, the `auto` algorithm is chosen along with the `ball_tree` algorithm. The ball tree algorithm partitions all the

data points into hyperspheres, which allows for quicker calculation of the nearest neighbours.

The `metric` parameter which determines the way distance is calculated between two data points is left as the Minkowski metric, and is modified using the parameter `p`. When `p` is 1 the distance measurement used corresponds to Manhattan distance, and when `p` is 2 (default), the metric used corresponds to Euclidean distance. Because a lower `p` is better for high dimensional data the range explored is 1-5.

The following two equations demonstrate Euclidean and Manhattan distance respectively, where x and y are data points (feature vectors), x_i is the i^{ith} feature, and n is the total number of features:

$$\texttt{euclidean\_distance} = \sqrt{(x_0 - y_0)^2 + ... + (x_n - y_n)^2} \quad (1)$$

$$\texttt{manhattan\_distance} = |x_0 - y_0| + ... + |x_n - y_n| \quad (2)$$

ANN. For the MLP, only one hidden layer is explored, as usually adding more layers adds more computation without improving results. The node numbers explored for the hidden layer are from 26 (the number of output nodes) to 200, to allow for the complexity of the input data. The default for this parameter is one hidden layer, with 100 nodes. Too few nodes may lead to not capturing the complexity of the data, whereas too many nodes may lead to over-fitting.

The activation function determines when a node is activated and for which values the activation 'pulse' must be propagated, eventually leading to the output layer and outputting the result. The activation functions explored are the `identity`, `logistic`, `tanh` and `relu` (default) functions. The `relu` function is the identity function, but 0 for all negative values. The activation function corresponds to the distribution of the underlying data, but because the distribution of the data in this case is unknown, all the available functions are tested.

The parameter `alpha` is the regularization term to balance over-fitting. A lower `alpha` encourages a more complicated decision boundary, and a higher value encourages a simpler boundary. [5] The range explored is from -5 to 3, in logarithmic space.

The `solver` parameter determines the method used for optimization of the node connection weights. The solvers explored are `lbfgs` (quasi-Newton solver), `sgd`, and `adam` (default). The SGD algorithm is an incremental gradient descent solver, which allows a user specified learning rate, while the Adam solver uses adaptive learning to get to a solution quicker. The SGD solver requires a learning rate functions, which are described next.

The learning rate of an ANN can be either constant or decreasing with time to encourage convergence to a solution. A higher learning rate means that the solver will reduce the weight of previous observations more readily when a new, contradicting, observation is made. A low learning rate means that new information is incorporated slower. The functions explored as the learning rate are: constant (default), inverse scaling (decreasing exponentially), and adaptive (if training loss does not decrease, divide the learning rate by 5)[6]. The `learning_rate_init` value is the initial value if the algorithm needs one. The range explored is 10^{-6} to 1. [28]

Voting classifier. Finally, the Voting classifier takes into account the decisions of all three classifiers described, and outputs the most likely class. This classifier is included to take advantage of the strengths of all three classifiers, in the hopes of getting a better overall accuracy. The voting type used is soft, which takes into account the probability/confidence that a classifier outputs, rather than just the decision by using weighted average probabilities.[7] This strategy allows a weighting of importance to be given to every classifier, however this is not used because preliminary testing did not show significant differences between the accuracies of the three classifiers. Given a particular gesture to classify, the classifiers return 26 probabilities each, all corresponding to the percentage probability that the given gesture belongs to a particular class. For every class, the average of the three probabilities from the three classifiers is calculated, and the Voting classifier outputs the class with the highest average probability as the most likely class.

4.5 Evaluation

Cross-validation. The classifiers are tested using a leave-one-out approach. A single run involves 34 participants as the train set, and 1 participant as the test set. This is repeated 35 times, with each run using a different participant as the test set. See Figure 3 for an overview of a single run. While this approach does mean that the models are trained on a lot of data, this is believed to be a more real-world evaluation, as the test data is all new. Only the training data set is used for decisions about data processing, feature selection, and for parameter tuning. In the data processing and the feature selection stages, the decisions made using the training set are used to modify the test data blindly so as to ensure the same features are used in both training and testing. For example, if the feature selection process decides features 3 and 6 are irrelevant, those features are removed in the test set without judging how effective they are at predicting the test data. Naturally, only the training set is used to train the models, and the test set is left to test the models.

Error analysis. Several metrics are recorded in order to facilitate analysis of the results: the confusion matrix, the precision, recall, and f1 scores for each class and the average for of each of these values for each classifier.

The confusion matrix plots the actual classes vs. the predicted classes. It allows a clear visual analysis of the misclassification of gestures: It shows which gestures were accurately classified and which were confused for other gestures. It is a visual representation of the true positive, true negative, false positive, and false negative values.

A true positive (T_p) occurs when the the predicted class is the same as the actual class. A false positive (F_p) occurs when a classifier classifies a gesture as the incorrect class. A true negative (T_n) is when the classifier correctly predicts a gesture as not part of an incorrect class. A false negative (F_n) occurs when the classifier does not classify a gesture into the correct class.

Precision (P) is the ratio of correctly identified instances of a class, to the total number of times the algorithm identified a gesture as that class. In formula, precision is given by $\frac{T_p}{T_p+F_p}$. Recall (R) is the ratio of correctly identified instances of a class to all the instances where the gesture was truly that class. In formula, recall

[5] http://scikit-learn.org/stable/auto_examples/neural_networks/plot_mlp_alpha.html

[6] http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

[7] http://scikit-learn.org/stable/modules/ensemble.html#voting-classifier

is given by $\frac{T_p}{T_p+F_n}$.

The f1 score is the harmonic mean of precision and recall, given by $2\frac{PR}{P+R}$. However, this value is ignored for analysis because the classes are well balanced by design and so will not give much more information than the accuracy. The accuracy is the sum of all true positives over the total number of predictions. This score is focused on during the feature selection and the hyper-parameter tuning stages.

Ideally, the accuracy, recall, and precision are all high. However, in the interests of interpreting a language, the classifiers giving the best precision and thus the lowest false positive values are prioritised.

5 RESULTS

A summary of the final accuracies of the explored classifiers is given in Table 5. The data set and the full logs for all the experiments run for this investigation are given on the project website.

5.1 Preprocessing

The features that were had no variance and thus removed were the `*_basis_origin_*` features and the direction features of the metacarpal bone (closest to the palm) of the thumb (`hand_finger_0_bone_0_direction_*`). The metacarpal bone of the thumb has no length in the Leap hand model, and thus, all the `hand_finger_0_bone_0_*` features do not provide any information and are irrelevant. The positional values of the metacarpal bone of the thumb are kept because they change with the hand centre, and thus are not 0 variance. However these features do not provide any new information and thus do not perform well in the feature selection stage. The basis specifies the orientation axes of a feature such as a bone, hand, or arm. The origin is a vector specifying translation factors on all three axes. Because the basis is independent of the positional values, its origin vector is always the same. The combination of removing 0 variance features and scaling the data improved the accuracy by about 5% in preliminary experimentation.

5.2 Feature selection

The most frequently selected feature by the randomised grid search is the transformed thumb tip position (relative to the hand centre), which corresponds with heuristic analysis, considering the number of gestures distinguished solely by the thumb position. The general hand descriptors such as pinch strength, grab strength and grab angle, were not selected as often as the raw finger features, however, the pinch strength feature does make it into the top 50. Using the top 50 most commonly chosen features during preliminary testing worsened performance, because taking the most selected features does not consider the importance of combinations of features. Some features have very high correlation, for example: the thumb tip position and the 'next joint' of the thumb distal bone actually refer to the same feature. An aggregate of features values both these features equally highly, while in a single run of feature elimination only one of these would be chosen, as the first one adds all the value and the second one adds nothing new. A potential method of combining the best of both approaches is by running recursive feature elimination on the top 100 features, and using the resultant features as a constant set for every run. However, because feature selection is not the time bottleneck of the pipeline, the feature selection is allowed to run for every test (see Figure 3).

5.3 Parameter tuning

The parameter tuning phase adds a lot of extra computational time, so some parameters were made constant based on preliminary experimentation. The most commonly selected parameter over several runs of the randomised grid search was chosen, and the resultant classifier was compared against a classifier with the parameter selected by randomised grid search in order to validate that the chosen parameter value is indeed optimal. This is run several times, and more parameters are made constant incrementally. Using this strategy, the accuracies of the two classifiers converge because the values not kept constant have the least effect on the accuracy of the classifier for this problem. Table 4 shows the parameters kept constant for each of the classifiers after preliminary testing. Setting any other parameters constant worsened the performance of the classifier on average (possibly due to non-comprehensive preliminary testing) and so are left to be tuned by the randomised grid search as shown in Table 3. In particular, the MLP was resistant to constant parameters, however this could be because the search space was greater than for any of the other classifiers.

Table 4: Parameters kept constant after preliminary testing

Classifier	Constant parameters
SVM	kernel='rbf', decision_function_shape='ovo'
kNN	weights='distance', algorithm='ball_tree'
MLP	None

5.4 Classifier scores

Overall, the accuracies achieved by the classifiers are very similar to each other (around 50%). However, the accuracies achieved using different participants vary greatly, from about 30% to 60%, but mostly lie around the 50% mark. Figure 6 summarises the accuracies achieved by the various classifiers over the 35 independent tests.

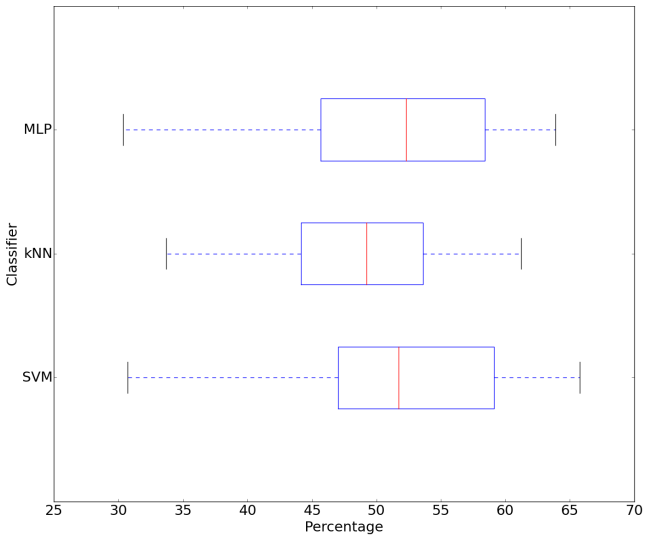

Figure 6: Box and Whisker plot of the accuracies for the SVM, kNN, and MLP over 35 independent tests

The mean recall, precision and accuracy values for each of the classifiers may be found in Table 5. At 52.2% accuracy, the SVM performs better than both the kNN (48.8% accuracy) and the MLP

Table 5: Means of scores and times over all 35 participants

Classifier	Recall	Precision	Accuracy (std)	Time (s)
kNN	48.82%	50.09%	48.82% (6.88)	0.395
SVM	52.16%	53.09%	52.16% (8.17)	0.312
MLP	51.14%	51.66%	51.14% (8.17)	0.012
Voting	52.07%	52.94%	52.07% (7.67)	0.720

(51.1% accuracy), but the differences between the classifiers are slight. This corresponds with the research of Chuan et al. [8], who found that the SVM performs marginally better than the kNN under the same circumstances. The voting classifier achieves 52.1% accuracy, but it does not provide any advantage over the individual classifiers. This could be because they all have the same strengths and weaknesses, possibly due to the underlying data. Indeed when the confusion matrices of the different classifiers are investigated, they all appear identical. Figure 7 shows the aggregate confusion matrix for all the classifiers over several runs.

In Table 5, the mean times taken in seconds to test the classifiers are shown. These are all tiny values, especially seeing as the time stated is the time taken to classify about 260 gestures, on average. The higher time for the kNN is due to the distance calculations which must be made during classification. The high test time for the voting classifier is due to the fact that all the other classifiers have to make their predictions before it can calculate its own prediction.

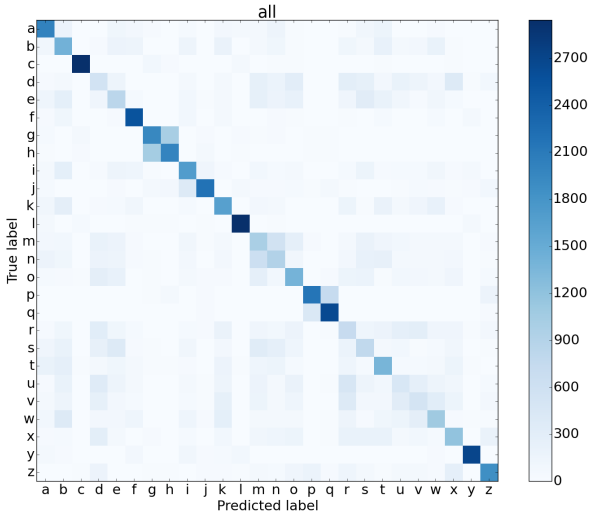

Figure 7: Confusion Matrix for all classifiers

5.5 Reliability of Data

While these accuracies are much better than random chance (≈ 3.8%), they are not as high as the accuracies reported in the literature. This is due to the realistic data gathering set up use for these experiments. For example, the Leap visualiser was not on display, so the participants did not have a chance to manoeuvre the Leap software into interpreting the hand correctly. The only requirement was that the hand gesture looks correct to a human, as this is the most realistic use case. Additionally, while the gestures were tilted slightly towards the device, priority was given to the human viewer being able to interpret the gesture. Thus, a suboptimal set-up for the Leap software was used, in order to mimic realistic use of the device. The result of this is the Leap software misinterpreting the data, and outputting an incorrect model of the hand.

As expected, similar looking gestures are confused for one another. For example, the gestures A, E, M, N, O, and S are similar to each other, due to their similar base, a fist (See Figure 1), and they are some of the worst classified gestures. This similarity, and oftentimes the occlusion of the thumb, results in poor interpretation of the hand by the Leap software. A similar issue occurs with the gestures D, U, and V as shown in Figure 1, however, an extra complication with this set is that the Leap struggles to detect raised fingers correctly. For example, D can be confused for E because the raised finger is simply not detected. This may be because the hand is not tilted down enough to be able to detect a raised finger. A milder example of this is the confusion of D for X. If a raised finger is detected, the perspective makes it unclear whether it is fully extended or slightly bent like in an X. Two gestures exempt from these problems are L and Y (Figure 1). This is because the thumb (and the pinky, in the case of Y), which is extended sideways, is not occluded by the rest of the palm and is clearly visible regardless of the tilt of the hand. As seen in Figure 7, the most confused gestures are U and V so these are investigated further by studying a visualisation of the recorded gestures. As seen in Figure 9, none of the visualised gestures were recorded correctly, and U and V look extremely similar, with the second extended finger not recognised at all. A recording of W is included to demonstrate further the problems Leap has when attempting to recognise raised fingers. As Figure 7 shows, W is actually not as badly recognised as U and V overall. This could be because the Leap recognises more raised fingers than in any of the other gestures, and so that is enough to classify it correctly sometimes. However, these are illustrations of only three gestures, and further analysis needs to be done to draw such conclusions.

Selecting the highest confidence frame improved the accuracies somewhat (by about 3%), however about a third of all the gestures still had a maximum confidence of less than 50%. These were not removed in order to keep the same number of gestures as used in testing the Myo and the Kinect.

(a) Gesture U **(b) Gesture V** **(c) Gesture W**

Figure 8: SASL gestures for U, V, and W

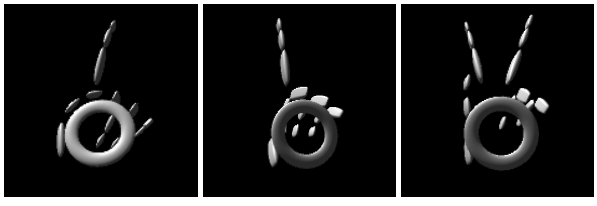

Figure 9: Visualisation of recorded gestures for U, V, and W respectively

5.6 Application

While a motivator for this research is a learning tool for SASL, the main focus is the Machine Learning challenge. For this reason, priority was given to the static gestures, and the dynamic gestures were treated as static gestures. While this results in relatively good accuracies for the dynamic gestures, this approach can only reasonably be extended to a SASL learning tool for the static gestures. This is because a static image of a dynamic gesture is not accurate enough to be able to verify the accuracy of the gesture in the real world. The reason they were classified well in this experiment is that even the static images tended to be dissimilar enough from the rest of the gestures to be classified as the correct gesture.

6 CONCLUSION AND FUTURE WORK

In agreement with the literature, the SVM performed best out of the three main classifiers with an accuracy of 52%. The MLP (the type of ANN used) performed second best with an average accuracy of 50%, and the kNN performed worst, with an accuracy of 49%. The voting classifier did not improve on the SVM, with an accuracy of 52%. The confusion matrices for all classifiers were visually indistinguishable, so underlying data was theorised to have been the bottleneck. This was shown to be correct using visualisations of the recorded data, and the effect was minimised by taking only the highest confidence frames for processing.

Because the data used for this experiment was gathered using the Leap, the Myo and the Kinect, the data can be combined validly in order to train a more accurate classifier. The different devices provide different perspectives on the gesture performed and so combining the data may prove useful.

In addition to its model of the hand, the Leap motion software also outputs raw depth images from its two cameras. This data was not gathered for this experiment, but may prove more useful than the Leap's hand model data.

During the recording phase, in addition to necessitating that a hand can be detected, a minimum confidence value could be required. While this does not take away from the person to person set-up (rather than person to Leap), it may take away from the convenience of the device. This experiment could be redone using the Orion Leap motion tracking software, which lacks the confidence value, but may recognise the hand more accurately.

Looking into taking advantage of multiple frames effectively may improve the accuracy, and allow the classifiers to be used for more than alphabet recognition. A simpler approach would be to select frames at certain time steps and simply extend the features.

ACKNOWLEDGMENTS

The author would like to thank Assoc. Prof. Deshendran Moodley and Assoc. Prof. James Gain for their insight and guidance; and Erin Versfeld and Shaheel Kooverjee for their support and contribution to the data gathering process. The NRF partially funded this research.

REFERENCES

[1] Asa Ben-Hur and Jason Weston. 2010. A user's guide to support vector machines. *Data mining techniques for the life sciences* (2010), 223–239.
[2] James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research* 13, Feb (2012), 281–305.
[3] MS Bhuvan, D Vinay Rao, Siddharth Jain, TS Ashwin, Ram Mohana Reddy Guddetti, and Sutej Pramod Kulgod. 2016. Detection and analysis model for grammatical facial expressions in sign language. In *Region 10 Symposium (TENSYMP), 2016 IEEE*. IEEE, 155–160.
[4] Christopher M Bishop. 2006. *Pattern recognition and machine learning*. springer.
[5] Leo Breiman. 2001. Random forests. *Machine learning* 45, 1 (2001), 5.
[6] Gavin C Cawley and Nicola LC Talbot. 2010. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research* 11, Jul (2010), 2079–2107.
[7] Hong Cheng, Lu Yang, and Zicheng Liu. 2016. Survey on 3D Hand Gesture Recognition. *IEEE Transactions on Circuits and Systems for Video Technology* 26, 9 (2016), 1659–1673.
[8] Ching-Hua Chuan, Eric Regina, and Caroline Guardino. 2014. American Sign Language recognition using leap motion sensor. In *Machine Learning and Applications (ICMLA), 2014 13th International Conference on*. IEEE, 541–544.
[9] Andrew Clark and Deshendran Moodley. 2016. A System for a Hand Gesture-Manipulated Virtual Reality Environment. In *Proceedings of the Annual Conference of the South African Institute of Computer Scientists and Information Technologists*. ACM, 10.
[10] Walter H Delashmit and Michael T Manry. 2005. Recent developments in multilayer perceptron neural networks. In *Proceedings of the seventh Annual Memphis Area Engineering and Science Conference, MAESC*. Citeseer.
[11] AS Elons, Menna Ahmed, Hwaidaa Shedid, and MF Tolba. 2014. Arabic sign language recognition using leap motion sensor. In *Computer Engineering & Systems (ICCES), 2014 9th International Conference on*. IEEE, 368–373.
[12] Makiko Funasaka, Yu Ishikawa, Masami Takata, and Kazuki Joe. 2015. Sign Language Recognition using Leap Motion Controller. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp), 263.
[13] Isabelle Guyon and André Elisseeff. 2003. An introduction to variable and feature selection. *Journal of machine learning research* 3, Mar (2003), 1157–1182.
[14] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, and others. 2003. A practical guide to support vector classification. (2003).
[15] Matt Huenerfauth, Pengfei Lu, and Andrew Rosenberg. 2011. Evaluating importance of facial expression in american sign language and pidgin signed english animations. In *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*. ACM, 99–106.
[16] Rajesh B Mapari and Govind Kharat. 2016. American Static Signs Recognition Using Leap Motion Sensor. In *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*. ACM, 67.
[17] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. 2014. Hand gesture recognition with leap motion and kinect devices. In *Image Processing (ICIP), 2014 IEEE International Conference on*. IEEE, 1565–1569.
[18] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. 2016. Hand gesture recognition with jointly calibrated Leap Motion and depth sensor. *Multimedia Tools and Applications* 75, 22 (2016), 14991–15015.
[19] M Mohandes, S Aliyu, and M Deriche. 2014. Arabic sign language recognition using the leap motion controller. In *Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on*. IEEE, 960–965.
[20] M Mohandes, S Aliyu, and M Deriche. 2015. Prototype Arabic Sign language recognition using multi-sensor data fusion of two leap motion controllers. In *Systems, Signals & Devices (SSD), 2015 12th International Multi-Conference on*. IEEE, 1–6.
[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[22] Leigh Ellen Potter, Jake Araullo, and Lewis Carter. 2013. The leap motion controller: a view on sign language. In *Proceedings of the 25th Australian computer-human interaction conference: augmentation, application, innovation, collaboration*. ACM, 175–178.
[23] Luis Quesada, Gustavo López, and Luis Guerrero. 2017. Automatic recognition of the American sign language fingerspelling alphabet to assist people living with speech or hearing impairments. *Journal of Ambient Intelligence and Humanized Computing* (2017), 1–11.
[24] Michael Seymour and Mohohlo Tšoeu. 2015. A mobile application for South African Sign Language (SASL) recognition. In *AFRICON, 2015*. IEEE, 1–5.
[25] Merkourios Simos and Nikolaos Nikolaidis. 2016. Greek sign language alphabet recognition using the leap motion device. In *Proceedings of the 9th Hellenic Conference on Artificial Intelligence*. ACM, 34.
[26] Paulo Trigueiros, Fernando Ribeiro, and Luis Paulo Reis. 2012. A comparison of machine learning algorithms applied to hand gesture recognition. In *Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on*. IEEE, 1–6.
[27] Ulrich Von Agris, Moritz Knorr, and Karl-Friedrich Kraiss. 2008. The significance of facial features for automatic sign language recognition. In *Automatic Face & Gesture Recognition, 2008. FG'08. 8th IEEE International Conference on*. IEEE, 1–6.
[28] D Randall Wilson and Tony R Martinez. 2003. The general inefficiency of batch training for gradient descent learning. *Neural Networks* 16, 10 (2003), 1429–1451.
[29] Jihai Zhang, Wengang Zhou, Chao Xie, Junfu Pu, and Houqiang Li. 2016. Chinese sign language recognition with adaptive HMM. In *Multimedia and Expo (ICME), 2016 IEEE International Conference on*. IEEE, 1–6.