ISSN: 0974-5823 Vol. 6 No. 3 December, 2021

International Journal of Mechanical Engineering

A Study of Deep Learning is used to Classify Hand Gestures with Reference to Matrix Representation

Dr. Vasanthakumari T.N

Assistant Professor of Mathematics

Govt. First Grade College , Tumakuru, Karnataka(India)

Abstract

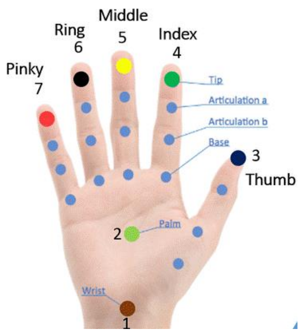
An entirely new dynamical hand gesture detection method was suggested in this paper based on joint data from the skeleton. Based on HIF3D's feature set, the suggested approach aims to represent entire body motions. Handwriting recognition was the inspiration for this feature set, which makes it unique. Our ultimate objective is to merge the representation and recognition of hand gestures and entire body gestures. Although we don't know when a gesture will begin or end, our dataset comprises gestures that are extremely varied in length, even though they are all part of the same class. This complicates the task of determining the sliding window's proper size. A deep learning algorithm is utilised to categorise hand motions based on a matrix representation in this research, which I have described in detail.

Keywords: Hand Gestures, Deep Learning, Machine Learning

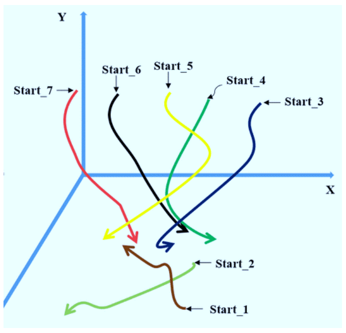
Introduction

Due to the fact that it is a component of Machine Learning, Deep Learning has risen in popularity in recent years. It has a variety of applications in today's world, ranging from categorising images and translating languages to developing the components of an autonomous vehicle. All of these functions are now performed entirely by computers, with no need for human intervention whatsoever.

Technological breakthroughs have made it possible to make tremendous progress in commercialized 3D sensing over the past few of years, which has greatly benefited the exploration of dynamic hand gesture detection. Whole-body movement recognition, on the other hand, has gained in popularity that since introduction of Kinect-style sensors. To anyone who pays attention, it should be evident that both research areas are focused with motions created by humans and that they are likely to encounter similar hurdles in their respective endeavours. In order to determine whether an action recognition feature-set is suitable for modelling dynamic hand gestures using skeleton data, the goal of this research is to evaluate its usefulness for this purpose. Even more concerning is that a large proportion of the known datasets is composed of previously divided movements, each of which is done with a single hand. A more complicated dataset resulted as a result of this, consisting of unbranched streams containing 13 different hand motion categories produced with such a single touch of two arms, which was much difficult to segment. The DHG dataset was used as a starting point for our evaluation, and then we collected our new data to complete the evaluation. It has been found that the results obtained with this technique are better than those achieved with previous techniques.



(a) The joints we've chosen to symbolise our dynamic hand gestures.



(b) Trajectories forming a 3D pattern are seen in this diagram.

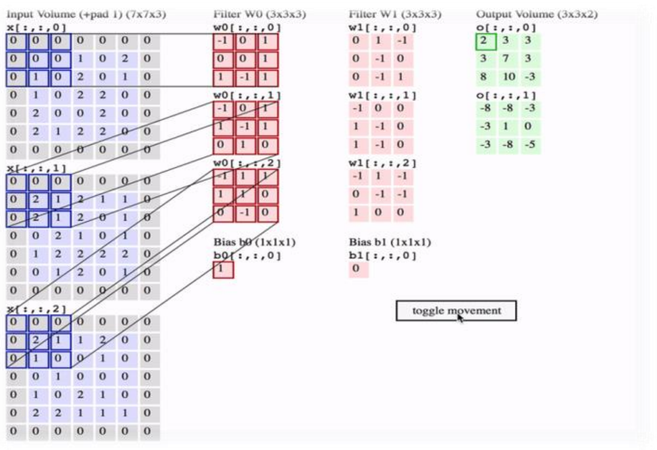
Figure (b) shows the generated 3D pattern S. For example, the first and second constructed trajectories represent the wrist and palm joints. The order of assembly trajectories remains: wrist, palm, thumb, index, middle, ring, pinky. From there it's on to HIF3D features extractor. The generated features reported increased level information about the 3D pattern's shape. The very first characteristics f_1, f_2 *and* f_3 (equation 1) represent the commencement of a gesture along the X, Y, and Z axes. In fact, the initial and end points' placements are often used to distinguish patterns.

$$\mathbf{f_1} = \frac{x_1 - c_x}{l} + \frac{1}{2}, \quad \mathbf{f_2} = \frac{y_1 - c_y}{l} + \frac{1}{2}, \quad \mathbf{f_3} = \frac{z_1 - c_z}{l} + \frac{1}{2} \quad (1)$$

c_x, c_y, and c_z are the coordinates with x_1, y_1 and z_1 of the centre of the box bounding S.

Furthermore, because HIF3D features can only represent spatial information (hand shape change), we need to capture temporal information as well. The only way to identify some movements, such as reversing gestures with the same spatial properties, is through temporal sequencing. So we extract HIF3D features using the Temporal Pyramid (TP). This technique extracts features from overlapping sub-sequences arising from a temporal split of a global gesture sequence.

The benefit of Deep Learning



We are living in the "Big Data Era" of technology, which means that we have a lot of data that cannot be processed using typical machine learning algorithms, but that can be processed using deep learning. It goes without saying that as the amount of data increases, deep learning will almost surely improve the performance of a model. As a result, Deep Learning is rising in popularity as a result of its superiority in terms of consistency when trained with a large volume of data.

Deep machine - learning technique wherein we provide an input X to it and then use it to estimate an output Y, such as when given several labelled photographs of dogs and cats and then using deep learning to formulate hypotheses for the new unbranded image of a cat or dog (for example). The question then becomes, how does deep learning accomplish this? So, fundamentally, a deep learning algorithms is comprised of a neuron that takes a labelled dataset and attempts to discover patterns between the input and its labels in the same way that the human brain does, and then makes predictions based on this experience.

It is composed of three layers: the input layer, the hidden layers, and the output layer, which is composed of a collections of neuron (or nodes). When working with images, the input layers accept an array of data (pixel values in this case), and predictions are made at the output nodes, while convolution layers are related with the majority of the calculation. When there are a significant number of neuron layers in a network, it is referred to as a deep neural network, and only deep learning is possible at that point. As a result, deep neural networks enhance their ability to mimic new complex functions as more neural layers are added.

There are several different types of neural networks.

There are several distinct types of neural networks that can be used for a variety of tasks, including Dense Deep Neural network, Convolutional Neural Network, Recurrent Neural Network, and others. Besides these layers, there are additional layers such as Pooling Layer, Dropout Layer, Activation Function, and so on, all of which work together to make the results of the model as efficient as possible.

Convolutional Neural Networks (CNN)

Computer vision, such as picture categorization, face recognition, and object detection, among other things, is where CNN is most commonly utilised. To classify images, CNNs are used, which are preceded by neural layer and activation functions, to take an input image and consider these as an array set of pixels, processing it by looking for patterns in the image, and then classify the image into the categories or classes specified in the statement of the problem. The image(the input layer) will accept perhaps an arrays of RGB matrix (3 channels) or a grayscale matrix as its input data (1 channel).

CNN's Architectural Design

CNN architecture would be built of convolutional layers (filters), pooling layer, fully connected layer (Dense), and finally output layers in deep learning.

Layer of Convolutional Neural Networks

In this case, the filter would be used to study features from an image through convolving it with the image, which is a mathematical function, and now a new matrix will be formed, which will contain the learned features such as edge detections, sharpening or blurring the image, among other things.

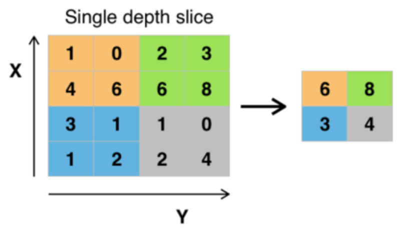
Matrix representation in a CNN

Layers of Pooling

Using this layer, you may do a down - sampling technique on the images that are fed into it, resulting in a compression of the input images, which means they go from a much more implemented in various to a less precise image that contains more information. It can take on a variety of forms:

1. Max Pooling (also known as maximum pooling)
2. Pooling on an Average Basis
3. Sum Pooling (also known as sum pooling)

A layer of pooling is used in CNN's architecture.



When using Max Pooling, the largest component from the corrected feature map is used, and when using Sum Pooling, it is used to take the sum among all components with in feature map.

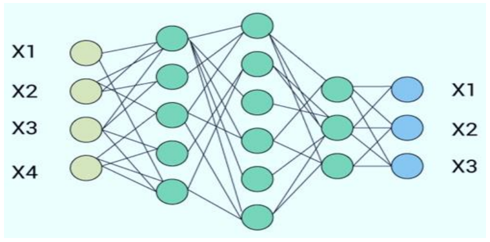
Layer that is completely connected

The matrix would be flattened at this layer, immediately following the CNN network and pooling, before being fed to the Fully Connected Layer, just like a neural networks. Next, at the last layer, an activation function such as softmax or sigmoid will be applied in classifying the output, with the number of neurons inside the final layer equal to the number of classifications in the output.

An input-output set of data layer thread

Activation of the Functionality

It is a nonlinear layer that is placed at the end of and in between



neural networks, and it is this layer that determines the output

```
import math
import numpy as np
import h5py
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.python.framework import ops
import keras
from keras import backend as K
```

of each node. The following activation functions are available: Sigmoid, ReLU, Leaky ReLU, Softmax, and so on.

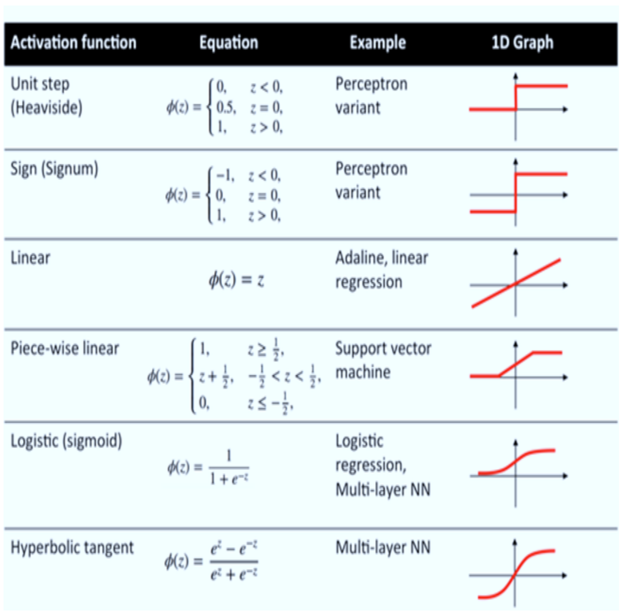
Activation Functions

Activation Functions List

Classification of Hand Gestures Using Python Programming

```
train_dataset = h5py.File('train_signs.h5', "r")
test_dataset =h5py.File('test_signs.h5',"r")
```

A dataset that contains photographs of various hand gestures,



Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer NN	

```
y_train=keras.utils.to_categorical(y_train,classes.shape[1])
y_test=keras.utils.to_categorical(y_test,classes.shape[1])

print ("number of training examples = " + str(x_train.shape[0]))
print ("number of test examples = " + str(x_test.shape[0]))
print ("X_train shape: " + str(x_train.shape))
print ("Y_train shape: " + str(y_train.shape))
print ("X_test shape: " + str(x_test.shape))
print ("Y_test shape: " + str(y_test.shape))
```

or by a fist, palm, displaying the thumb, and others, was utilised as in Hand Gesture Classification task. These images can be further used to indicate counts ranging from 1 to 5 using these hand motions can be found here.

There are numerous techniques in learning algorithms for classification, so we'll be applying Deep learning only with aid of Convolution Neural Networks (CNN) is stated above, as well as Keras to do this (an open source neural networks library wrote in a Python). Keras is simple and fast, and it also has support for CNN, and it performs flawlessly on both the CPU and the GPU at the same time.

Technologies that were used:

```
x_train=np.array(train_dataset['train_set_x'][:])
y_train=np.array(train_dataset['train_set_y'][:])
x_test=np.array(test_dataset['test_set_x'][:])
xxx_test=np.array(test_dataset['test_set_x'][:])
y_test=np.array(test_dataset['test_set_y'][:])
```

- Python3
- Tools and libraries that were used:

a) Google Collaborator.

b) Keras

c) Tensor-Flow

Import all of the necessary libraries.

Dataset loading

It is necessary to note that we have two sets of data – one for training and another for testing. Using a training data of 1080 photos and a test dataset of 120 images, with six labels to classify, ranging from 0 to 5, the training and test datasets are used to train the model. Each image will have a resolution of 64x64 pixels.

The h5py format has been used to store the dataset in question. An interface to the HDF5 binary datatype written in Python is provided by the h5py package. HDF5 allows you to store thousands of datasets in a single file with the help of a single file. Due to the fact that they allow standard mode like r/w/a, it is necessary to load our dataset, which comprises images in the r mode.

Following the loading of the dataset, the following stage will be to convert the image pixel values into a collection of arrays and labels, with each array representing a different training and test dataset.

Normalization of a dataset is a process in which In order to make the dataset more manageable, we'll divide the number of pixels by 255 and then normalise the dataset by multiplying the normalised values by 255.

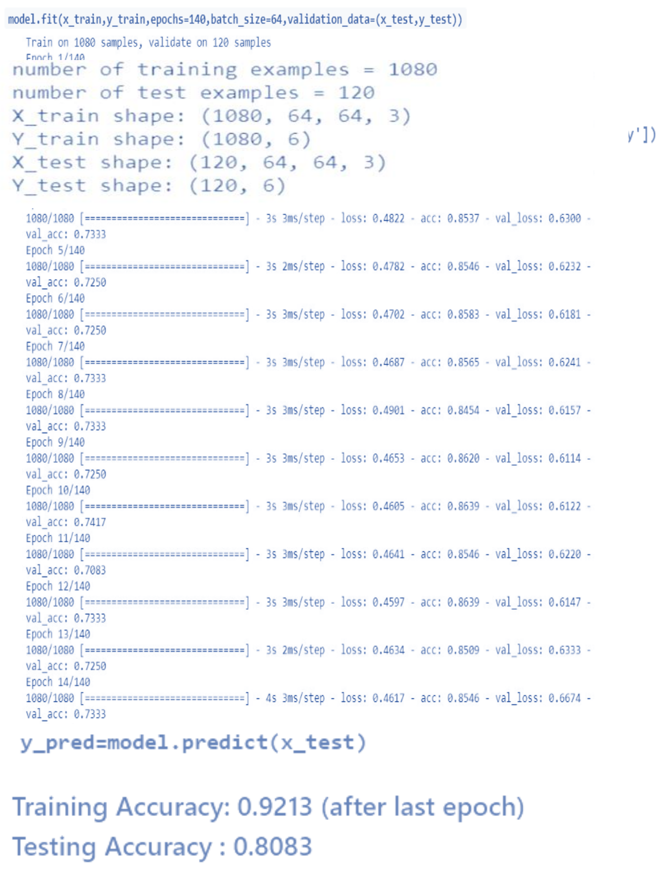
```
x_train=x_train/255
x_test=x_test/255
```

The process of converting basic values to classifier metric for use in both the training and test sets

Using Keras, construct a CNN architecture.

In deep learning, a sequentially model is a linear stacking of layers that is used to generate deep learning models such as the instance of the a Sequential model, which is created first and then layers are added to that too piecewise.

When you use the conv2d filter, you're simply creating a kernel that's been convolved only with input layer (at its original level, which is the picture) and then producing a new array (tensor). When building this layer, we must include the following arguments: filtering (number of outputs filter in the convolutions), kernel size (dimensions of a kernel), strides (number of steps in the convolution).



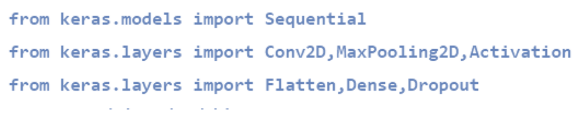
```
model.fit(x_train,y_train,epochs=140,batch_size=64,validation_data=(x_test,y_test))
  Train on 1080 samples, validate on 120 samples
  Epoch 1/140
number of training examples = 1080
number of test examples = 120
X_train shape: (1080, 64, 64, 3)
Y_train shape: (1080, 6)                                                   y'])
X_test shape: (120, 64, 64, 3)
Y_test shape: (120, 6)
  1080/1080 [==============================] - 3s 3ms/step - loss: 0.4822 - acc: 0.8537 - val_loss: 0.6300 -
  val_acc: 0.7333
  Epoch 5/140
  1080/1080 [==============================] - 3s 2ms/step - loss: 0.4782 - acc: 0.8546 - val_loss: 0.6232 -
  val_acc: 0.7250
  Epoch 6/140
  1080/1080 [==============================] - 3s 3ms/step - loss: 0.4702 - acc: 0.8583 - val_loss: 0.6181 -
  val_acc: 0.7250
  Epoch 7/140
  1080/1080 [==============================] - 3s 3ms/step - loss: 0.4687 - acc: 0.8565 - val_loss: 0.6241 -
  val_acc: 0.7333
  Epoch 8/140
  1080/1080 [==============================] - 3s 3ms/step - loss: 0.4901 - acc: 0.8454 - val_loss: 0.6157 -
  val_acc: 0.7333
  Epoch 9/140
  1080/1080 [==============================] - 3s 3ms/step - loss: 0.4653 - acc: 0.8620 - val_loss: 0.6114 -
  val_acc: 0.7250
  Epoch 10/140
  1080/1080 [==============================] - 3s 3ms/step - loss: 0.4605 - acc: 0.8639 - val_loss: 0.6122 -
  val_acc: 0.7417
  Epoch 11/140
  1080/1080 [==============================] - 3s 3ms/step - loss: 0.4641 - acc: 0.8546 - val_loss: 0.6220 -
  val_acc: 0.7083
  Epoch 12/140
  1080/1080 [==============================] - 3s 3ms/step - loss: 0.4597 - acc: 0.8639 - val_loss: 0.6147 -
  val_acc: 0.7333
  Epoch 13/140
  1080/1080 [==============================] - 3s 2ms/step - loss: 0.4634 - acc: 0.8509 - val_loss: 0.6333 -
  val_acc: 0.7250
  Epoch 14/140
  1080/1080 [==============================] - 4s 3ms/step - loss: 0.4617 - acc: 0.8546 - val_loss: 0.6674 -
  val_acc: 0.7333
 y_pred=model.predict(x_test)

Training Accuracy: 0.9213 (after last epoch)
Testing Accuracy : 0.8083
```

In addition to accepting some arguments such as pool size (the maximum number of pooling window), strides, paddings, and so on, MaxPooling2D performs a down - sampling procedure just on input of this layer, which means this will condense the input image with much more specifics to under detailed images like some features (such as boundaries) have already been identified. Conv2D also acknowledges some arguments such as pool size (the maximum number of pooling window), strides, paddings, and so on.

After the sequence of convolutional-pooling layers, a fully connected convolution layer will be applied, which resulting in an N-dimensional vectors, where N represents the size of classes wherein the model chooses the appropriate class.

It applies the Dropout function on the incoming data. In order to avoid the model from overfitting, this strategy involves setting the weights from some redundant neuron in a given layer to zero.

Import all of the module from Keras in order to build a



```
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Activation
from keras.layers import Flatten,Dense,Dropout
```

CNN infrastructure.

This function builds a model & add layers to it.

The code shows that we have generated a neural network with two convolution layers followed by a max pool layer to activation function "relu" in both layers, where first layer has eight filters of size (4,4) and the maximum pool size (8,8), and the second layer has sixteen filters of size (2,2) and the maximum pool is (44,4), and padding will be the same in each layer. In our output, we have six classes to forecast, so we will flatten the outputs of the convolutions and then add another fully - connected (Dense) with six nodes in it. This is because we have six classes to predict in our output.

This function prepares the models for training.

Models are being trained.

As previously said, we will train our models by feeding it the labelled dataset (as previously loaded), i.e. the training dataset will act as the training data and the testing dataset will act as the validating dataset. Using Keras, it will be simple to train our model using the fit technique, and throughout the training process, it will go off 140 iteration in order to update the weight values of a neural networks with the help of optimizers**.**

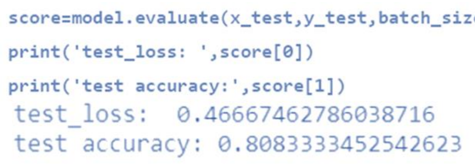
Modeling and testing are being carried out.

As soon as we have finished training the model, we will test it on a new dataset (test dataset). We will use an evaluation method that returns loss and accuracy values for the models in test mode; its input will be test images as well as target variables (i.e. the test images and the test variables), and its output will be the loss and accuracy values for the models in test mode;

Then, using the predict method, we will estimate the outputs for the input samples, which are the test datasets.

Conclusion

The Hand Gesture dataset was identified using Convolution Neural Networks with the assistance of Keras, a



```
score=model.evaluate(x_test,y_test,batch_size=64)
print('test_loss: ',score[0])
print('test accuracy:',score[1])
 test_loss:  0.46667462786038716
 test accuracy: 0.8083333452542623
```

Python-based neural-network library that is open-sourced and developed in Python. Keras is preferential because it rises the computation of deep neural networks architecture by providing a few functions that are required for optimization, such as backpropagation, gradient descent, and so on, and it reduces the computation time. Deep neural network neural network architecture is preferred because it is more efficient. In addition to pre-trained models and weights, Keras includes applications that give some pre-trained models and weights that can be used directly for prediction, extraction of features, and fine-tuning. Do take a look at Analytics Steps for additional updates on the subject.

Reference

[1] Said Yacine Boulahia, Eric Anquetil, Franck Multon, Richard Kulpa. Dynamic hand gesture recognition based on 3D pattern assembled trajectories. IPTA 2017 - 7th IEEE International Conference on Image Processing Theory, Tools and Applications, Nov 2017, Montreal, Canada. pp.1-6. ffhal01666377f

[2] Yogeesh N. "Zer0_The Amazing Contribution of Indians." TUMBE Group of International Journals, vol. 1, no. 1, 2018, pp. 1-5.

[3] Alexey Kurakin, Zhengyou Zhang, and Zicheng Liu. A real time system for dynamic hand gesture recognition with a depth sensor. In Signal Processing Conference (EUSIPCO), 2012 Proceedings of the 20th European, pages 1975–1979. IEEE, 2012.

[4] Yogeesh N. "Graphical Representation of Solutions to Initial and Boundary Value Problems of Second Order Linear Differential Equation Using FOOS (Free & Open Source Software)-Maxima." International Research Journal of Management Science and Technology (IRJMST), vol. 5, no. 7, 2014, pp. 168-176, www.irjmst.com/abstractview/7349.

[5] Giulio Marin, Fabio Dominio, and Pietro Zanuttigh. Hand gesture recognition with jointly calibrated leap motion and depth sensor. Multimedia Tools Appl., 75 (22):14991–15015, 2016.

[6] Yogeesh N. "Mathematical Approach to Representation of Locations Using K-Means Clustering Algorithm." International Journal of Mathematics And its Applications (IJMAA), vol. 9, no. 1, 2021, pp. 127-136.

[7] Camille Monnier, Stan German, and Andrey Ost. A multi-scale boosted detector for efficient and robust gesture recognition. In ECCV Workshops (1), pages 491– 502, 2014.

[8] Chih-Chung Chang and Chih-Jen Lin. Libsvm: a library for support vector machines. ACM transactions on intelligent systems and technology (TIST), 2(3):27, 2011

[9] Adrien Delaye and Eric Anquetil. Hbf49 feature set: A first unified baseline for online symbol recognition. Pattern Recognition, 46(1):117–130, 2013.

[10] R. N. Khushaba, “Correlation analysis of electromyogram signals for multiuser myoelectric interfaces,” IEEE Transactions on Neural Systems and Rehabilitation Engineering, vol. 22, no. 4, pp. 745–755, 2014.

[11] R. Chattopadhyay, N. C. Krishnan, and S. Panchanathan, “Topology preserving domain adaptation for addressing subject based variability in semg signal.” in AAAI Spring Symposium: Computational Physiology, 2011, pp. 4–9.

[12] T. Tommasi et al., “Improving control of dexterous hand prostheses using adaptive learning,” IEEE Transactions on Robotics, vol. 29, no. 1, pp. 207–219, 2013.

[13] Yogeesh N. "Study on Clustering Method Based on K-Means Algorithm." Journal of Advances and Scholarly Researches in Allied Education (JASRAE), vol. 17, no. 1, 2020, pp. 485-489(5), www.ignited.in//I/a/305304.