UNIVERSITY OF THESSALY
SCHOOL OF SCIENCE
COMPUTER SCIENCE AND BIOMEDICAL INFORMATICS

SIGN LANGUAGE RECOGNITION USING NEURAL NETWORKS

TOUTOU EVANGELIA

THESIS

Supervisor
Delibasis Konstantinos
Associate Professor
Co-Supervisor
Sotirios Tasoulis
Assistant Professor

Lamia, 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Εφαρμογές Νευρωνικών Δικτύων στην Αναγνώριση Νοηματικής Γλώσσας

Τουτού Ευαγγελία

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Επιβλέπων
Δελήμπασης Κωνσταντίνος
Αναπληρωτής Καθηγητής
Συνεπιβλέπων
Τασουλής Σωτήριος
Επίκουρος Καθηγητής

Λαμία, 2021

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί* ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. *Δέχομαι ότι η αυτολεξεί* ***παράθεση χωρίς εισαγωγικά****, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια*

4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: 31/05/2021

Η Δηλ.

ΤΟΥΤΟΥ ΕΥΑΓΓΕΛΙΑ

(Υπογραφή)

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

Εφαρμογές Νευρωνικών Δικτύων στην Αναγνώριση Νοηματικής Γλώσσας

Τουτού Ευαγγελία

Τριμελής Επιτροπή:

Δελήμπασης Κωνσταντίνος, Αναπληρωτής Καθηγητής (επιβλέπων)

Τασουλής Σωτήριος, Επίκουρος Καθηγητής (συνεπιβλέπων)

Πλαγιανάκος Βασίλειος, Καθηγητής

TABLE OF CONTENTS

LIST OF FIGURES

ΠΕΡΙΛΗΨΗ

Η επικοινωνία είναι μια σημαντική πτυχή της καθημερινής ζωής. Από τα παλιά χρόνια, τα ανθρώπινα όντα προσπαθούν να βρουν τρόπους να επικοινωνήσουν μεταξύ τους γεγονός που οδήγησε στη δημιουργία των γλωσσών. Οι περισσότερες ανεπτυγμένες γλώσσες εξαρτώνται από τη λεκτική επικοινωνία, καθιστώντας τις αδύνατες για χρήση από τους κωφούς και από τα άτομα με προβλήματα στην ομιλία. Για να ξεπεραστεί δυσκολία αυτή, δημιουργήθηκαν νοηματικές γλώσσες επιτρέποντας στους κωφούς/μουγκούς να επικοινωνούν μεταξύ τους και να συνδεθούν με την κοινωνία. Δεδομένου ότι οι κωφοί και οι μουγγοί άνθρωποι αποτελούν μειονότητα, η διδασκαλία των νοηματικών γλωσσών δεν είναι απαραίτητη , δημιουργώντας έτσι προβλήματα και δυσκολίες στην επικοινωνία μεταξύ των κωφών/μουγκών και των ατόμων που δε γνωρίζουν κάποια νοηματική γλώσσα. Το πρόβλημα αυτό έχει απασχολήσει αρκετούς ερευνητές, οι οποίοι προσπαθούν να το ξεπεράσουν δημιουργώντας συστήματα αναγνώρισης νοηματικής γλώσσας.

Η παρούσα εργασία επικεντρώνεται στην αλφάβητο της Αμερικανικής Νοηματικής Γλώσσας και προτείνει ένα νέο μοντέλο αναγνώρισης νοηματικής γλώσσας χρησιμοποιώντας εφαρμογές της MediaPipe και Deep Learning (Βαθιά Μάθηση). Βίντεο που αναπαριστούν τα γράμματα της Αμερικανικής αλφαβήτου στη νοηματική γλώσσα επιλέχθηκαν, τα οποία στη συνέχεια μετατράπηκαν σε ακολουθίες εικόνων. Οι εικόνες που προέκυψαν τροφοδοτήθηκαν στην εφαρμογή της MediaPipe για αναγνώριση των σημείων χεριών (Hands solution) , η οποία επιστρέφει 21 τρισδιάστατα σημεία, το καθένα από τα οποία αντιπροσωπεύει μια άρθρωση στην παλάμη, συμπεριλαμβανομένης της άρθρωσης του καρπού. Επιπλέον, παρουσιάζονται δύο νέα χαρακτηριστικά με βάση τις συντεταγμένες των σημείων. Για κάθε χαρακτηριστικό δημιουργήθηκε μια βάση δεδομένων (dataset), χρησιμοποιώντας δημόσια διαθέσιμα βίντεο από το Youtube . Για την ταξινόμηση των χαρακτηριστικών σε γράμματα της Αμερικάνικης Νοηματικής αλφαβήτου χρησιμοποιήθηκαν δύο διαφορετικές τοπολογίες ενός πολυεπίπεδου perceptron (MLP). Η ακρίβεια στην αναγνώριση των γραμμάτων αλφαβήτου της Αμερικανικής Νοηματικής Γλώσσας, ήταν 99% και 94% για κάθε χαρακτηριστικό , αντίστοιχα.

Λέξεις - Κλειδιά: Αμερικανική Νοηματική Γλώσσα, αλφάβητος, βαθιά μάθηση, MediaPipe, σημεία χεριού, Νευρωνικά Δίκτυα

SUMMARY

Communication is an important aspect of everyday life. Since the beginning of time, human beings are trying to find ways to communicate with each other. Most developed languages depend on verbal communication, making them impossible for deaf and mute people to use. To overcome this difficulty, sign languages were created allowing those people to convey messages and to connect with society. Since deaf and mute people are a minority, sign languages are not commonly learned, thus creating miscommunication and difficulties in the society between deaf and most of people who cannot use the sign language (non-signers). This problem has been focused by many researchers, who try to overcome it by creating computer-based sign language recognition systems.

This thesis focuses on American Sign Language alphabet and proposes a new model for sign language recognition using Mediapipe's solutions and deep learning. Several videos were selected and turned into image sequences. The resulting images were fed into Mediapipe's Hands solution python API, which returns 21 3D landmarks, each representing a joint in the palm including the wrist joint. This thesis presents two new features based on the coordinates of the landmarks. A dataset was created for each proposed feature definition, using videos publicly available from Youtube to extract the features. Two different topologies of a multi-layer perceptron (MLP) were used to classify the features into one of the ALS alphabet. The achieved accuracy in recognizing American Sign Language alphabet letters, was 99% and 94% respectively.

Keywords: American Sign Language, alphabet, deep learning, MediaPipe, hand landmarks, neural network

I. INTRODUCTION

As a social being, communication and understanding are essential to human socialization. In deaf communities, the form of communication is non-verbal as the deaf communicates through Sign Language. Sign Language is a visual language that use the hands, face, head, and upper torso. It is visually processed and has been developed as a language to meet the communication need of the Deaf. The awareness arises when a deaf person is trying to communicate with someone with normal hearing but does not understand sign language. In these cases, the sign language interpreter would come in handy. Except for a person, a computer can also be an interpreter, between a deaf and a non-signer person, using computer vision and deep learning.

a) Artificial Intelligence

Deep Learning is a subset of Machine Learning, which by itself is a subset of Artificial Intelligence as it is shown in the figure 1. Artificial Intelligence (AI) is a general term that refers to algorithm-based intelligence demonstrated by machines, unlike the natural intelligence exhibited by humans and animals. Leading AI textbooks define the field of AI as the study of "intelligent agents": any device that perceives its environment and takes actions that maximize its chance of successfully achieving its goals. In other words, the term artificial intelligence is often used to describe machines or computers that mimic cognitive functions that humans associate with the human mind, such as "learning" and "problem-solving". In recent years, the requirements of computer applications have become increasingly complex, which led to a number of emerging trends in AI research and their adoption. Some of the applications of AI are:

- Robotics
- Problem solving
- Neural Network design
- Intelligent agent systems
- Computer Vision
- Big Data

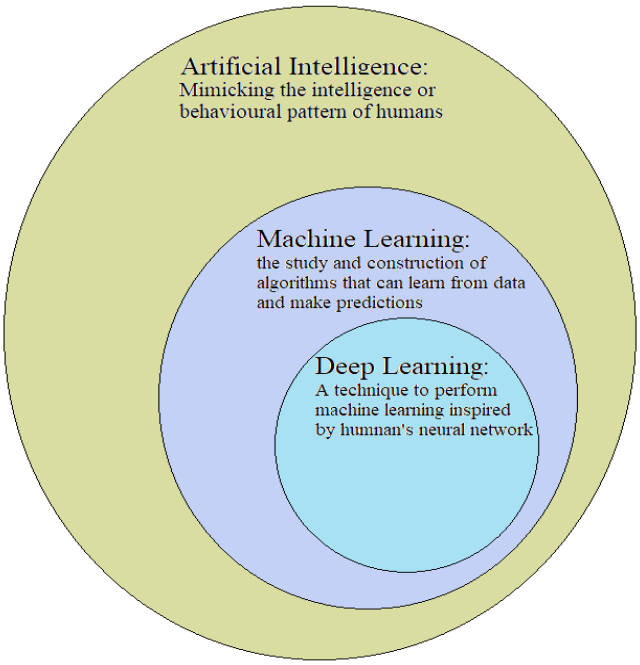


Figure 1. The difference between AI, Machine Learning and Deep Learning

Machine learning (ML) is a subfield of computer science, developed from the study of pattern recognition and computational learning theory in AI. ML explores the study and construction of algorithms that can learn from data and generate predictions from them. Such algorithms work by constructing models from experimental data to make predictions based on data or to make decisions that are expressed as the result. ML enables algorithms to learn and improve from experience without being explicitly programmed. Therefore, it is the tool to make AI applications. Machine Learning algorithms can be classified as supervised and unsupervised.

Supervised ML algorithms are trained on labeled exemplar data and can apply what has been learned in the past to label or predict new. A supervised ML algorithm compares its output to the correct output and uses the error to modify its model accordingly. In this way it produces an inferred function to make predictions based on the input data. The model can predict labels or decide actions for any new, unseen input after sufficient training.

In contrast, unsupervised ML algorithms are used when no label information is available for training in the input data. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system can't figure out the right output, but it can draw inferences from datasets to describe hidden structures from unlabeled data, by exploring them.

Semi-supervised machine learning algorithms fall somewhere in between supervised and unsupervised learning since they use both labeled and unlabeled data for training. The

systems that use this method can improve learning accuracy considerably. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources to learn from it.

b) Computer Vision

Computer Vision (CV) is the application of Artificial Intelligence and ML to digital images and videos, acquired by cameras, or other imaging systems. CV and recently deep-learning models can be used to train computers to interpret and understand the visual world, i.e., accurately detect, identify/classify, segment, or track objects in images and videos in the same way that humans do. In order to perform these tasks, CV algorithms are trained by processing labelled data multiple times until they discern distinctions and ultimately perform the required image-related task. Deep learning is an essential technology to accomplish this.

c) Deep Learning

Deep Learning (DL) is an application of Machine Learning inspired by the structure of the human brain. Deep learning algorithms attempt to draw similar conclusions and enable computers to perform tasks that come naturally to humans: learn by example. To achieve this, DL uses artificial neural networks (ANN), a multi-layered structure of algorithms. An artificial neural network is not pre-trained and, until configured and trained by the user, has no practical function on its own. Like biological neural networks, artificial neural networks rely on neurons to function. An artificial neuron is an algorithmic construct that supports its basic principles in the fundamental functions of the biological neuron. It is the fundamental processing unit in neural networks. Like biological neurons, artificial neurons receive signals from other neurons and transmit signals to subsequent neurons with which they are connected. A neural network has in general three kinds of layers: The input layer, the hidden layers, and the output layer. The input layer contains a number of neurons equal to the dimensionality of the feature vector, which receive the input data that transmits them to the hidden neurons of the next layer. The hidden layers add non-linearity to the behavior of an ANN. A neural network can have multiple hidden layers, which directly affects its effectiveness and precision. The output layer consists of output neurons, which generate the ANN's output (regression, or classification result).

An artificial neuron or else a perceptron is a linear classifier the most basic of all neural networks, being a fundamental building block of more complex neural networks. It simply connects an input cell and an output cell, as it is shown in Figure 3. The response of a perceptron to one of its inputs is declared as:

$$f(x) = \begin{cases} 1 & if\ w \cdot x + b > 0 \\ 0 & else \end{cases}$$

There are three basic rules for training a perceptron:

1. If the output for an input X is 1 and should be 1, its synaptic weight remains the same
2. If the output is 0 while it should be 1, the synaptic weight increases.
3. Otherwise, if it is 1 while it should be 0, it decreases.

A modified simple model of perceptron is shown in Figure 2. Every connection has a weight attached with a positive or negative value. Positive weights activate the neuron, while negative weights inhibit it. Weights are equivalent to biological synaptic weights,

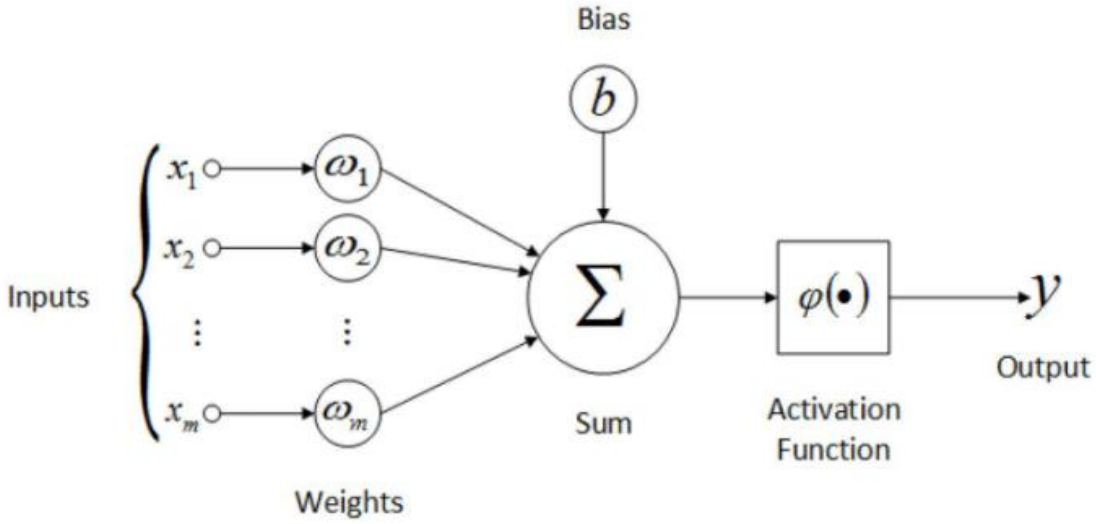


Figure 2. Architecture of a simple neuron

where the influence of each signal depends on how good the connection is between the neuron-sender and the neuron-receiver of the signal. These weights are not fixed, their values are being adapted during training, to minimize output error and equivalently get the desired result. Figure 2 shows the detailed model of a typical artificial neuron with inputs (x_1, x_2, ..., x_m) being connected to neuron with weights on each connection. The neuron sums all the signals it receives, with each signal being multiplied by its associated weights on the connection. This output is then passed through a transfer or activation function that is normally non-linear to give the final output, *y*. A transfer function can be any kind of

function, depending on the topology and the purposes of the network, and implements the calculation of the proper response according to the inputs. The most commonly used function is the sigmoid (logistic function) because of its easily differentiable properties. For the training process in a neural network, it is necessary to split the dataset into three parts:

- **The training dataset**, along with the correct output is presented to the ANN during training to adjust the weights between the nodes/neurons and the biases.
- **The validation dataset**, contains unseen data, and it is used for fine-tuning the network's performance and determine the termination of training.
- **Test dataset**, is used to calculate the accuracy and the loss of the trained ANN.

Neural Network Architectures

Although there are numerous neural network architectures, bellow are presented some of the most popular ones, split into three general categories: the standard networks, the recurrent networks, and the convolutional networks.

The Standard Networks

Feed-Forward Networks: The feed-forward networks are a collection of perceptrons and are the commonest type of neural network in practical applications. In this type of network, there are three fundamental types of layers: input layers, hidden layers, and output layers. If there is more than one hidden layer, they are called “deep” neural networks. During each connection, the signal from the previous layer is multiplied by a weight, added to a bias, and passed through an activation function. They compute a series of transformations that change the similarities between cases. Feed-forward networks use backpropagation to iteratively update the parameters until it achieves a desirable performance.

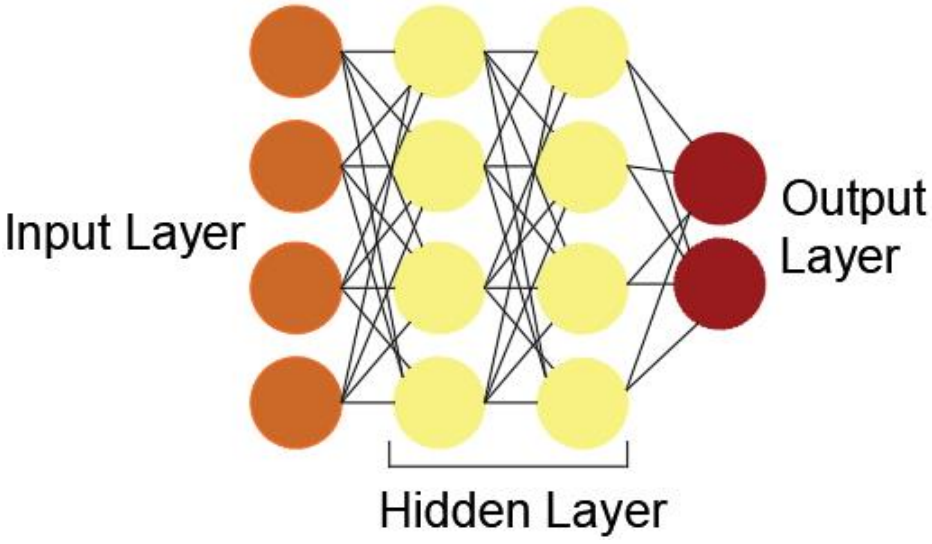


Figure 3. A Feed-Forward Network

Recurrent Networks

- The Recurrent Neural Network: A recurrent neural network is a specialized type of network that has directed cycles in its connection graph which means it can sometimes get back to where it started by following the arrows, and that is why it is called recurrent. Allowing for information to be stored in the network, RNNs are using reasoning from previous training to make better and more informed decisions about upcoming events using the previous predictions as 'context signals'. They are more biologically realistic and are equivalent to very deep nets with one hidden layer per time slice, except that they use the same weights at every time slice and they get input at every time slice. They can remember information in their hidden state for a long time but is hard to train them to use this potential. Because of their nature, RNNs are commonly used to handle sequential tasks and can handle inputs of any size.

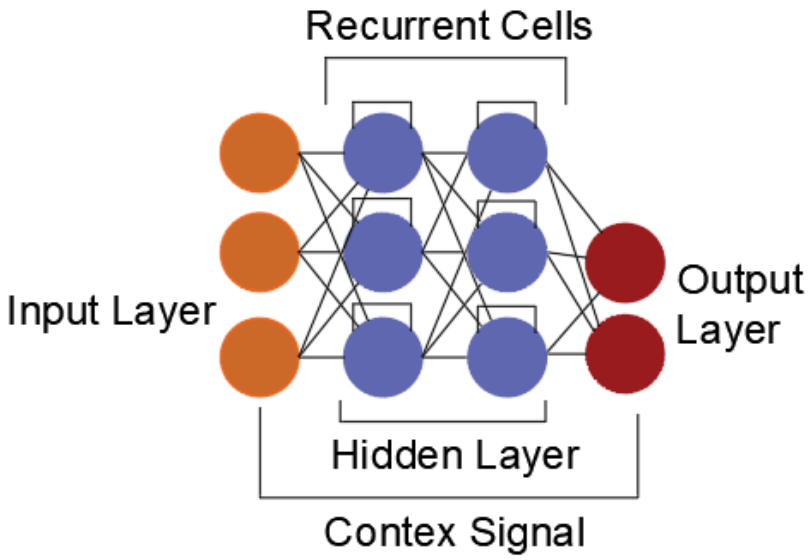


Figure 4. A Recurrent Neural Network

- The Long Short Term Memory Network (LSTM): Even with good initial weights, RNNs have difficulty dealing with long-range dependencies. The influence (backpropagated error) of a given input on an input of the hidden layer, either blows up exponentially, or decays to zero as it is cycled around the network's connections. The solution to this problem is a Long Short-Term Memory Network or an LSTM. This RNN architecture is specifically designed to address the *vanishing gradient problem*, providing the structure with memory blocks. These blocks are using logistic and linear units with multiplicative interactions and having three gates: input, output and forget, which are equivalent to write, read, and reset respectively. Information enters the cell whenever its "write" gate is on and can be read from the cell by turning on its "read" gate.

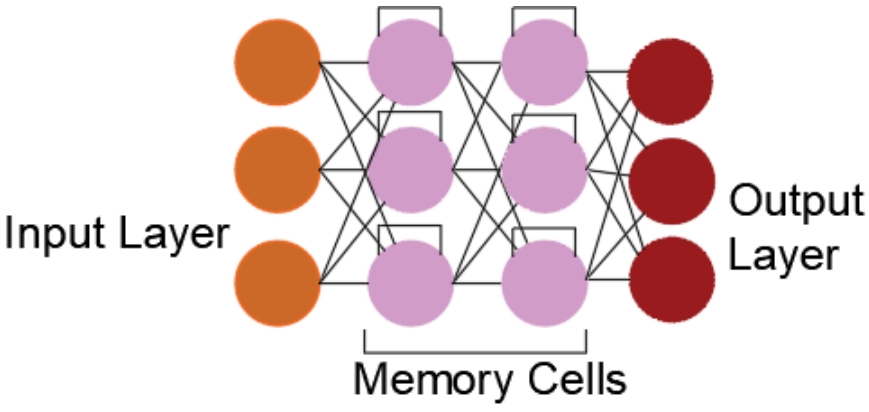


Figure 5. The LSTM Network

Convolutional Neural Networks

Convolutional Neural Networks (CNNs) is a class of Neural Networks that specialize in processing data that has a grid-like topology, such as an image. A CNN typically consists of three types of layers: a convolutional layer, a pooling layer, and a fully connected layer. The Convolutional layer is the first layer to extract features from an input image and performs a dot product between two matrices (equivalent to the operator of convolution). One of the matrices is the set of learnable parameters, known as a kernel or filter, and the other matrix is a restricted portion of the image called *receptive field*. The kernel is spatially smaller than an image has the same depth as the image i.e., if the image is composed of three channels (RGB) the filter will also have 3 channels.

The Pooling layer is responsible for the parameter's reduction of the input image without losing important information. This can be with different types of pooling: max pooling, average pooling and sum pooling. Max pooling takes the largest element from the rectified feature map. Average pooling involves calculating the average for each patch of the feature map and sum of all elements in the feature map call as sum pooling. The pooling layer is usually followed by the activation layer. Several repetitions of these three layers are built in succession, until the dimensionality of the image has been adequately reduced.

A small number of fully connected layers follow, similar to a feed forward network, whose input is the flattened (1-dimensional) output of the last pooling layer. The final layer of the fully connected layer classifies the image using the softmax activation function. This function is used to get probabilities of the input being in a particular class (classification).

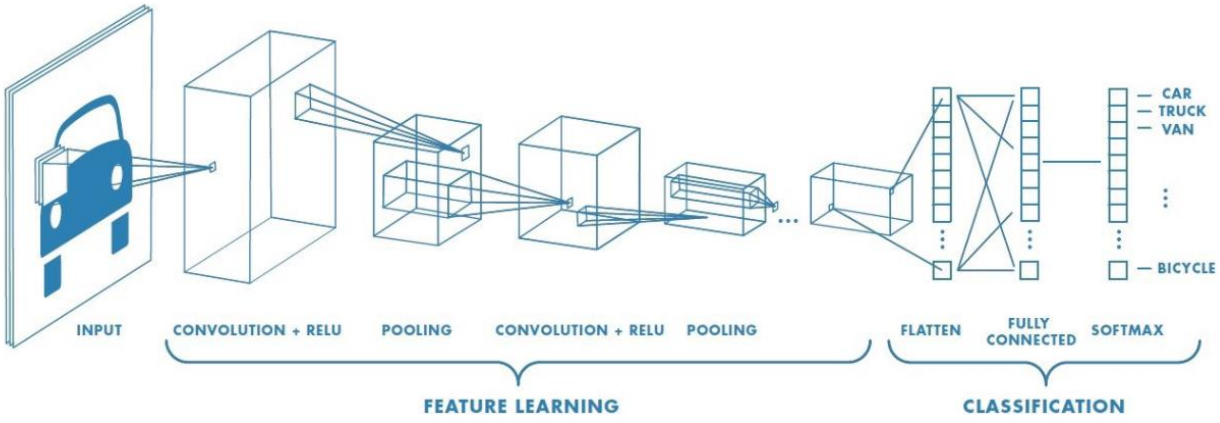


Figure 6. Architecture of a Convolutional Neural Network [16]

d) Tensorflow & Keras

Implementing a successful complex deep neural network model is a challenging task that thanks to the modular structure of the networks and standard inference tools, several software frameworks are available that speed up the design and training of deep neural networks. TensorFlow [1] is the newest addition to this toolbox and is an open-source library developed by Google, primarily for deep learning applications. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks, while it also supports traditional machine learning. It provides several improvements, such as graphical visualization and improved compilation time. Tensorflow is nominally used for the Python programming language, although there is access to the underlying C++ API.

Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow. It was developed with a focus on enabling fast experimentation. More specifically, Keras is the high-level API of the newest version of Tensorflow, Tensorflow 2: an approachable, highly-productive interface for solving machine learning problems, with a focus on modern deep learning. It provides essential building blocks for developing and shipping machine learning solutions with high iteration velocity. The core data structures of Keras are layers, for example, the input layer, the dropout layer, and the dense layer, and models like the Sequential model which is the simplest type of model consisting of a linear stack of layers. In addition to standard neural networks, Keras has support for convolutional and recurrent neural networks. It supports other common utility layers like dropout, batch normalization, and pooling.

This work was implemented using the Python programming language and the construction of neural network was achieved using both Keras and Tensorflow API's.

e) MediaPipe

MediaPipe is a cross-platform framework for building machine learning applications for the analysis of multimodal data, like video, audio, or any time-series data. With MediaPipe, a perception pipeline can be built as a graph of modular components, including inference models, for example, Tensorflow, and media processing functions. At the time of writing MediaPipe offers a number of solutions [2] for real-time object detection, such as Face Mesh, Face Detection, Iris tracking, hand tracking, Pose, Holistic, Hair Segmentation,

Object Detection, Box Tracking, Instant Motion Tracking, and 3D Object Detection (Objectron).

MediaPipe can be easily implemented in a Python environment as it offers ready-to-use yet customizable Python solutions as a prebuilt Python package. In this work, the Hands MediaPipe solution [3], which employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame, is used for American Sign Language Recognition. The figure 8 depicts the landmarks of the hand and their code names and it was taken from [4].

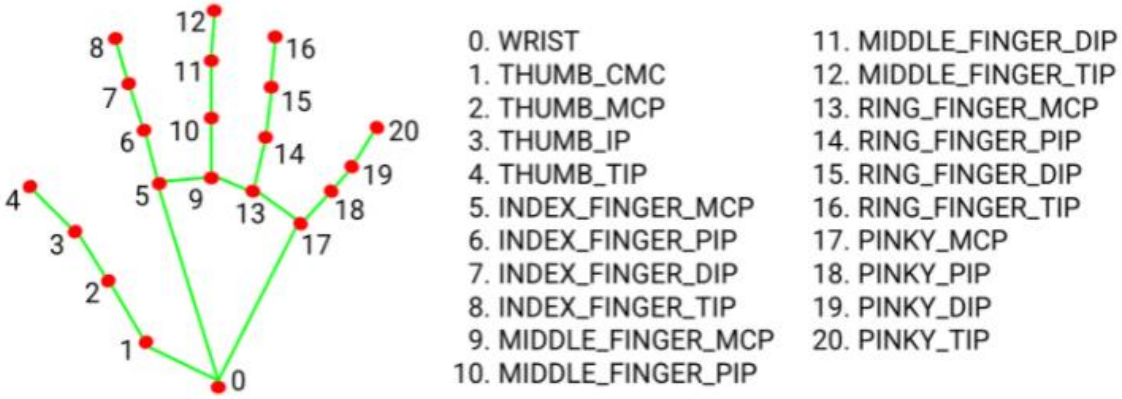


Figure 7. Hand Landmarks [4]

The end goal of this work is to create a vision-based system, able to recognize American Sign Language alphabet letters in video sequences, acquired under normal every-day conditions. To this end we resorted to Mediapipe's Hands solution for the detection of hand landmarks combined with Artificial Neural Networks for the classification of the signed letters. Our priority is to propose a feature invariant in translation, rotation and scale so as to be more useful and easily implemented in daily life applications for sign language recognition (SLR).

II. RELATED WORK

Literature review of the problem shows that there have been several approaches to address the issue of miscommunication between deaf / mute people and non - sign language speakers (non – signers). In [5] the authors used Microsoft Kinect, a low-cost depth camera, to create a new method for ASL (American Sign Language) alphabet recognition. More specifically, they first obtained a segmented hand configuration by using a depth contrast feature based per-pixel classification algorithm. Then they developed a method to localize hand joint positions under kinematic constrains. The results were implemented to a Random Forest classifier, built to recognize ASL signs using the joint angles, and managed to achieve above 90% accuracy in recognizing 24 static ASL alphabet signs.

Sarfaraz Masood, Harish Chandra Thuwal and Adhyan Srivastava in their work [6] created a system based on vision to identify finger – spelled letters of ASL. To achieve this, they used [7], a contribution to a dataset of standard American Sign Language (ASL) hand gestures containing 2425 images from 5 individuals. These images were used as input for the pre-trained VGG16 model, a vision model developed by the Vision Geometry Group from oxford. The accuracy of the model obtained using the Convolution Neural Network was about 96%.

In [8] the authors also used Deep Learning and Computer Vision to create a vision-based application which offers sign language translation to text thus aiding communication between signers and non-signers. The proposed model of this work takes video sequences and extracts temporal and spatial features. More specifically, they proposed to use a Convolutional Neural Network (CNN) model named Inception to extract spatial features from the video stream. Then by using a LSTM (Long Short – Term Memory), a Recurrent Neural Network model, the temporal features are extracted using the outputs from the Softmax and the Pool layer of the CNN, respectively. The dataset used is the American Sign Language Dataset.

Teak-Wei Chong and Boon-Giin Lee in [9] developed a sign language recognition prototype using the Leap Motion Controller (LMC) and aimed for full American Sign Language (ASL) recognition, which consists of 26 letters (both static and dynamic) and 10 digits. The features were extracted from the LMC device, which was connected to a desktop PC and placed on the table to detect and track the hand and finger gestures and then were

implemented in Support Vector Machine (SVM) and Deep Neural Network (DNN) classifiers, who were compared for ASL recognition. The experimental results revealed that the sign language recognition rates for the 26 letters using a support vector machine (SVM) and a deep neural network (DNN) are 80.30% and 93.81%, respectively. As for the recognition rates for a combination of 26 letters and 10 digits they were 72.79% for the SVM and 88.79% for the DNN.

In a similar work, Luis Quesada, Gustavo López and Luis Guerrero [10], developed a system based on hand tracking devices, Leap Motion and Intel RealSense, used for signs recognition. The system uses a Support Vector Machine for sign classification. In this paper three types of evaluations of the system were performed: (1) hand trackers recognition potential using both the Leap Motion and Intel RealSense, (2) SVM classification potential, executed by the researchers using the Leap Motion and the Intel RealSense, and (3) a user assessment was performed by external participants using only the Leap Motion. The results were remarkable with 100% accuracy achieved in recognizing some signs.

In another noticeable work [11] authors showed that a late fusion approach to multimodality in sign language recognition improves the overall ability of the model in comparison to the singular approaches of image classification and Leap Motion data classification. More precisely, the dataset was created with 18 BSL (British Sign Language) gestures collected from multiple subject and two deep neural networks (the Vision model and the Leap Motion model) were benchmarked and compared to derive a best topology for each. The Vision model was implemented by a Convolutional Neural Network and optimized Artificial Neural Network, and the Leap Motion model was implemented by an evolutionary search of Artificial Neural Network topology. After this step, the authors used transfer learning, a machine learning method where a model developed for a task is reused as the starting point for a model on a second task and the weights trained via British Sign language were implemented as the initial weights in a new model for American Sign Language classification, scoring 82,55% accuracy.

Beyond American and British Sign Language, researchers have also developed applications for other recognized sign languages, as the issue of miscommunication between deaf people and non - sign language speakers is global. For example, Douglas F. L. Lima *et al.* [12] proposed a solution for fingerspelling recognition in Brazilian Sign Language (Libras) using Convolutional Neural Networks. This approach does not use gloves, armbands or visual markers in the images, recognizing these Libras gestures, only images, and considering different backgrounds, signers, hand positions and illumination

patterns trying to approach real practical situations for Brazilian deaf users. The system uses a 224.000 images dataset created the authors' team, which represents the letters of the Libras alphabet signed by 12 people in different backgrounds, body arm, hand positions, and lighting patterns. The results showed that this solution had an average accuracy of approximately 99% in a dependent person scenario and had an average accuracy of 71% in an independent person scenario.

A more general contribution to sign language translation using machine learning is [13], where a method of handshapes recognition based on skeletal data is described and a new feature vector is proposed. This work can be applied to any sign language as the dataset can be created by the user. Two datasets were used in this work, one created by the authors containing forty-eight static handshapes, occurring in Polish Finger Alphabet (PFA) and Polish Sign Language (PSL) and the database provided in [14] which contains the recordings of 10 letters from ASL, performed 10 times by 14 people and acquired by jointly calibrated LMC and depth sensor. More specifically, this paper encodes the relative differences between vectors associated with the pointing directions of the fingers and the palm normal. Different classifiers are tested on the demanding dataset, containing 48 handshapes performed 500 times by five users and were considered two different sensor configurations and significant variation in the hand rotation.

An overview of the main research works based on the Sign Language recognition system can be found in [15]. In this paper every method that can be applied in sign language recognition is discussed and the strengths and disadvantages that contribute to the system functioning perfectly or otherwise are highlighted by invoking major problems associated with the developed systems. In addition, a novel method for designing SLR system based on combining EMG sensors with a data glove is proposed. This method is based on electromyography (EMG) signals recorded from hands muscles for allocating word boundaries for streams of words in continuous SLR.

III. METHODOLOGY

a) Overview of the algorithm

An overview of the suggested system's pipeline is depicted in Fig.9. Keeping in mind the end goal, which is to create a system that uses machine learning to recognize American sign language alphabet letters, it is very important to carefully select the data that will be fed into the system. In this case, four videos depicting the American Sign Language alphabet were chosen and modified to be focused on the hand, thus clearer for the recognition and the extraction of the landmarks. The modified videos were converted into image sequences frame by frame and then each frame was imported to the Mediapipe's Hands solution which returned the 21 3D landmarks for each frame.

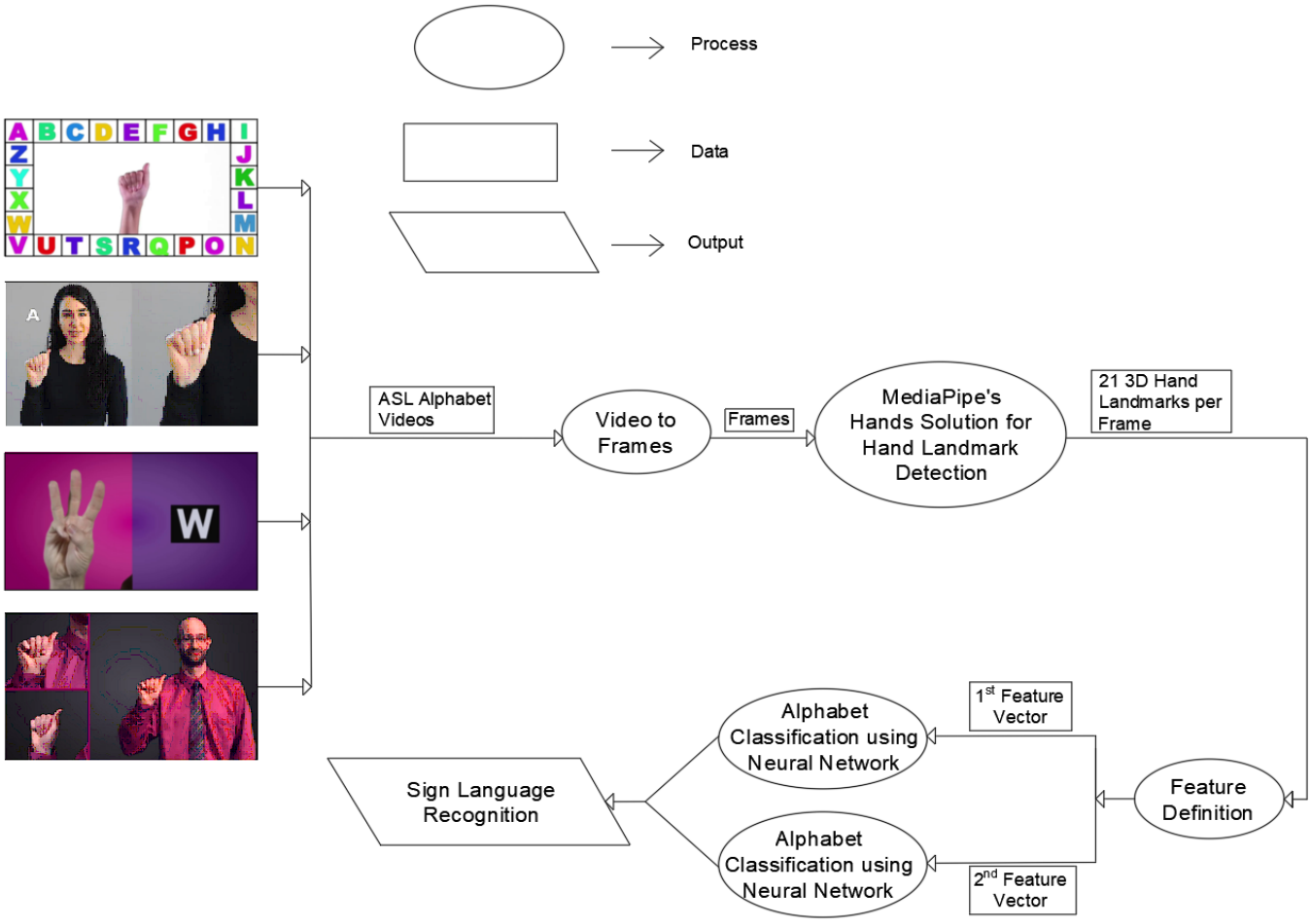


Figure 8. System Pipeline

The letters J and Z of the alphabet were excluded as they are non-static signs and cannot be recognized by a single frame. Some examples of what was returned can be seen in the figures below.

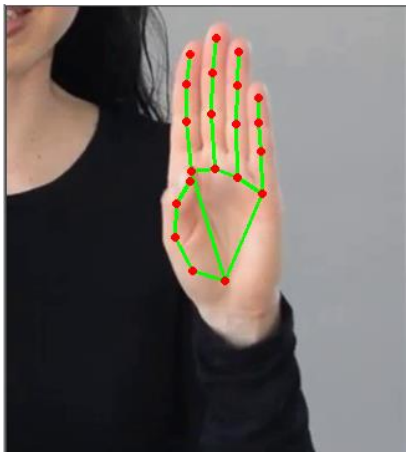


Figure 11. Letter 'B' Landmarks

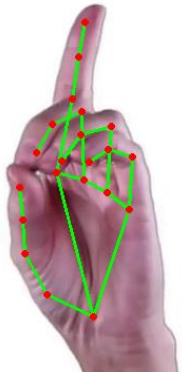


Figure 10. Letter 'D' Landmarks

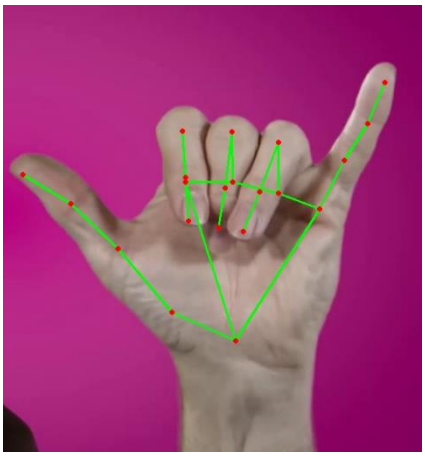


Figure 9. Letter 'Y' Landmarks

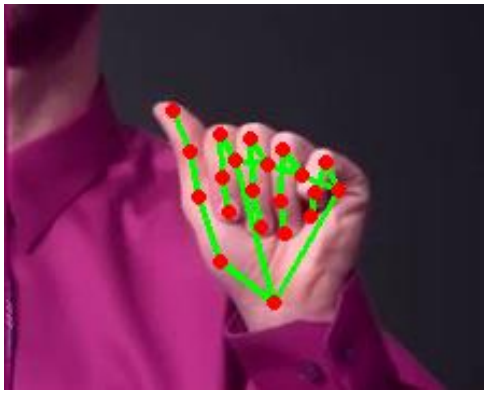


Figure 12. Letter 'A' Landmarks

These landmarks formed the base to feature definition process which was the next step for sign language recognition. This process led to the extraction of the two proposed feature vectors. Each feature vector was implemented into a neural network for alphabet classification, thus for sign language recognition.

b) Hand Landmark detection

In a detailed explanation of how Mediapipe's Hands Solution works can be found. hand tracking solution utilizes an ML pipeline consisting of two models working together: A palm detector that operates on a full input image and locates palms via an oriented hand bounding box and a hand landmark model that operates on the cropped hand bounding box provided by the palm detector and returns high-fidelity 2.5D landmarks. A demonstration of the hand landmark model's architecture can be seen in Fig.14. the model has three outputs sharing a feature extractor. Each head is trained by correspondent datasets marked in the same color.

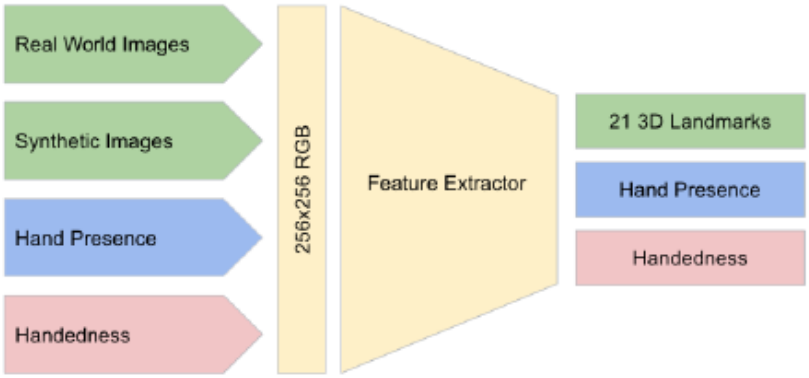


Figure 13. Model's Architecture [2]

More specifically, after running palm detection over the whole image, the subsequent hand landmark model performs precise landmark localization of 21 2.5D coordinates inside the detected hand regions via regression. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions. The model has three outputs Fig.13:

1. 21 hand landmarks consisting of *x*, *y* coordinates and relative depth (defined as the *z* coordinate), all normalized with respect to image dimensions.
2. A hand flag indicating the probability of hand presence in the input image.
3. A binary classification of handedness, e.g., left or right hand.

c) Feature Definition

The landmarks were converted to vectors to facilitate mathematical notation. The next important step was to find a way to use these vectors in order to create features able to train a neural network that will recognize the American Sign Language signs. The basic factor chosen to calculate the letter signed was the angle between the vectors. The main reason why the angle was chosen as the basic factor of the feature vectors is that the proposed features should be invariant in any translations, rotations and change of scale, which easily occur when the orientation or the distance of the person from the camera are changed. This led to the definition of the following two features vectors.

1ST FEATURE VECTOR

For the extraction of the first feature vector, the angle of any two vectors, defined by the wrist landmark and the rest of the landmarks, were considered. To this end, the first step is to subtract the wrist's coordinates V_0 from each landmark V_i (Equation 1). The *cosine* of the required angle is calculated by the dot product of the unit vectors (Equation 2).

$$\mathbf{V_i'} = V_i - V_0 \quad (1)$$

$$\cos(\theta_{ij}) = \frac{\mathbf{V_i}' \cdot \mathbf{V_j}'}{|\mathbf{V_i}'||\mathbf{V_j}'|} \quad (2)$$

$$\theta_{ij} = cos^{-1}(\theta_{ij}) \quad (3)$$

An example of the definition of the θ_{ij} between the 3rd (THUMB_IP) an 5th landmark (INDEX_FINGER_MCP) is depicted in Fig. 14.

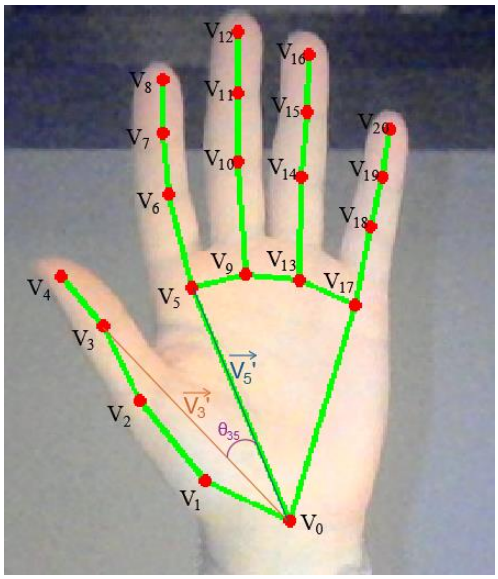


Figure 14. First feature calculations

This process resulted in 190 angles for every frame forming an entry of the training dataset. Depending on the depicted letter, the corresponding label was added to the data entry.

2ND FEATURE VECTOR

The second feature vector contains the angles between the vectors defined by every two successive landmarks in each finger. First the vectors are defined as following (Equation 4)

$$\mathbf{p_i} = \begin{cases} V_i - V_0, i = 1,5,9,13,17 \\ V_i - V_{i-1}, i \neq 0,1,5,9,13,17 \end{cases} \quad (4)$$

A similar process that was used in the extraction of the first feature was followed, that is, the required angle was found by the dot product of the vectors.

$$\varphi_i = cos^{-1}\left(\frac{\mathbf{p_i} \cdot \mathbf{p_{i-1}}}{|\mathbf{p_i}||\mathbf{p_{i-1}}|}\right) \quad (5)$$

The proposed feature vector is defined as

$$F_2 = \left(\underbrace{\phi_2, \phi_3, \phi_4}_{\text{thumb}}, \underbrace{\phi_6, \phi_7, \phi_8}_{\text{indexfinger}}, \underbrace{\phi_{10}, \phi_{11}, \phi_{12}}_{\text{middlefinger}}, \underbrace{\phi_{14}, \phi_{15}, \phi_{16}}_{\text{ringfinger}}, \underbrace{\phi_{18}, \phi_{19}, \phi_{20}}_{\text{pinky}} \right) \quad (6)$$

An example of the definition of the φ_i between the 1st (THUMB_CMC) an 2nd landmark (THUMB_MCP) is depicted in Fig. 15.

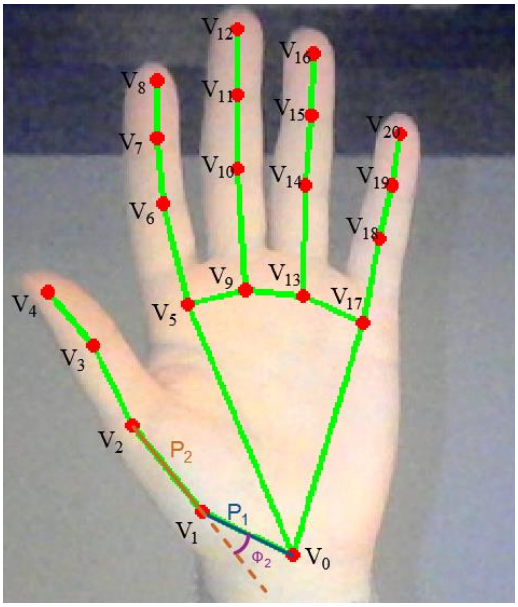
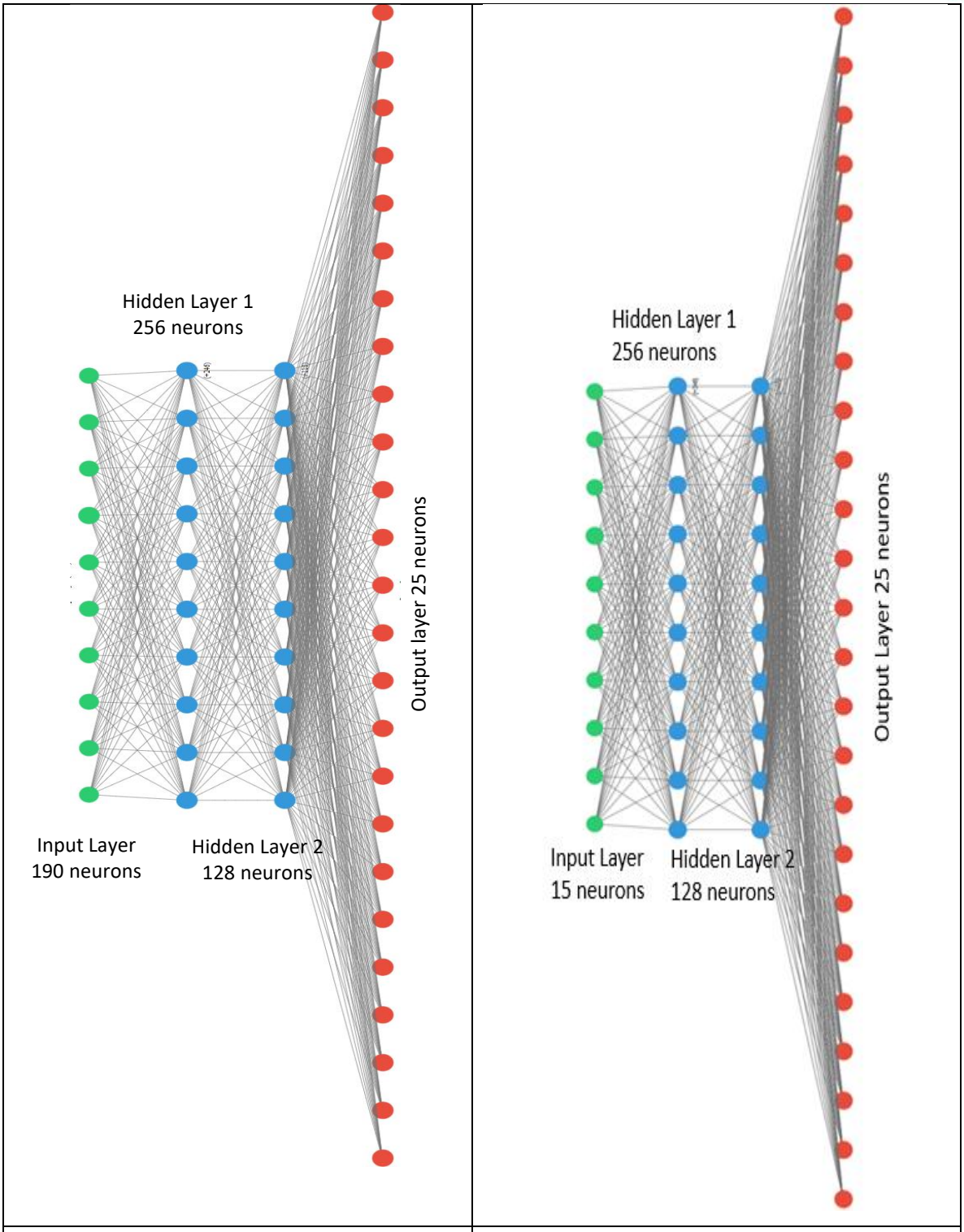


Figure 15. Second Feature Calculations Example

Consequently, 3 angles are created for each finger, resulting in 15 angles per hand in each frame. Each set of angles represent an entry to the second training dataset. Depending on the depicted letter, the corresponding label was given to the entry.

d) Alphabet classification using ANN

In order to achieve sign language recognition using the above features the use of a neural network, for each feature vector was necessary. Each neural network consists of 4 layers: 1 input layer, 2 hidden layers and the output layer. Implementing two hidden layers leads to creating a deep neural network. In the input layer, 190 neurons were used for the first feature and 15 for the second one due to the size of the entries in the training dataset for each feature. The two hidden layers consist of 256 and 128 neurons, respectively while the output layer has 25 neurons for the 24 letters of the ASL alphabet (0,25). All of the layers, except the output layer, use the rectified linear (ReLu) activation function that is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. On the contrary, the softmax activation function is used for the output layer, an activation function that converts a vector into a vector with probabilities of each possible outcome. It is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes. The architectures for each network are depicted in Fig.16. The only difference between the two Neural Networks is the size of the input layer which is formed based on the entry size of the training dataset.



ANN Topology for the first feature vector	ANN Topology for the second feature vector

Figure 16. The two ANN topologies tested in this work.

IV. RESULTS

a) Dataset description

To create the dataset four American Sign Language alphabet videos were selected. Two of them were shown two or three different perspectives of the signs so they were cropped to two and three videos, respectively, as shown in the figures 10 and 11, which resulted in reaching the final number of seven videos that enriched the dataset. Then these videos were converted into image sequences, with over 1500 frames for each video. The table depicts the actual number of frames that were created per video and the number of the frames for each ASL alphabet sign. Some of the frames were not included as they were unclear or illustrated the transition from the current sign to the next.

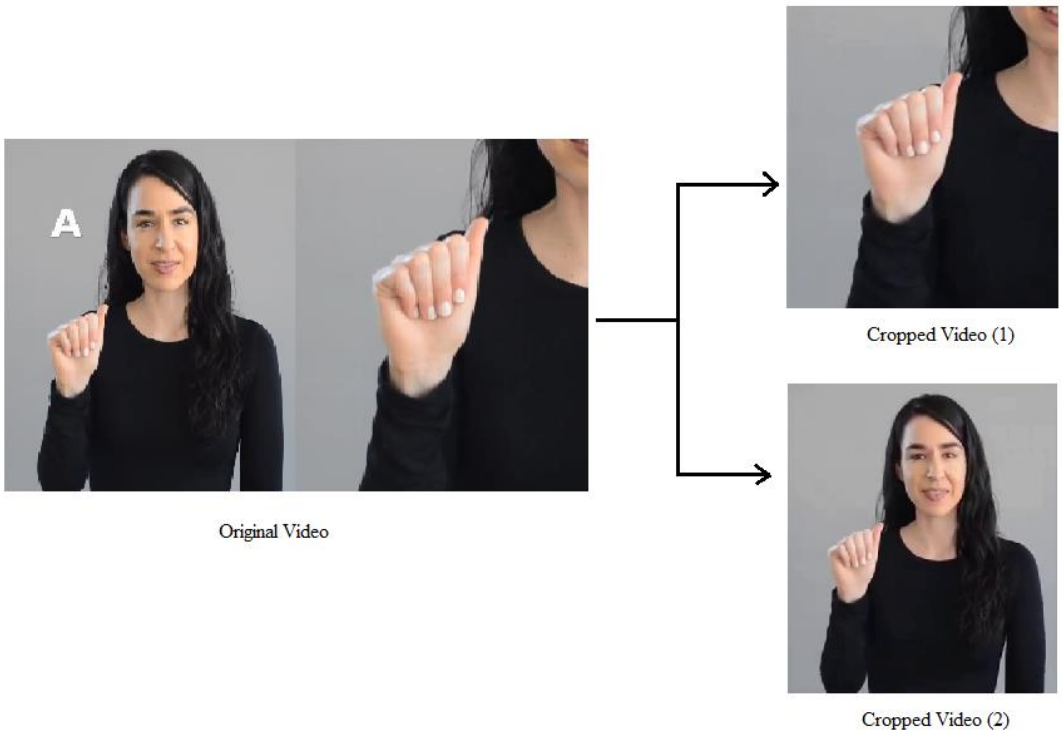


Figure 17. Cropping Process

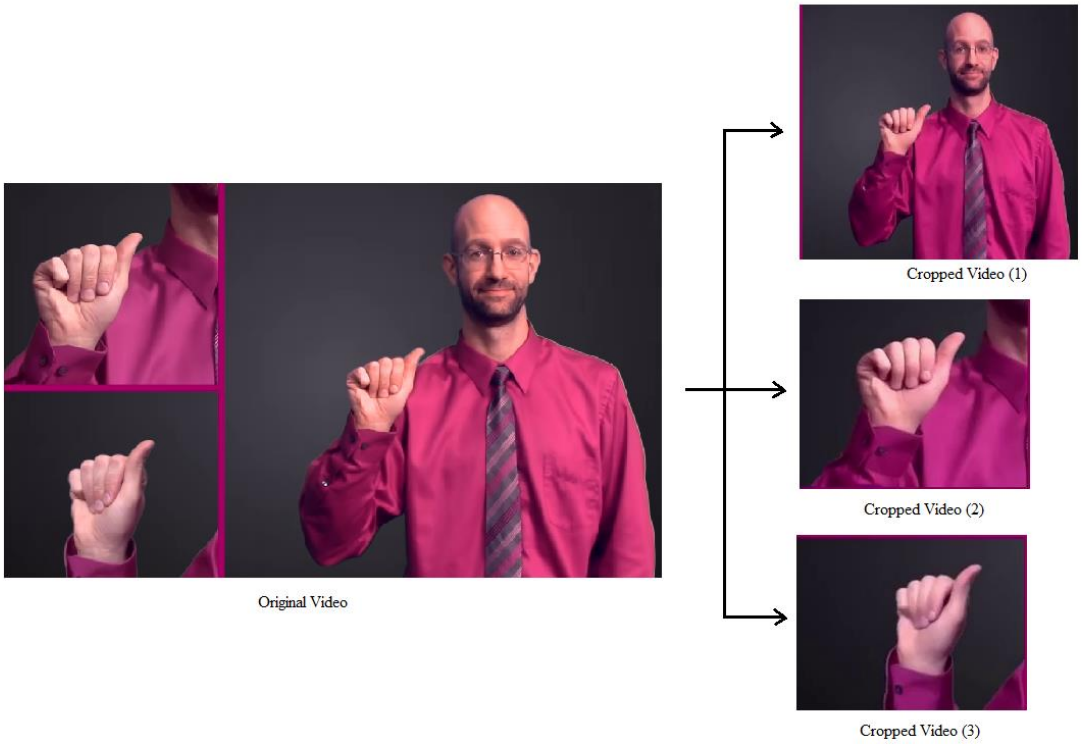


Figure 18. Cropping Process

Table 1. Number of Frames for each letter per Video

	Videos						
Number of Frames	**No 1**	**No 2**	**No 3**	**No 4**	**No 5**	**No 6**	**No 7**
A	51	122	40	53	70	70	70
B	66	188	77	77	103	103	103
C	65	160	83	77	103	103	103
D	64	152	79	75	97	97	97
E	72	124	74	68	101	101	101
F	61	153	71	72	98	98	98
G	65	125	86	83	99	99	99
H	54	117	81	78	86	86	86
I	59	126	65	58	89	89	89
K	69	147	59	59	89	89	89
L	65	136	75	77	85	85	85

M	43	121	58	65	74	74	74
N	38	121	67	63	99	99	99
O	53	107	66	56	67	67	67
P	44	171	63	68	59	59	59
Q	50	130	68	64	103	103	103
R	72	138	54	58	103	103	103
S	45	129	40	49	103	103	103
T	52	137	51	49	67	67	67
U	55	130	49	49	72	72	72
V	77	132	67	70	65	65	65
W	49	118	61	59	90	90	90
X	59	126	57	55	93	93	93
Y	51	116	53	43	57	57	57
Total Frames	**1926**	**4806**	**2009**	**2014**	**2479**	**2479**	**2479**
Included Frames	**1379**	**3226**	**1544**	**1525**	**2072**	**2072**	**2072**

The above frames were implemented in Mediapipe's python API for static images and the coordinates for the twenty-one hand landmarks, in which the two features are based, were returned. The final dataset for each feature consists of 9862 entries from which 7889 are used for training the Neural Network and the other 1973 entries are used for testing.

b) Quantitative Results

The results are remarkable as we managed to achieve 99% accuracy in sign language recognition for the first feature and 94% for the second one. The complete results can be seen in the figures below. Fig.20 depicts the Confusion Matrix for the first feature vector. The predicted labels are placed on the x-axis and the true labels on the y-axis. The blue cells running from the top left to bottom right contain the number of samples that the model accurately predicted. The white cells contain the number of samples that were incorrectly predicted. There are 1973 total samples in the test set. Looking at the confusion matrix, it

is clear that the model accurately predicted 1884 out of 1973 total samples. The model incorrectly predicted 89 out of the 1973.

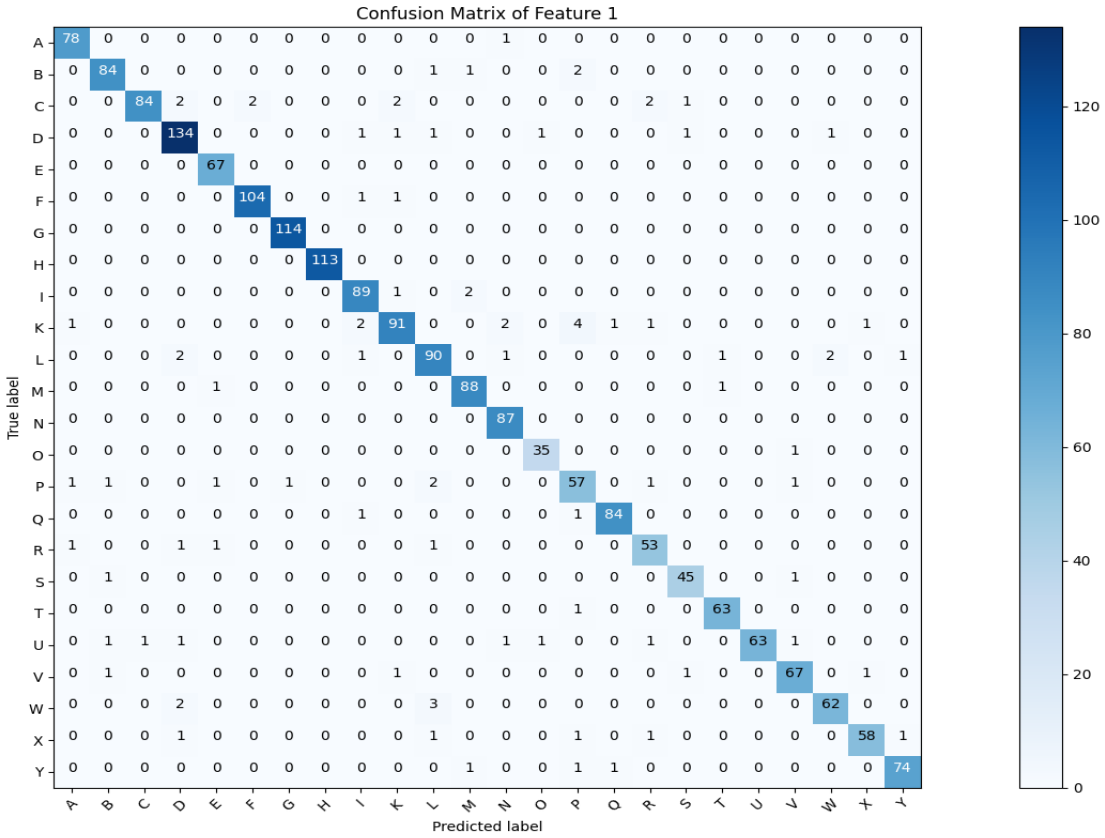


Figure 19. Confusion Matrix of the first feature

Figure 20 presents the loss and the accuracy of the first feature vector in one training.

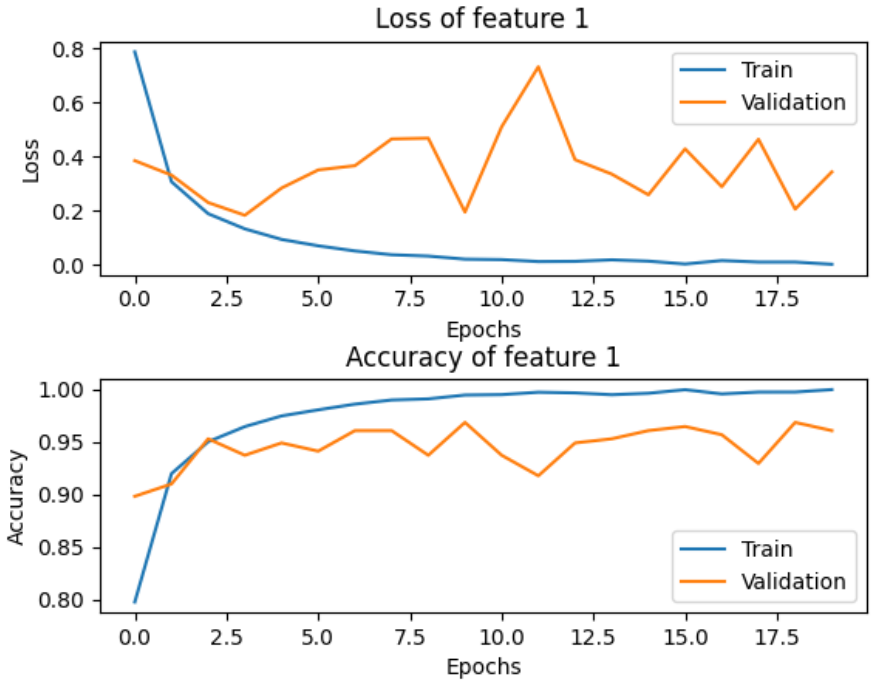


Figure 20. Train Loss and Accuracy of the first feature

Figure 21 depicts the Confusion Matrix for the second feature vector. The red cells running from the top left to bottom right contain the number of samples that the model accurately predicted. The white cells contain the number of samples that were incorrectly predicted. This time the model predicted accurately 1798 and 175 incorrectly out of 1973 samples.

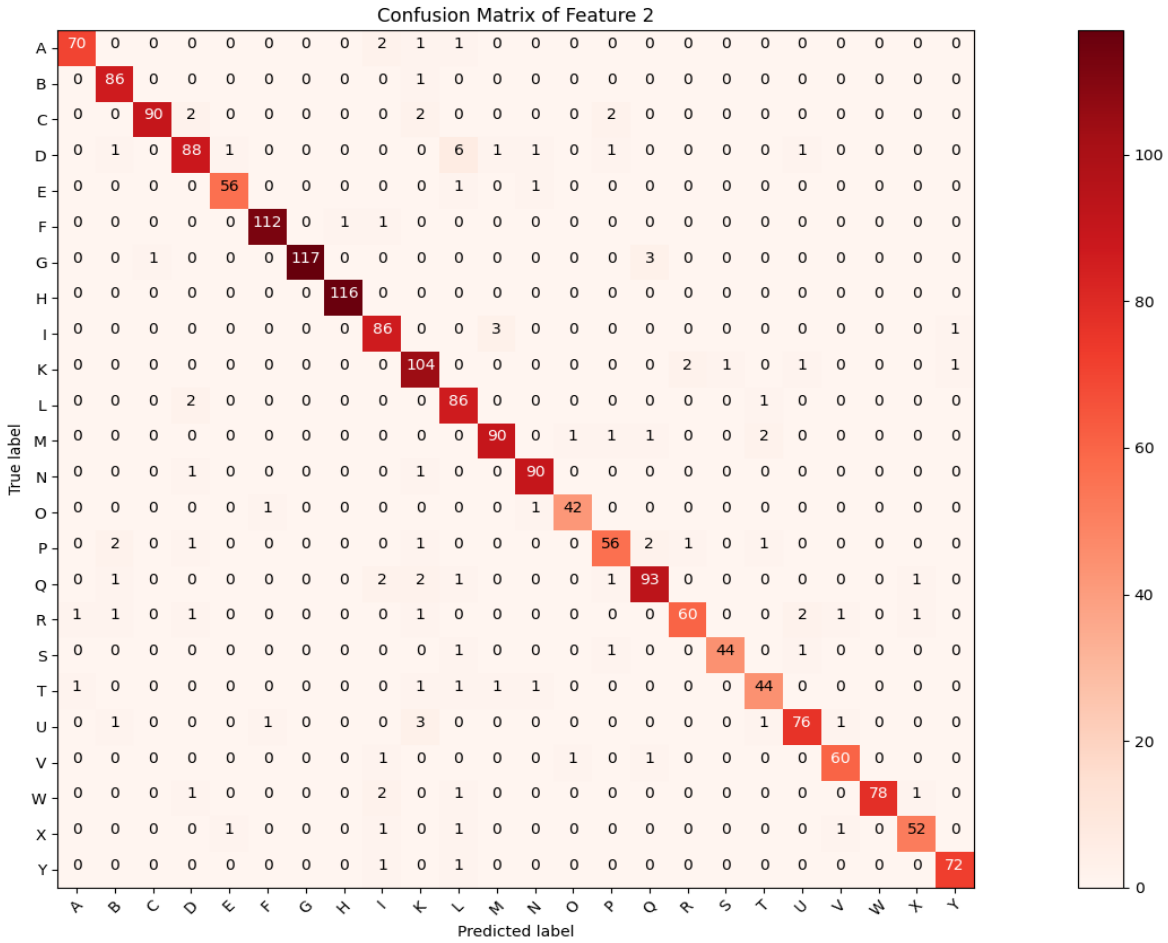


Figure 21. Confusion Matrix of the second feature

Figure 22 depicts the loss and the accuracy of the model for one training of the second feature vector.

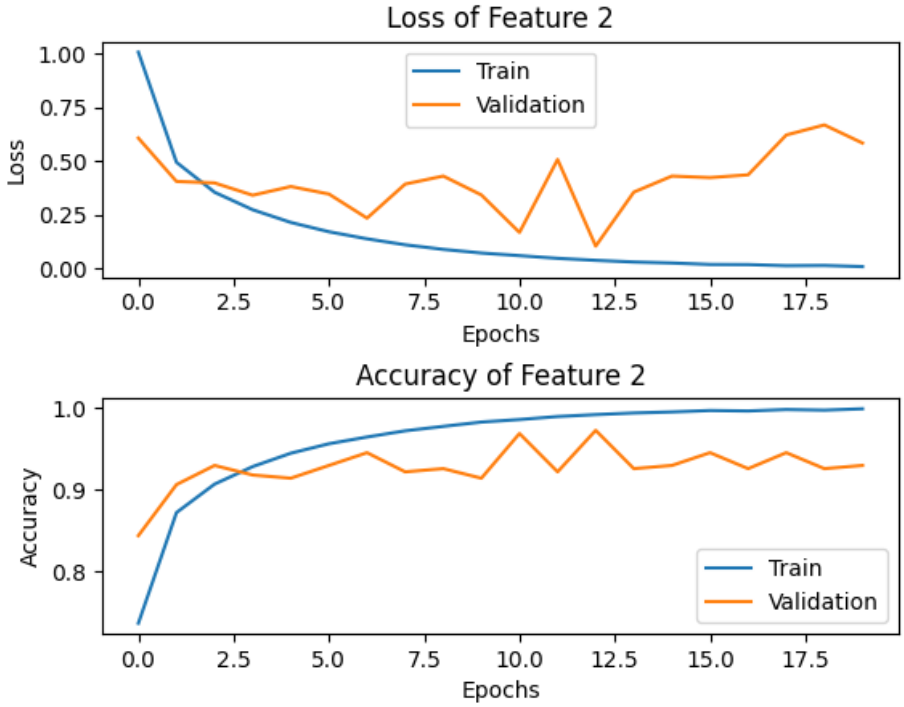


Figure 22. Train Loss and Accuracy of the second feature

To get clearer results the Neural Network was trained 10 times and the Standard Deviation and the Average of train loss and validation loss were calculated resulting in the figures below that demonstrate the efficacy of the network.

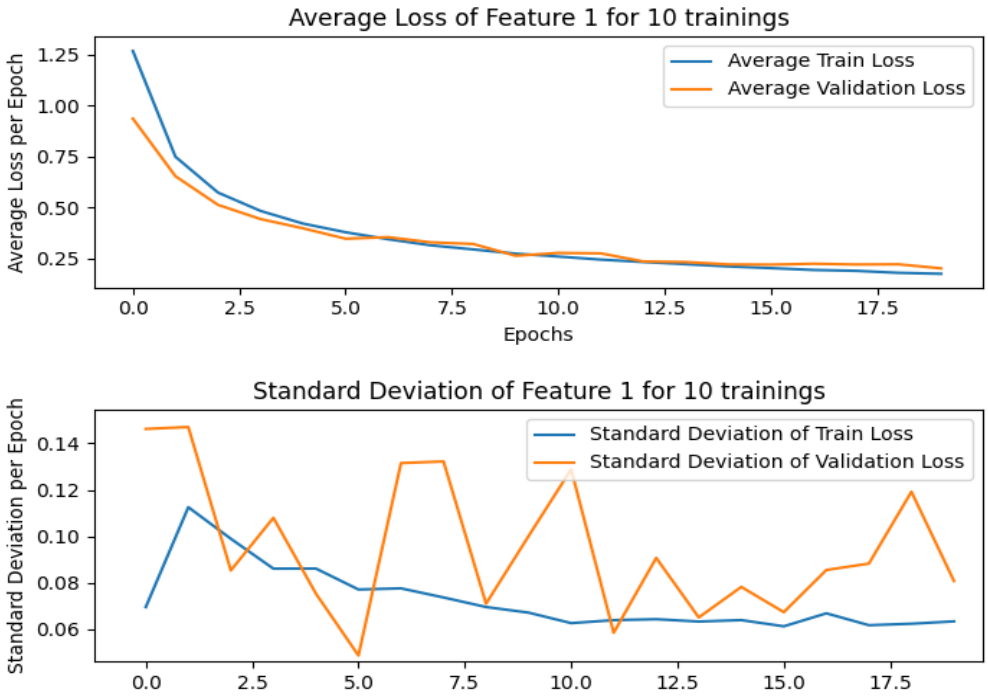


Figure 23. Average Loss and Standard Deviation of the first feature for 10 trainings

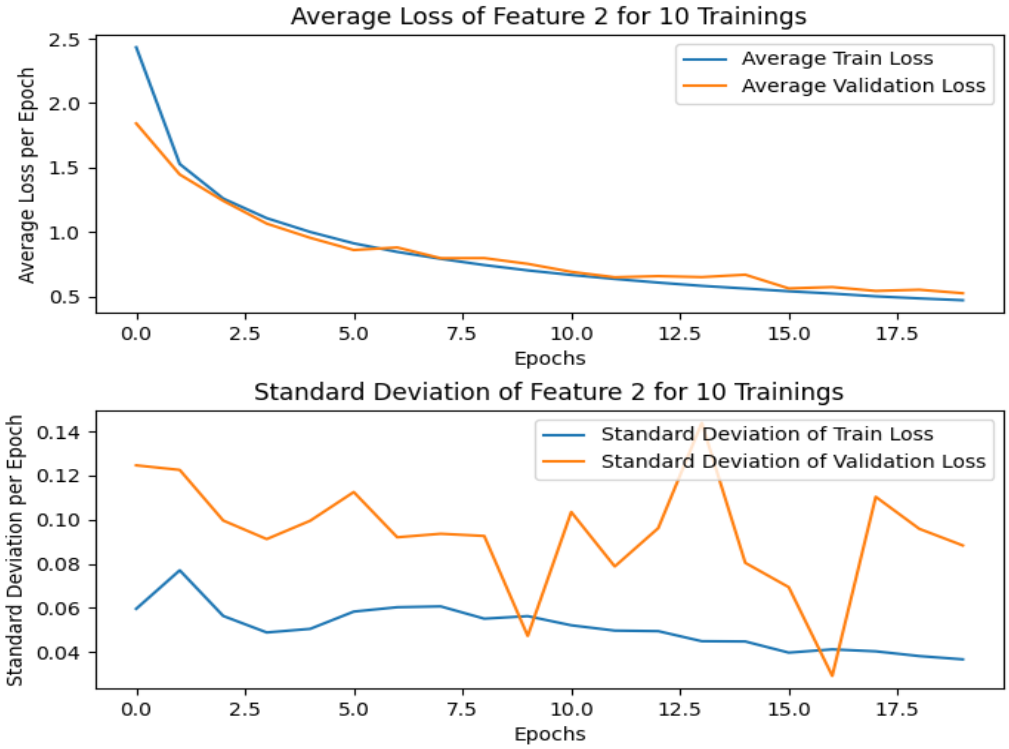


Figure 24. Average Loss and Standard Deviation of the second feature for 10 trainings

c) Demonstration of feature invariance

As it was previously mentioned, the proposed feature vectors should be invariant in translation, rotation and scale. Figures 25, 26 and 27 demonstrate the invariance of the selected feature vectors. More precisely, the system was tested for the recognition of the letter B three times. The first time the letter was signed directly to the camera, just like the videos that were used as input. The second time the letter 'B' was displayed translated and rotated (see Fig. 26), whereas in Fig. 27 the same letter was signed at a different viewing angle (side view). All three times the system managed to recognize correctly the letter confirming the feature invariance. This is a very important feature of the proposed system, since it can robustly identify the signed letter under very different geometric settings.

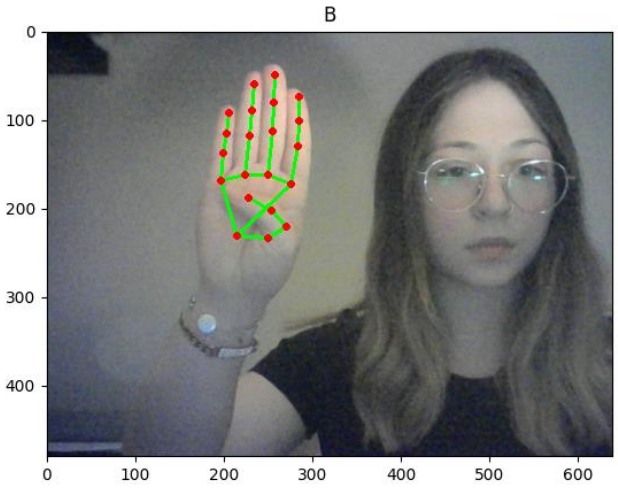


Figure 25. Letter 'B' Signed directly to the camera.

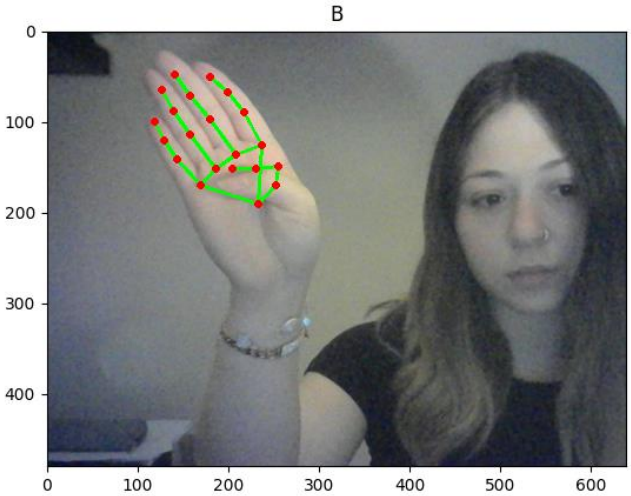


Figure 26. Letter 'B' signed translated and rotated.

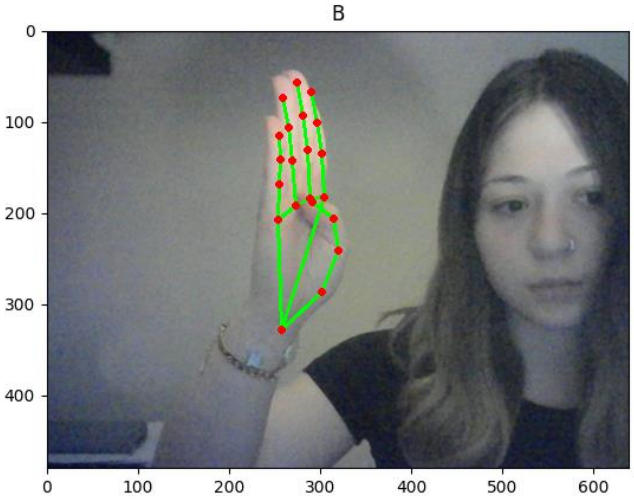


Figure 27. Letter 'B' signed from side view.

d) System's complexity and execution time

The basic operations of the system were tested to see its efficiency. Table 2 presents the execution time for each operation.

Table 2. Average execution times

Operation	Time (ms)
MediaPipe Hand landmark detection	40
Feature extraction	<< 0.1
ANN forward pass Feature 1	38
ANN forward pass Feature 2	35

Table 3. Fps per feature

SLR	FPS
1st Feature Vector	12,8
2nd Feature Vector	13,3

V. DISCUSSION AND CONCLUSIONS

Computer Vision using Deep Learning is a powerful combination that can have several applications in making daily life easier and less complicated. One such possible application, which has not received extensive attention by researchers, is Sign Language Recognition. It is very important to recognize real-time Sign Language through a computer to strengthen the socialization of the deaf without the need for an extra person.

Over the last decade many works of research have been directed toward developing a sign recognition system for different sign languages and it was concluded that such a system is challenging for various disciplines including gestures capturing method, machine learning classifiers, and natural language processing. The complexity in the sign recognition system arises from the fact that sign languages are the least identical, with large vocabularies and referential language, thus making the task of recognizing isolated or continuous signing highly multifaceted.

The system developed in this work, exploits these technologies to recognize successfully ASL alphabet letters. The proposed system can achieve automatic ASL letter recognition approximately equal for both of the proposed features at 13 fps, using minimal computer hardware, without any special camera requirements.

A possible and easily implemented expansion of the model could be the recognition of the ASL numbers, or recognition of numbers and letters of any other static Sign Language. The use of this system in phrase recognition could also be investigated.

VI. REFERENCES

[1] "Tensorflow," Google Brain, 2015. [Online]. Available: https://www.tensorflow.org/overview.

[2] Google, "MediaPipe Hands," 2020. [Online]. Available: https://google.github.io/mediapipe/solutions/hands.html.

[3] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov,Andrei Tkachenka, George Sung, Chuo-Ling Chang, Matthias Grundmann,, "MediaPipe Hands: On-device Real-time Hand Tracking," p. 5, 2020.

[4] Google, "MediaPipe," Google, 2020. [Online]. Available: https://google.github.io/mediapipe/.

[5] Cao Dong, Ming C Leu , Zhaozheng Yin, "American Sign Language Alphabet Recognition Using Microsoft Kinect," *tMissouri University of Science and Technology,* p. 52, 2015.

[6] Sarfaraz Masood, Harish Chandra Thuwal and Adhyan Srivastava, "American Sign Language Character Recognition Using Convolution Neural Network," *Smart Computing and Informatics,* pp. 403-412, 2018.

[7] Barczak, A.L.C., N.H. Reyes, M. Abastillas, A. Piccio, and T. Susnjak, "A new 2D static hand gesture colour image dataset for asl gestures," *Res. Lett. Inf. Math. Sci,* vol. 15, pp. 12-20, 2011.

[8] Kshitij Bantupalli,Ying Xie, "American Sign Language Recognition using Deep Learning and Computer Vision," *IEEE International Conference on Big Data (Big Data),* pp. 4896 - 4899, 2018.

[9] Teak-Wei Chong, Boon-Giin Lee, "American Sign Language Recognition Using Leap Motion Controller with Machine Learning Approach," *Sensors,* vol. 18, no. 3554, p. 17, 2018.

[10] Luis Quesada, Gustavo López, Luis Guerrero, "Automatic recognition of the American sign language fingerspelling alphabet to assist people living with speech or hearing impairments," *Springer-Verlag Berlin,* p. 11, 2017.

[11] Jordan J. Bird , Anikó Ekárt and Diego R. Faria, "British Sign Language Recognition via Late Fusion of Computer Vision and Leap Motion with Transfer Learning to American Sign Language," *Sensorw,* vol. 20, no. 5151, p. 19, 2020.

[12] Douglas F. L. Lima, Armando S. Salvador Neto, Ewerton N. Santos, Tiago Maritan U. Araujo, Thais Gaudencio do Rêgo, "Using Convolutional Neural Networks for Fingerspelling Sign Recognition in Brazilian Sign Language," *Brazilian Symposium on Multimedia and the Web,* pp. 109-115, 2019.

[13] Tomasz Kapuscinski, Patryk Organisciak, "Handshape Recognition Using Skeletal Data," *Sensors,* vol. 18, no. 2557, p. 17, 2018.

[14] Giulio Marin, Fabio Dominio, Pietro Zanuttigh, "Hand Gesrure Recognition With Leap Motion And Kinect Devices," *In Proceedings of the 2014 IEEE International Conference on Image Processing,* pp. 1565-1569, 2014.

[15] M. Ebrahim Al-Ahdal, Nooritawati Md TahiR, "Review in Sign Language Recognition Systems," *IEEE Symposium on Computers & Informatics,* pp. 52-57, 2012.

[16] Towards Data Science, "Towards Data Science," [Online]. Available: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

VII. CODE

a) Code for video to frame conversion

```
import os
import cv2

cam = cv2.VideoCapture("C:\\Users\\lia\\Desktop\\video
input\\asl6.mp4")

try:

    if not os.path.exists('data6'):
        os.makedirs('data6')


except OSError:
    print('Error: Creating directory of data6')

# frame
currentframe = 0

while True:

    ret, frame = cam.read()

    if ret:
        # if video is still left continue creating images
        name = './data6/frame' + str(currentframe) + '.jpg'
        print('Creating...' + name)

        # writing the extracted images
        cv2.imwrite(name, frame)

        currentframe += 1
    else:
        break

cam.release()
cv2.destroyAllWindows()
```

b) Code for Landmarks Detection and First Feature Vector

```
import mediapipe as mp

import pandas as pd

import xlsxwriter

import cv2
```

```
import numpy as np

import math


df = pd.read_csv(r'C:\Users\lia\Desktop\video
input\Entries6.txt')


a = df.loc[:, 'A']

b = df.loc[:, 'B']

c = df.loc[:, 'C']

d = df.loc[:, 'D']

e = df.loc[:, 'E']

f = df.loc[:, 'F']

g = df.loc[:, 'G']

h = df.loc[:, 'H']

i = df.loc[:, 'I']

k = df.loc[:, 'K']

l = df.loc[:, 'L']

m = df.loc[:, 'M']

n = df.loc[:, 'N']

o = df.loc[:, 'O']

p = df.loc[:, 'P']

q = df.loc[:, 'Q']

r = df.loc[:, 'R']

s = df.loc[:, 'S']

t = df.loc[:, 'T']

u = df.loc[:, 'U']

v = df.loc[:, 'V']

w = df.loc[:, 'W']
```

```
x = df.loc[:, 'X']

y = df.loc[:, 'Y']


alphabet = [a, b, c, d, e, f, g, h, i, k, l, m, n, o, p, q,
r, s, t, u, v, w, x, y]

c = -1

row = 0

col = 0

q = 0

workbook = xlsxwriter.Workbook('Angles7.xlsx')

worksheet1 = workbook.add_worksheet()


mp_drawing = mp.solutions.drawing_utils

mp_hands = mp.solutions.hands

for j in alphabet:

    q = q + 1

    for i in j:

        i = int(i)

        file =
'C:/Users/lia/PycharmProjects/pythonProject/data6/frame' +
str(i) + '.jpg'

        with mp_hands.Hands(

                static_image_mode=True,

                max_num_hands=2,

                min_detection_confidence=0.5) as hands:

            # Read an image, flip it around y-axis for
correct handedness output (see

            # above).

            image = cv2.flip(cv2.imread(file), 1)
```

```
            # Convert the BGR image to RGB before
processing.

            results = hands.process(cv2.cvtColor(image,
cv2.COLOR_BGR2RGB))

            nl = '\n'

            # Print handedness and draw hand landmarks on
the image.

            print('Handedness:', results.multi_handedness)

            if not results.multi_hand_landmarks:

                continue

            image_height, image_width, _ = image.shape

            annotated_image = image.copy()

            for hand_landmarks in
results.multi_hand_landmarks:

                c = c + 1

                wristx =
hand_landmarks.landmark[mp_hands.HandLandmark.WRIST].x

                wristy =
hand_landmarks.landmark[mp_hands.HandLandmark.WRIST].y

                wristz =
hand_landmarks.landmark[mp_hands.HandLandmark.WRIST].z

                ##THUMP

                ##Landmark1 : Thump cmc coordinates:

                XCMCthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_CMC].x

                YCMCthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_CMC].y

                ZCMCthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_CMC].z


                ##Landmark2 : Thump mcp coordinates:
```

```
                XMCPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_MCP].x

                YMCPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_MCP].y

                ZMCPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_MCP].z


                ##Landmark3 : Thump ip coordinates:

                XIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_IP].x

                YIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_IP].y

                ZIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_IP].z


                ##Landmark4 : Thump tip coordinates:

                XTIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].x

                YTIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].y

                ZTIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].z

                ##INDEX

                ##Landmark5 : Index finger mcp coordinates:

                XMCPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_M
CP].x

                YMCPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_M
CP].y

                ZMCPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_M
CP].y
```

```
                ##Landmark6 : Index finger pip coordinates:

                XPIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_P
IP].x

                YPIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_P
IP].y

                ZPIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_P
IP].z


                ##Landmark7 : Index finger dip coordinates:

                XDIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_D
IP].x

                YDIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_D
IP].y

                ZDIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_D
IP].z


                ##Landmark8 : Index finger tip coordinates:

                XTIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_T
IP].x

                YTIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_T
IP].y

                ZTIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_T
IP].z


                ##MIDDLE FINGER
```

```
                ##Landmark9 :  Middle finger mcp
coordinates:

                XMCPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
MCP].x

                YMCPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
MCP].y

                ZMCPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
MCP].z


                ##Landmark10 : Middle finger pip
coordinates:

                XPIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
PIP].x

                YPIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
PIP].y

                ZPIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
PIP].z


                ##Landmark11 : Middle finger dip
coordinates:

                XDIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
DIP].x

                YDIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
DIP].y

                ZDIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
DIP].z
```

```
                ##Landmark12 : Middle finger tip
coordinates:

                XTIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
TIP].x

                YTIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
TIP].y

                ZTIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
TIP].z


                ##RING FINGER

                ##Landmark13 : Ring finger mcp coordinates:

                XMCPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_MC
P].x

                YMCPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_MC
P].y

                ZMCPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_MC
P].z


                ##Landmark14 : Ring finger pip coordinates:

                XPIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_PI
P].x

                YPIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_PI
P].y

                ZPIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_PI
P].z
```

```
                ##Landmark15 : Ring finger dip coordinates:

                XDIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_DI
P].x

                YDIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_DI
P].y

                ZDIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_DI
P].z


                ##Landmark16 : Ring finger tip coordinates:

                XTIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_TI
P].x

                YTIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_TI
P].y

                ZTIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_TI
P].z


                ##PINKY

                ##Landmark17 : Pinky mcp coordinates:

                XMCPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_MCP].x

                YMCPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_MCP].y

                ZMCPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_MCP].z


                ##Landmark18 : Pinky pip coordinates:
```

```
                XPIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_PIP].x

                YPIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_PIP].y

                ZPIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_PIP].z


                ##Landmark19 : Pinky dip coordinates:

                XDIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_DIP].x

                YDIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_DIP].y

                ZDIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_DIP].z


                ##Landmark20 : Pinky tip coordinates:

                XTIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_TIP].x

                YTIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_TIP].y

                ZTIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_TIP].z


                ##THUMB

                newCMCthumbx = XCMCthumb - wristx

                newCMCthumby = YCMCthumb - wristy

                newCMCthumbz = ZCMCthumb - wristz


                newMCPthumbx = XMCPthumb - wristx

                newMCPthumby = YMCPthumb - wristy

                newMCPthumbz = ZMCPthumb - wristz
```

```
newIPthumbx = XIPthumb - wristx

newIPthumby = YIPthumb - wristy

newIPthumbz = ZIPthumb - wristz


newTIPthumbx = XTIPthumb - wristx

newTIPthumby = YTIPthumb - wristy

newTIPthumbz = ZTIPthumb - wristz


##INDEX FINGER

newMCPindexx = XMCPindex - wristx

newMCPindexy = YMCPindex - wristy

newMCPindexz = ZMCPindex - wristz


newPIPindexx = XPIPindex - wristx

newPIPindexy = YPIPindex - wristy

newPIPindexz = ZPIPindex - wristz


newDIPindexx = XDIPindex - wristx

newDIPindexy = YDIPindex - wristy

newDIPindexz = ZDIPindex - wristz


newTIPindexx = XTIPindex - wristx

newTIPindexy = YTIPindex - wristy

newTIPindexz = ZTIPindex - wristz


##MIDDLE FINGER
```

```
newMCPmiddlex = XMCPmiddle - wristx

newMCPmiddley = YMCPmiddle - wristy

newMCPmiddlez = ZMCPmiddle - wristz


newPIPmiddlex = XPIPmiddle - wristx

newPIPmiddley = YPIPmiddle - wristy

newPIPmiddlez = ZPIPmiddle - wristz


newDIPmiddlex = XDIPmiddle - wristx

newDIPmiddley = YDIPmiddle - wristy

newDIPmiddlez = ZPIPmiddle - wristz


newTIPmiddlex = XTIPmiddle - wristx

newTIPmiddley = YTIPmiddle - wristy

newTIPmiddlez = ZTIPmiddle - wristz


##RING FNGER

newMCPringx = XMCPring - wristx

newMCPringy = YMCPring - wristy

newMCPringz = ZMCPring - wristz


newPIPringx = XPIPring - wristx

newPIPringy = YPIPring - wristy

newPIPringz = ZPIPring - wristz


newDIPringx = XDIPring - wristx

newDIPringy = YDIPring - wristy
```

```
                newDIPringz = ZDIPring - wristz


                newTIPringx = XTIPring - wristx
                newTIPringy = YTIPring - wristy
                newTIPringz = ZTIPring - wristz


                ##PINKY
                newMCPpinkyx = XMCPpinky - wristx
                newMCPpinkyy = YMCPpinky - wristy
                newMCPpinkyz = ZMCPpinky - wristz


                newPIPpinkyx = XPIPpinky - wristx
                newPIPpinkyy = YPIPpinky - wristy
                newPIPpinkyz = ZPIPpinky - wristz


                newDIPpinkyx = XDIPpinky - wristx
                newDIPpinkyy = YDIPpinky - wristy
                newDIPpinkyz = ZDIPpinky - wristz


                newTIPpinkyx = XTIPpinky - wristx
                newTIPpinkyy = YTIPpinky - wristy
                newTIPpinkyz = ZTIPpinky - wristz


                thumb_cmc_vec = np.array([newCMCthumbx,
newCMCthumby, newCMCthumbz])
                thumb_mcp_vec = np.array([newMCPthumbx,
newMCPthumby, newMCPthumbz])
```

```
                thumb_ip_vec = np.array([newIPthumbx,
newIPthumby, newIPthumbz])

                thumb_tip_vec = np.array([newTIPthumbx,
newTIPthumby, newTIPthumbz])


                index_mcp_vec = np.array([newMCPindexx,
newMCPindexy, newMCPindexz])

                index_pip_vec = np.array([newPIPindexx,
newPIPindexy, newPIPindexz])

                index_dip_vec = np.array([newDIPindexx,
newDIPindexy, newDIPindexz])

                index_tip_vec = np.array([newTIPindexx,
newTIPindexy, newTIPindexz])


                middle_mcp_vec = np.array([newMCPmiddlex,
newMCPmiddley, newMCPmiddlez])

                middle_pip_vec = np.array([newPIPmiddlex,
newPIPmiddley, newPIPmiddlez])

                middle_dip_vec = np.array([newDIPmiddlex,
newDIPmiddley, newDIPmiddlez])

                middle_tip_vec = np.array([newTIPmiddlex,
newTIPmiddley, newTIPmiddlez])


                ring_mcp_vec = np.array([newMCPringx,
newMCPringy, newMCPringz])

                ring_pip_vec = np.array([newPIPringx,
newPIPringy, newPIPringz])

                ring_dip_vec = np.array([newDIPringx,
newDIPringy, newDIPringz])

                ring_tip_vec = np.array([newTIPringx,
newTIPringy, newTIPringz])
```

```
                pinky_mcp_vec = np.array([newMCPpinkyx,
newMCPpinkyy, newMCPpinkyz])

                pinky_pip_vec = np.array([newPIPpinkyx,
newPIPpinkyy, newPIPpinkyz])

                pinky_dip_vec = np.array([newDIPpinkyx,
newDIPpinkyy, newDIPpinkyz])

                pinky_tip_vec = np.array([newTIPpinkyx,
newTIPpinkyy, newTIPpinkyz])


                vectors = [thumb_cmc_vec, thumb_mcp_vec,
thumb_ip_vec, thumb_tip_vec, index_mcp_vec, index_pip_vec,

                           index_dip_vec, index_tip_vec,
middle_mcp_vec, middle_pip_vec, middle_dip_vec,
middle_tip_vec,

                           ring_mcp_vec, ring_pip_vec,
ring_dip_vec, ring_tip_vec, pinky_mcp_vec, pinky_pip_vec,

                           pinky_dip_vec, pinky_tip_vec]
                list= []
                for i in range(len(vectors) - 1):
                    vector = vectors[i]
                    for j in range(i + 1, len(vectors)):
                        next_vector = vectors[j]
                        M = (np.linalg.norm(vector) *
np.linalg.norm(next_vector))
                        ES = np.dot(vector, next_vector)
                        th = math.acos(ES / M)
                        list.append([th])
                worksheet1.activate()


                worksheet1.write(row + c, 0, q)
                for col, data in enumerate(list):
```

```
                    worksheet1.write_row(row + c, col + 1,
data)


workbook.close()
```

c) Code for Landmarks Detection and Second Feature Vector

```
import mediapipe as mp
import pandas as pd
import xlsxwriter
import cv2
import numpy as np
import math

def Calculations (vec1,vec2,vec3) :
    vec4 = vec2-vec1
    vec5 = vec3-vec2
    M = np.linalg.norm(vec4)*np.linalg.norm(vec5)
    ES = np.dot(vec4, vec5)
    th2 = math.acos(ES/M)
    return [th2]



df = pd.read_csv(r'C:\Users\lia\Desktop\video
input\Entries6.txt')

a = df.loc[:, 'A']
b = df.loc[:, 'B']
c = df.loc[:, 'C']
d = df.loc[:, 'D']
e = df.loc[:, 'E']
f = df.loc[:, 'F']
g = df.loc[:, 'G']
h = df.loc[:, 'H']
i = df.loc[:, 'I']
k = df.loc[:, 'K']
l = df.loc[:, 'L']
m = df.loc[:, 'M']
n = df.loc[:, 'N']
o = df.loc[:, 'O']
p = df.loc[:, 'P']
q = df.loc[:, 'Q']
r = df.loc[:, 'R']
s = df.loc[:, 'S']
```

```
t = df.loc[:, 'T']
u = df.loc[:, 'U']
v = df.loc[:, 'V']
w = df.loc[:, 'W']
x = df.loc[:, 'X']
y = df.loc[:, 'Y']

alphabet = [a, b, c, d, e, f, g, h, i, k, l, m, n, o, p, q,
r, s, t, u, v, w, x, y]
c = -1
row = 0
col = 0
q = 0
workbook = xlsxwriter.Workbook('NewAngles7.xlsx')
worksheet1 = workbook.add_worksheet()

mp_drawing = mp.solutions.drawing_utils
mp_hands = mp.solutions.hands
for j in alphabet:
    q = q + 1
    for i in j:
        i = int(i)
        file =
'C:/Users/lia/PycharmProjects/pythonProject/data6/frame' +
str(i) + '.jpg'
        with mp_hands.Hands(
                static_image_mode=True,
                max_num_hands=2,
                min_detection_confidence=0.5) as hands:
            # Read an image, flip it around y-axis for
correct handedness output (see
            # above).
            image = cv2.flip(cv2.imread(file), 1)
            # Convert the BGR image to RGB before
processing.
            results = hands.process(cv2.cvtColor(image,
cv2.COLOR_BGR2RGB))
            nl = '\n'
            # Print handedness and draw hand landmarks on
the image.
            print('Handedness:', results.multi_handedness)
            if not results.multi_hand_landmarks:
                continue
            image_height, image_width, _ = image.shape
            annotated_image = image.copy()
            for hand_landmarks in
results.multi_hand_landmarks:
                c = c + 1
                wristx =
hand_landmarks.landmark[mp_hands.HandLandmark.WRIST].x
```

```
                wristy =
hand_landmarks.landmark[mp_hands.HandLandmark.WRIST].y
                wristz =
hand_landmarks.landmark[mp_hands.HandLandmark.WRIST].z
                ##THUMP
                ##Landmark1 : Thump cmc coordinates:
                XCMCthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_CMC].x
                YCMCthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_CMC].y
                ZCMCthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_CMC].z

                ##Landmark2 : Thump mcp coordinates:
                XMCPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_MCP].x
                YMCPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_MCP].y
                ZMCPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_MCP].z

                ##Landmark3 : Thump ip coordinates:
                XIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_IP].x
                YIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_IP].y
                ZIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_IP].z

                ##Landmark4 : Thump tip coordinates:
                XTIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].x
                YTIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].y
                ZTIPthumb =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP].z
                ##INDEX
                ##Landmark5 : Index finger mcp coordinates:
                XMCPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_M
CP].x
                YMCPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_M
CP].y
                ZMCPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_M
CP].y
                ##Landmark6 : Index finger pip coordinates:
                XPIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_P
IP].x
```

```
                YPIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_P
IP].y
                ZPIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_P
IP].z

                ##Landmark7 : Index finger dip coordinates:
                XDIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_D
IP].x
                YDIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_D
IP].y
                ZDIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_D
IP].z

                ##Landmark8 : Index finger tip coordinates:
                XTIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_T
IP].x
                YTIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_T
IP].y
                ZTIPindex =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_T
IP].z

                ##MIDDLE FINGER
                ##Landmark9 :  Middle finger mcp
coordinates:
                XMCPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
MCP].x
                YMCPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
MCP].y
                ZMCPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
MCP].z

                ##Landmark10 : Middle finger pip
coordinates:
                XPIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
PIP].x
                YPIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
PIP].y
```

```
                ZPIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
PIP].z

                ##Landmark11 : Middle finger dip
coordinates:
                XDIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
DIP].x
                YDIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
DIP].y
                ZDIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
DIP].z

                ##Landmark12 : Middle finger tip
coordinates:
                XTIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
TIP].x
                YTIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
TIP].y
                ZTIPmiddle =
hand_landmarks.landmark[mp_hands.HandLandmark.MIDDLE_FINGER_
TIP].z

                ##RING FINGER
                ##Landmark13 : Ring finger mcp coordinates:
                XMCPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_MC
P].x
                YMCPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_MC
P].y
                ZMCPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_MC
P].z

                ##Landmark14 : Ring finger pip coordinates:
                XPIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_PI
P].x
                YPIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_PI
P].y
                ZPIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_PI
P].z
```

```
                ##Landmark15 : Ring finger dip coordinates:
                XDIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_DI
P].x
                YDIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_DI
P].y
                ZDIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_DI
P].z

                ##Landmark16 : Ring finger tip coordinates:
                XTIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_TI
P].x
                YTIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_TI
P].y
                ZTIPring =
hand_landmarks.landmark[mp_hands.HandLandmark.RING_FINGER_TI
P].z

                ##PINKY
                ##Landmark17 : Pinky mcp coordinates:
                XMCPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_MCP].x
                YMCPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_MCP].y
                ZMCPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_MCP].z

                ##Landmark18 : Pinky pip coordinates:
                XPIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_PIP].x
                YPIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_PIP].y
                ZPIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_PIP].z

                ##Landmark19 : Pinky dip coordinates:
                XDIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_DIP].x
                YDIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_DIP].y
                ZDIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_DIP].z

                ##Landmark20 : Pinky tip coordinates:
                XTIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_TIP].x
```

```
                YTIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_TIP].y
                ZTIPpinky =
hand_landmarks.landmark[mp_hands.HandLandmark.PINKY_TIP].z



                wrist_vec = np.array([wristx, wristy,
wristz])
                thumb_cmc_vec = np.array([XCMCthumb,
YCMCthumb, ZCMCthumb])
                thumb_mcp_vec = np.array([XMCPthumb,
YMCPthumb, ZMCPthumb])
                thumb_ip_vec = np.array([XIPthumb, YIPthumb,
ZIPthumb])
                thumb_tip_vec = np.array([XTIPthumb,
YTIPthumb, ZTIPthumb])

                index_mcp_vec = np.array([XMCPindex,
YMCPindex, ZMCPindex])
                index_pip_vec = np.array([XPIPindex,
YPIPindex, ZPIPindex])
                index_dip_vec = np.array([XDIPindex,
YDIPindex, ZDIPindex])
                index_tip_vec = np.array([XTIPindex,
YTIPindex, ZTIPindex])

                middle_mcp_vec = np.array([XMCPmiddle,
YMCPmiddle, ZMCPmiddle])
                middle_pip_vec = np.array([XPIPmiddle,
YPIPmiddle, ZPIPmiddle])
                middle_dip_vec = np.array([XDIPmiddle,
YDIPmiddle, ZDIPmiddle])
                middle_tip_vec = np.array([XTIPmiddle,
YTIPmiddle, ZTIPmiddle])

                ring_mcp_vec = np.array([XMCPring, YMCPring,
ZMCPring])
                ring_pip_vec = np.array([XPIPring, YPIPring,
ZPIPring])
                ring_dip_vec = np.array([XDIPring, YDIPring,
ZDIPring])
                ring_tip_vec = np.array([XTIPring, YTIPring,
ZTIPring])

                pinky_mcp_vec = np.array([XMCPpinky,
YMCPpinky, ZMCPpinky])
                pinky_pip_vec = np.array([XPIPpinky,
YPIPpinky, ZPIPpinky])
                pinky_dip_vec = np.array([XDIPpinky,
YDIPpinky, ZDIPpinky])
```

```
                pinky_tip_vec = np.array([XTIPpinky,
YTIPpinky, ZTIPpinky])

                print(wrist_vec)
                list = []
                Calculations(wrist_vec, thumb_cmc_vec,
thumb_mcp_vec)
                list.append(Calculations(wrist_vec,
thumb_cmc_vec, thumb_mcp_vec))

                Calculations(thumb_cmc_vec, thumb_mcp_vec,
thumb_ip_vec)
                list.append(Calculations(thumb_cmc_vec,
thumb_mcp_vec, thumb_ip_vec))

                Calculations(thumb_mcp_vec, thumb_ip_vec,
thumb_tip_vec)
                list.append(Calculations(thumb_mcp_vec,
thumb_ip_vec, thumb_tip_vec))

                Calculations(wrist_vec, index_mcp_vec,
index_pip_vec)
                list.append(Calculations(wrist_vec,
index_mcp_vec, index_pip_vec))

                Calculations(index_mcp_vec, index_pip_vec,
index_dip_vec)
                list.append(Calculations(index_mcp_vec,
index_pip_vec, index_dip_vec))

                Calculations(index_pip_vec, index_dip_vec,
index_tip_vec)
                list.append(Calculations(index_pip_vec,
index_dip_vec, index_tip_vec))

                Calculations(wrist_vec, middle_mcp_vec,
middle_pip_vec)
                list.append(Calculations(wrist_vec,
middle_mcp_vec, middle_pip_vec))

                Calculations(middle_mcp_vec, middle_pip_vec,
middle_dip_vec)
                list.append(Calculations(middle_mcp_vec,
middle_pip_vec, middle_dip_vec))

                Calculations(middle_pip_vec, middle_dip_vec,
middle_tip_vec)
                list.append(Calculations(middle_pip_vec,
middle_dip_vec, middle_tip_vec))
```

```
                Calculations(wrist_vec, ring_mcp_vec,
ring_pip_vec)
                list.append(Calculations(wrist_vec,
ring_mcp_vec, ring_pip_vec))

                Calculations(ring_mcp_vec, ring_pip_vec,
ring_dip_vec)
                list.append(Calculations(ring_mcp_vec,
ring_pip_vec, ring_dip_vec))

                Calculations(ring_pip_vec, ring_dip_vec,
ring_tip_vec)
                list.append(Calculations(ring_pip_vec,
ring_dip_vec, ring_tip_vec))

                Calculations(wrist_vec, pinky_mcp_vec,
pinky_pip_vec)
                list.append(Calculations(wrist_vec,
pinky_mcp_vec, pinky_pip_vec))

                Calculations(pinky_mcp_vec, pinky_pip_vec,
pinky_dip_vec)
                list.append(Calculations(pinky_mcp_vec,
pinky_pip_vec, pinky_dip_vec))

                Calculations(pinky_pip_vec, pinky_dip_vec,
pinky_tip_vec)
                list.append(Calculations(pinky_pip_vec,
pinky_dip_vec, pinky_tip_vec))

                print(list)

                mp_drawing.draw_landmarks(
                    annotated_image, hand_landmarks,
mp_hands.HAND_CONNECTIONS)

                cv2.imwrite(
                    '/data6/annotated_image' + str(i) +
'.png', cv2.flip(annotated_image, 1))
                cv2.imshow('Hand Landmarks',
annotated_image)
                worksheet1.activate()

                worksheet1.write(row + c, 0, q)
                for col, data in enumerate(list):
                    worksheet1.write_row(row + c, col + 1,
data)


workbook.close()
```

e) Code for First Feature's Neural Network

```
import numpy as np
import keras
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import itertools

from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from ann_visualizer.visualize import ann_viz
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files
(x86)/Graphviz2.38/bin/'

df = pd.read_csv('C:/Users/lia/Desktop/Angles.csv',
header=None)
print(df.head)
X = np.array(df.loc[:, 1:191])
Y = np.array(df.loc[:, 0])
print(X.shape, Y.shape)
x_train, x_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2)
print(x_train.shape, x_test.shape)


def preprocess(x, y):
    x = tf.cast(x, tf.float32)
    y = tf.cast(y, tf.int64)

    return x, y


def create_dataset(xs, ys, n_classes=25):
    ys = tf.one_hot(ys, depth=n_classes)
    return tf.data.Dataset.from_tensor_slices((xs, ys)) \
        .map(preprocess) \
        .shuffle(len(ys)) \
        .batch(128)


train_dataset = create_dataset(x_train, y_train)
val_dataset = create_dataset(x_test, y_test)

model = keras.Sequential([
    keras.layers.InputLayer(input_shape=(190,)),
    keras.layers.Dense(units=256, activation='relu'),
    keras.layers.Dense(units=128, activation='relu'),
    keras.layers.Dense(units=25, activation='softmax')
])
```

```
model.compile(optimizer='adam',

loss=tf.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(
    train_dataset.repeat(),
    epochs=20,
    steps_per_epoch=1000,
    validation_data=val_dataset.repeat(),
    validation_steps=2
)
ann_viz(model, view=True, filename='network.gv',
title='First Feature Vector Neural Network')
# plot loss during training
plt.subplot(211)
plt.title('Train Loss of feature 1')
plt.plot(history.history['loss'], label='Train')
plt.plot(history.history['val_loss'], label='Test')
plt.ylabel('Loss')
plt.xlabel('Epochs')

plt.legend()
# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy of feature 1')
plt.plot(history.history['accuracy'], label='Train')
plt.plot(history.history['val_accuracy'], label='Test')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend()
plt.show()

predictions = model.predict(x_test, batch_size=10,
verbose=0)
rounded_predictions = np.argmax(predictions, axis=-1)


def get_number():
    while True:
        num = input("Pick a number that represents an entry
in test set: ")
        if num.isdigit():
            num = int(num)
            if 0 <= num <= 1000:
                return int(num)
        else:
            print("Try again...")


def predict(model, input, correct_label):
```

```
    class_names = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
                   'V', 'W', 'X', 'Y']
    prediction = model.predict(np.array([input]))
    predicted_class = class_names[np.argmax(prediction)]
    print('Correct class of entry No', num, ':',
class_names[correct_label], 'Predicted class of entry No',
num, ':', predicted_class)


num = get_number()
input = x_test[num]
label = y_test[num]
predict(model, input, label)

cm = confusion_matrix(y_true=y_test,
y_pred=rounded_predictions)


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else
"black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()


cm_plot_labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
                  'V', 'W', 'X', 'Y']
plt.figure()
plot_confusion_matrix(cm=cm, classes=cm_plot_labels,
title='Confusion Matrix of Feature 1')
```

f) Code for Second Feature's Neural Network

```
import numpy as np
import keras
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import itertools
import statistics
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from tensorflow.python.keras.models import save_model
import graphviz
import ann_visualizer
from ann_visualizer.visualize import ann_viz
import os
os.environ["PATH"] += os.pathsep + 'C:/Program Files
(x86)/Graphviz2.38/bin/'

df = pd.read_csv('C:/Users/lia/Desktop/NewAngles.csv',
header=None)
print(df.head)
X = np.array(df.loc[:, 1:16])
Y = np.array(df.loc[:, 0])
print(X.shape, Y.shape)
x_train, x_test, y_train, y_test = train_test_split(X, Y,
test_size=0.2)
print(x_train.shape, x_test.shape)
print(x_test[0])


def preprocess(x, y):
    x = tf.cast(x, tf.float32)
    y = tf.cast(y, tf.int64)

    return x, y


def create_dataset(xs, ys, n_classes=25):
    ys = tf.one_hot(ys, depth=n_classes)
    return tf.data.Dataset.from_tensor_slices((xs, ys)) \
        .map(preprocess) \
        .shuffle(len(ys)) \
        .batch(128)


train_dataset = create_dataset(x_train, y_train)
val_dataset = create_dataset(x_test, y_test)

model = keras.Sequential([
    keras.layers.InputLayer(input_shape=(15,)),
```

```
    keras.layers.Dense(units=256, activation='relu'),
    keras.layers.Dense(units=128, activation='relu'),
    keras.layers.Dense(units=25, activation='softmax')
])
model.compile(optimizer='adam',

loss=tf.losses.CategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(
    train_dataset.repeat(),
    epochs=20,
    steps_per_epoch=1000,
    validation_data=val_dataset.repeat(),
    validation_steps=2
)
ann_viz(model, view=True, filename='network1.gv',
title='Second Feature Vector Neural Network')
filepath = './saved_model'
save_model(model, filepath)
sum = 0
# plot loss during training
plt.subplot(211)
plt.title('Train Loss of Feature 2')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend()
# plot accuracy during training
plt.subplot(212)
plt.title('Accuracy of Feature 2')
plt.plot(history.history['accuracy'], label='train')
plt.plot(history.history['val_accuracy'], label='test')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend()
plt.show()

predictions = model.predict(x_test, batch_size=10,
verbose=0)
rounded_predictions = np.argmax(predictions, axis=-1)


def get_number():
    while True:
        num = input("Pick a number: ")
        if num.isdigit():
            num = int(num)
            if 0 <= num <= 1000:
                return int(num)
```

```
        else:
            print("Try again...")


def predict(model, input, correct_label):
    class_names = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
                   'V', 'W', 'X', 'Y']
    prediction = model.predict(np.array([input]))
    predicted_class = class_names[np.argmax(prediction)]
    print('Correct class:', class_names[correct_label],
'Predicted class:', predicted_class)


num = get_number()
input = x_test[num]
label = y_test[num]
predict(model, input, label)

cm = confusion_matrix(y_true=y_test,
y_pred=rounded_predictions)


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Reds):
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else
"black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()


cm_plot_labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U',
                  'V', 'W', 'X', 'Y']
```

```
plot_confusion_matrix(cm=cm, classes=cm_plot_labels,
title='Confusion Matrix of Feature 2')
```

g) Code for 10 Trainings (1st Feature)

```
import numpy as np
import keras
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import itertools
import statistics
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

loop = 0
loss_lists = []
val_loss_lists = []
while loop < 10:
    print(loop)
    df = pd.read_csv('C:/Users/lia/Desktop/Angles.csv',
header=None)
    X = np.array(df.loc[:, 1:191])
    Y = np.array(df.loc[:, 0])
    print(X.shape, Y.shape)
    x_train, x_test, y_train, y_test = train_test_split(X,
Y, test_size=0.2)


    def preprocess(x, y):
        x = tf.cast(x, tf.float32)
        y = tf.cast(y, tf.int64)

        return x, y


    def create_dataset(xs, ys, n_classes=25):
        ys = tf.one_hot(ys, depth=n_classes)
        return tf.data.Dataset.from_tensor_slices((xs, ys))
\
            .map(preprocess) \
            .shuffle(len(ys)) \
            .batch(128)


    train_dataset = create_dataset(x_train, y_train)
    val_dataset = create_dataset(x_test, y_test)

    model = keras.Sequential([
        keras.layers.Dense(units=15, activation='relu'),
```

```
        keras.layers.Dense(units=256, activation='relu'),
        keras.layers.Dense(units=128, activation='relu'),
        keras.layers.Dense(units=25, activation='softmax')
    ])
    model.compile(optimizer='adam',

loss=tf.losses.CategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy']
                  )

    history = model.fit(
        train_dataset.repeat(),
        epochs=20,
        steps_per_epoch=100,
        validation_data=val_dataset.repeat(),
        validation_steps=2

    )

    predictions = model.predict(x_test, batch_size=10,
verbose=0)
    rounded_predictions = np.argmax(predictions, axis=-1)

    cm = confusion_matrix(y_true=y_test,
y_pred=rounded_predictions)


    def plot_confusion_matrix(cm, classes,
                              normalize=False,
                              title='Confusion matrix',
                              cmap=plt.cm.Reds):
        plt.imshow(cm, interpolation='nearest', cmap=cmap)
        plt.title(title)
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        plt.xticks(tick_marks, classes, rotation=45)
        plt.yticks(tick_marks, classes)

        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
            plt.text(j, i, cm[i, j],
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else
"black")

        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        '''plt.show()'''
```

```
    cm_plot_labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G',
'H', 'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
                      'U',
                      'V', 'W', 'X', 'Y']
    plot_confusion_matrix(cm=cm, classes=cm_plot_labels,
title='Confusion Matrix')

    loss = (history.history['loss'])
    val_loss = (history.history['val_loss'])

    loss_lists.append(loss)
    val_loss_lists.append(val_loss)
    loop = loop + 1


loss1 = np.array(loss_lists[0])
loss2 = np.array(loss_lists[1])
loss3 = np.array(loss_lists[2])
loss4 = np.array(loss_lists[3])
loss5 = np.array(loss_lists[4])
loss6 = np.array(loss_lists[5])
loss7 = np.array(loss_lists[6])
loss8 = np.array(loss_lists[7])
loss9 = np.array(loss_lists[8])
loss10 = np.array(loss_lists[9])

element1 = [loss1[0], loss2[0], loss3[0], loss4[0],
loss5[0], loss6[0], loss7[0], loss8[0], loss9[0], loss10[0]]
element2 = [loss1[1], loss2[1], loss3[1], loss4[1],
loss5[1], loss6[1], loss7[1], loss8[1], loss9[1], loss10[1]]
element3 = [loss1[2], loss2[2], loss3[2], loss4[2],
loss5[2], loss6[2], loss7[2], loss8[2], loss9[2], loss10[2]]
element4 = [loss1[3], loss2[3], loss3[3], loss4[3],
loss5[3], loss6[3], loss7[3], loss8[3], loss9[3], loss10[3]]
element5 = [loss1[4], loss2[4], loss3[4], loss4[4],
loss5[4], loss6[4], loss7[4], loss8[4], loss9[4], loss10[4]]
element6 = [loss1[5], loss2[5], loss3[5], loss4[5],
loss5[5], loss6[5], loss7[5], loss8[5], loss9[5], loss10[5]]
element7 = [loss1[6], loss2[6], loss3[6], loss4[6],
loss5[6], loss6[6], loss7[6], loss8[6], loss9[6], loss10[6]]
element8 = [loss1[7], loss2[7], loss3[7], loss4[7],
loss5[7], loss6[7], loss7[7], loss8[7], loss9[7], loss10[7]]
element9 = [loss1[8], loss2[8], loss3[8], loss4[8],
loss5[8], loss6[8], loss7[8], loss8[8], loss9[8], loss10[8]]
element10 = [loss1[9], loss2[9], loss3[9], loss4[9],
loss5[9], loss6[9], loss7[9], loss8[9], loss9[9], loss10[9]]
element11 = [loss1[10], loss2[10], loss3[10], loss4[10],
loss5[10], loss6[10], loss7[10], loss8[10], loss9[10],
loss10[10]]
```

```
element12 = [loss1[11], loss2[11], loss3[11], loss4[11],
loss5[11], loss6[11], loss7[11], loss8[11], loss9[11],
loss10[11]]
element13 = [loss1[12], loss2[12], loss3[12], loss4[12],
loss5[12], loss6[12], loss7[12], loss8[12], loss9[12],
loss10[12]]
element14 = [loss1[13], loss2[13], loss3[13], loss4[13],
loss5[13], loss6[13], loss7[13], loss8[13], loss9[13],
loss10[13]]
element15 = [loss1[14], loss2[14], loss3[14], loss4[14],
loss5[14], loss6[14], loss7[14], loss8[14], loss9[14],
loss10[14]]
element16 = [loss1[15], loss2[15], loss3[15], loss4[15],
loss5[15], loss6[15], loss7[15], loss8[15], loss9[15],
loss10[15]]
element17 = [loss1[16], loss2[16], loss3[16], loss4[16],
loss5[16], loss6[16], loss7[16], loss8[16], loss9[16],
loss10[16]]
element18 = [loss1[17], loss2[17], loss3[17], loss4[17],
loss5[17], loss6[17], loss7[17], loss8[17], loss9[17],
loss10[17]]
element19 = [loss1[18], loss2[18], loss3[18], loss4[18],
loss5[18], loss6[18], loss7[18], loss8[18], loss9[18],
loss10[18]]
element20 = [loss1[19], loss2[19], loss3[19], loss4[19],
loss5[19], loss6[19], loss7[19], loss8[19], loss9[19],
loss10[19]]


loss_elements = [element1, element2, element3, element4,
element5, element6, element7, element8, element9, element10,
element11, element12, element13, element14, element15,
element16, element17, element18, element19, element20]
print(loss_elements)

val_loss1 = np.array(val_loss_lists[0])
val_loss2 = np.array(val_loss_lists[1])
val_loss3 = np.array(val_loss_lists[2])
val_loss4 = np.array(val_loss_lists[3])
val_loss5 = np.array(val_loss_lists[4])
val_loss6 = np.array(val_loss_lists[5])
val_loss7 = np.array(val_loss_lists[6])
val_loss8 = np.array(val_loss_lists[7])
val_loss9 = np.array(val_loss_lists[8])
val_loss10 = np.array(val_loss_lists[9])

val_element1 = [val_loss1[0], val_loss2[0], val_loss3[0],
val_loss4[0], val_loss5[0], val_loss6[0], val_loss7[0],
val_loss8[0], val_loss9[0], val_loss10[0]]
```

```
val_element2 = [val_loss1[1], val_loss2[1], val_loss3[1],
val_loss4[1], val_loss5[1], val_loss6[1], val_loss7[1],
val_loss8[1], val_loss9[1], val_loss10[1]]
val_element3 = [val_loss1[2], val_loss2[2], val_loss3[2],
val_loss4[2], val_loss5[2], val_loss6[2], val_loss7[2],
val_loss8[2], val_loss9[2], val_loss10[2]]
val_element4 = [val_loss1[3], val_loss2[3], val_loss3[3],
val_loss4[3], val_loss5[3], val_loss6[3], val_loss7[3],
val_loss8[3], val_loss9[3], val_loss10[3]]
val_element5 = [val_loss1[4], val_loss2[4], val_loss3[4],
val_loss4[4], val_loss5[4], val_loss6[4], val_loss7[4],
val_loss8[4], val_loss9[4], val_loss10[4]]
val_element6 = [val_loss1[5], val_loss2[5], val_loss3[5],
val_loss4[5], val_loss5[5], val_loss6[5], val_loss7[5],
val_loss8[5], val_loss9[5], val_loss10[5]]
val_element7 = [val_loss1[6], val_loss2[6], val_loss3[6],
val_loss4[6], val_loss5[6], val_loss6[6], val_loss7[6],
val_loss8[6], val_loss9[6], val_loss10[6]]
val_element8 = [val_loss1[7], val_loss2[7], val_loss3[7],
val_loss4[7], val_loss5[7], val_loss6[7], val_loss7[7],
val_loss8[7], val_loss9[7], val_loss10[7]]
val_element9 = [val_loss1[8], val_loss2[8], val_loss3[8],
val_loss4[8], val_loss5[8], val_loss6[8], val_loss7[8],
val_loss8[8], val_loss9[8], val_loss10[8]]
val_element10 = [val_loss1[9], val_loss2[9], val_loss3[9],
val_loss4[9], val_loss5[9], val_loss6[9], val_loss7[9],
val_loss8[9], val_loss9[9], val_loss10[9]]
val_element11 = [val_loss1[10], val_loss2[10],
val_loss3[10], val_loss4[10], val_loss5[10], val_loss6[10],
val_loss7[10], val_loss8[10], val_loss9[10], val_loss10[10]]
val_element12 = [val_loss1[11], val_loss2[11],
val_loss3[11], val_loss4[11], val_loss5[11], val_loss6[11],
val_loss7[11], val_loss8[11], val_loss9[11], val_loss10[11]]
val_element13 = [val_loss1[12], val_loss2[12],
val_loss3[12], val_loss4[12], val_loss5[12], val_loss6[12],
val_loss7[12], val_loss8[12], val_loss9[12], val_loss10[12]]
val_element14 = [val_loss1[13], val_loss2[13],
val_loss3[13], val_loss4[13], val_loss5[13], val_loss6[13],
val_loss7[13], val_loss8[13], val_loss9[13], val_loss10[13]]
val_element15 = [val_loss1[14], val_loss2[14],
val_loss3[14], val_loss4[14], val_loss5[14], val_loss6[14],
val_loss7[14], val_loss8[14], val_loss9[14], val_loss10[14]]
val_element16 = [val_loss1[15], val_loss2[15],
val_loss3[15], val_loss4[15], val_loss5[15], val_loss6[15],
val_loss7[15], val_loss8[15], val_loss9[15], val_loss10[15]]
val_element17 = [val_loss1[16], val_loss2[16],
val_loss3[16], val_loss4[16], val_loss5[16], val_loss6[16],
val_loss7[16], val_loss8[16], val_loss9[16], val_loss10[16]]
val_element18 = [val_loss1[17], val_loss2[17],
val_loss3[17], val_loss4[17], val_loss5[17], val_loss6[17],
val_loss7[17], val_loss8[17], val_loss9[17], val_loss10[17]]
```

```
val_element19 = [val_loss1[18], val_loss2[18],
val_loss3[18], val_loss4[18], val_loss5[18], val_loss6[18],
val_loss7[18], val_loss8[18], val_loss9[18], val_loss10[18]]
val_element20 = [val_loss1[19], val_loss2[19],
val_loss3[19], val_loss4[19], val_loss5[19], val_loss6[19],
val_loss7[19], val_loss8[19], val_loss9[19], val_loss10[19]]

val_loss_elements = [val_element1, val_element2,
val_element3, val_element4, val_element5, val_element6,
val_element7, val_element8, val_element9, val_element10,
val_element11, val_element12, val_element13, val_element14,
val_element15, val_element16, val_element17, val_element18,
val_element19, val_element20]
print(val_loss_elements)

final_loss =
loss1+loss2+loss3+loss4+loss5+loss6+loss7+loss8+loss9+loss10
final_val_loss =
val_loss1+val_loss2+val_loss3+val_loss4+loss5+val_loss6+val_
loss7+val_loss8+val_loss9+val_loss10

print(final_loss, final_val_loss)
av_loss_list = []
for value in final_loss:
    av_loss = value/20
    av_loss_list.append(av_loss)
av_val_loss_list = []
for value in final_val_loss:
    av_val_loss = value/20
    av_val_loss_list.append(av_val_loss)

plt.subplot(211)
plt.title('Average Loss of Feature 1 for 10 trainings')
plt.plot(av_loss_list, label='Average Train Loss')
plt.plot(av_val_loss_list, label='Average Validation Loss')
plt.ylabel('Average Loss per Epoch')
plt.xlabel('Epochs')
plt.legend()

st_dev_loss_list = []
for values in loss_elements:
    st_dev_loss = statistics.stdev(values)
    st_dev_loss_list.append(st_dev_loss)

st_dev_val_loss_list = []
for values in val_loss_elements:
    st_dev_val_loss = statistics.stdev(values)
    st_dev_val_loss_list.append(st_dev_val_loss)

plt.subplot(212)
```

```
plt.title('Standard Deviation of Feature 1 for 10
trainings')
plt.plot(st_dev_loss_list, label='Standard Deviation of
Train Loss')
plt.plot(st_dev_val_loss_list, label='Standard Deviation of
Validation Loss')
plt.ylabel('Standard Deviation per Epoch')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```

h) Code for 10 Trainings (2nd Feature)

```
import numpy as np
import keras
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
import itertools
import statistics
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix

loop = 0
loss_lists = []
val_loss_lists = []
while loop < 10:
    print(loop)
    df = pd.read_csv('C:/Users/lia/Desktop/NewAngles.csv',
header=None)
    print(df.head)
    X = np.array(df.loc[:, 1:16])
    Y = np.array(df.loc[:, 0])
    print(X.shape, Y.shape)
    x_train, x_test, y_train, y_test = train_test_split(X,
Y, test_size=0.2)
    print(x_train.shape, x_test.shape)


    def preprocess(x, y):
        x = tf.cast(x, tf.float32)
        y = tf.cast(y, tf.int64)

        return x, y


    def create_dataset(xs, ys, n_classes=25):
        ys = tf.one_hot(ys, depth=n_classes)
        return tf.data.Dataset.from_tensor_slices((xs, ys))
\
            .map(preprocess) \
```

```
            .shuffle(len(ys)) \
            .batch(128)


    train_dataset = create_dataset(x_train, y_train)
    val_dataset = create_dataset(x_test, y_test)

    model = keras.Sequential([
        keras.layers.Dense(units=15, activation='relu'),
        keras.layers.Dense(units=256, activation='relu'),
        keras.layers.Dense(units=128, activation='relu'),
        keras.layers.Dense(units=25, activation='softmax')
    ])
    model.compile(optimizer='adam',

loss=tf.losses.CategoricalCrossentropy(from_logits=True),
                  metrics=['accuracy']
                  )

    history = model.fit(
        train_dataset.repeat(),
        epochs=20,
        steps_per_epoch=100,
        validation_data=val_dataset.repeat(),
        validation_steps=2

    )

    predictions = model.predict(x_test, batch_size=10,
verbose=0)
    rounded_predictions = np.argmax(predictions, axis=-1)

    cm = confusion_matrix(y_true=y_test,
y_pred=rounded_predictions)


    def plot_confusion_matrix(cm, classes,
                              normalize=False,
                              title='Confusion matrix',
                              cmap=plt.cm.Reds):
        plt.imshow(cm, interpolation='nearest', cmap=cmap)
        plt.title(title)
        plt.colorbar()
        tick_marks = np.arange(len(classes))
        plt.xticks(tick_marks, classes, rotation=45)
        plt.yticks(tick_marks, classes)

        thresh = cm.max() / 2.
        for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
            plt.text(j, i, cm[i, j],
```

```
                    horizontalalignment="center",
                    color="white" if cm[i, j] > thresh else
"black")

        plt.tight_layout()
        plt.ylabel('True label')
        plt.xlabel('Predicted label')
        '''plt.show()'''


    cm_plot_labels = ['A', 'B', 'C', 'D', 'E', 'F', 'G',
'H', 'I', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T',
                      'U',
                      'V', 'W', 'X', 'Y']
    plot_confusion_matrix(cm=cm, classes=cm_plot_labels,
title='Confusion Matrix')

    loss = (history.history['loss'])
    val_loss = (history.history['val_loss'])

    loss_lists.append(loss)
    val_loss_lists.append(val_loss)
    loop = loop + 1


loss1 = np.array(loss_lists[0])
loss2 = np.array(loss_lists[1])
loss3 = np.array(loss_lists[2])
loss4 = np.array(loss_lists[3])
loss5 = np.array(loss_lists[4])
loss6 = np.array(loss_lists[5])
loss7 = np.array(loss_lists[6])
loss8 = np.array(loss_lists[7])
loss9 = np.array(loss_lists[8])
loss10 = np.array(loss_lists[9])

element1 = [loss1[0], loss2[0], loss3[0], loss4[0],
loss5[0], loss6[0], loss7[0], loss8[0], loss9[0], loss10[0]]
element2 = [loss1[1], loss2[1], loss3[1], loss4[1],
loss5[1], loss6[1], loss7[1], loss8[1], loss9[1], loss10[1]]
element3 = [loss1[2], loss2[2], loss3[2], loss4[2],
loss5[2], loss6[2], loss7[2], loss8[2], loss9[2], loss10[2]]
element4 = [loss1[3], loss2[3], loss3[3], loss4[3],
loss5[3], loss6[3], loss7[3], loss8[3], loss9[3], loss10[3]]
element5 = [loss1[4], loss2[4], loss3[4], loss4[4],
loss5[4], loss6[4], loss7[4], loss8[4], loss9[4], loss10[4]]
element6 = [loss1[5], loss2[5], loss3[5], loss4[5],
loss5[5], loss6[5], loss7[5], loss8[5], loss9[5], loss10[5]]
element7 = [loss1[6], loss2[6], loss3[6], loss4[6],
loss5[6], loss6[6], loss7[6], loss8[6], loss9[6], loss10[6]]
```

```
element8 = [loss1[7], loss2[7], loss3[7], loss4[7],
loss5[7], loss6[7], loss7[7], loss8[7], loss9[7], loss10[7]]
element9 = [loss1[8], loss2[8], loss3[8], loss4[8],
loss5[8], loss6[8], loss7[8], loss8[8], loss9[8], loss10[8]]
element10 = [loss1[9], loss2[9], loss3[9], loss4[9],
loss5[9], loss6[9], loss7[9], loss8[9], loss9[9], loss10[9]]
element11 = [loss1[10], loss2[10], loss3[10], loss4[10],
loss5[10], loss6[10], loss7[10], loss8[10], loss9[10],
loss10[10]]
element12 = [loss1[11], loss2[11], loss3[11], loss4[11],
loss5[11], loss6[11], loss7[11], loss8[11], loss9[11],
loss10[11]]
element13 = [loss1[12], loss2[12], loss3[12], loss4[12],
loss5[12], loss6[12], loss7[12], loss8[12], loss9[12],
loss10[12]]
element14 = [loss1[13], loss2[13], loss3[13], loss4[13],
loss5[13], loss6[13], loss7[13], loss8[13], loss9[13],
loss10[13]]
element15 = [loss1[14], loss2[14], loss3[14], loss4[14],
loss5[14], loss6[14], loss7[14], loss8[14], loss9[14],
loss10[14]]
element16 = [loss1[15], loss2[15], loss3[15], loss4[15],
loss5[15], loss6[15], loss7[15], loss8[15], loss9[15],
loss10[15]]
element17 = [loss1[16], loss2[16], loss3[16], loss4[16],
loss5[16], loss6[16], loss7[16], loss8[16], loss9[16],
loss10[16]]
element18 = [loss1[17], loss2[17], loss3[17], loss4[17],
loss5[17], loss6[17], loss7[17], loss8[17], loss9[17],
loss10[17]]
element19 = [loss1[18], loss2[18], loss3[18], loss4[18],
loss5[18], loss6[18], loss7[18], loss8[18], loss9[18],
loss10[18]]
element20 = [loss1[19], loss2[19], loss3[19], loss4[19],
loss5[19], loss6[19], loss7[19], loss8[19], loss9[19],
loss10[19]]


loss_elements = [element1, element2, element3, element4,
element5, element6, element7, element8, element9, element10,
element11, element12, element13, element14, element15,
element16, element17, element18, element19, element20]
print(loss_elements)

val_loss1 = np.array(val_loss_lists[0])
val_loss2 = np.array(val_loss_lists[1])
val_loss3 = np.array(val_loss_lists[2])
val_loss4 = np.array(val_loss_lists[3])
val_loss5 = np.array(val_loss_lists[4])
val_loss6 = np.array(val_loss_lists[5])
val_loss7 = np.array(val_loss_lists[6])
```

```
val_loss8 = np.array(val_loss_lists[7])
val_loss9 = np.array(val_loss_lists[8])
val_loss10 = np.array(val_loss_lists[9])

val_element1 = [val_loss1[0], val_loss2[0], val_loss3[0],
val_loss4[0], val_loss5[0], val_loss6[0], val_loss7[0],
val_loss8[0], val_loss9[0], val_loss10[0]]
val_element2 = [val_loss1[1], val_loss2[1], val_loss3[1],
val_loss4[1], val_loss5[1], val_loss6[1], val_loss7[1],
val_loss8[1], val_loss9[1], val_loss10[1]]
val_element3 = [val_loss1[2], val_loss2[2], val_loss3[2],
val_loss4[2], val_loss5[2], val_loss6[2], val_loss7[2],
val_loss8[2], val_loss9[2], val_loss10[2]]
val_element4 = [val_loss1[3], val_loss2[3], val_loss3[3],
val_loss4[3], val_loss5[3], val_loss6[3], val_loss7[3],
val_loss8[3], val_loss9[3], val_loss10[3]]
val_element5 = [val_loss1[4], val_loss2[4], val_loss3[4],
val_loss4[4], val_loss5[4], val_loss6[4], val_loss7[4],
val_loss8[4], val_loss9[4], val_loss10[4]]
val_element6 = [val_loss1[5], val_loss2[5], val_loss3[5],
val_loss4[5], val_loss5[5], val_loss6[5], val_loss7[5],
val_loss8[5], val_loss9[5], val_loss10[5]]
val_element7 = [val_loss1[6], val_loss2[6], val_loss3[6],
val_loss4[6], val_loss5[6], val_loss6[6], val_loss7[6],
val_loss8[6], val_loss9[6], val_loss10[6]]
val_element8 = [val_loss1[7], val_loss2[7], val_loss3[7],
val_loss4[7], val_loss5[7], val_loss6[7], val_loss7[7],
val_loss8[7], val_loss9[7], val_loss10[7]]
val_element9 = [val_loss1[8], val_loss2[8], val_loss3[8],
val_loss4[8], val_loss5[8], val_loss6[8], val_loss7[8],
val_loss8[8], val_loss9[8], val_loss10[8]]
val_element10 = [val_loss1[9], val_loss2[9], val_loss3[9],
val_loss4[9], val_loss5[9], val_loss6[9], val_loss7[9],
val_loss8[9], val_loss9[9], val_loss10[9]]
val_element11 = [val_loss1[10], val_loss2[10],
val_loss3[10], val_loss4[10], val_loss5[10], val_loss6[10],
val_loss7[10], val_loss8[10], val_loss9[10], val_loss10[10]]
val_element12 = [val_loss1[11], val_loss2[11],
val_loss3[11], val_loss4[11], val_loss5[11], val_loss6[11],
val_loss7[11], val_loss8[11], val_loss9[11], val_loss10[11]]
val_element13 = [val_loss1[12], val_loss2[12],
val_loss3[12], val_loss4[12], val_loss5[12], val_loss6[12],
val_loss7[12], val_loss8[12], val_loss9[12], val_loss10[12]]
val_element14 = [val_loss1[13], val_loss2[13],
val_loss3[13], val_loss4[13], val_loss5[13], val_loss6[13],
val_loss7[13], val_loss8[13], val_loss9[13], val_loss10[13]]
val_element15 = [val_loss1[14], val_loss2[14],
val_loss3[14], val_loss4[14], val_loss5[14], val_loss6[14],
val_loss7[14], val_loss8[14], val_loss9[14], val_loss10[14]]
```

```
val_element16 = [val_loss1[15], val_loss2[15],
val_loss3[15], val_loss4[15], val_loss5[15], val_loss6[15],
val_loss7[15], val_loss8[15], val_loss9[15], val_loss10[15]]
val_element17 = [val_loss1[16], val_loss2[16],
val_loss3[16], val_loss4[16], val_loss5[16], val_loss6[16],
val_loss7[16], val_loss8[16], val_loss9[16], val_loss10[16]]
val_element18 = [val_loss1[17], val_loss2[17],
val_loss3[17], val_loss4[17], val_loss5[17], val_loss6[17],
val_loss7[17], val_loss8[17], val_loss9[17], val_loss10[17]]
val_element19 = [val_loss1[18], val_loss2[18],
val_loss3[18], val_loss4[18], val_loss5[18], val_loss6[18],
val_loss7[18], val_loss8[18], val_loss9[18], val_loss10[18]]
val_element20 = [val_loss1[19], val_loss2[19],
val_loss3[19], val_loss4[19], val_loss5[19], val_loss6[19],
val_loss7[19], val_loss8[19], val_loss9[19], val_loss10[19]]

val_loss_elements = [val_element1, val_element2,
val_element3, val_element4, val_element5, val_element6,
val_element7, val_element8, val_element9, val_element10,
val_element11, val_element12, val_element13, val_element14,
val_element15, val_element16, val_element17, val_element18,
val_element19, val_element20]
print(val_loss_elements)

final_loss =
loss1+loss2+loss3+loss4+loss5+loss6+loss7+loss8+loss9+loss10
final_val_loss =
val_loss1+val_loss2+val_loss3+val_loss4+loss5+val_loss6+val_
loss7+val_loss8+val_loss9+val_loss10

print(final_loss, final_val_loss)
av_loss_list = []
for value in final_loss:
    av_loss = value/10
    av_loss_list.append(av_loss)
av_val_loss_list = []
for value in final_val_loss:
    av_val_loss = value/10
    av_val_loss_list.append(av_val_loss)

plt.subplot(211)
plt.title('Average Loss of Feature 2 for 10 Trainings')
plt.plot(av_loss_list, label='Average Train Loss')
plt.plot(av_val_loss_list, label='Average Validation Loss')
plt.ylabel('Average Loss per Epoch')
plt.xlabel('Epochs')
plt.legend()

st_dev_loss_list = []
for values in loss_elements:
    st_dev_loss = statistics.stdev(values)
```

```
    st_dev_loss_list.append(st_dev_loss)

st_dev_val_loss_list = []
for values in val_loss_elements:
    st_dev_val_loss = statistics.stdev(values)
    st_dev_val_loss_list.append(st_dev_val_loss)

plt.subplot(212)
plt.title('Standard Deviation of Feature 2 for 10
Trainings')
plt.plot(st_dev_loss_list, label='Standard Deviation of
Train Loss')
plt.plot(st_dev_val_loss_list, label='Standard Deviation of
Validation Loss')
plt.ylabel('Standard Deviation per Epoch')
plt.xlabel('Epochs')
plt.legend()
plt.show()
```