UNIVERSITY *of* MARYLAND
EASTERN SHORE
™

SCHOOL *of* BUSINESS AND TECHNOLOGY
Department of Engineering and Aviation Sciences

Design of Sign AI

A Smart ASL Translator

Josheb Dayrit

Advisor: Dr. Zhang

Spring 2020

List of Contents

List of Tables

List of Tables

Abstract

By the end of the project, summarize the project into short text and put here.

1. Introduction

The following project seeks to develop an ASL (American Sign Language) translation tool for deaf persons. In addition, it will be hosted on a microcomputer, like the 32-bit Raspberry Pi. Finally, a deep learning model will be trained to recognize ASL through pre-processed sEMG (surface electromyography) data collected by sEMG electrodes. Needless to say, each part in the development process is crucial – doubly so for the deep learning model, which is in charge of gesture classification and translation.

As far as additional hardware is concerned, there will be an LCD providing the user with an interface on which spoken words that are converted to text are displayed. Communication is a two-way street, after all; the LCD enables the user to not only express, but to also understand. The end product is expected to offer its deaf users a "give" as well as a "receive" functionality. That is, it facilitates two-sided conversation, where the user (who is assumed to be deaf) is able to speak as well as be spoken to.

For Sign AI, ease of use as well as translation accuracy are of utmost importance. Ease of use is incorporated in the design process through accessories which increase portability and convenience by virtue of being small, little, and therefore easy-to-carry. Meanwhile, translation accuracy hinges on the deep learning model and its ability to differentiate between different types of EMG data. In the end, what Sign AI aims to offer is a portable and reliable ASL translator that creates a user-friendly experience by automating the translation process. Everything is done in the backend and little input is required from the user.

1.1 Motivation

Different means of communication have existed since time immemorial. While speech stands as the most common mode of communication, people are also able to communicate through writing. It is no surprise that unique modes of communication are as plenty as they are ubiquitous. In technology, computers use a scheme of 0s and 1s to perform complicated tasks and display the outputs; in nature, all scores of animals are able to encode meaning in the noises they produce. For modern humans, speech is intrinsic to communication. However, the two are not at all exclusive to each other. In a paper titled "The Origin of Human Multi-Model Communication," two researchers (Stephen Levinson and Judith Holler) from the Max Planck Institute for Psycholinguistics and Donders Centre for Brain, Cognition and Behavior stated that "the modern human communication system is, on a biological time scale, a recent innovation" [1]. Citing the Gestural Theory of Language Origin, Levinson and Holler believed that the initial humans (the "cavemen," or Homo troglodytes) must have developed and employed a gesture-based language. They also believed that changes in anatomy triggered a gradual but inevitable shift from gesture-based communication to vocal communication in humans.

In a society that now communicates (primarily) through speech, it will be difficult for deaf persons to assimilate – and, more importantly, interact – with their non-deaf peers. According to the Gallaudet Research Institute, in year 2006, "nearly 10,000,000 persons (Americans) are hard of hearing and close to 1,000,000 are functionally deaf" [2]. This is also corroborated by more recent (2012) statistics from the Center of Disease Control and Prevention (CDC), which state that "approximately 15% of American adults (37.5 million) aged 18 and over report some trouble hearing" [3]. While the CDC clarifies that "the overall annual prevalence of hearing loss (has) dropped slightly" [3] with the passing of time (from 16% in 1999 to 14% in 2012), additional

findings from the National Deaf Center on Post-Secondary Outcomes reveal an alarming truth about deafness: “In 2014, only 48% of deaf people were employed, compared to 72% of hearing people” [4].

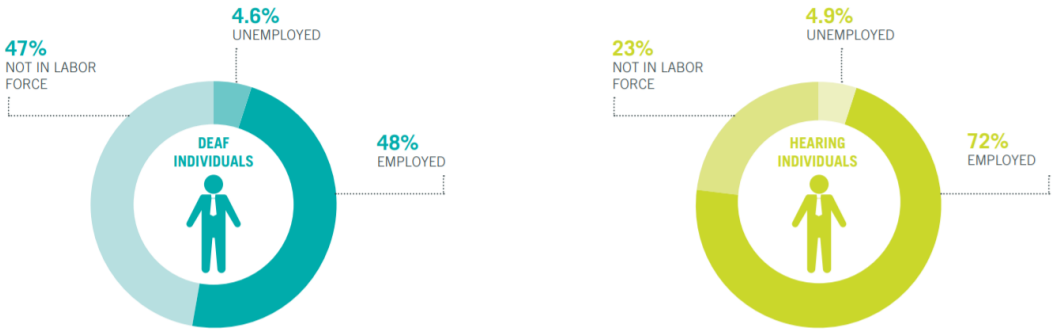


Figure 1: Rates of employment in 2014, according to the National Deaf Center on Post-Secondary Outcomes

Unfortunately, disparities in opportunities and (to some extent) outcomes are due in large part to physical disabilities, as shown by the countless statistics on deaf people. Since deaf people are not capable of speech, they choose to communicate through gesture-based languages such as ASL (American Sign Language). Because of this, however, prospects for work are limited for deaf people. The harsh reality is that many employers are unwilling to hire deaf employees. While the ADA (American Discrimination Act) outlaws discriminatory practices based on race, gender, or disability, it does not prevent them entirely. In fact, in an American study which closely scrutinized successful discrimination lawsuits made against companies, the EEOC (Equal Employment Opportunity Commission) concluded that “young people who are deaf or hard of hearing may be advised to anticipate some resistance from employers” [5].

As outrageous as it sounds, even if a deaf candidate applies for a position while possessing qualifications that meet or exceed a company’s standards, the factor that determines their employment is whether or not they require excessive accommodation. From a corporate

perspective, hiring employees who will only place undue burden on a company and its resources is generally not worth the cost. This idea also aligns with the EEOC's secondary conclusion, which states that disabled applicants who "articulate well their qualifications" and provide long-term work-arounds or solutions to their disabilities are "more likely than others to find success in the world of work."

One solution that Sign AI brings to the table is the function of a basic, real-time, AI-based ASL translation tool. The inability to speak properly is one disadvantage of deaf persons that either prevents them from fully pursuing specific career paths or leads them to be unjustifiably discriminated against in their search for employment. This inability stems from the fact that speech is a product of both the ears and the mouth; the mouth produces the sounds, while the ears regulate them. Without coordination from both the mouth and ears, it becomes increasingly difficult to produce intelligible speech. Many non-deaf employers simply do not have the time to learn sign language for their deaf applicants. In the hiring process, most are likely to turn their attention to the non-deaf candidates, even if the deaf candidates have the necessary qualifications. Such discrimination is, by all accounts, unfair. However, there is some wisdom to be found in it, even if morally questionable. All businesses are utilitarian in that their goal is to maximize productivity while minimizing expenditure of any sort (human, financial, etc.). If a prospective employee demands more from a company than their fellow competitors, that company will likely settle for those competitors in hopes of making things easier on themselves. While Sign AI cannot shift this mindset, it can remedy, to an extent, the one disadvantage encumbering deaf persons when competing with their non-deaf peers.

As stated before, while deaf people face many barriers of entry regarding employment, communication is, by far, the most pressing issue. In the workplace, collaboration among

employees is important. Typically, company projects are completed not by a single person, but by teams of people making collective contributions. A pre-requisite to effective collaboration is effective communication, especially in the nascent stages of a project, during which first impressions are gathered, and ideas are put forth for evaluation by others. For deaf persons, communication comes in the form of sign language; however, for non-deaf persons, this is not the case. Deaf persons have a hard time expressing their ideas to non-deaf persons, because sign language is fundamentally different to spoken language. To bridge the gap between sign language and spoken language, a linguistic interface is needed between the two languages to translate one to the other. Sign AI strives to fulfill this role through a trained AI, a TTS (text-to-speech) feature that will speak out the meaning of various ASL gestures, and an STT (speech-to-text) feature that will translate from spoken word back to text.

1.2 Objective

The objective of Sign AI is to facilitate basic conversations between the deaf and non-deaf.

1.3 Design Requirements

- The final product is a gesture-based translator, providing translation for each gesture performed.
- The final product can translate a total of 100 gestures.
- The final product can differentiate between gestures with an accuracy of 70-90 percent.
- The final product will have gesture-to-speech and speech-to-text capabilities.
- The final product will have an LCD to display speech-to-text outputs.

1.4 Design Constraints

Although design constraints are few in number due to the "hands-off" nature of an AI project, undoubtedly, they do exist. One constraint lies in the ASL language itself, while others in hardware and cost.

- Some gestures are not arm-based or hand-based, which means that they cannot be measured by muscle sensors. In ASL, pronouns such as "he," "she," or "they" are expressed through a singular gesture: pointing a finger. With the current setup of sensors, there is no way to determine if a gesture meant for a pronoun is referring to a "he," "she," or "they."
- Since most resources for the project are located on the cloud, connection to the Internet is required for operation.
- Regarding expenses, the project will cost $300 in total.

1.5 Design Method

It should be noted that the classification process hinges upon not only the accuracy of the neural network (AI), but also the reliability of the data that is being used to train it. This data originates from the sensors, and the particular brand of sensors that will be implemented is the MyoWare Muscle Sensor developed by Advancer Technologies. The definition of "reliable data" changes from project to project. In the case of Sign AI, however, it is data that can capture via EMG measurements the disparate characteristics of each gesture.

Naturally, in a project that is wholly dependent on data, the first step is to collect it. Then comes the pre-processing stage, where EMG data is formatted in a way that can be passed on to a neural network for training. Data collection is expected to be a major bottleneck in the development of Sign AI, as it is a time-consuming endeavor. Not to mention, the tasks of collecting data and training the neural network cannot be pipelined because the neural network takes in the collected data as an input. As a solution, pseudo-data will be randomly generated in place of real sensor data.

Before the pseudo-data is generated, however, a plan has to be drafted relating to how it will be implemented into the project. To recap, the end goal is to develop a neural network that can differentiate between various types of sensor data. Considering this, the pseudo-data must represent a wide range of data. One idea was to plot a sample of points on an x-y graph, with each bounded by a pre-determined y-limit. First, the graphs will be generated by a script; then, they will be formatted accordingly for use of the neural network. These procedures encompass the second step.

The third step would be the design of the neural network itself. Architectures for neural networks are plenty, and some even specialize in solving niche problems. For example, there exists the recurrent neural network (or RNN, for short) which is used for pattern recognition is text and speech. When it comes to image processing, the ConvNet (or convolutional neural network) was developed to fulfill the need. Due to its universal application in the realm of image processing, the standard ConvNet design (found on the TensorFlow website) will be used as a base by the Sign AI. Justifications on why a ConvNet is being used can be found in the implementation plan. Following the third step is the fourth, which is exporting the finalized neural network on a Raspberry Pi and assembling the necessary hardware for the gesture-to-speech and speech-to-text functionalities.

2. Project Description

2.1 System Description

The system has 3 main parts: the neural network, the sensors, and the communication features. The neural network is the crux of the system, as it is for all intents and purposes the translator itself. Working in tandem with the neural network is the sensors, which feed data to it. This data is processed by the neural network to determine which gesture is being performed by the user. Lastly, there are the communication features in the form of gesture-to-speech and speech-to-text. As stated before, the Sign AI seeks to provide deaf users with the capability to speak as well as be spoken to. This means that the Sign AI must not only translate the gestures, but also develop a method of allowing deaf user to understand spoken word; one way this can be done is through a speech-to-text functionality.

2.2 System Diagram

The system diagram is displayed in Figure 2. To begin, the elements underneath the Raspberry Pi are the software programs installed on it. The hardware elements on the left (i.e. the microphone and the sensors) provide the Raspberry Pi with inputs, while the hardware elements on the right (i.e. the speaker and LCD display) actualize the outputs.

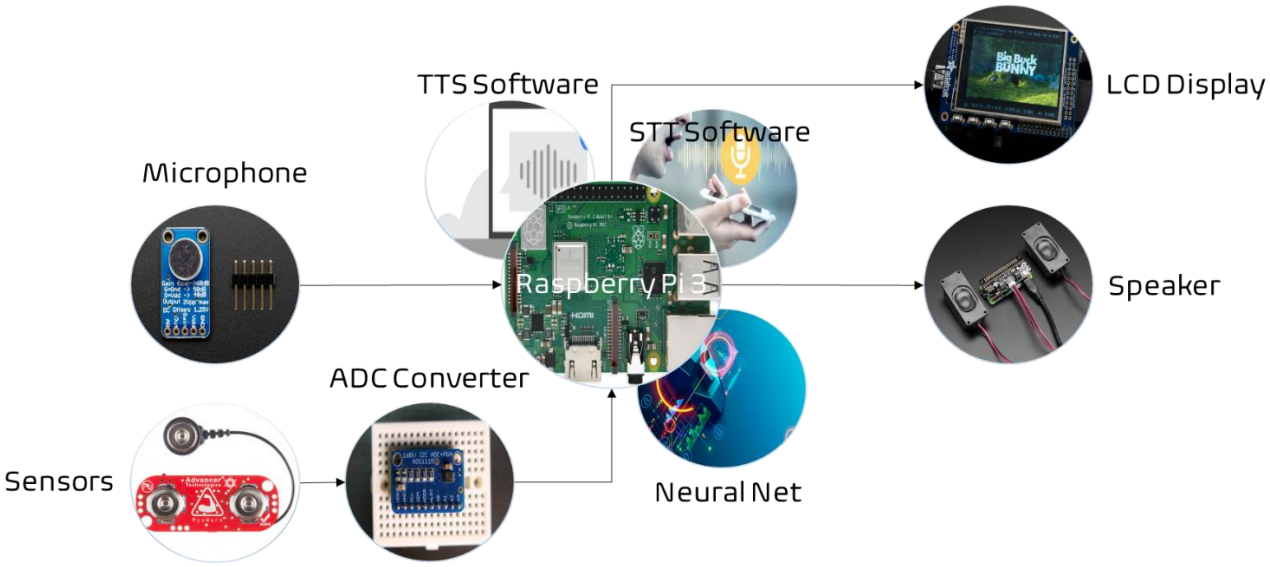


Fig. 2. The system diagram

2.3 System Functions

The system diagram can be decomposed into 3 separate sub-systems: the sensor-to-Pi subsystem, mic-to-display subsystem, and AI-to-speaker subsystem. Each subsystem, when put together, is responsible for facilitating the flow of data throughout the overall system. The sub-systems are explained in more detail in the coming pages.

Unlike the Arduino, the Raspberry Pi 3 does not have an innate method of processing analog signals. In fact, it does not even have analog pins; every pin on a Raspberry Pi is for digital information. Therefore, an ADC converter is required to connect data coming from the MyoWare muscle sensors to the Raspberry Pi. Obviously, some data will be lost when the analog-to-digital conversion is performed. However, this is a necessary evil due to the limitations of the Raspberry Pi.

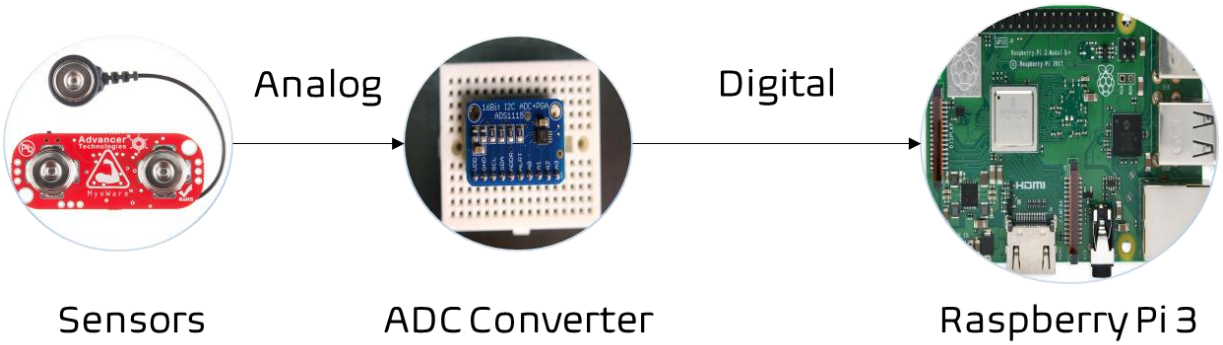


Fig. 3. The sensor-to-Pi subsystem

One of the primary functions of the Sign AI is to allow deaf users to be able to understand their non-deaf peers through a speech-to-text function. This speech-to-text function is achieved through the use of a microphone (which encodes audio input into a data file) and a speech-to-text software (which processes the data file in question and returns a text file). First, an audio file will flow from the microphone to the Raspberry Pi. Then, audio-to-text conversion happens once the speech-to-text software is allowed run by the Raspberry Pi; the output is a text file. In the final step, this text file will appear in the LCD display for the deaf user to read.

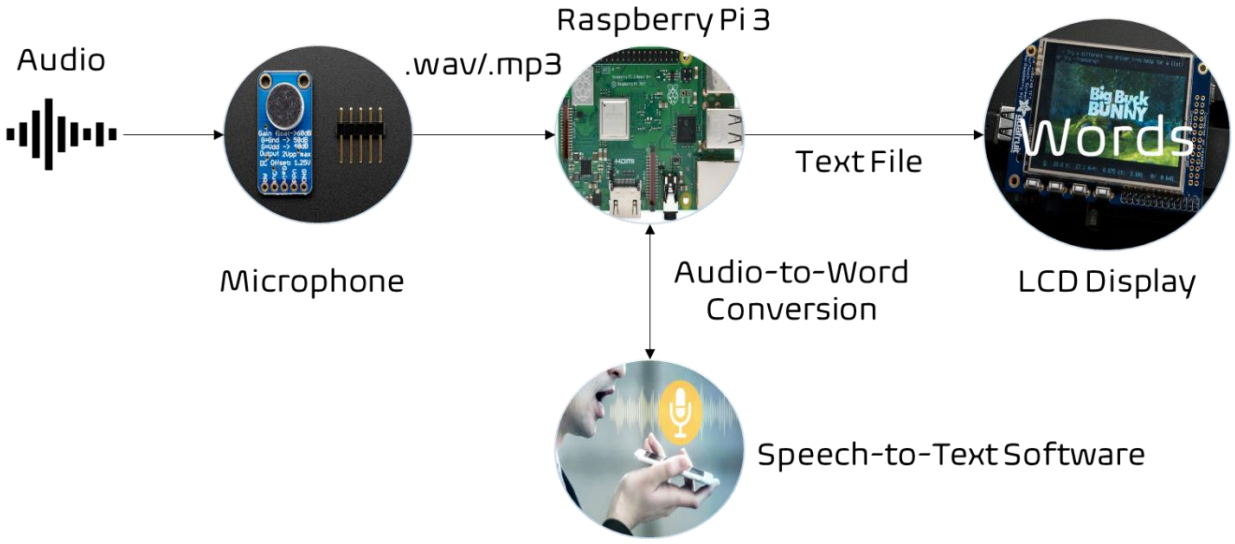


Fig. 4. The mic-to-display subsystem

Last but not least is the subsystem involving the neural network and speaker; both components are required to perform text-to-speech conversion. Once it receives input from the muscle sensors, the neural network then classifies the data that is fed into it as one of the gestures in its database. After this process, the neural network packages its classification into a text file that is subsequently read by the Raspberry Pi. Flaring to life afterwards is the text-to-speech software, which processes the most recent file and returns an audio file that is broadcast and amplified by the speaker.

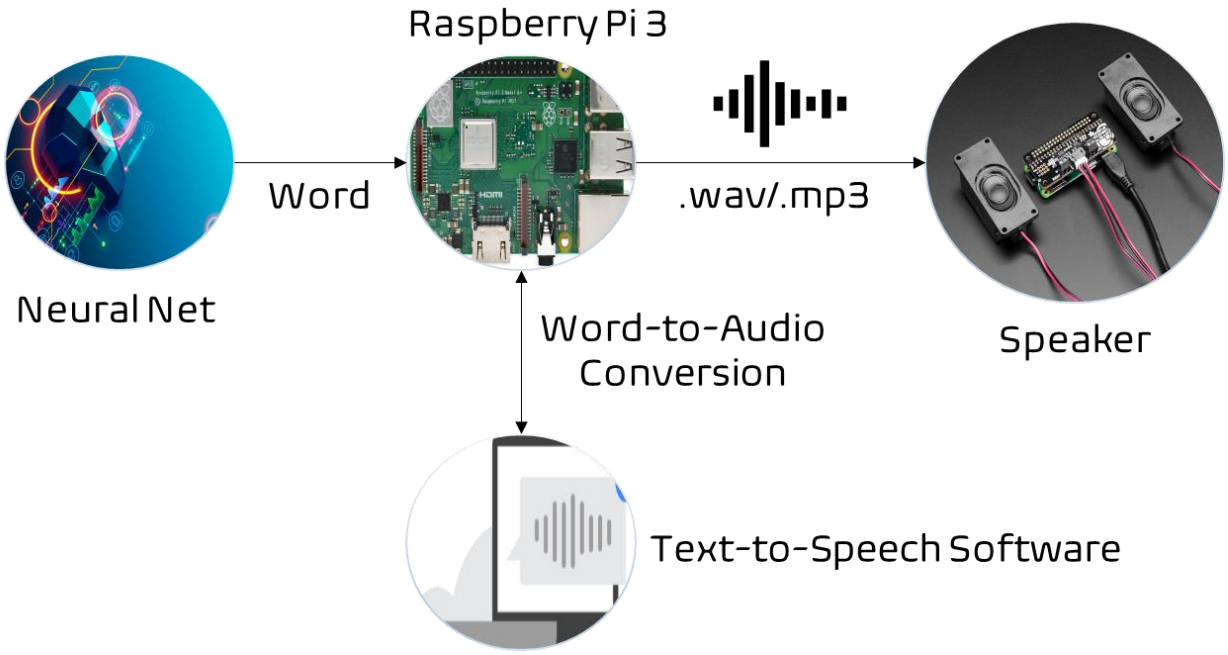


Fig. 5. The AI-to-speaker subsystem

3. Implementation Plan

3.1 Tasks

- Task 1: Data Acquisition and Data Processing

    Subtask 2.1: Recruit test subjects for project.

    Subtask 2.2: Interface microcomputer and sensing devices; then, collect data.

    Subtask 2.3: Write code to automatically document all sensor outputs.

    Subtask 2.4: Write code to automatically filter undesired data.

    Subtask 2.5: Write code to automatically format data for use in a neural network.

- Task 2: Neural Network Design

    Subtask 1.1: Propose a data visualization for sensor data.

    Subtask 1.2: Determine a suitable neural network architecture for the project.

    Subtask 1.3: Verify the feasibility of proposed data visualization method and neural network architecture.

- Task 3: Neural Network Implementation

    Subtask 3.1: Create a TensorFlow model based on the results of Task 1.

    Subtask 3.2: Train TensorFlow model using acquired sensor data.

    Subtask 3.3: Refine TensorFlow model to optimize accuracy.

    Subtask 3.4: Determine method of deployment for TensorFlow model.

    Subtask 3.5: Develop a script for facilitating data flow to TensorFlow model.

    Subtask 3.6: Test finalized TensorFlow model in real time.

- Task 4: Design of Gesture-to-Speech Feature

    Subtask 4.1: Determine required hardware and software.

    Subtask 4.2: Research part specifications.

Subtask 4.3: Interface the software with the hardware.

Subtask 4.4: Assemble the required hardware.

- Task 5: Design of Speech-to-Text Feature

  Subtask 5.1: Determine required hardware and software.

  Subtask 5.2: Research part specifications.

  Subtask 5.3: Interface the software with the hardware.

  Subtask 5.4: Assemble the required hardware.

- Task 6: System Assembly, Testing, and Refinement

3.2 Timeline/Milestones/Delivery Plan

Table 1. PROJECT TIMELINE AND DELIVERY PLAN

Time	Task	Comments	Responsible Personnel
Week 1-5	Project Planning		Josheb Dayrit
Week 6-7	Subtask 2.1 & 2.2	Data visualization method and neural network architecture.	Josheb Dayrit
Week 8-11	Subtask 2.3, Subtask 4.1 & 4.2, Subtask 5.1 & 5.2	Are the proposed data visualization method and neural network architecture viable for use in project?	Josheb Dayrit
Week 12-15	Subtask 1.1, 1.2, 1.3, 1.4, & 1.5	Collection and processing of sensor data.	Josheb Dayrit
Week 16-19	Subtask 3.1, 3.2, 3.3, 3.4, 3.5, 3.6	Deployment of neural network on microcomputer.	Josheb Dayrit
Week 20-22	Subtask 4.3 & 4.4, Subtask 5.3 & 5.4, Task 6	Putting all the pieces together.	Josheb Dayrit

4. Implementation

4.1 Implementation of Task 2.

4.1.1. Implementation of Subtask 2.1

How data is represented, in addition to the data itself, has an impact on the performance of a neural network. For a project that seeks to perform the translation of ASL gestures using a neural network, data is traditionally represented as static images of the ASL gestures themselves. While this method of visualizing data has produced serviceable outcomes for most ASL projects, it is not a method that is without flaws.

To start, it should be noted that static images can indeed depict certain sets of gestures perfectly. For these gestures, the final position of hand is the distinguishing characteristic, and it can usually be contained in a single frame. However, this is not the case for gestures where rotation or movement is a key characteristic, as – unlike other gestures – they cannot be contained in a single frame. Fig. 6 and Fig 7., which are found in the next page, show just how similar some gestures are when only considering their final position. The ASL gestures described by Fig.6 and Fig.7 are "try" and "rock," respectively. Though the orientation of both gestures is a bit different in the figures specified, it should be noted that the palms are balled up for both gestures. Not to mention, there are other gestures where the palm is balled up (i.e. the ASL for the letter "A"). These overlaps in final hand position will make it hard for the neural network to differentiate between gestures if given that information alone. The reason why most AI-based ASL projects do not venture beyond certain gestures is because the method through which their data is expressed is narrowly-focused, suitable for some circumstances but not others. There is an inherent constraint to how their data is visualized, and that constraint prevents their neural network from doing translation work on other gestures.

Fig. 6. ASL for “try” (final hand position)

Fig. 7. ASL for “rock” (final hand position)

Another inherent defect of image data is the presence of image noise, which is often hard to decouple from the desired parts of an image. According to a paper on techniques for image noise reduction, Indian engineering professors Mythili and Kavitha describe 8 types of image noise in great detail: amplifier noise (gaussian noise), salt-and-pepper noise (impulse noise), shot noise, quantization noise (uniform noise), film grain, on-isotropic noise, speckle noise (multiplicative noise), and periodic noise. Though some discussion is devoted to each category of noise, discussions on impulse noise, gaussian noise, and speckle noise are particularly lengthy, as these are the most common types of image noise found in images.

The first type of noise discussed by the paper is amplifier noise (or Gaussian noise), wherein it is stated that amplifier noise is "additive" [6]. Mathematically speaking, Mythili and Kavitha disclose that, in ideal cases, amplifier noise can be quite easily modeled by a Gaussian distribution. Armed with this information, various filters have been developed and implemented by DSP (digital signal processing) researchers in order to combat it. On the fundamental level, amplifier noise is simply noise that adds a bias to the original red, green, and blue values of an image signal.

Fig. 8. Examples of gaussian noise affecting an image (left is original) [7]

In Fig. 6, a ResearchGate paper by professors from 2 European universities applied Gaussian noise on an image to develop techniques relating to facial recognition. A progression is shown of

an image as it is degraded by noise of increasing severity. Based on the differences that can be observed from the image at each stage of the "noising" process, it becomes clear exactly how each pixel is being affected the applied noise. Comparing the original, noiseless image to its noisy counterparts, one can see how pixel intensity is changed at every location in the image, with some pixels becoming more gray and other less gray. These random fluctuations in pixel intensity can be attributed to the fact that Gaussian noise is – according to Mythili and Kavitha – "additive," causing parts of an image to be lighter or darker than they originally are.

Another commonplace noise type that is discussed in Mythili's and Kavitha's paper is impulse noise or salt-and-pepper noise, which is similar in some ways to amplifier or Gaussian noise. In Gaussian noise, the image is affected as a whole. However, in impulse noise, only a select number of pixels are affected, which make them stand out from the rest. The primary difference between Gaussian noise and impulse noise is that impulse noise is smaller in scale (lesser number of pixels are affected) and caused on the hardware side. Some hardware mishaps which can lead to impulse noise include non-functional analog-to-digital converters in cameras, overheating of equipment, and dust in lenses.

Fig. 9. Example of salt-and-pepper noise affecting an image (left is original) [8]

Fig.7 makes the differences between impulse noise and Gaussian noise rather apparent. Firstly, it is clear to see that the original image in Fig.7 does not lose most of its integrity when the impulse noise was applied. By comparison, the original image in Fig.6 essentially became indiscernible when the Gaussian noise was applied. In an image that is affected by impulse noise, there are fewer corrupted pixels, which means that the image is left intact apart from those pixels. The reason behind why impulse noise is also known as salt-and-pepper noise lies precisely in its localized effect on images. Mythili and Kavitha describe it best when they state that images altered by salt-and-pepper noise contain patterns of black and white dots reminiscent of salt and pepper.

The final noise type covered in detail by Mythili and Kavitha is speckle noise, which, unlike Gaussian noise, is multiplicative rather than additive. In particular, satellite and radar technologies are said to be more acutely affected by speckle noise, as it is a direct result of interference from the surrounding area.

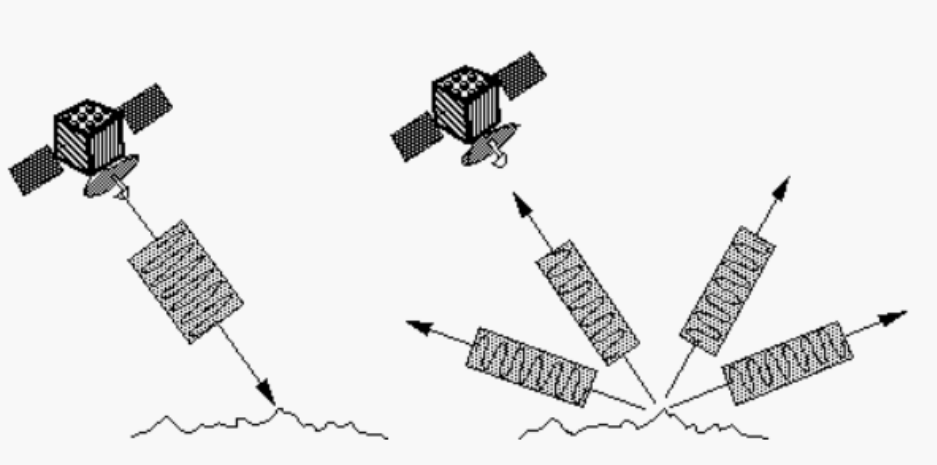


Fig. 10. An imaging radar at work [9]

Most image-capturing hardware, ranging from mundane devices like cameras to more sophisticated contraptions such as the Synthetic Aperture Radar, rely on the concept of backscattering. In the case of the Synthetic Aperture Radar, an image is captured through a mixture of steps involving the radar itself and the specimen. First, the radar illuminates the specimen with a light of its own. Then, sensors on the radar pick up the light that is reflected back to it. This back-and-forth interaction between the radar and the specimen is known as backscattering. Once the reflection is received by the radar, it is processed digitally; based on this reflection, the radar reconstructs its version of the specimen.

In the concept of backscattering, the parameter that matters most is the energy that is reflected by the specimen. An article by NASA on the topic of radar technology reveals that the energy of the echoed photon signal can determine the brightness of an image that is produced. Specifically, NASA states that "bright features (in an image) mean that a large fraction of the radar energy was reflected back to the radar, while dark features imply that very little energy was reflected." In addition to the energy of the reflected signal, there are also environmental and material-specific factors that can influence the quality of an image. For example, in radar images, specimens which contain more water generally appear brighter than their dry counterparts. This is because water has an effect on an object's electrical properties. When water molecules comes into contact with an object, they cohere with other molecules and form a barrier on top of the skin. The resistance of the skin is reduced because it is connected in parallel to the layer of water. In short, variance in resistance can also dictate if an object appears brighter or darker in a radar image.

Radars and cameras are similar in that backscattering can affect the quality of images that both produce. For cameras, the negative impacts of backscattering can be particularly observed in Fig.11, where droplets of water appear as clear or white imperfections in the depicted image.

Fig. 11. Water droplets causing distortions in image

Whenever there are small particles littered about in the frame of a camera, the chance of producing an image with undesirable affectations increases drastically. In addition to water droplets, fine substances like dust or sand can also produce similar affectations in an image. The factor that usually determines the severity of these affectations is distance. According to the inverse-square law, light is reflected more strongly if it is closer to its source (which, in this case, is the camera). By virtue of being closer to the camera lens, the water droplets in Fig.11 appear brighter (and, by extension, more prominent) than the rest of the image.

Image noise, in addition to most gestures being dynamic rather than static, is one primary reason why data from image-capturing devices proves to be unreliable for use in a neural network. There are too many factors (some controllable, while others not) that can affect raw image data from devices such as radars or cameras, and it will take a lot of work just to process the data and isolate the desired parts. Even if a researcher decides to train the neural network to be able to

recognize noise from the gestures, it will take a large pool of data to tackle such a complicated problem. Collecting this data will likely take longer than the two semesters allotted for the Sign AI project. The complication is due in large part by the myriad types of noise that can exist in a single image. To be able to recognize them all, a neural network would need an abundant number of samples. Otherwise, the alternative would be to develop numerous robust filters capable of suppressing noise to an acceptable if not ideal level.

In this project, image processing will still be performed. However, unlike in most AI-based ASL projects, it will not be performed on raw image data from an image-capturing devices; rather, it will be performed on sensor data visualized in the form of scatter plots. Most ASL projects tend to use static photographs of the ASL gestures (particularly their final position, as this position can be a distinguishing characteristic of some gestures) and pass them on to a neural network for processing and training. One would think that this is how data on the gestures should naturally be visualized. The underlying assumption is that photographs of the ASL gestures themselves are able to display the differences between each ASL gesture. As discussed previously, however, this method of data visualization and differentiation proves to be flawed in an ample number of ways. Firstly, not all gestures are static, as some are characterized by their motion and not their final position. In addition, images from image-capturing devices are always subject to corruption due to noise.

4.1.2. Implementation of Subtask 2.2

The convolutional neural network (or ConvNet, for short) is a neural network architecture whose specialty lies in the field of image processing. Many ASL projects have used the ConvNet for the translation of ASL gestures, often achieving successful results. The typical approach of such projects is to use images of the ASL language being performed in real time as training material for the neural network.

In an ASL project conducted by a computer scientist (Vivek Bheda) and a linguist (Dianna Radpour) from the University of New York, the pair attempted to use the ConvNet for classifying the most basic ASL hand signs, which are the letter and number hand signs. In total, 36 hand signs were processed by the ConvNet, consisting of letters from A to Z and numbers from 0 to 10. Supervised learning is the name of the game for dozens of machine learning models (including the model designed in the paper), and it is a kind of learning where the inputs and outputs are pre-established. In supervised learning, the task of the neural network is to engineer a way of mapping inputs to the correct outputs.

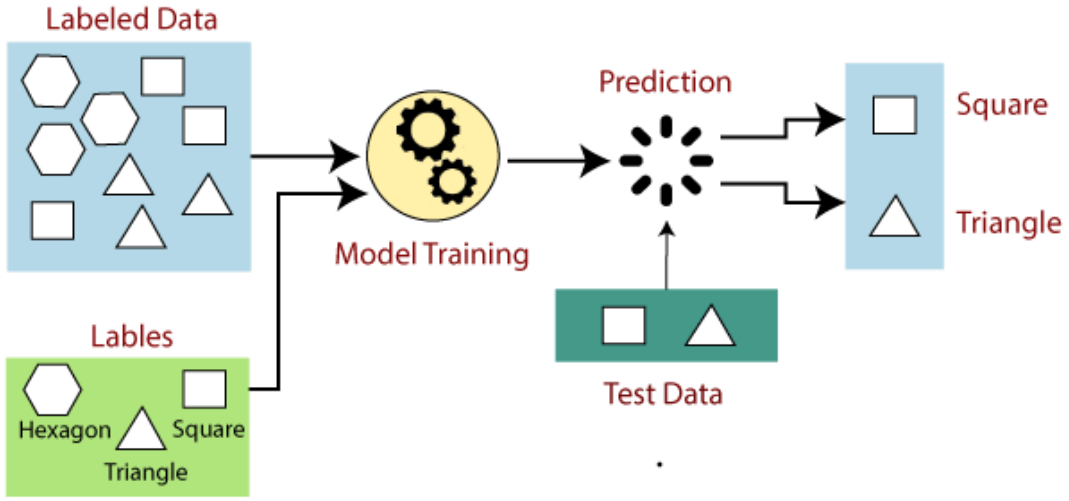


Fig. 12. A diagram of supervised learning in AI [10]

In terms of design, the model developed by the team of Bheda and Radpour also adhered to the conventional structure of a ConvNet, which is a conglomeration of convolutional layers, dense (or fully-connected) layers, and other supplementary layers used for minimizing computational load. The layer that differentiates the ConvNet from neural network architectures similar to it is the convolutional layer, a special layer that processes an image piece-wise rather than as a whole. Using the standard ConvNet architecture, Bheda and Radpour were able to achieve accuracies ranging from 80 to 90 percent, which meant that their ConvNet only failed to distinguish a gesture 10 to 20 percent of the time, depending on the gesture. Sign AI hopes to adopt a similar architecture as the one outlined in Bheda and Radpour's paper. The ConvNet is the go-to neural network architecture when it comes to image-processing projects. Since image processing will still be performed in the Sign AI project, the ConvNet is the neural network architecture that is likely the most suitable and will yield the best accuracy among all other neural network architectures.

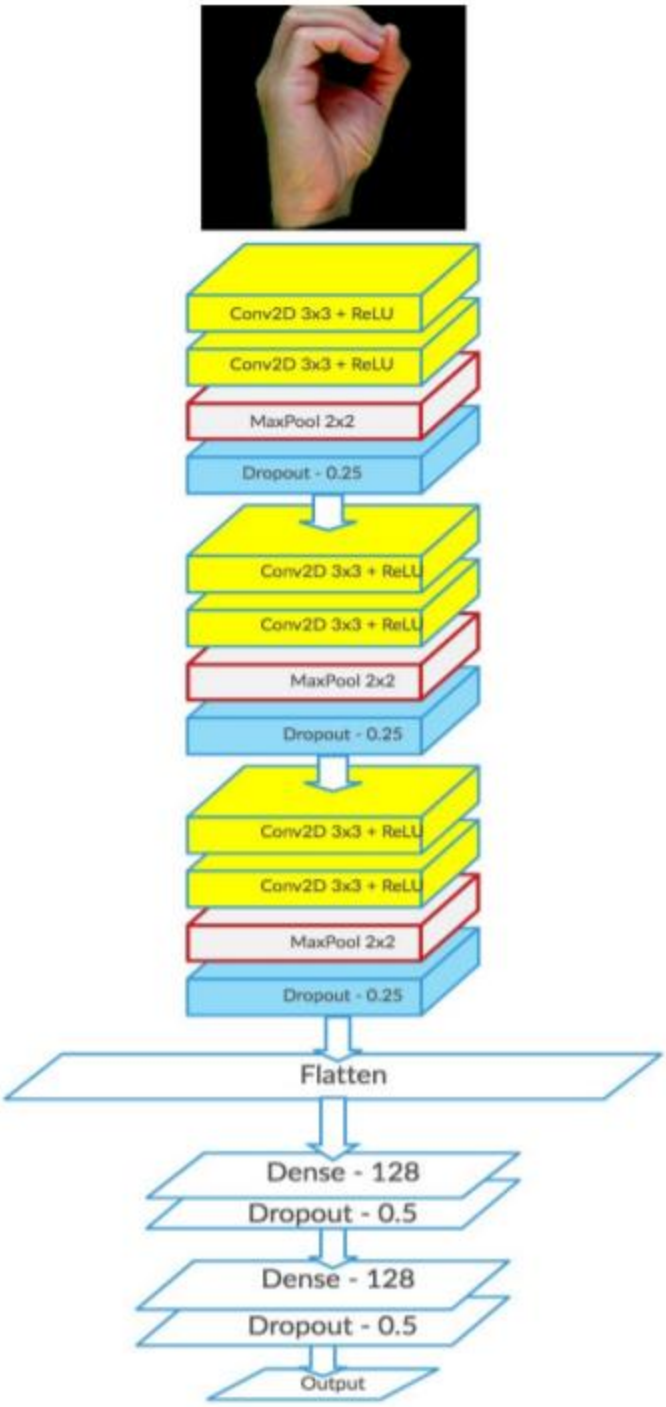


Fig. 13. Architecture of Bheda and Radpour's neural network [11]

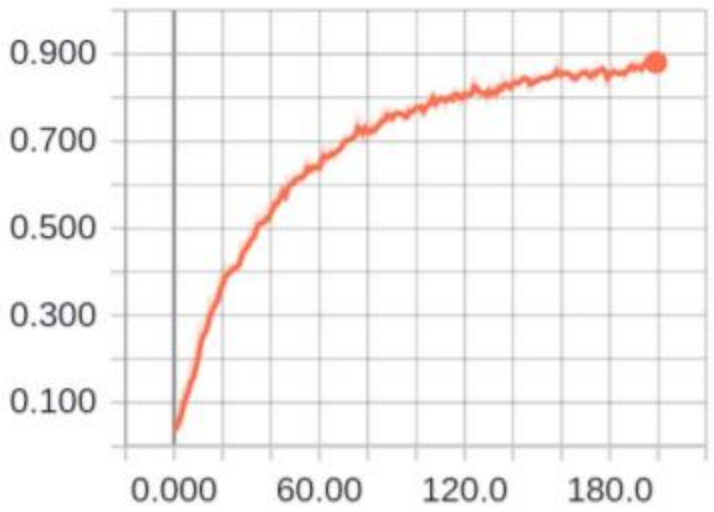


Fig. 14. Performance of Bheda and Radpour's neural network for letters (x-axis: training cyles, y-axis: accuracy)

[11]

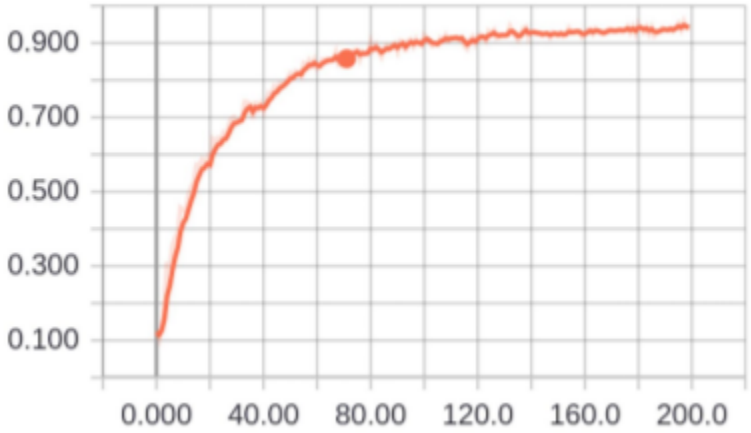


Fig. 15. Performance of Bheda and Radpour's neural network for numbers (x-axis: training cyles, y-axis: accuracy)

[11]

Accuracy is an important parameter in the Sign AI project, and as stated before in the design requirements, the Sign AI will be able to recognize and differentiate between gesture with an accuracy ranging from 70-90 percent. The reason behind this accuracy range lies in the previous successes of similar AI-based ASL projects as well as their scope. Past ASL projects have reached accuracy values close to and even beyond the specified accuracy threshold. However, it should be noted that the scope of these projects were more limited than Sign AI in terms of the number of gestures classified.

One example is a project done by professors from a Pakistani institution. In a paper titled "American Sign Language Translation through Sensory Glove," they attempted to use flex and contact sensors alongside an accelerometer to recognize differences in the activation of muscles when performing ASL gestures. "The glove was found to have an accuracy of 92 percent," [12] the paper's abstract read.

Like other projects, however, the sensory glove had glaring limitations, particularly in the number of gestures that it can translate. While the glove is capable of translating the alphabet with 92 percent accuracy, that is the extent of its translation. The limited scope of the project might have played a part in achieving its high accuracy, as having to do less translation work would mean there are less sources of error. If the glove were to process 100 gestures as opposed to 26 gestures, realistically-speaking, it would have a harder time differentiating between them, especially if some gestures do not produce distinct measurements. The takeaway is that, the more gestures there are to classify, the greater the risk for misclassification becomes. This is why the aim of Sign AI is not 90 percent accuracy, but, rather, 70-90 percent accuracy. The tolerance is necessary to account for the undetermined increase in misclassification due to the wider scope of the project.

To expound on the design requirements, the 100 gestures were not just gestures that were selected at random. This design requirement of 100 gestures was inspired from Dr. Bill Vicars, who is both an ASL professor as well as a deaf person. On his website called “HandSpeak,” he compiled a list of the 100 gestures in the ASL language that are used with the most frequency.

5. Conclusion (Discussion and Future Plans)

By the end of the project, conclude the project and your learning experience.

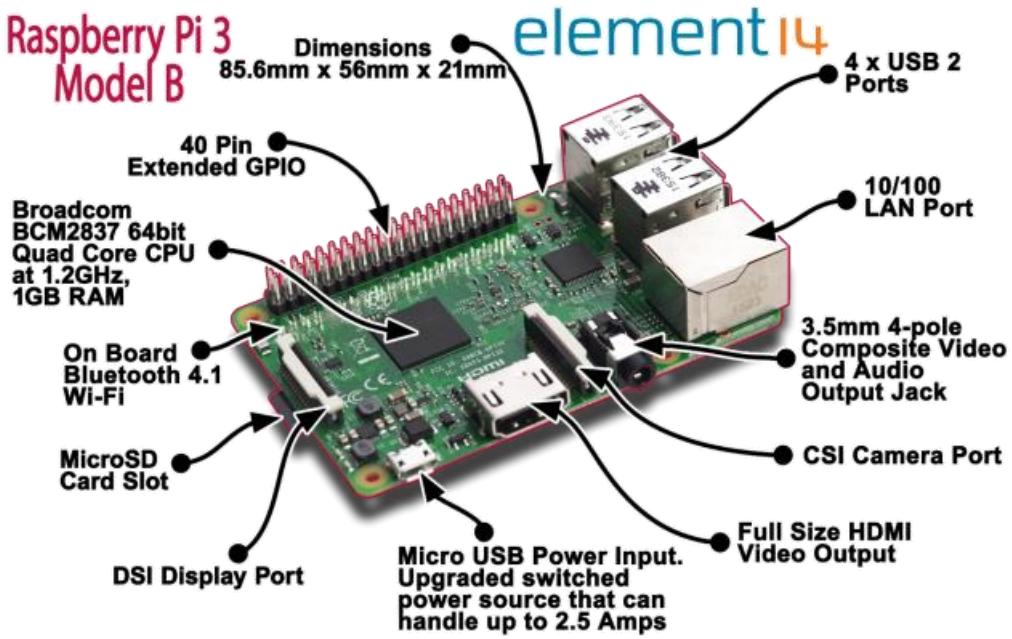
Acknowledgment

If you get help or support from someone else (besides the team member and the advisor) and want to show your appreciation, put here (**do not include the advisor**).

Appendix

A. Component Specs

1. *Specs of Raspberry Pi*



B. Source Code.

1. *Source Code of Pseudo Data for Task 2.3*

```
"""SignAI_PseudoDataGen

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1XV_9cBxyFKjI7VS0jAAOlKn32i68D_ew
"""

# Commented out IPython magic to ensure Python compatibility.
# %tensorflow_version 1.x
import tensorflow as tf

print(tf.__version__)

import pandas as pd
from google.colab import files
import matplotlib.pyplot as plt
```

```
import random

plot_num = 50
samples = 10

i = 0
j = 0

sensor_dat = pd.DataFrame(columns=list('ABCD'))

# generate and download pseudo-random data (Type 1)

while i < plot_num:
   while j < samples:
      new_dat = pd.DataFrame([[float(random.randrange(3000, 3500) / 100),
                           float(random.randrange(2500, 4000) / 100),
                           float(random.randrange(2000, 4500) / 100),
                           float(random.randrange(1500, 5000) / 100)]],
                          columns=list('ABCD'))
      sensor_dat = sensor_dat.append(new_dat, ignore_index='True')
      j = j + 1

   if j == samples:
      j = 0

   sd_scat = sensor_dat.reset_index().plot(xlim=(-1, 10), ylim=(0, 65), kind='scatter', x='index',
y='A',
                                    color='Blue');
   sensor_dat.reset_index().plot(kind='scatter', x='index', y='B',
                            color='Red', ax=sd_scat)
   sensor_dat.reset_index().plot(kind='scatter', x='index', y='C',
                            color='Brown', ax=sd_scat)
   sensor_dat.reset_index().plot(kind='scatter', x='index', y='D',
                            color='Green', ax=sd_scat)
   sd_scat.xaxis.set_label_text("")
   sd_scat.yaxis.set_label_text("")
   plt.grid()
   plt.savefig("type1_" + str(i + 151) + ".png")
   files.download("type1_" + str(i + 151) + ".png")
   plt.clf()
   sensor_dat = sensor_dat.iloc[0:0]  # clears dataframe
   new_dat = new_dat.iloc[0:0]  # same as above

   i = i + 1

# generate and download pseudo-random data (Type 2)
```

```
while i < plot_num:
   while j < samples:
      new_dat = pd.DataFrame([[float(random.randrange(1500, 5000) / 100),
                               float(random.randrange(2000, 4500) / 100),
                               float(random.randrange(2500, 4000) / 100),
                               float(random.randrange(3000, 3500) / 100)]],
                             columns=list('ABCD'))
      sensor_dat = sensor_dat.append(new_dat, ignore_index='True')
      j = j + 1

   if j == samples:
      j = 0

   sd_scat = sensor_dat.reset_index().plot(xlim=(-1, 10), ylim=(0, 65), kind='scatter', x='index',
y='A',
                                           color='Blue');
   sensor_dat.reset_index().plot(kind='scatter', x='index', y='B',
                                 color='Red', ax=sd_scat)
   sensor_dat.reset_index().plot(kind='scatter', x='index', y='C',
                                 color='Brown', ax=sd_scat)
   sensor_dat.reset_index().plot(kind='scatter', x='index', y='D',
                                 color='Green', ax=sd_scat)
   sd_scat.xaxis.set_label_text("")
   sd_scat.yaxis.set_label_text("")
   plt.grid()
   plt.savefig("type2_" + str(i + 151) + ".png")
   files.download("type2_" + str(i + 151) + ".png")
   plt.clf()
   sensor_dat = sensor_dat.iloc[0:0]  # clears dataframe
   new_dat = new_dat.iloc[0:0]  # same as above

   i = i + 1

# generate and download pseudo-random data (Type 3)

while i < plot_num:
   while j < samples:
      new_dat = pd.DataFrame([[float(random.randrange(2500, 4000) / 100),
                               float(random.randrange(3000, 3500) / 100),
                               float(random.randrange(1500, 5000) / 100),
                               float(random.randrange(2000, 4500) / 100)]],
                             columns=list('ABCD'))
      sensor_dat = sensor_dat.append(new_dat, ignore_index='True')
      j = j + 1
```

```
    if j == samples:
        j = 0

    sd_scat = sensor_dat.reset_index().plot(xlim=(-1, 10), ylim=(0, 65), kind='scatter', x='index',
y='A',
                                    color='Blue');
    sensor_dat.reset_index().plot(kind='scatter', x='index', y='B',
                              color='Red', ax=sd_scat)
    sensor_dat.reset_index().plot(kind='scatter', x='index', y='C',
                              color='Brown', ax=sd_scat)
    sensor_dat.reset_index().plot(kind='scatter', x='index', y='D',
                              color='Green', ax=sd_scat)
    sd_scat.xaxis.set_label_text("")
    sd_scat.yaxis.set_label_text("")
    plt.grid()
    plt.savefig("type1_" + str(i + 151) + ".png")
    files.download("type1_" + str(i + 151) + ".png")
    plt.clf()
    sensor_dat = sensor_dat.iloc[0:0]  # clears dataframe
    new_dat = new_dat.iloc[0:0]  # same as above

    i = i + 1

import pandas as pd
from google.colab import files
import matplotlib.pyplot as plt
import random

plot_num = 50
samples = 10

i = 0
j = 0

sensor_dat = pd.DataFrame(columns=list('ABCD'))

# generate and download pseudo-random data (Type 1 modified)

while i < plot_num:
    while j < samples:
        new_dat = pd.DataFrame([[float(random.randrange(3250, 3500) / 100),
                             float(random.randrange(2750, 4000) / 100),
                             float(random.randrange(2250, 4500) / 100),
                             float(random.randrange(1750, 5000) / 100)]],
                           columns=list('ABCD'))
        sensor_dat = sensor_dat.append(new_dat, ignore_index='True')
```

```
      j = j + 1

   if j == samples:
      j = 0

   sd_scat = sensor_dat.reset_index().plot(xlim=(-1, 10), ylim=(0, 55), kind='scatter', x='index',
y='A',
                                    color='Blue');
   sensor_dat.reset_index().plot(kind='scatter', x='index', y='B',
                                 color='Red', ax=sd_scat)
   sensor_dat.reset_index().plot(kind='scatter', x='index', y='C',
                                 color='Brown', ax=sd_scat)
   sensor_dat.reset_index().plot(kind='scatter', x='index', y='D',
                                 color='Green', ax=sd_scat)
   sd_scat.xaxis.set_label_text("")
   sd_scat.yaxis.set_label_text("")
   plt.grid()
   plt.savefig("type1_" + str(i + 151) + ".png")
   files.download("type1_" + str(i + 151) + ".png")
   plt.clf()
   sensor_dat = sensor_dat.iloc[0:0]  # clears dataframe
   new_dat = new_dat.iloc[0:0]  # same as above

   i = i + 1

# generate and download pseudo-random data (Type 2 modified)

while i < plot_num:
   while j < samples:
      new_dat = pd.DataFrame([[float(random.randrange(1750, 5000) / 100),
                               float(random.randrange(2250, 4500) / 100),
                               float(random.randrange(2750, 4000) / 100),
                               float(random.randrange(3250, 3500) / 100)]],
                             columns=list('ABCD'))
      sensor_dat = sensor_dat.append(new_dat, ignore_index='True')
      j = j + 1

   if j == samples:
      j = 0

   sd_scat = sensor_dat.reset_index().plot(xlim=(-1, 10), ylim=(0, 55), kind='scatter', x='index',
y='A',
                                    color='Blue');
   sensor_dat.reset_index().plot(kind='scatter', x='index', y='B',
                                 color='Red', ax=sd_scat)
   sensor_dat.reset_index().plot(kind='scatter', x='index', y='C',
```

```
                        color='Brown', ax=sd_scat)
    sensor_dat.reset_index().plot(kind='scatter', x='index', y='D',
                        color='Green', ax=sd_scat)
    sd_scat.xaxis.set_label_text("")
    sd_scat.yaxis.set_label_text("")
    plt.grid()
    plt.savefig("type2_" + str(i + 151) + ".png")
    files.download("type2_" + str(i + 151) + ".png")
    plt.clf()
    sensor_dat = sensor_dat.iloc[0:0]  # clears dataframe
    new_dat = new_dat.iloc[0:0]  # same as above

    i = i + 1

# generate and download pseudo-random data (Type 3 modified)

while i < plot_num:
    while j < samples:
        new_dat = pd.DataFrame([[float(random.randrange(2750, 4000) / 100),
                        float(random.randrange(3250, 3500) / 100),
                        float(random.randrange(1750, 5000) / 100),
                        float(random.randrange(2250, 4500) / 100)]],
                       columns=list('ABCD'))
        sensor_dat = sensor_dat.append(new_dat, ignore_index='True')
        j = j + 1

    if j == samples:
        j = 0

    sd_scat = sensor_dat.reset_index().plot(xlim=(-1, 10), ylim=(0, 55), kind='scatter', x='index',
y='A',
                                color='Blue');
    sensor_dat.reset_index().plot(kind='scatter', x='index', y='B',
                        color='Red', ax=sd_scat)
    sensor_dat.reset_index().plot(kind='scatter', x='index', y='C',
                        color='Brown', ax=sd_scat)
    sensor_dat.reset_index().plot(kind='scatter', x='index', y='D',
                        color='Green', ax=sd_scat)
    sd_scat.xaxis.set_label_text("")
    sd_scat.yaxis.set_label_text("")
    plt.grid()
    plt.savefig("type1_" + str(i + 151) + ".png")
    files.download("type1_" + str(i + 151) + ".png")
    plt.clf()
    sensor_dat = sensor_dat.iloc[0:0]  # clears dataframe
    new_dat = new_dat.iloc[0:0]  # same as above
```

```
    i = i + 1
```

2. Source Code of Neural Network for Task 2.3

```
"""Copy of GraphClassification_CNN

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1m38O0rxuRoQym4_CpR2-YFP5Q9q9DFMj
"""

import numpy as np
import matplotlib.pyplot as plt
import os
import cv2
from tqdm import tqdm

DATADIR = "/content/drive/My Drive/PseudoDataTrain2"

CATEGORIES = ["type_1", "type_2", "type_3"]

from google.colab import drive
drive.mount('/content/drive')

training_data = []
IMG_WIDTH = 300
IMG_HEIGHT = 300

def create_training_data():
  for category in CATEGORIES:
    path = os.path.join(DATADIR,category)
    class_num = CATEGORIES.index(category)  # if 0, type 1 data; 1, type 2 data; 2, type 3 data.
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img))  # convert to array
        new_array = cv2.resize(img_array, (IMG_WIDTH, IMG_HEIGHT))
        training_data.append([new_array, class_num])

create_training_data()

print(len(training_data))

DATADIR_VAL = "/content/drive/My Drive/PseudoDataValid2"

validation_data = []
```

```
def create_validation_data():
  for category in CATEGORIES:
    path = os.path.join(DATADIR_VAL,category)
    class_num = CATEGORIES.index(category)  # if 0, type 1 data; 1, type 2 data; 3, type 3 data.
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img))  # convert to array
        new_array = cv2.resize(img_array, (IMG_WIDTH, IMG_HEIGHT))
        validation_data.append([new_array, class_num])

create_validation_data()

print(len(validation_data))

import random

random.shuffle(training_data)
random.shuffle(validation_data)

X = []
y = []

X_val = []
y_val = []

for features, label in training_data:
  X.append(features)
  y.append(label)

X = np.array(X).reshape(-1, IMG_WIDTH, IMG_HEIGHT, 3)

for features, label in validation_data:
  X_val.append(features)
  y_val.append(label)

X_val = np.array(X_val).reshape(-1, IMG_WIDTH, IMG_HEIGHT, 3)

import pickle

pickle_out = open('X.pickle', 'wb')
pickle.dump(X, pickle_out)  # dumps X into pickle_out
pickle_out.close()

pickle_out = open('y.pickle', 'wb')
pickle.dump(y, pickle_out)  # dumps y into pickle_out
pickle_out.close()
```

```
pickle_out = open('X_val.pickle', 'wb')
pickle.dump(X_val, pickle_out)  # dumps X_val into pickle_out
pickle_out.close()

pickle_out = open('y_val.pickle', 'wb')
pickle.dump(y_val, pickle_out)  # dumps y_val into pickle_out
pickle_out.close()

from tensorflow.keras import layers, models
import pickle

train_images = pickle.load(open('X.pickle', 'rb'))  # the data
train_labels = pickle.load(open('y.pickle', 'rb'))  # labels for each data

test_images = pickle.load(open('X_val.pickle', 'rb'))  # the data
test_labels = pickle.load(open('y_val.pickle', 'rb'))  # labels for each data

train_images, test_images = train_images / 255.0, test_images / 255.0

model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(IMG_WIDTH,
IMG_HEIGHT, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(3, activation='softmax'))

model.summary()

model.compile(optimizer='adam',
            loss='sparse_categorical_crossentropy',
            metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

DATADIR_EVAL = "/content/drive/My Drive/PseudoDataReal2"

evaluation_data = []
```

```
def create_evaluation_data():
  for category in CATEGORIES:
    path = os.path.join(DATADIR_EVAL,category)
    class_num = CATEGORIES.index(category)  # if 0, type 1 data; 1, type 2 data; 2, type 3 data.
    for img in os.listdir(path):
        img_array = cv2.imread(os.path.join(path,img))  # convert to array
        new_array = cv2.resize(img_array, (IMG_WIDTH, IMG_HEIGHT))
        evaluation_data.append([new_array, class_num])

create_evaluation_data()

print(len(evaluation_data))

X_eval = []
y_eval = []

random.shuffle(evaluation_data)

for features, label in evaluation_data:
  X_eval.append(features)
  y_eval.append(label)

X_eval = np.array(X_eval).reshape(-1, IMG_WIDTH, IMG_HEIGHT, 3)
X_eval = X_eval/255.0

model.evaluate(X_eval,  y_eval, verbose=1)
```

REFERENCES

[1] S. C. Levinson and J. Holler, "The origin of human multi-modal communication," *Philos. Trans. R. Soc. B Biol. Sci.*, vol. 369, no. 1651, p. 20130302, 2014.

[2] C. Lucas, *Multicultural aspects of sociolinguistics in deaf communities*, vol. 2. Gallaudet University Press, 2001.

[3] J. Fileccia, "Sensitive care for the deaf: a cultural challenge," *Creat. Nurs.*, vol. 17, no. 4, pp. 174–179, 2011.

[4] C. Lou Garberoglio, S. Cawthon, and M. Bond, "Deaf People and Employment in the United States," *Washington, DC US Dep. Educ. Off. Spec. Educ. Programs, Natl. Deaf Cent. Postsecond. Outcomes*, 2016.

[5] F. G. Bowe, B. T. McMahon, T. Chang, and I. Louvi, "Workplace discrimination, deafness and hearing impairment: The national EEOC ADA research project," *Work*, vol. 25, no. 1, pp. 19–25, 2005.

[6] C. Mythili and V. Kavitha, "Efficient technique for color image noise reduction," *Res. Bull. Jordan, ACM*, vol. 1, no. 11, pp. 41–44, 2011.

[7] A. S. Andotra and S. Sharma, "A Novel Analysis of Noise and Filtering Mechanism for Image Enhancement-A Review," 2017.

[8] C. Song, S. Sudirman, and M. Merabti, "A Spatial and Frequency Domain Analysis of the Effect of Removal Attacks on Digital Image Watermarks‖," Jan. 2010.

[9] A. Baraldi and F. Parmiggiani, "A refined Gamma MAP SAR speckle filter with improved geometrical adaptivity," *IEEE Trans. Geosci. Remote Sens.*, vol. 33, no. 5, pp. 1245–1257, 1995.

[10] T. Hastie, R. Tibshirani, and J. Friedman, "Overview of supervised learning," in *The elements of statistical learning*, Springer, 2009, pp. 9–41.

[11] V. Bheda and D. Radpour, "Using deep convolutional networks for gesture recognition in American sign language," *arXiv Prepr. arXiv1710.06836*, 2017.

[12] J. Bukhari, M. Rehman, S. I. Malik, A. M. Kamboh, and A. Salman, "American sign language translation through sensory glove; signspeak," *Int. J. u-and e-Service, Sci. Technol.*, vol. 8, no. 1, pp. 131–142, 2015.