AFEKA

The Academic College of Engineering in Tel Aviv

Hand Sign recognition in Hebrew

Prepared for

[dr. Dina Bar-Goren]

Prepared by

[Yaniv Erlich-205958333]
[Bar Moshe-206921777]

2. Special Thanks

Thank you for all your guidance and support during Our final project on computer vision at Afeka College.

We appreciate the resources and facilities provided by the college which made our project possible.

We are grateful for your support and guidance.

Table of Contents

4. Executive Summary

This project aims to develop a system that can accurately translate Hebrew sign language into text. The system will use machine learning algorithms to analyze sign language gestures captured through a camera and then convert them into written text.

The project will involve improving a database of sign language gestures commonly used in Hebrew sign language and using this data to train the machine learning algorithms.

Key Objectives:

- To develop a system that accurately translates Hebrew sign language into text.
- To learn and improve a database of Hebrew sign language
- To train machine learning algorithms using the database of sign language

Benefits:
The development of a system that can accurately translate Hebrew sign language into text will have several benefits. It will facilitate communication between hearing-impaired individuals and those who do not understand sign language, enabling them to communicate more effectively. It will also provide a tool for sign language interpreters to quickly and accurately translate sign language into text, reducing the need for human interpreters in certain situations.

The development of a system that can translate Hebrew sign language into text has the potential to greatly improve communication for hearing-impaired individuals. This project aims to improve a database of sign language letters, train machine learning algorithms to accurately recognize and translate these gestures and design a system that can adapt to various dialects and regional variations. The result will be a valuable tool for hearing-impaired individuals and sign language interpreters.

5. Introduction

The field of computer science has advanced significantly in the last few decades, leading to a surge of technological innovations that have made significant impacts on various aspects of our lives.
One area that has received increasing attention is the development of tools to enhance accessibility for individuals with disabilities, particularly those with hearing impairments.
The Hebrew Sign Language Translator project aims to develop a system that can accurately translate Hebrew sign language into written Hebrew, thereby improving communication accessibility for the deaf and hard-of-hearing community in Israel.

5.1 Motivation:

Sign languages are an essential means of communication for millions of individuals worldwide, particularly those who are deaf or hard of hearing.
However, while sign languages are structured communication systems that are naturally developed and governed by a set of linguistic rules similar to spoken languages, creating accurate translation tools for sign language remains challenging.
This project's motivation is to fill the gap in the availability of accessible translation tools that provide to the unique nuances of Hebrew sign language.

5.2 Problem Definition:

The project aims to address the problem of a lack of accessible tools for smooth communication between deaf and hearing individuals.
With over 300 sign languages being used across the world, an estimated 70 million deaf people rely on them for communication. Hand sign language recognition is the communication barrier faced by individuals who are deaf or hard of hearing when interacting with those who do not understand sign language.This can result in miscommunication, exclusion, and limited access to information and services.Hand sign language recognition technology has the potential to bridge this communication gap by accurately interpreting sign language gestures and translating them into written language, enabling effective communication between deaf and hearing individuals in various contexts.

5.3 Goals:

1. Goal: To develop a system that accurately and reliably translates Hebrew sign language to text in real-time Objectives:

- Conduct research on the most effective machine learning algorithms for recognizing and translating sign language.
- Collect a large dataset of sign language letters to train and test the system.
- Develop and test the system with a small group of users, refining and optimizing the algorithms based on user feedback.

2. Goal: To improve accessibility and inclusivity for deaf and hard-of-hearing individuals in various settings. Objectives:

- Conduct a needs assessment to identify the specific communication needs of the target population in different contexts (e.g., healthcare, education, workplace).
- Develop a user-friendly interface for the system that can be easily accessed by both deaf and hearing individuals.
- Collaborate with community organizations, healthcare providers, educational institutions, and employers to promote and implement the use of the system.

6. Literature Review

"A New Benchmark on American Sign Language Recognition using Convolutional Neural Network."

The article proposes a new approach to recognizing American Sign Language (ASL) using convolutional neural networks (CNNs). The researchers designed a new benchmark dataset called ASL-LEX that contains 2,000 sign language videos with 500 unique signs. They used this dataset to evaluate their proposed CNN-based ASL recognition system and compared it to existing approaches.

The authors report that their proposed method achieved state-of-the-art performance on the ASL-LEX benchmark dataset. They achieved an accuracy of 94.6% on the ASL-LEX test set, which is a significant improvement over existing method. The authors believe that their approach could be used in various applications, such as ASL translation systems and assistive technologies for the deaf and hard of hearing. Overall, this article provides valuable insights into the use of CNNs for ASL recognition and sets a new standard for evaluating future ASL recognition systems.

“A hybrid approach for Bangla sign language recognition using deep transfer learning model with random forest classifier.”

The article discusses a study on the recognition of Bangla sign language using a hybrid approach of deep transfer learning model with a random forest classifier. The study aimed to improve the accuracy of recognizing Bangla sign language using a combination of deep learning and transfer learning. The researchers used convolutional neural networks (CNN) to extract features from the input images of sign language gestures, and then fine-tuned a pre-trained CNN model using transfer learning techniques. They then used a random forest classifier to classify the sign language gestures based on the features extracted by the CNN.

The results of the study showed that the hybrid approach of deep transfer learning with a random forest classifier achieved high accuracy in recognizing Bangla sign language. The researchers found that the combination of CNN-based feature extraction and transfer learning improved the recognition accuracy compared to using a CNN or random forest classifier alone. The study demonstrated the potential of using deep transfer learning models with random forest classifiers for improving the recognition of sign language gestures.

“Text2Sign: Towards Sign, Language Production Using Neural Machine Translation and Generative Adversarial Networks”

The article discusses a study on the use of deep learning techniques for image denoising, which is the process of removing noise or unwanted information from an image. The study proposed a new deep learning model called Residual Encoder-Decoder Convolutional Neural Network (RED-CNN) for image denoising. The researchers used the RED-CNN model to learn the mapping between the noisy input image and the corresponding clean image. The model was trained on a large dataset of images to learn the features that can help in denoising the images effectively.

The results of the study showed that the proposed RED-CNN model outperformed other state-of-the-art image denoising methods in terms of both objective evaluation metrics and subjective visual quality. The researchers found that the RED-CNN model was able to effectively remove noise from various types of images while preserving the details and edges in the images. The study demonstrated the potential of using deep learning techniques for image denoising, and the proposed RED-CNN model can be useful in various applications such as medical imaging, surveillance, and remote sensing.

7. Competitive Analysis

1.OmniBridge

Connecting Deaf and hearing through AI .
https://omnibridge.ai/

OmniBridge provides a solution to the communication gap by enabling bi-directional, real-time conversations between people who use American Sign Language (ASL) and those who speak English by harnessing the power of AI-driven machine translation technology.

2.SignAll

Enabling spontaneous communication between the Deaf and hearing via technology
https://www.signall.us/

SignAll has developed technology leveraging AI and computer vision that is able to recognize and translate sign language. It can be used in both business and education.

3.Byron:

Translations service
https://eby.co.il/

Byron is one of the leading companies in Israel for technical writing, translations and language training. Our goal is to promote fluent communication, and we do so with the best experts who work with us. Byron was founded in 1987.

We conducted market research for our project and found various companies, both technical and non-technical, that work with sign language. We evaluated them and discovered both advantages and disadvantages compared to our solution.

Our advantage over non-technical companies is that our service is affordable and easy to use. Technical companies use AI to translate sign language, but they don't support all 300 sign languages, including Hebrew.

Our solution offers automatic translation of sign language without the need for physical translation and is compatible with Hebrew.

criteria /Company	**OmniBridge**	**SignAll**	**Byron**	**Our Project**
supports Hebrew	x	x	y	y
needs a human translator ?	x	x	y	x
uses technology's	y	y	x	y
real time translation	y	y	y	y
Accuracy	n/a	n/a	100% as it doesn’t depend on computer	**Expected 80%+**

8. Another possible Solutions

1. Motion Capture with YOLOv6: Utilizing YOLOv6, an object detection algorithm, along with motion capture technology can enhance the recognition system's accuracy. YOLOv6 can detect hand movements and track them in real-time, providing a more comprehensive understanding of the sign. The captured motion data can then be fed into machine learning models like recurrent or convolutional neural networks for sign recognition.

2. Depth Cameras: Another approach is to use specialized cameras that can capture 3D information about the sign. This provides more detailed information than a regular camera, enabling the recognition system to identify signs with higher accuracy.

3. Hybrid Approaches: Combining different methods can also improve the recognition system's accuracy. For instance, a hybrid approach that utilizes both motion capture and depth camera data can provide more comprehensive and accurate information for sign recognition. Another approach could be to combine transfer learning with motion capture data to train a more efficient recognition model.

9. Architecture

9.1 First Stage – Image Input:

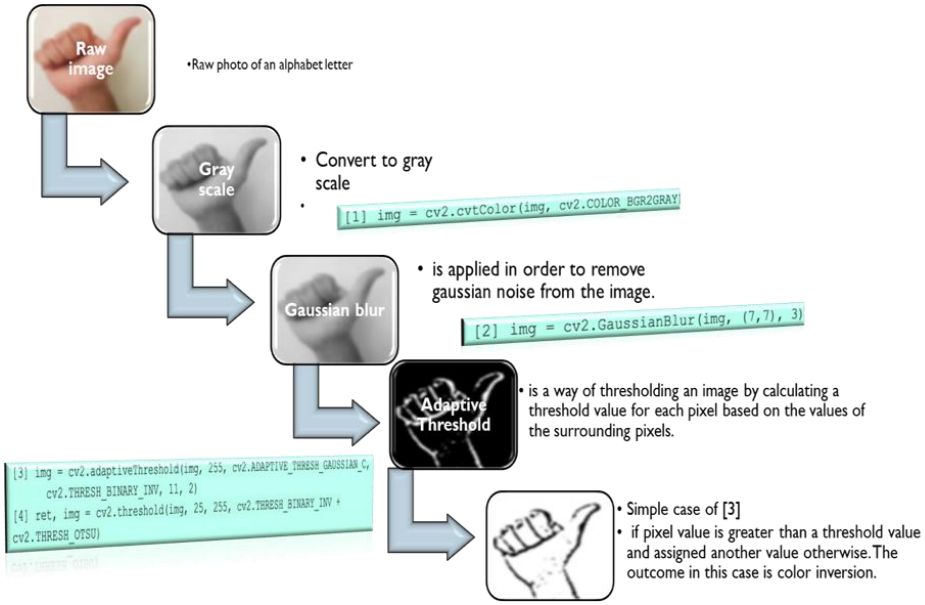
The first stage of the process involves capturing still images from a video stream that is received through a web camera.

These images are taken from a specific region of the video, which is defined by a 128x128 pixel red rectangle that is marked in red. This region of interest (ROI) is chosen to ensure that only the relevant area is captured. The images are captured at a specific frequency, which is known as the sample rate. This frequency determines how often the images are captured and is an important factor in ensuring the accuracy of the process.

9.2 Second Stage - Image Processing:

In the second stage of the process, the raw image of an alphabet letter is subjected to image processing.

Firstly, the image is converted to grayscale to simplify further operations. Then, a Gaussian blur is applied to reduce any noise present in the image. An adaptive thresholding technique is then used to determine a threshold value for each pixel, based on the values of the surrounding pixels. If a pixel's value exceeds the threshold value, it is assigned another value, otherwise, the original value is retained. As a result, the image undergoes color inversion. These techniques enhance the image, making it easier to recognize the alphabet letter.



*based on git how many fingers.

9.3 Third Stage – Feature Selection :

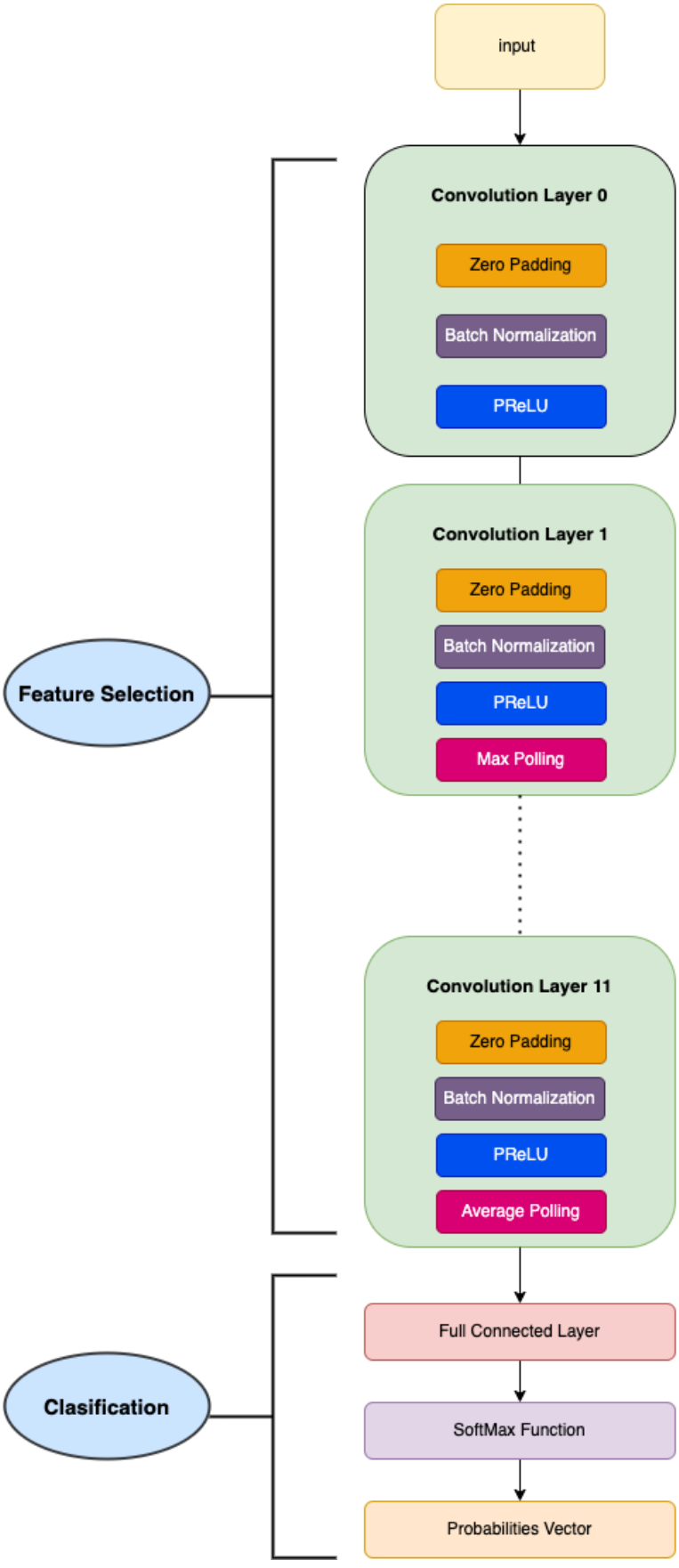
In the third stage, we select features using a method that involves repetitive blocks that are mostly the same. The main difference between these blocks is the number of Convolutional Filters used, which ranges from 32 to 1028.
The deeper layers can detect more abstract features. This process is repeated a total of N times until the last Convolutional block is reached. The result is a small dimensional vector for a fully connected layer.
Each block contains a Convolutional function with 3x3 filters and bias, zero padding to match the dimensions of the next convolution block, batch normalization to stabilize the network, an activation function (PReLU), and max pooling (every second block).
The final block executes average pooling instead of max pooling. After evaluating several models, we chose the CNN12 architecture because it provided the best performance in terms of accuracy and computational efficiency.

The repetitive blocks in this architecture help to extract features from the input image, and the deeper layers enable the detection of more complex features.

9.4 Final Stage – Classification:

In the final stage of the process, the image that has been processed by the Convolutional layers reaches the fully connected layers. These layers, along with the implementation of the SoftMax function, convert the data into a probability vector. The predefined threshold for the probability vector is 80%. If the probability vector does not contain a value that passes this threshold, the sign is ignored and not displayed. Each cell in the vector represents the probability of an Israeli Sign Language letter, including the space and delete signs. The system output is the letter with the highest probability value. This approach allows the system to accurately classify the sign language gesture into its corresponding letter or symbol.

Model’s Block Diagram



10. Design

10.1 Artificial Neural Networks & Deep Learning

Deep learning is a type of machine learning that tries to copy the way our brains work when processing information and recognizing patterns. It uses Artificial Neural Networks, which can learn from both labeled and unlabeled data.
These networks are given large amounts of data to learn from so that they can figure out important features that will help them solve classification problems. They produce a list of possible answers, with the most likely one being chosen.

Neural Networks are a set of tools that help machine learning algorithms process data inputs. They are made up of layers of Artificial Neurons, which have an Activation function that determines the output of each neuron. The input layer is the first layer of the network, and the output layer is the last. The layers in between are called hidden layers because they don't represent input or output.

10.2 Convolutional Neural Networks (CNN):

Convolutional Neural Networks (CNN) are commonly used for classification tasks, such as image, speech, and audio recognition. CNNs learn to identify features from large sets of diverse examples to produce a probability vector that helps classify new examples. CNNs consist of an input layer, multiple hidden layers (including Convolutional Layers), and an output layer.
Convolutional Layers detect patterns using filters (weights) and transform the input using convolution operations. Optional image processing layers may contain predefined filters. For sign language recognition using images, a CNN is often used, and the last layers of the CNN are called Fully Connected Layers. In a Fully Connected Layer, each output neuron is connected to each input neuron with a specific weight, forming the parameters of the layer. The connection is represented mathematically using a dot product between the weights and input neurons to produce an output neuron.

10.3 CNN-12:

CNN12 is a type of model that can classify grayscale images.
The program takes images that are 128x128 pixels in size as input. The model has 12 layers that perform special operations on the input image called convolutional layers.
After each convolutional layer, some additional operations are performed to help the program better understand the image.

Every two convolutional layers, the program performs another type of operation called max pooling to simplify the image. The last pooling layer is different, and it performs an operation called average pooling. The final two layers of the program are used to make the actual prediction of which category the input image belongs to. These two layers use a technique called dropout to improve the program's accuracy.
Finally, the program uses a mathematical function called SoftMax to output a probability for each of the 25 categories, indicating how likely it is that the input image belongs to each category. This program was inspired by some tutorials from Stanford University and was used to identify wild animals from camera-trap images in a research project.

10.4 Perceptron's

Perceptrons are simple artificial neurons that take in a set of inputs, represented as binary values, and weights, which tell the Perceptron how important each input is. The output of the Perceptron is determined by comparing the sum of the inputs multiplied by their respective weights to a threshold value, which is set using a bias. If the sum is greater than the threshold value, the output is one; otherwise, it is zero.

The problem with Perceptrons is that if the weights or bias are changed even a little bit, it can cause the output to flip from one to zero or vice versa. This can make it difficult for the network to learn from the data and adjust the weights accordingly.

$$y = 1 \quad if(w * x + b) > 0$$

$$y = 0 \quad otherwise$$

- y is the predicted output
- w is a vector of weights, which are adjusted during training to optimize the model's performance
- x is a vector of inputs
- b is the bias term, which can be thought of as the intercept or threshold of the model.

10.5 ReLU

The ReLU function works better than the Sigmoid function for a problem called Vanishing Gradient. This problem happens when an Activation function squeezes the input to a small range of output values in a non-linear way. This means that many different input values are all getting mapped to the same small output value, so the model can't learn from the data very well. The ReLU function solves this problem, but it has another problem where it turns all negative input values into 0, which can also cause problems with learning. To fix this, Leaky ReLU was created, which multiplies negative input values by a small number instead of turning them into 0.

10.6. Pooling Function

A Pooling Layer is a part of a Convolution Neural Network that helps to make the network faster and prevent it from getting too good at recognizing just the training data. It does this by shrinking the size of the data that the network is working with. This makes it easier and faster for the network to process the information, and it also means there are fewer things for the network to remember, which can help prevent it from becoming too specialized to the training data.

10.6.1 Max Pooling

Max Pooling is a part of a Convolution Neural Network that makes the network faster and helps it recognize images better. It does this by looking at small squares of the input data and only keeping the biggest number in each square. This means the network doesn't have to remember as much information, which can make it faster and prevent it from getting too good at recognizing just the training data. The Max Pooling layer uses 2x2 squares, and it jumps over the input data by 2 squares each time. The Max Pooling layer is used to get rid of less important information in the input data, so the network can still recognize the image correctly.

10.6.2 Average Pooling

Average Pooling is a technique used in Convolution Neural Networks that helps to make the network faster and prevent it from overfitting. It works by dividing the input data into small squares and calculating the average value of each square. This reduces the size of the input data and makes it easier for the network to process. Average Pooling is similar to Max Pooling, but instead of keeping the largest value in each square, it keeps the average value. This helps to smooth out the input data and remove some of the noise. Average Pooling is often used after Convolutional Layers in a neural network to help reduce the number of parameters and improve performance.

10.7. SoftMax Function

The SoftMax function is used in the last layer of the CNN model to generate a probability distribution vector, where each value in the vector represents a possible image category. The function takes an input vector and normalizes it to produce an output vector where the sum of all values equals 1.

The function includes a temperature coefficient (c), which affects the distribution of the output vector. Increasing the temperature coefficient makes high values even higher and low values lower, while decreasing it flattens the distribution by decreasing high values and increasing low values.

In equation form, the SoftMax function can be written as:

$$y_i = \frac{e^{cx_i}}{\sum_{k=1}^{m} e^{cx_k}}$$

In this equation, the output value (y_i) for each category i is determined by taking the exponential of the input value (x_i) multiplied by the temperature coefficient (c), and then dividing by the sum of the exponential values of all input values multiplied by the temperature coefficient.

10.7. Conclusion from last design

We decided to use a Convolutional Neural Network (CNN) for our image classification task, and after some research, we chose the cnn12 architecture from a Github repository.
cnn12 is based on VGG16, a popular CNN model known for its strong performance in image classification.
However, cnn12 has fewer layers, making it faster and more efficient while still achieving high accuracy on various image datasets. By using cnn12, we hope to improve the speed and accuracy of our image classification system.

11. Alpha Product

11.1 Algorithms

```
def show_test_popup(roi):
    if isOn == False:

isOn = True
TestInput = tkinter.Toplevel()
TestInput.geometry("350x700")
TestInput.title("Popup")
current_roi = Image.open(roi)
current_roi_tk = ImageTk.PhotoImage(current_roi)
img_component = tkinter.Label(TestInput, image=current_roi_tk)
img_component.image = current_roi_tk
img_component.place(x=75, y=200)
label = tkinter.Label(
TestInput, text=f" Current prediction is: {pred}\nChoose the right
letter if prediction is wrong")
label.pack()
variable = tkinter.StringVar(TestInput, alphaBet[0])
option_menu = tkinter.OptionMenu(
TestInput, variable, *hebrew_to_english.keys())
option_menu.pack()
button_frame = tkinter.Frame(TestInput)
button_frame.pack()
Righr_button = tkinter.Button(
button_frame, text="Prediction Right", command=on_yes_click)
Righr_button.pack()
wrong_button = tkinter.Button(
button_frame, text="Prediction Wrong", command=on_no_click)
wrong_button.pack()
TestInput.protocol("WM_DELETE_WINDOW", on_no_click)
```

The **show_test_popup** function is a Python function that displays a popup window with an image and buttons to confirm or correct a prediction. The function takes one argument, **roi**, which is a string that represents the path to an image to be displayed in the popup.
The function returns **None**.
When the function is called, it defines two sub-functions, **on_yes_click** and **on_no_click**, which are executed when the "prediction right" or "prediction wrong" buttons are clicked, respectively. These functions are responsible for writing the prediction and user's choice to a CSV file and saving a copy of the image with a unique filename. They also increment the counter and close the popup window.
The **show_test_popup** function first checks whether the popup window is already open or not. If it's not open, the function creates the popup window using the Tkinter module. It then loads the image and creates a Tkinter PhotoImage object to display the image in the window. The function also creates a label and option menu for selecting the correct letter and buttons to confirm or correct the prediction.
Finally, the function configures the window to close if the user tries to close it, disables other windows while the popup is open, and waits for the popup window to be destroyed.

```
TestInput.grab_set()
TestInput.wait_window()
```

```
class VideoFrame:
    def __init__(self, video_source=0):
        self.testMode = False
        self.vid = cv2.VideoCapture(video_source)

        if not self.vid.isOpened():
            raise ValueError("Unable to open video source", video_source)

        self.width = self.vid.get(cv2.CAP_PROP_FRAME_WIDTH)
        self.height = self.vid.get(cv2.CAP_PROP_FRAME_HEIGHT)
        self.showMask, self.predict = 0, 0
        self.fx, self.fy, self.fh = 10, 50, 45
        self.x0, self.y0, self.predWidth = 400, 50, 224

    def get_frame(self):
        global dataColor
        global count, pred

        if self.vid.isOpened():
            ret, self.frame = self.vid.read()
            self.frame = cv2.flip(self.frame, 1)  # mirror
            frame = copy.deepcopy(self.frame)
            cv2.rectangle(frame, (self.x0, self.y0),
                          (self.x0 + self.predWidth - 1,
                           self.y0 + self.predWidth - 1),
                          dataColor, 12)
            roi = self.frame[self.y0:self.y0 + self.predWidth,
                             self.x0:self.x0 + self.predWidth]
            roi = binaryMask(roi)
            if self.showMask:
                img = cv2.cvtColor(roi, cv2.COLOR_GRAY2BGR)
                frame[self.y0:self.y0 + self.predWidth,
                      self.x0:self.x0 + self.predWidth] = img
            if self.predict:
                loop = asyncio.get_event_loop()
                loop.run_until_complete(
                    predictImg(roi, test_mode=self.testMode))

            if self.predict and not self.testMode:
                dataColor = (0, 250, 0)
                cv2.putText(frame, 'Strike ' + 'P' + ' to pause', (self.fx, self.fy - 15),
cv2.FONT_HERSHEY_SIMPLEX,
```

The **VideoFrame** class is defined with an **__init__** method that takes a **video_source** parameter with a default value of **0**. Inside the method, it sets the **testMode** instance variable to **False**. It then opens the video source by creating a **cv2.VideoCapture** object with the **video_source** parameter and raises a **ValueError** if the video source cannot be opened. The method then gets the width and height of the video source and sets them as instance variables. It also initializes several other instance variables, including **showMask**, **predict**, **fx**, **fy**, **fh**, **x0**, **y0**, and **predWidth**.
The class also has a **get_frame** method that reads a frame from the video source and applies some image processing to it. It flips the frame horizontally, draws a rectangle on it, and applies a binary mask to a region of interest (ROI) within the rectangle. Depending on the values of **showMask** and **predict**, it either displays the processed ROI on the frame or applies a prediction algorithm to the ROI. It then adds text to the frame based on the values of **predict**, **testMode**, and some other variables. Finally, it adds a letter prediction and a sample timer to the frame using **PIL** and **ImageDraw** and returns the frame as a boolean success flag and a BGR image.
The **VideoFrame** class also has a **__del__** method that releases the video source when the object is destroyed. image and writes the prediction and user's choice to a CSV file.

```
                    0.6, dataColor, 2, 1)
        elif self.testMode:
            dataColor = (250, 0, 20)
            cv2.putText(frame, 'Strike ' + 'T' + ' to stop test', (self.fx, self.fy - 15),
cv2.FONT_HERSHEY_SIMPLEX,
                    0.6, dataColor, 2, 1)
        else:
            dataColor = (0, 0, 250)
            cv2.putText(frame, 'Strike ' + 'P' + ' to start' + ' or T to Test', (self.fx, self.fy -
10), cv2.FONT_HERSHEY_SIMPLEX,
                    0.8, dataColor, 2, 1)

        # Add Letter prediction
        img_pil = Image.fromarray(frame)
        draw = ImageDraw.Draw(img_pil)
        draw.text((self.fx, self.fy + self.fh), "Prediction: %s" %
                pred, font=font, fill=dataColor)
        draw.text((self.fx, self.fy + 380), 'Sample Timer: %d ' %
                count, font=font, fill=dataColor)
        # noinspection PyAttributeOutsideInit
        self.frame = np.array(img_pil)

        if ret:
            # Return a boolean success flag and the current frame converted to BGR
            return ret, cv2.cvtColor(self.frame, cv2.COLOR_BGR2RGB)
        else:
            return ret, None
    else:
        return None

  # Release the video source when the object is destroyed
  def __del__(self):
    if self.vid.isOpened():
        self.vid.release()
```

```
async def predictImg(roi, test_mode=False):

  Asynchronously prediction.
  :param roi: preprocessed image.
  global count, sentence
  global pred, prevPred, textForm
  img = cv2.resize(roi, (imgDim, imgDim))
  img = np.float32(img) / 255.
  img = np.expand_dims(img, axis=-1)
  img = np.expand_dims(img, axis=0)
```

This function **predictImg** takes in an image region of interest (**roi**) as input and asynchronously predicts the class of the image.
The **roi** parameter is expected to be preprocessed before being passed to this function.
The function first resizes the image to a fixed size (**imgDim**), converts it to a floating-point representation, scales it to values between 0 and 1, and adds dimensions to represent the channel and batch size dimensions.

```
    vec = model.predict(img)
    pred =
convertEnglishToHebrewLetter(classes[np.argmax(vec[0])])
    maxVal = np.amax(vec)
    if maxVal < threshold or pred == '':
        pred = ''  # reset the prediction
        count = sample_rate  # reset the counter
    elif pred != prevPred:
        prevPred = pred  # update the previous prediction
        count = sample_rate  # reset the counter
    else:  # maxVal >= Threshold && pred == prevPred
        count = count - 1
        if count == 0:
            count = sample_rate  # reset the counter
            if test_mode:
                cv2.imwrite("lastRoi.jpg", cv2.cvtColor(
                    roi, cv2.COLOR_RGB2BGR))
                if isOn == False:
                    show_test_popup("lastRoi.jpg")
            else:
                if pred == 'del':
                    sentence = sentence[:-1]
                else:
                    sentence = sentence + pred
                if pred == ' ':
                    pred = 'space'
                textForm.config(state=NORMAL)
                textForm.delete(0, END)
                textForm.insert(0, (finalizeHebrewString(sentence)))


                textForm.config(state=DISABLED)
```

The preprocessed image is then passed to a pre-trained machine learning model (**model**) to obtain a prediction vector.
The prediction vector contains the predicted probability of the input image belonging to each class in the classification task.
The function uses the **argmax** function from NumPy to get the index of the class with the highest probability and maps this index to its corresponding Hebrew letter using a dictionary.
The function then checks if the predicted class probability is greater than a certain threshold value (**threshold**).

If the maximum value in the prediction vector is less than the threshold, the prediction is considered invalid, and the function resets the prediction and a counter. Otherwise, if the prediction is valid and different from the previous prediction, the function updates the previous prediction and the counter. If the prediction is valid and the same as the previous prediction, the counter is decremented, and if it reaches zero.

If the **test_mode** parameter is set to **True**, the function saves the input image region to a file named "lastRoi.jpg" and displays a popup window containing the image.
Otherwise, the function modifies a **sentence** variable based on the predicted class, updates a **textForm** object with the contents of the **sentence** variable, and disables the text box to prevent the user from modifying its contents.
The function returns no output, and its results are instead stored in global variables **pred**, **prevPred**, **textForm**, **count**, and **sentence**.

11.2 Code

Video explaining the running process

The project consists of two main sections: the training and testing of the CNN12 model using Google Colab and the development of the alpha product in Python.

The project's root directory contains several files, including "globals.py" and "utilities.py," which are Python modules that provide shared functions and variables for use in the project.

- The "**Model**" directory contains the trained model weights and related files, including cnn12_model, trainWeights.
  These files are used to load and test the CNN12 model.
- The "**fonts**" directory contains the font file "arial.ttf," which is used for text display in the alpha product.
- The "**icons**" directory contains the images "delete.png" and "save.png," which are used as icons in the alpha product.
- The "**Final_Project.ipynb**" file is a Jupyter notebook that was used for training and testing the CNN12 model in Google Colab.
- The "**main_program.py**" file is the alpha product itself, which contains the code for loading the trained model and performing image preprocessing and recognition.
- the "**requirements.txt**" file lists the necessary Python packages for running the alpha product.

To install the necessary packages for running the "main_program.py" file, you need to follow these easy steps:

Go to the project's main folder using the command line or terminal.

Open the "requirements.txt" file using a text editor.

Type the command "pip install -r requirements.txt" and press enter to install the required packages.

After you have installed the necessary packages, follow these simple steps to run the "main_program.py" file:

Go to the project's main folder using the command line or terminal.

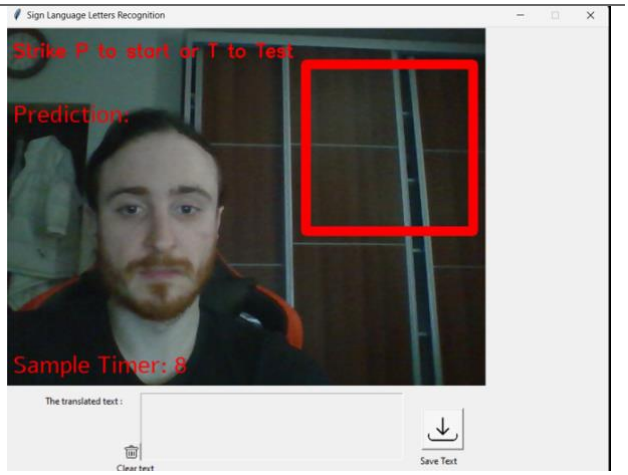
Type "python main_program.py" and press enter.

The alpha product will open, and you can use it to recognize images by selecting them using the "Choose File" button and clicking "Predict."

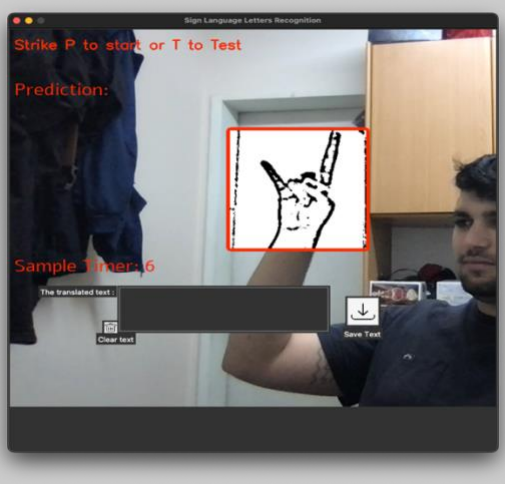
Please make sure that the required model and weight files are in the "Model" folder and that the "fonts" and "icons" folders are in the main project folder before running the "main_program.py" file.

11.3 Alpha Manual

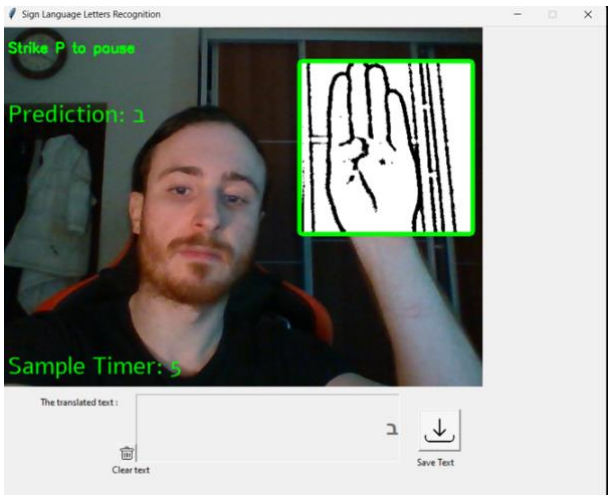
When you open the program, a screen will appear that looks like the image provided. On this screen, you will see a small red box and three options: "P" for prediction mode, "T" for testing mode, and "M" to start image preprocessing. Each option is explained in detail below.



If you click on "M," the program will implement preprocessing filters on the region of interest (ROI) and show a preview of the image as it will appear to the model during the image preprocessing stage.



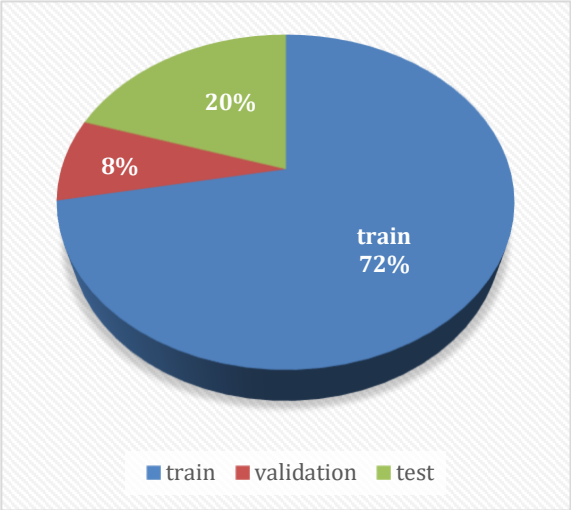
Prediction Mode:When you press "P," the program will enter Prediction mode. By placing your hand on the small green box, the program will read your hand sign and translate it to Hebrew language, displaying the text in the black box below. You have the option to clear the text by clicking Clear text or save the text as a file by clicking Save Text , converting the hand signs to written text for later use.



12. Assessment

12.1 Data Set

The dataset comprises 150,000 images consisting of 25 categories, which includes 22 letters of Israeli Sign Language, a space sign, a delete sign, and a class containing solely white background images.
Prior to training, the images underwent preprocessing to reduce the training time.



The dataset was partitioned into three subsets, namely Train, Test, and Validation data, with 72% allocated for training, 8% for validation, and 20% for testing.
When looking at the letters in Hebrew sign language, it might be hard for a model to recognize some of the letters that look similar. For example, it could confuse the letters "נ" ,"ת", and "מ", or the letters "ו" and "י".


Example of the letter 'מ'


Example of the letter 'נ'

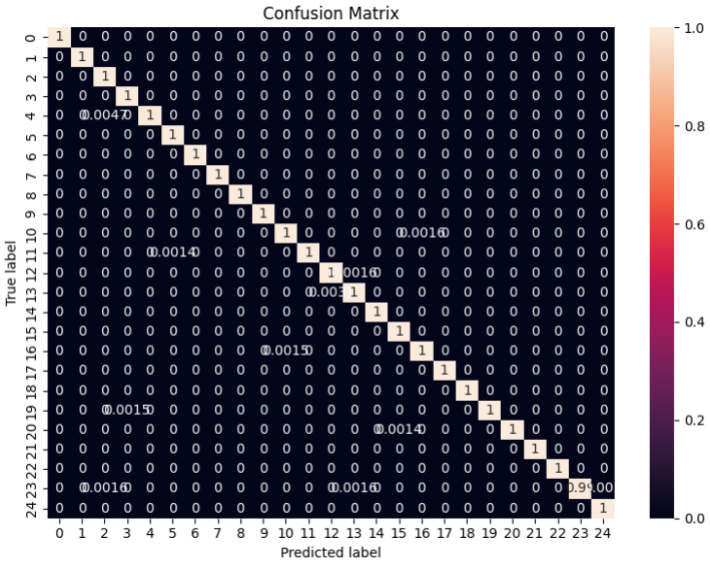

Example of the letter 'ת'

To improve how well the model worked when it was tested, we saved the images and made sure they were spread out evenly between the groups used for training, testing, and validation.

We will explain more about this in the report's evaluation section.

12.2 Evaluations

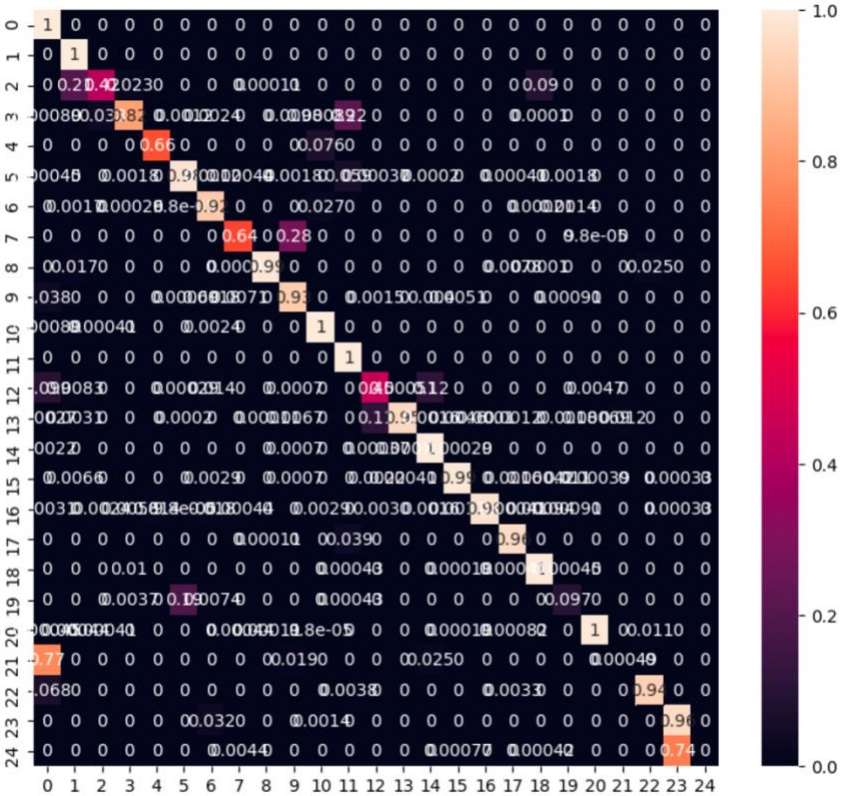
Initially, the model evaluation was conducted using 20% of the Test Data, as explained in the Training section. The results were impressive, with a high accuracy rate of 99%. However, this led to the conclusion that the test data closely resembled the train data, thus requiring further evaluation.



To ensure the system's accuracy was thoroughly tested, a smaller dataset was created, consisting of 140 images per letter, totaling 3500 images.

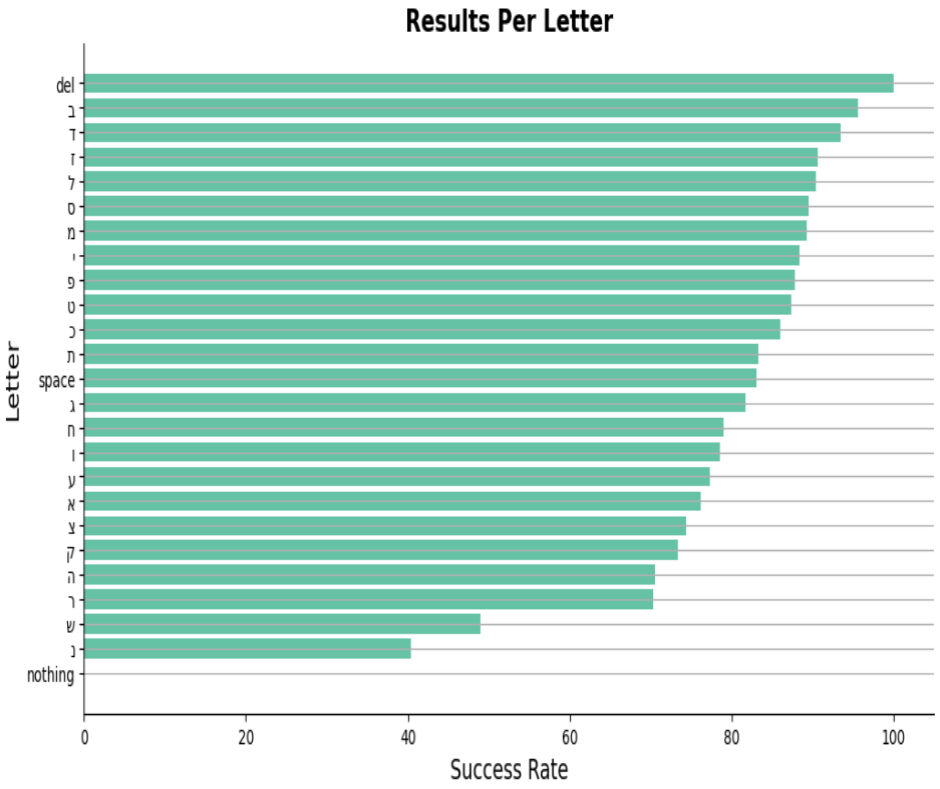
The new dataset was used to assess the accuracy of the recognition process using an evaluation script.

The accuracy rate of the application was calculated to be 89.66% , and a confusion matrix was generated, displaying the confusion rate for each Hand Sign class.



For our final project, we wanted to test our model's accuracy by using the test mode that we discussed earlier. We tested each letter and performed around 3500 tests in total. We then calculated the success rate for each letter and found that the average success rate across all letters was 81.23%. This information is useful because it gives us an idea of how well our model performs overall and can help us identify any areas for improvement.

letter	success rate	letter_counter
נ	40.38	104
ש	48.98	98
ר	70.31	128
ה	70.53	190
ק	73.33	120
צ	74.32	148
א	76.19	210
ע	77.27	88
ו	78.57	112
ח	79.03	124
ג	81.69	142
space	83.08	130
ת	83.33	264
כ	85.96	114
ט	87.3	126
פ	87.84	148
י	88.33	240
מ	89.23	130
ס	89.47	76
ל	90.41	146
ז	90.62	128
ד	93.48	184
ב	95.59	136
del	100	92



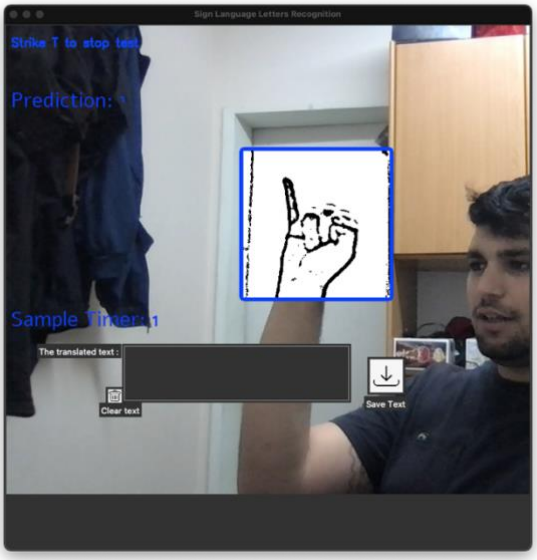
12.3 Testing

Test Mode:

Pressing the "T" key activates Testing mode. When you place your hand on the small blue box, the program will read the sign and translate it into Hebrew.

The predicted letter will be displayed near the prediction on the top left of the screen.

A prompt window will then appear with the current prediction, allowing you to edit it if it's incorrect. You will have two options to choose from to indicate whether the prediction is correct or incorrect. The popup window picture appears below.



During testing, a pop-up message will appear indicating that a prediction has been made. If the prediction is accurate, the user should select the "Prediction Right" button, which will save the image to the "tempimages" folder, located within a subfolder of the corresponding letter/category.

If the prediction is incorrect, the user should choose the correct prediction (letter/category) and click the "Prediction Wrong" button. Regardless of the button selected, the action will be logged in the test.csv file.

13. Results

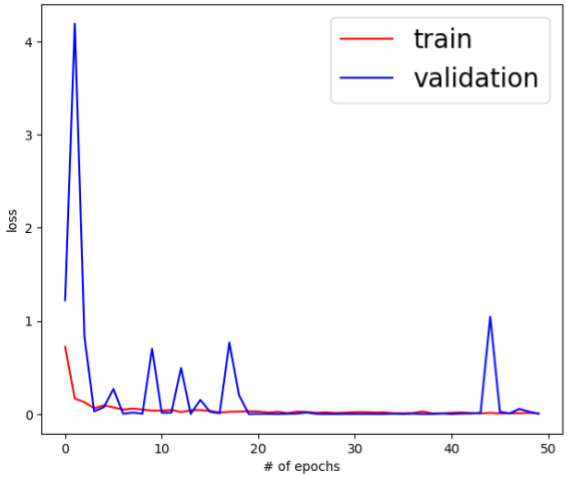
We used Google Collab's GPU to train the model.

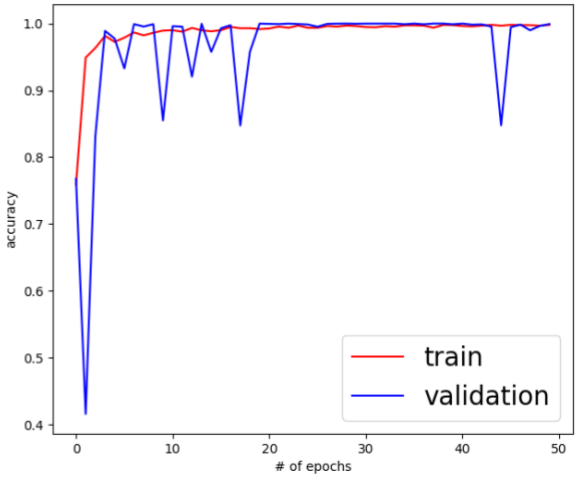
The settings we used for training were: Standard Learning Rate of 0.001, Batch size of 128, and 50 Epochs.

To measure how well the model was learning, we used the Categorical Cross-Entropy loss function. Additionally, we compiled the model with the Adam optimizer, which is a special algorithm designed for Deep Neural Networks to help them learn faster.

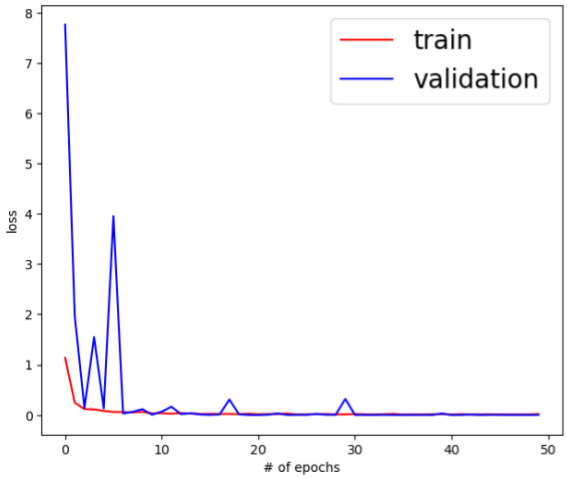
Three different models were trained and tested for an image classification task using CNN12 architecture.

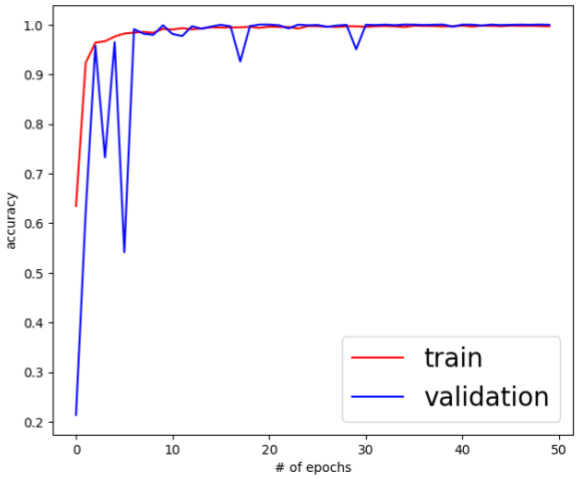
The **first model** was trained without an empty class and without dropout regularization. It achieved the best accuracy and loss values between Epoch 20 to Epoch 41, but using any of these weights resulted in poor performance due to overfitting.



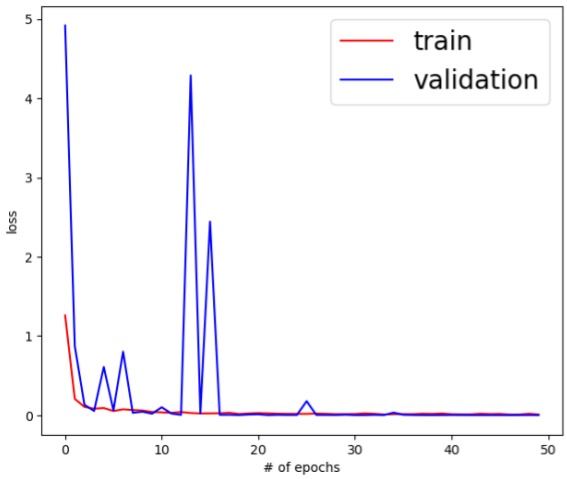


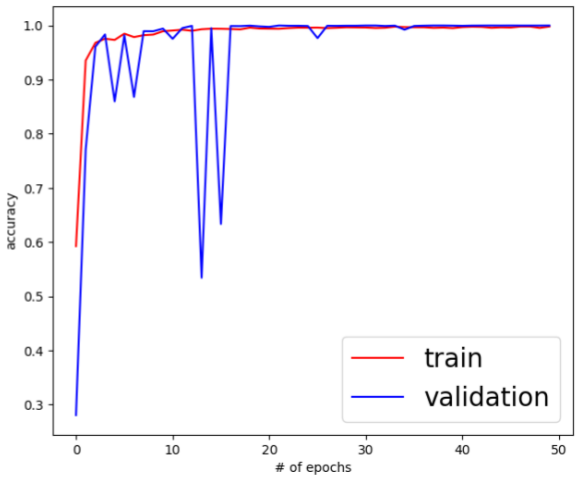
The **second model** was trained without an empty class but with a dropout rate of 30%. Although it achieved the best accuracy and loss values from Epoch 18, using any of these weights led to poor model performance. Also, the model had a problem of recognizing blank input images as deletion signs with 99% accuracy. Therefore, an empty class was added to address this issue.





The **third model** was trained with an empty class and a dropout rate of 30%, achieving the best performance in the research. The weights used for this model were from epoch 19 to epoch 25.





14. Summary and conclusions

After executing the project, we can draw some conclusions regarding the achievement of our goals. Overall, our application was able to recognize handwritten Hebrew letters with an accuracy level of 80% and above. This means that if the model recognized a letter with an accuracy level above 0.8, it was printed to the text box. Otherwise, the letter was only shown in the main window but not printed.

However, our experience with the application highlighted some difficulties in identifying similar signs such as "ת" ,"נ" ,"מ", or "י" ,"ו". These similar signs significantly decreased the accuracy of our model, and the final recognition result of these letters was mostly wrong. To increase the accuracy in these specific signs, we recommend using a larger dataset of various people, along with retraining and evaluation of a new model.

During the project execution, we faced some problems such as the need for a clean white background in the Region of Interest (ROI). This limitation limits the environment in which the software could operate. However, we were able to overcome some of the challenges and achieve our goal of creating a functional application that recognizes handwritten Hebrew letters.

In conclusion, while we were able to achieve our main goal, there is still room for improvement in the recognition process, especially in identifying similar signs. To achieve better accuracy, we recommend implementing our suggestions for future work, including a larger and more diverse dataset, retraining the model, and addressing the limitations of the ROI.

15. Sources List

1. Bhowmik, A., Ahmed, M. U., Mahmud, S., & Uddin, M. S. (2021). A hybrid approach for Bangla sign language recognition using deep transfer learning model with random forest classifier. Neural Computing and Applications, 33(19), 14579-14592. https://ieeexplore.ieee.org/document/9067974
2. Dutta, T. K., & Banerjee, P. (2021). Text2Sign: Towards Sign Language Production Using Neural Machine Translation and Generative Adversarial Networks. In 2021 IEEE Region 10 Symposium (TENSYMP) (pp. 688-693). IEEE. https://www.sciencedirect.com/science/article/pii/S0957417422019327
3. Rajput, M. A., Hussain, S. M., & Naqvi, R. A. (2019). A New Benchmark on American Sign Language Recognition using Convolutional Neural Network. In 2019 International Conference on Robotics and Automation for Humanitarian Applications (RAHA) (pp. 1-6). IEEE. https://link.springer.com/article/10.1007/s11263-019-01281-2
4. Vakade, H. (2021). CNN-for-Image-Classification. GitHub repository. Retrieved from https://github.com/hemavakade/CNN-for-Image-Classification
5. Vasquez, J. (2021). CNN-HowManyFingers. GitHub repository. Retrieved from https://github.com/jaredvasquez/CNN-HowManyFingers