Morse Code to English Translator using Finger Gesture Recognition

Major UG Project Presentation

Department of Computer Science & Engineering,
LBS College of Engineering, Kasaragod

Presented by :
Akarsh S Shetty (KSD17IT001)
Mohammed Jareer K (KSD17IT019)
Nikhil N (KSD17IT021)
Ziad NK (KSD17IT026)

Guide : Rahul C
(Asst Professor)
Dept of CSE

Outline

1. Introduction
2. Literature Survey
3. Theoretical Background
4. Experimental Results & Analysis
5. Screenshots & Outputs
6. Conclusion & Future Work
7. References

Introduction

Problem Statement : In today's fast paced technological advance, it has become difficult for Deaf-Blind people to communicate with Digital Medias. They are overly dependent on others for basic digital tasks like Information Retrieval, Document Preparation, Social Media, SOS message, etc.

Abstract : Morse Code is a detailed system of dots, dashes, and spaces used to represent numbers, punctuation, and letters of the alphabet. It is used both as a code and a way to communicate without the ability or need to use actual characters.

- Finger gestures are a highly controllable action that can be successfully used for Human-Computer Interaction (HCI).

Literature Survey

PAPER - I

Title: **Analysis of Vision-Based Text Entry using Morse Code generated by Tongue gestures.**

Authors: **Luis Ricardo Sapaico and Makoto Sato - Tokyo Institute of Technology, Tokyo, Japan.**

Objective : Proposed a Text Entry Interface based on the detection of tongue protrusion gestures using Computer Vision methods.

Result : A text entry system was developed by using Morse Codes with tongue gestures, therefore, it is possible to ‘type’ a letter by using human tongues; the system gained 84.78% overall accuracy. According to the tests for the disabled, preliminary results showed that the system has usability, usefulness and universality.

PAPER - II

Title: **Online Hand Gesture Recognition Classification for Deaf & Blind**

Authors: **Nitesh S. Soni, Prof. Dr. M.S. Nagmode, Mr. R. D. Komati-MIT, Pune, India**

Objective : A fingertip with a bright visual colored glove is applied to finger gesture recognition as an input; seven gestures can be successfully identified.

Result : A static-hand recognition of international sign language was implemented by using principal component analysis (PCA). The methods can recognize 25 hand gestures. The recognition rate is approximately 99% compared to other models, but the gestures must be shown in front of a black background to increase the overall recognition rate.

PAPER - III

Title: **Morse Codes Enter Using Finger Gesture Recognition**

Authors : **Ricky Li, Minh Nguyen, Wei Qi Yan- Auckland University of Technology, New Zealand**

Objective : Analysis of Morse Codes enter using the finger to desk-tap contact recognition using RBF classifier & BPNN Machine Learning Algorithms.

Result : RBF-based rate of finger gesture recognition for Morse codes is above 84% and for the BPNN algorithm, the overall recognition rate is about 76%. If the distance between a camera and a finger is too far, the object usually will be very small and unclear.

PAPER - IV

PAPER - V

Our proposed approach

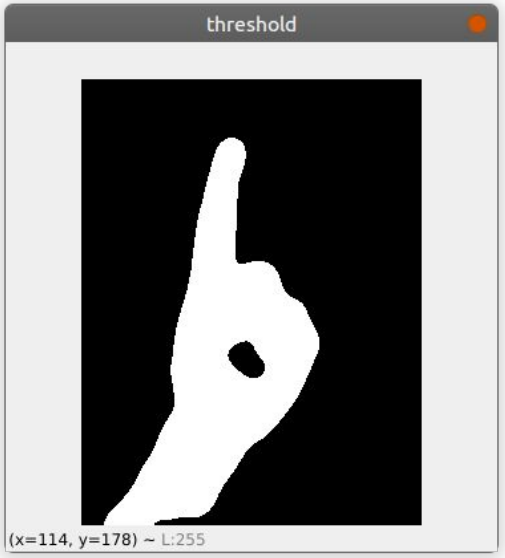
Step 1. Classification of fingers to different classes. And each class signifies the corresponding Morse code character.

Finger Gesture	**Morse Code Character**
1 fingers	●
2 fingers	-
3 fingers	Reset/ Start
4 fingers	Space
5 fingers	Stop and predict the character

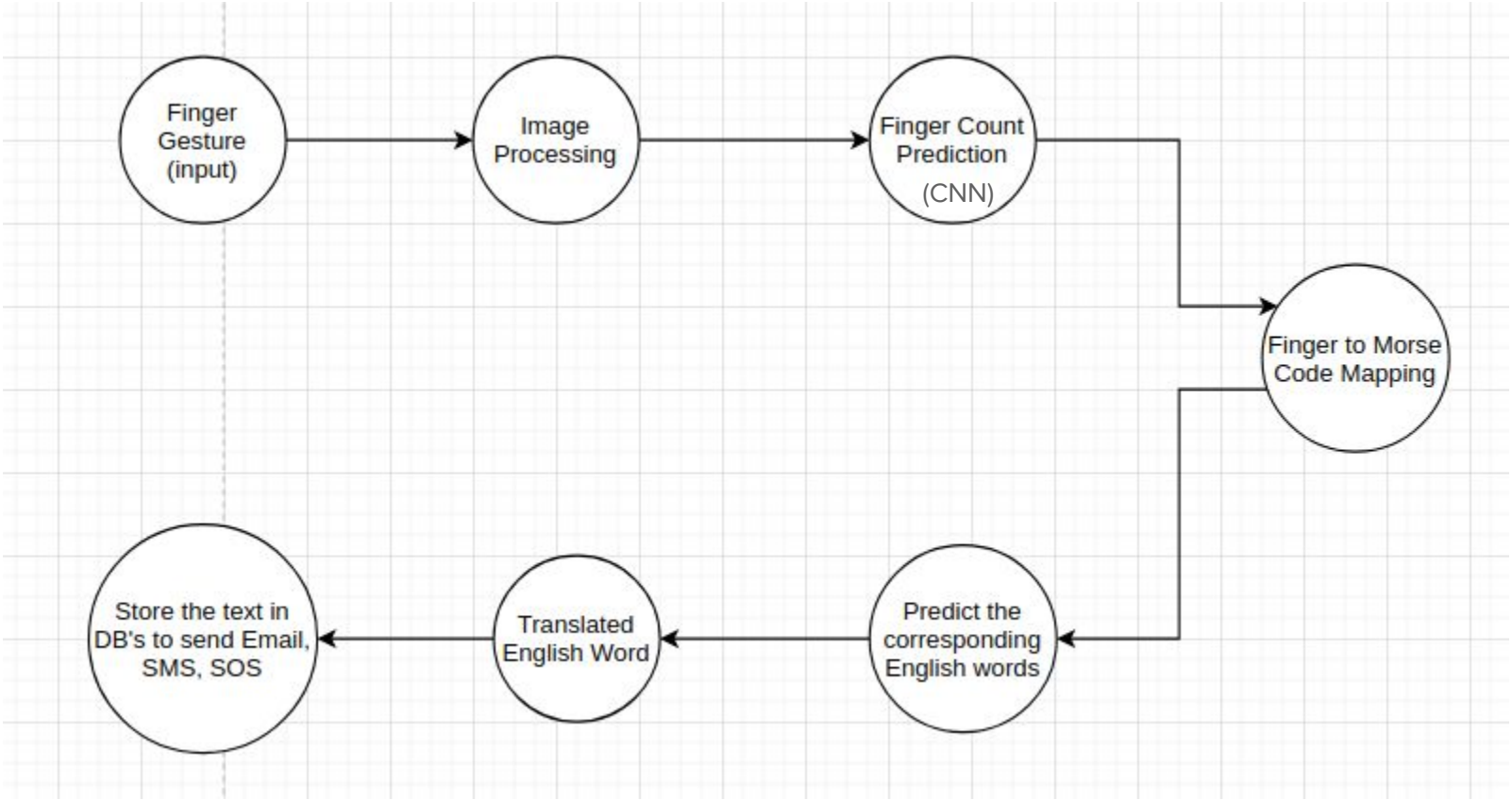
Step 2. Detect (the input) fingers using Computer Vision techniques

Step 3. Predict the character for the detected finger(s) gestures using (CNNs) Convolutional Neural Networks

Step 4. Store the text in the database for entering text, SMS, e-mail, SOS, Information Retrieval etc



Flowchart Representation



Challenges faced

To develop a robust dialogue system based on Morse codes, there are several challenges such as

- First challenge : **finger colours, environment, lighting conditions, and imitations of various Algorithms.**
- Second challenge : is to predict the morse code from **large no. of frames** - solved it by reducing the frame rate & introduced mechanisms like start & end of a character
- **Zero fingers** - to differentiate between dits and dahs

Requirements (Tools/ Libraries)

Python

Tensorflow

Keras

Open CV

Matplotlib

Sci-py

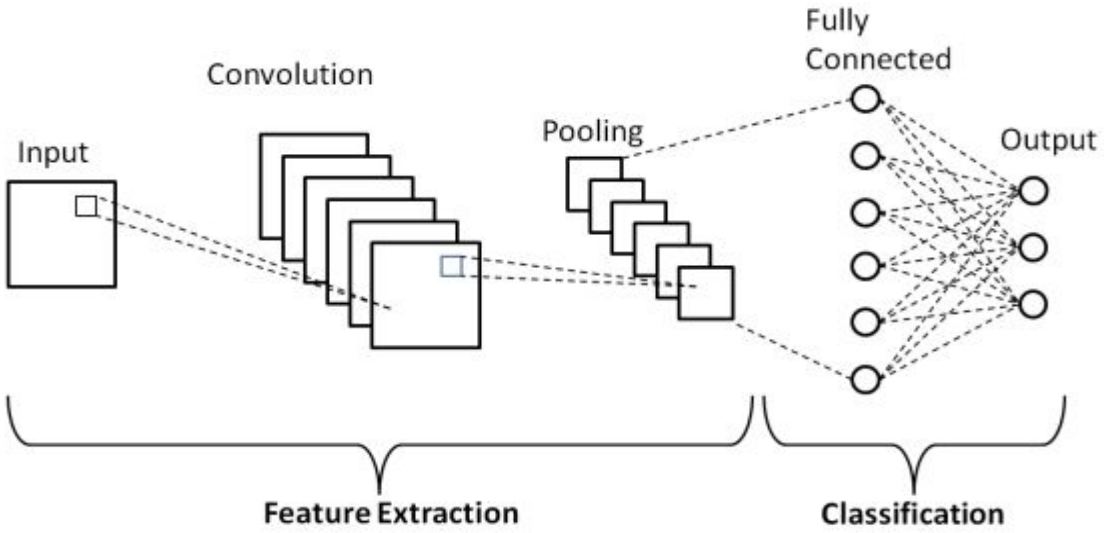
Jupyter Notebook

Theoretical Background

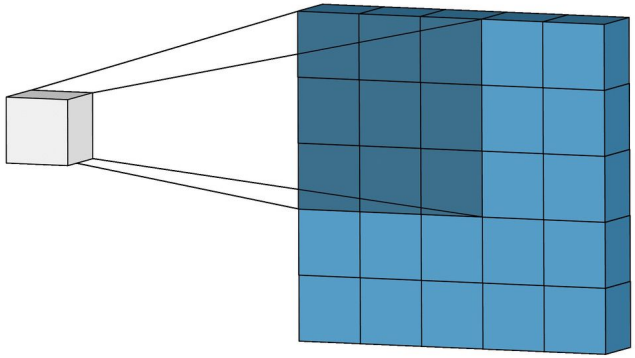
What Are Convolutional Neural Networks?

- Convolutional Neural Networks, like neural networks, are made up of **neurons** with **learnable weights** and **biases**.

- Each **neuron** receives several inputs, takes a weighted **sum** over them, **pass** it through an **activation function** and responds with an **output**.



Why Convolutional Neural Networks?



1. **Feature engineering** not required.

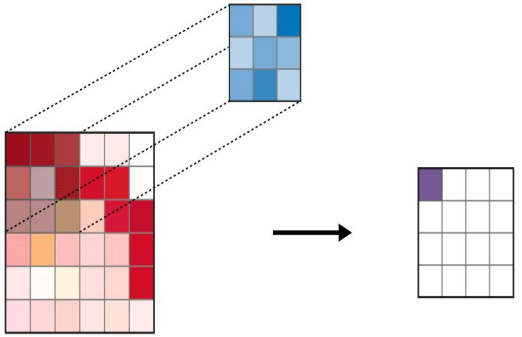
2. When we compare handcrafted features with CNN, **CNN performs well** and it gives better accuracy.

3. Covers **local** and **global** features. It also learns different features from images.

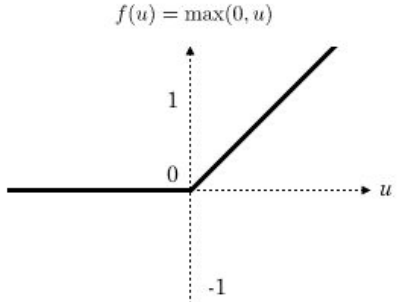
4. Concept of **dimensionality reduction** suits the huge number of parameters in an image.

ConvNet Layers (i)

CONVOLUTION layer : are the layers where filters are applied to the original image, or to other feature maps in a deep CNN.



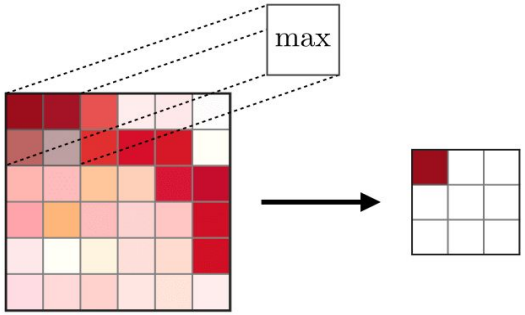
ReLU layer : In this layer we remove every negative values from filtered image & replace it with 0's. This is done to avoid the values from summing up to 0.



POOL layer : Reduces dimensionality;

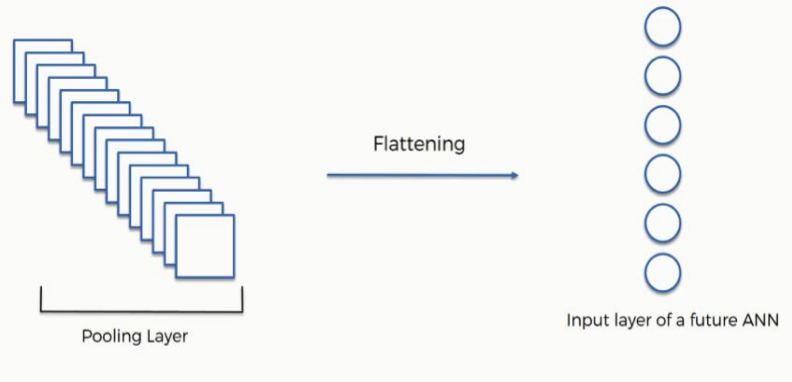
Sliding Window approach (Strides)

From each window, takes the max value.



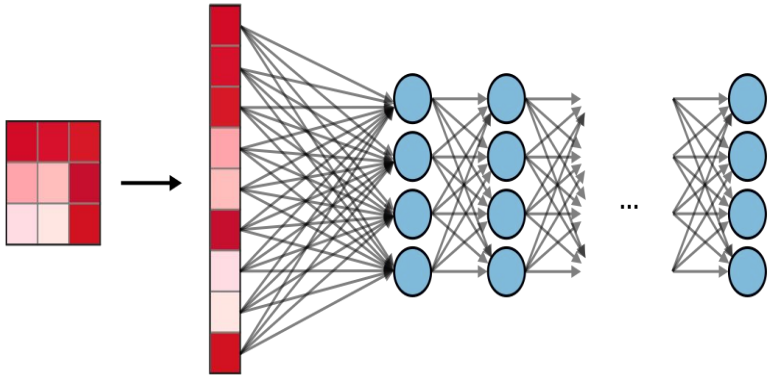
ConvNet Layers (ii)

FLATTENING layer : involves transforming the entire pooled feature map matrix into a single column which is then fed to the neural network for processing.



FULLY CONNECTED layer : aggregate information from final feature maps;

Generate final classification



Computer Vision

Computer Vision is a field of study focused on the problem of helping computers to see.

Image processing is the process of creating a new image from an existing image, typically simplifying or enhancing the content in some way.

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.

It is cross platform and aids in real-time computer vision tasks.

Framework Approach

1. Data collection
   - ➔ Defining the problem and assembling a dataset
2. Data preparation
   - ➔ Preparing your data
3. Choose model
4. Train model
   - ➔ Developing a model that does better than a baseline
5. Evaluate model
   - ➔ Choosing a measure of success
   - ➔ Deciding on an evaluation protocol
6. Parameter tuning
   - ➔ Scaling up: developing a model that overfits
   - ➔ Regularizing your model and tuning your parameters
7. Predict

Data Collection

- We are making use of pre-collected data.
- Fingers dataset from Kaggle : https://www.kaggle.com/koryakinp/fingers
- Author : Pavel Koryakin

0

1

2

3

4

5

Data Preparation - (i)

Task : Multiclass Classification

Content : 31200 images of left and right hands fingers.

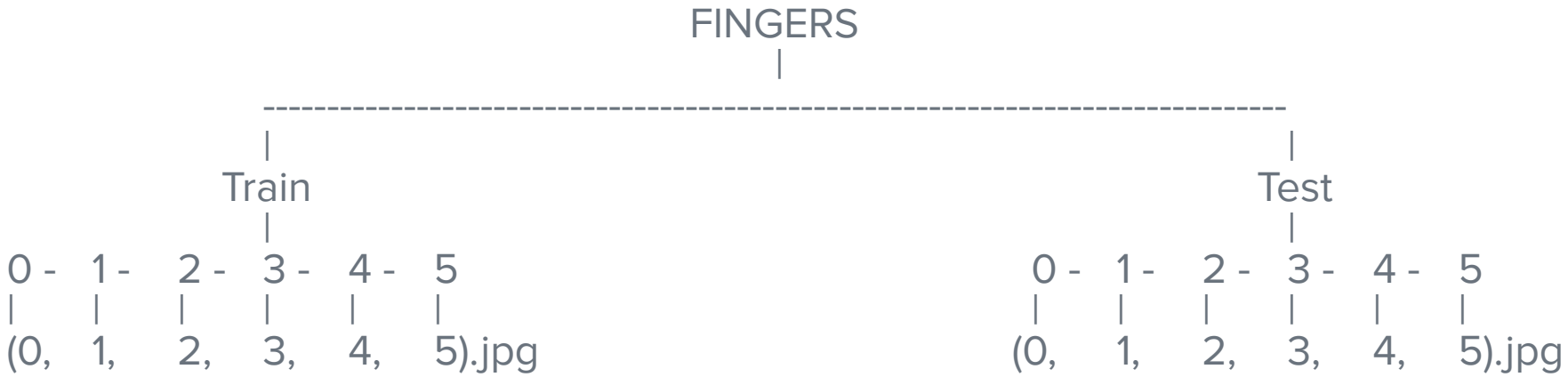
All images are 128 by 128 pixels.

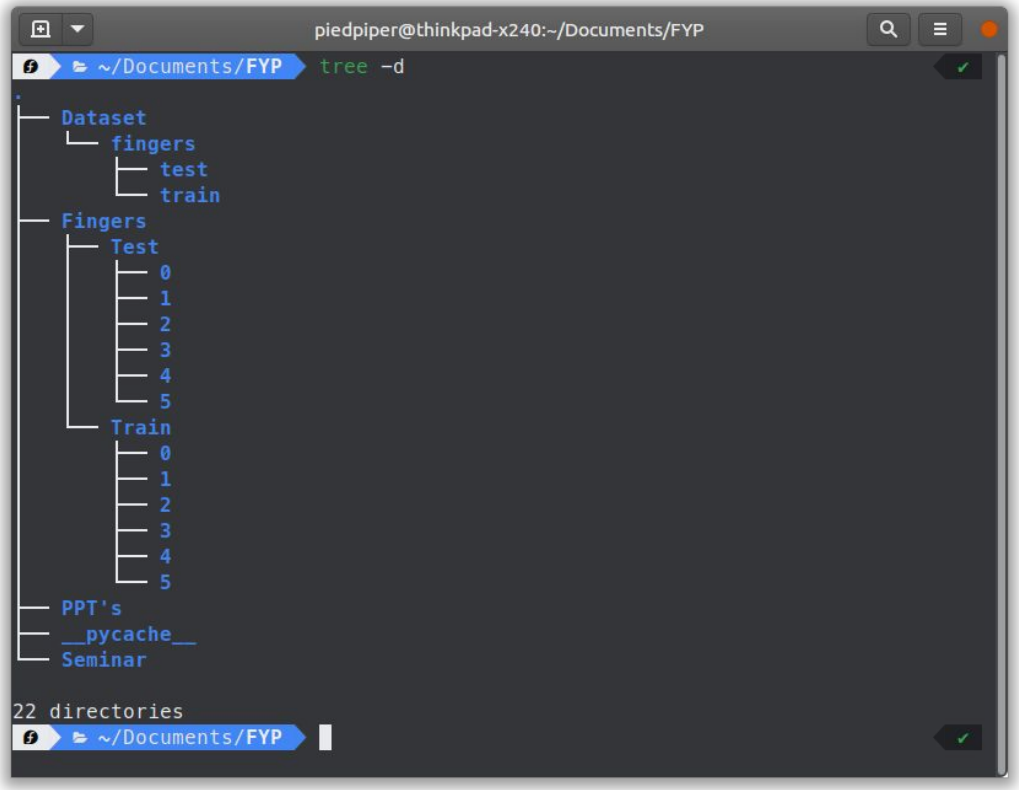
- Training set: 22200 images
- Test set: 9000 images
- Images are centered by the center of mass
- Noise pattern on the background

Note : Images of left hand were generated by flipping images of right hand.

Data Preparation - (ii)

- Fingers is the Root folder and inside that we have two folders called Train and Test.
- Using Tensorflow's Image Data Generator to generate branches of tensor image data with real-time data augmentation (**tf.keras.preprocessing.image.ImageDataGenerator**).
- Dataset should be in this format :





Directory Structure

Model Working

Two Step Process :

1. Detect number of fingers opened.
   - Image Generator - (Tensorflow)
   - Model the Architecture - (CNN)
   - Train Model
2. Generate Corresponding Morse Code.
   - Set Parameters for hand detection (ROI) - (OpenCV)
   - Load the trained model from previous process
   - Capture Video - (OpenCV)
   - Predict the Morse Code

Process-1 : Model Architecture - CNN (ConvNet 2D)

- We'll use Conv2D because we are converting our dataset into Gray Scale images.
- Keras is a deep learning API written in Python, running on top of the machine learning platform TensorFlow.
    - Scalable
    - User friendly and fast deployment
    - Deeply integrates with Tensorflow ML framework.
    - Cross-platform
- TensorFlow 2.0 is an end-to-end, open-source machine learning platform. You can think of it as an infrastructure layer for differentiable programming.

Overview of CNN implementation using Keras - (i)

1. Sequential Model : A linear stack of arrays (initializes Neural Network)

```
from tensorflow.keras.models import Sequential

model = Sequential()
```

2. Convolution2D : is used to make the convnet that deals with the images

```
tf.keras.layers.Conv2D(32, (3,3), activation = 'relu', input_shape=(150,150,1))
```

3. MaxPooling2D : layer is used to add the pooling layers.

```
tf.keras.layers.MaxPooling2D(2, 2)
```

4. Flatten : is the function that converts the pooled feature map to a single column that is passed to the fully connected layer.

```
tf.keras.layers.Flatten()
```

Overview of CNN implementation using Keras - (ii)

5. Dense : adds the fully connected layer to the neural network.

```
tf.keras.layers.Dense(512, activation = 'relu')
```

6. Dropout : is easily implemented by randomly selecting nodes to be dropped-out with a given probability each weight update cycle.

```
tf.keras.layers.Dropout(0.3)
```

7. Once your model looks good, configure its learning process with .compile():

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

CNN Model Architecture

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 148, 148, 32)      320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 72, 72, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 34, 34, 128)       73856
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 17, 17, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 512)               3211776
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 6)                 3078
=================================================================
Total params: 3,455,110
Trainable params: 3,455,110
Non-trainable params: 0
_________________________________________________________________
None
```

Train the model

8. **Train the model using model.fit()**

```
model.fit(train_data,
          steps_per_epoch = no_of_steps_per_epoch,
          epochs = no_of_epochs,
          validation_data = test_data,
          validation_steps = no_of_validation_steps)
```

9. **Evaluate your test loss and metrics in one line:**

```
loss_and_metrics = model.evaluate(testing)
```

Model

Two Step Process :

1. Detect number of fingers opened.
   - Image Generator - (Tensorflow)
   - Model the Architecture - (CNN)
   - Train Model
2. Generate Corresponding Morse Code.
   - Set Parameters for hand detection (ROI) - (OpenCV)
   - Load the pre-trained model from previous process
   - Capture Video - (OpenCV)
   - Predict the Morse Code

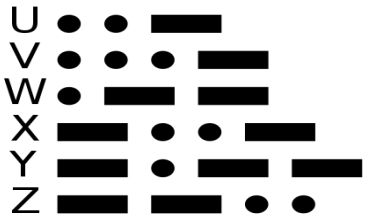
Process 2 : Set Parameters for hand detection

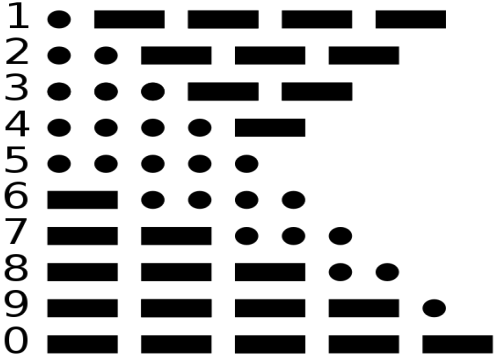
- **Mask window** will show you the masked image of your hand by removing the background.
- **Blur Window** will show you the blur grey image of your hand by removing the background.
- **Threshold Window** will show you the Threshold image of your hand.
- **Predicted Window** will show the predicted number of fingers opened.
- **Test Window** will show the predicted text from morse code.
- If you open **One finger** it is small beep(Dot in Morse code).
- If you open **Two fingers** it is Long beep(Dash in Morse code).
- If you open **Three fingers** it is to start capturing for a new character or reset the current character.
- If you open **Five fingers** it is to Stop capturing and predict the character from morse code pattern.
- If you open **Four fingers** it is to add space between characters.

Morse Code (dits & dahs representation)

International Morse Code

1. The length of a dot is one unit.
2. A dash is three units.
3. The space between parts of the same letter is one unit.
4. The space between letters is three units.
5. The space between words is seven units.





Morse Code Rep^{n} using Python Dictionaries

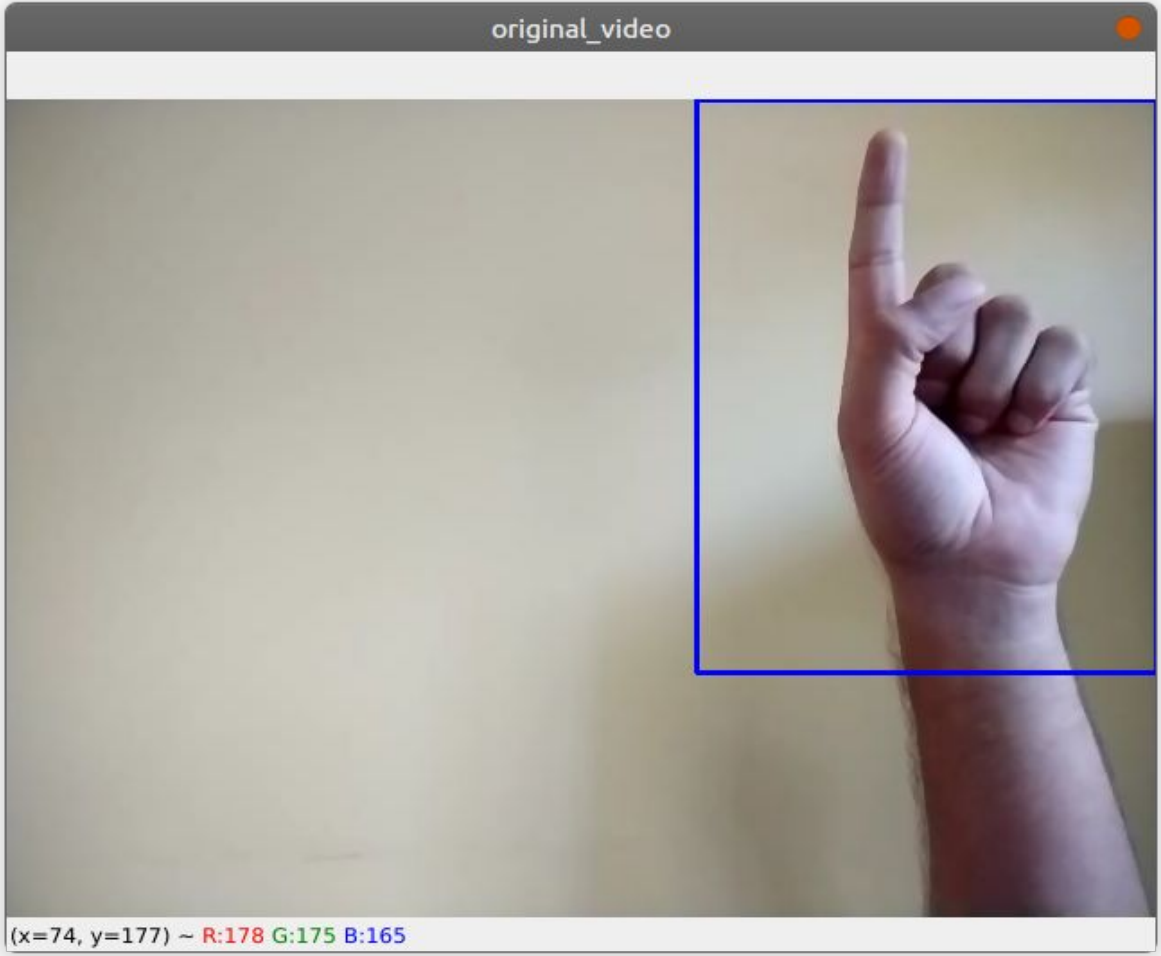
```
{"12":"A",    "1111":"H",   "222":"O",    "1112":"V",   "11122":"3",
"2111":"B",   "11":"I",     "1221":"P",   "122":"W",    "11112":"4",
"2121":"C",   "1222":"J",   "2212":"Q",   "2112":"X",   "11111":"5",
"211":"D",    "212":"K",    "121":"R",    "2122":"Y",   "21111":"6",
"1":"E",      "1211":"L",   "111":"S",    "2211":"Z",   "22111":"7",
"1121":"F",   "22":"M",     "2":"T",      "1222":"1",   "22211":"8"
"221":"G",    "21":"N",     "112":"U",    "11222":"2",  "22221":"9",
                            "22222":"0"}
```

Original Video Window

It takes live feeds from webcam as input.

Fingers are to be shown in the blue rectangular box.

- ➔ ‘b’ to start capturing
- ➔ ‘r’ to Reset the character pattern
- ➔ ‘ESC’ to exit

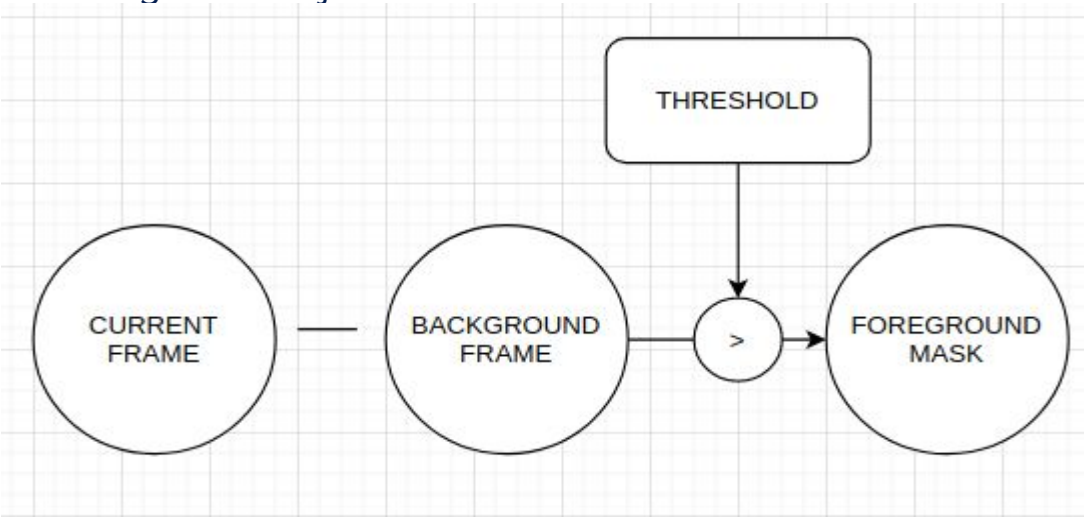


Masking

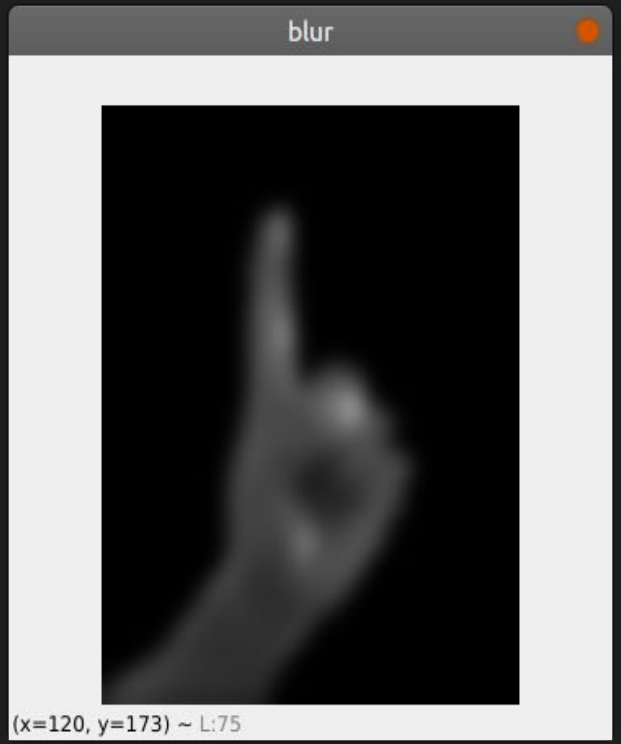
A **mask** allows us to focus only on the portions of the image that interests us.

Background Subtraction : It's a common & widely used technique for generating a foreground mask.

Erosion : It erodes away the boundaries of the foreground objects



Blurring



Converts BGR image to Grayscale.

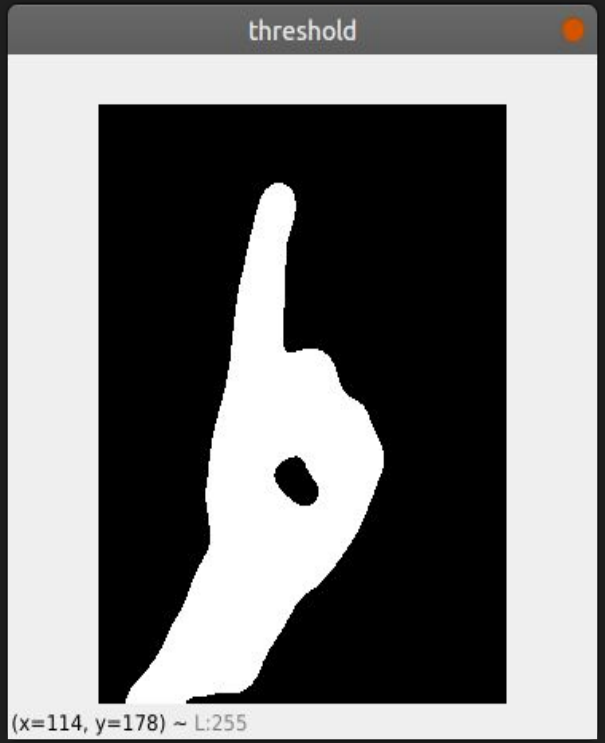
Applied before Edge Detection/ Thresholding to reduce the amount of high-frequency content - noise & edges.

It helps in **smoothing** the image.

Gaussian Blur : Uses weighted mean to get a naturally blurred images. Makes use of Gaussian Kernel.

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (self.blur_alue,
self.blur_alue), 0)
cv2.imshow('blur', blur)
```

Thresholding



Thresholding : Separates foreground from the background.

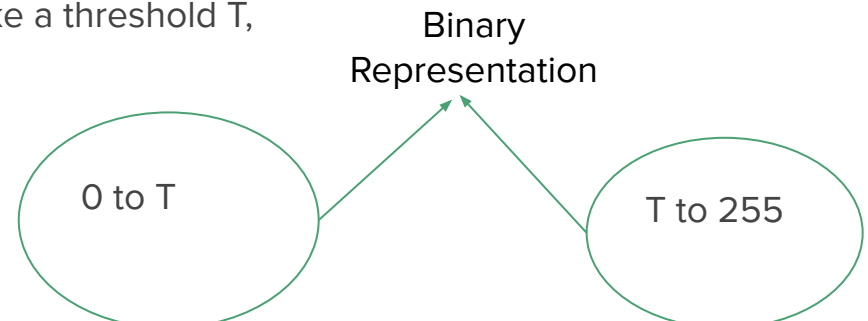
Image Segmentation method; based on intensity, the pixels in an image get divided by comparing the pixel's intensity with a threshold value.

Binarization of image :

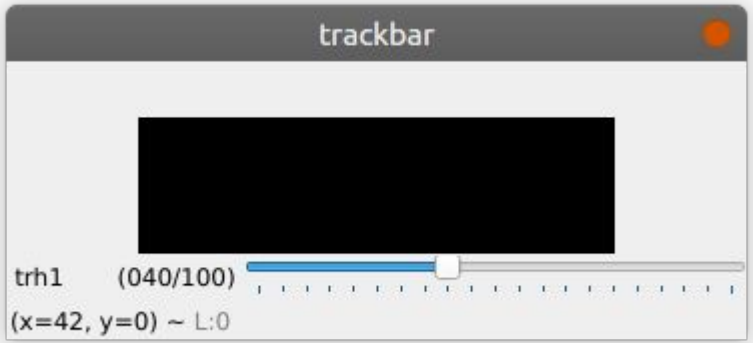
Grayscale images → Binary Image Representation

Pixel values (Intensity) : **0 - Black**; **255 - White**

Take a threshold T,

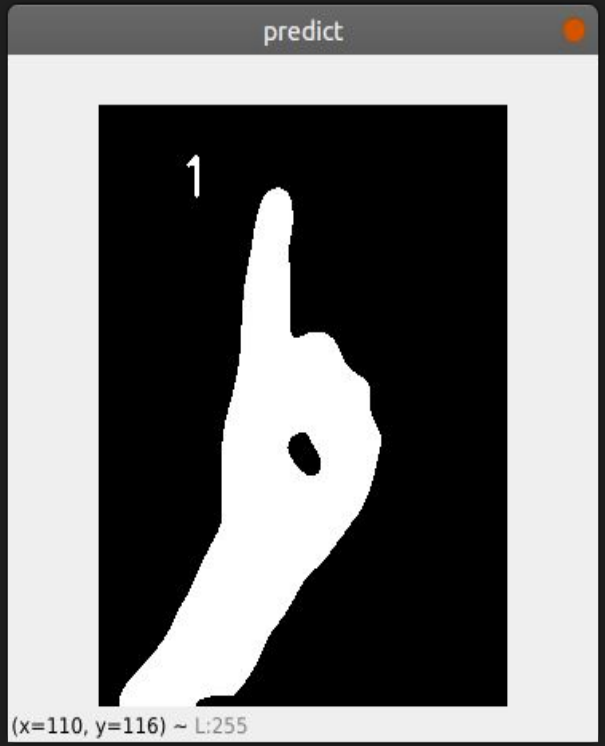


After **Masking** and **Blurring** each frame is sent through two filters one is a grey image and another is threshold image. We will be using this threshold image and pass it to the **pre-trained model** to detect no of fingers opened.

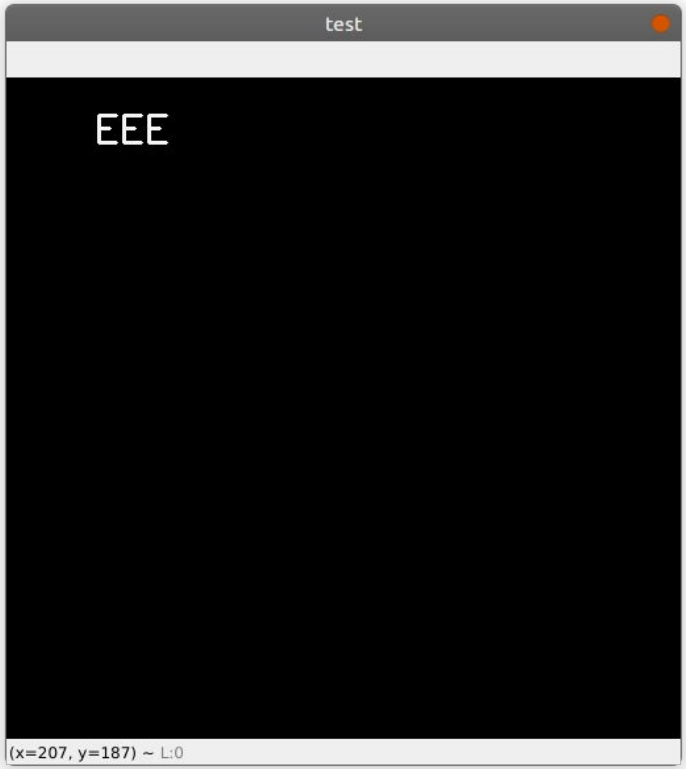


Trackbar : This window allows the user to change threshold values (T) in real-time.

Prediction



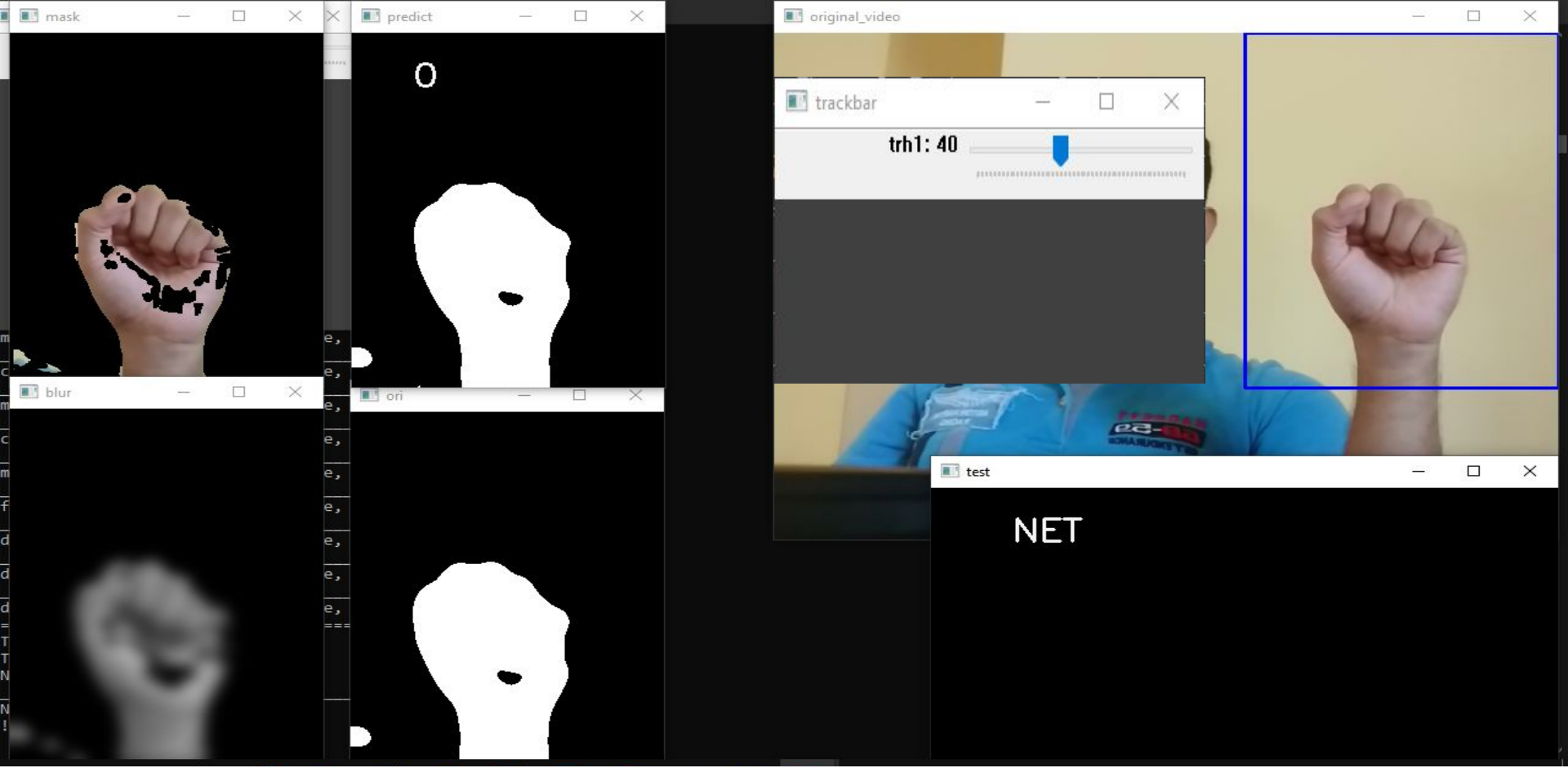
Translated Text Window



This window shows the **translated** Morse code in English.

Based on **no. of fingers** opened Morse code will be generated.

Screenshots and Outputs



Morse Code to English Translation

Conclusion and Future Work

Conclusion : The deaf-blind people will benefit from this application, if perfected. Hence, supporting their independence.

Future Work :

- Further improve the current logic
- Minimize use of resources
- Extend the functionality to a chatbot (AI)
- Deploy it as an Android app or Web app
- Implement using eyes, tongue and other Human Computer Interaction Mechanisms

References

1. Pradumn Kumar, Upasana Dugal, “Tensorflow Based Image Classification using Advanced Convolutional Neural Network”, International Journal of Recent Technology and Engineering (IJRTE) ISSN: 2277-3878, Volume-8 Issue-6, March 2020
2. Anirudh Rao, Edureka (CNN) – Developing An Image Classifier In Python Using TensorFlow
3. Diederik Kingma from OpenAI, Jimmy Ba from University of Toronto, “Adam: A method for stochastic optimization”, 2015 ICLR
4. Keras Documentation, Google Tensorflow Documentation, OpenCV Documentation, Kaggle

Thank You