DL model optimization for Lightweight Gesture Recognition

Goswami Pranav (B20CS016), Harshita Kalani (B20CS019)

May, 2023

Abstract

The project presents the development of a multiplatform application (Mobile Application, Desktop Application and website) for hand gesture detection using deep learning models. The system can be used to recognize and classify hand gestures captured through a camera on various platforms such as desktop, mobile, and embedded systems. The application uses convolutional neural network (CNN) models trained on large datasets of hand gesture images to detect and recognize a variety of gestures. The system is designed to be fast and efficient, allowing real-time recognition of hand gestures with high accuracy. The results show that the application can be used in a wide range of applications, including human-computer interaction, virtual reality, and robotics. The proposed system demonstrates the potential of deep learning techniques in creating efficient and accurate hand gesture recognition systems.

1. Introduction

Hand gesture recognition is the process of automatically identifying hand movements and translating them into meaningful commands or actions. With the increasing prevalence of mobile devices and microcontrollers, the ability to recognize gestures in real-time has become increasingly important for a wide range of applications, including augmented reality, virtual assistants, and gaming. In this project, we have used deep learning models to train the dataset for lightweight gesture recognition, which can operate efficiently on resource-constrained devices. We have used various techniques to reduce the size and complexity of the model, while maintaining high accuracy and real-time performance.
The project involves the preprocessing of a dataset of hand gestures, designing and training deep learning models and using the trained dataset to create applications which can run on multiplatforms such as mobile phone, desktop and web with minimal power consumption and memory usage.
Overall, the project aims to contribute to the development of lightweight gesture recognition systems that can enable new forms of human-computer interaction on resource-constrained devices.

2. Dataset Description

We have utilized two different datasets to train and test our models. The first dataset contains 10 labels and the other contains 3 labels, namely, swing, fist and plam. The deployed models are trained over the dataset containing 10 labels.

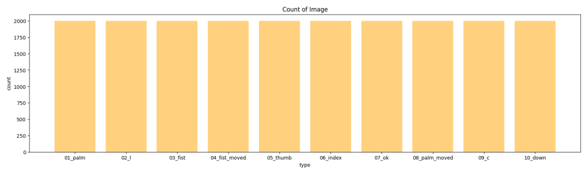


Figure 1: Visualising count of each class

2.1. Hand Gesture Recognition Database

We are using Hand Gesture Recognition Database for efficient training of various Deep learning models. It contains a total of 20,000 images and 10 labels named as *palm, L, fist, moved, thumb, index, ok, palm moved, C, down*, each containing 2000 images. Hand gesture recognition database is composed by a set of near-infrared images acquired by the Leap Motion sensor.

2.2. Hand Gesture Detection 3 classes

This dataset is imported from Hand Gesture Recognition Using Background Ellimination and Convolution Neural Network that contains 3 classes named *thumb, fist, swing* each containing 2000 images summing to a total of 6000 images.

3. Data Visualisation

From Figure 1 we can observe that count of each class is equal to 2000 in all 10 classes.

4. Data preprocessing

In order to prepare the dataset for training our deep learning model, we first imported the data

Figure 2: Visualisation of data items

Figure 3: Visualisation of data items

from a drive. The dataset consisted of images that were annotated with the corresponding hand gestures. However, the imported images were in color, which may not be helpful in identifying the hand gestures accurately. Therefore, we converted the images to grayscale and RGB formats, which are more suitable for deep learning models.
Additionally, we found that the images in the dataset had varying sizes, which could potentially cause issues during training. To address this, we resized all images to a fixed size of 224x224 pixels. This ensures that the images have a consistent size, which allows the model to learn more effectively from the data.
We also used the U-Net segmentation algorithm to segment the hand gesture images. Image segmentation is the process of dividing an image into multiple segments or regions, each of which represents a different object or feature in the image.
The Unet algorithm is a popular deep learning technique used for image segmentation, which involves training a deep neural network to predict the segmentation masks for the input images. By segmenting the hand gesture images, we were able to isolate the hand from the background, which can greatly improve the accuracy of the model in recognizing hand gestures.
Once the images were preprocessed, we imported them into TensorFlow, a popular deep learning framework, using its built-in functions for loading image datasets. This allowed us to easily access and manipulate the dataset within our Python code.

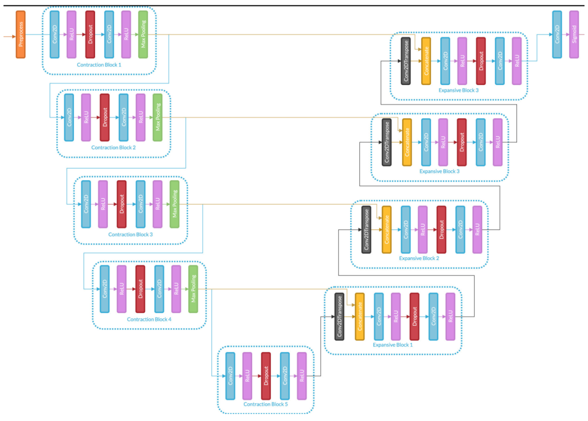


Figure 4: U-Net architecture

5. Segmentation using UNet Architecture

The U-Net architecture for image segmentation has two parts: the contracting path captures the context of the input image using convolutional layers with max pooling, while the expanding path localizes the objects of interest by upsampling feature maps and concatenating them with corresponding features from the contracting path. This is followed by more convolutional layers to refine the segmentation masks. Finally, a 1x1 convolutional layer produces the final segmentation mask with a label for each pixel indicating the object class.
In Figure 4, Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

6. Deep Learning Models

The imported dataset was converted into tensors of batch size 32 each and they were then used for training and validating the models

6.1. VGG 19

Very Deep Convolutional Networks designed to carry out Large-Scale Image Recognition baseline model is used in our multi-class classification as it attains a significant accuracy on image classification and localization tasks. Due to its inherent strength in processing X-Ray image recognition, we used it for our experiments. We implemented the Transfer Learning technique on the VGG-19 (baseline) and customized the VGG-19 model during the training on X-Ray datasets. Pre-trained weights of the ImageNet dataset were used. We used discriminative learning rates to preserve the lower-level features and regulate the higher-level features for optimum

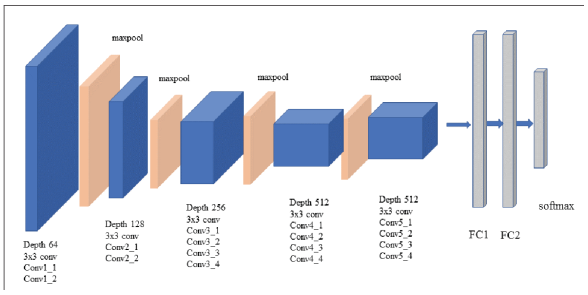


Figure 5: VGG 19 architecture

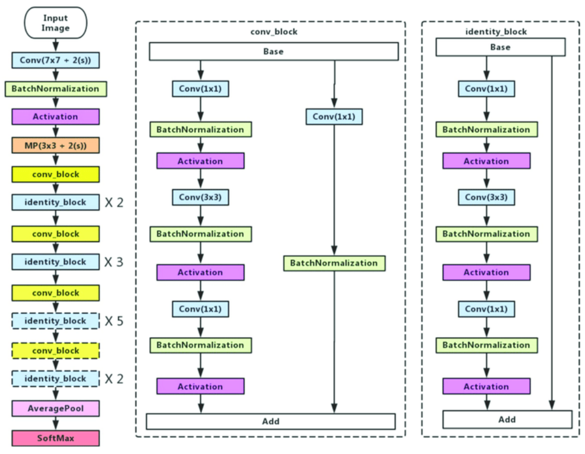


Figure 6: ResNet 50 architecture

results. VGG19 is a variant of VGG model which in short consists of 19 layers (16 convolution layers, 3 Fully connected layer, 5 MaxPool layers and 1 SoftMax layer). To reduce the time and space complexity we removed some of the convolution layers and used the 3 fully connected layer, 5 MaxPool layers and 1 SoftMax layer of 2 Neurons as it is binary classification problem. The model was compiled using "adam" optimiser and "categorical cross entropy" as loss function.

6.2. ***ResNet50***

ResNet is known to be a better deep learning architecture as it is relatively easy to be optimized and can attain higher accuracy. Furthermore, there is always a problem of vanishing gradient, which is resolved using the skip connections in the network. As the number of layers in the deep network architecture in- creases, the time complexity of the network increases. This complexity can be reduced by utilizing a bottleneck design. As a consequence, ResNet50 is a pre- ferred pretrained model using weights from imagenet to build up our framework. The model is compiled using stochastic gradient descent optimizer and sparse categorical cross entropy as loss function.

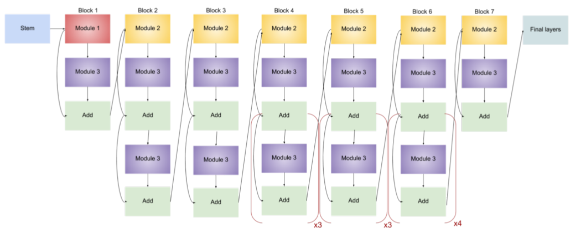


Figure 7: EfficientNet B3 architecture

6.3. ***EfficientNet B3***

he EfficientNet network is based on a novel scaling strategy for CNN mod- els. It employs a straightforward compound coefficient that is quite successful. Unlike existing approaches that scale network parameters such as width, depth and resolution, EfficientNet evenly scales each dimension with a given set of scaling factors. Scaling individual dimensions increases model performance in practice, but balancing all network dimensions in relation to available resources significantly enhances overall performance. Pre-trained weights from imagenet were used to load the model and then is compiled using stochastic gradient descent optimizer and sparse categorical cross-entropy as loss function

6.4. ***CNN Implemented in Paper***

We have tried to perform a scratch implementation of CNN presented in research paper. The architecture of the deep learning model that we used for lightweight gesture recognition is a Sequential model with multiple layers. The first layer of the model is a Convolution 2D layer with a filter size of 64 and a 3x3 convolution kernel. The input shape for this layer is set to 224x224x1. The output of the first Convolution 2D layer is fed into a max pooling layer of size 2x2, which performs a downsampling operation on the output. This is followed by another identical Convolution 2D layer, which is also passed through a max pooling layer.
To prevent overfitting and improve generalization, we applied dropout regularization to the output of the second max pooling layer, dropping out 25% of the data points randomly. The output of the dropout layer is then flattened, which converts the 3D tensor output into a 1D vector.
The flattened output is then fed into a Dense layer with 256 neurons, which applies a linear transformation to the input data. Similar to the previous layer, we applied dropout regularization with a rate of 25% to the output of the Dense layer.
The same block of Convolution 2D, max pooling, and dropout layers are repeated again, followed by another Dense layer with 10 neurons and softmax

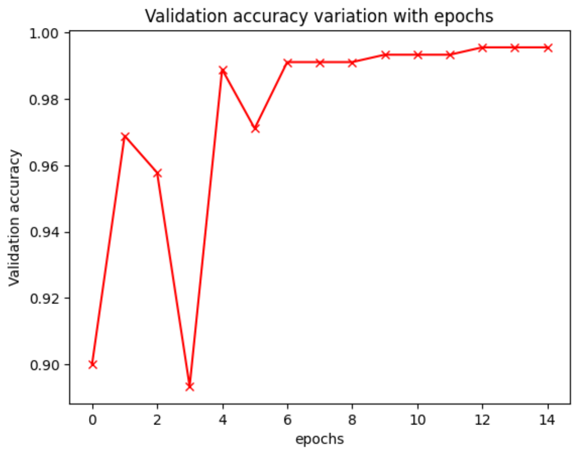


Figure 8: Validation Accuracy vs Epochs

activation, which performs the final classification of the input hand gestures into different classes. In the intermediate layers, we used the rectified linear unit (ReLU) activation function, which is a common choice for deep learning models due to its computational efficiency and ability to handle sparse inputs. The final layer uses the softmax activation function, which normalizes the output of the layer into a probability distribution over the different classes.

7. Comparative Analysis

In this section, we have provided validation accuracy, loss of each model applied in form of graphs and tables.

7.1. *VGG 19 Results*

We were able to achieve validation accuracy upto 99.5556% on epoch 12 and validation loss came out to be 0.024584. It can be seen from Figure 8 and Figure 9 that accuracy increases and loss decreases as we go higher on each epoch.

7.2. *Scratch CNN Results*

We were able to achieve validation accuracy upto 0.973333% at epoch 12 and validation loss came out to be 0.093229. It can be seen from Figure 10 and Figure 1 that accuracy increases and loss decreases as we go higher on each epoch.

8. Deployment

Based on a comparative analysis, it was determined that a Convolutional Neural Network (CNN) implementation from a research paper was the best choice for deployment. This implementation was found to be robust, preventing both overfitting and

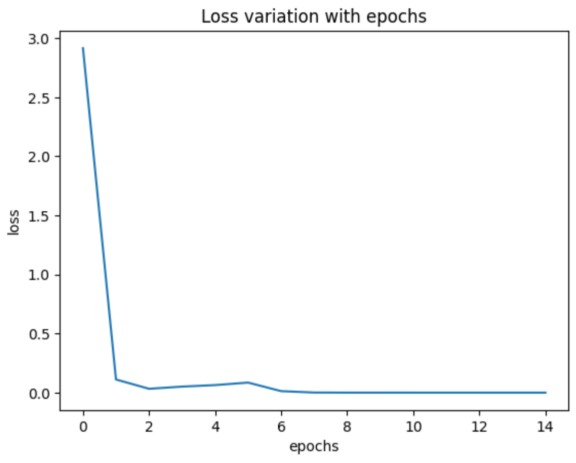


Figure 9: Loss vs Epochs

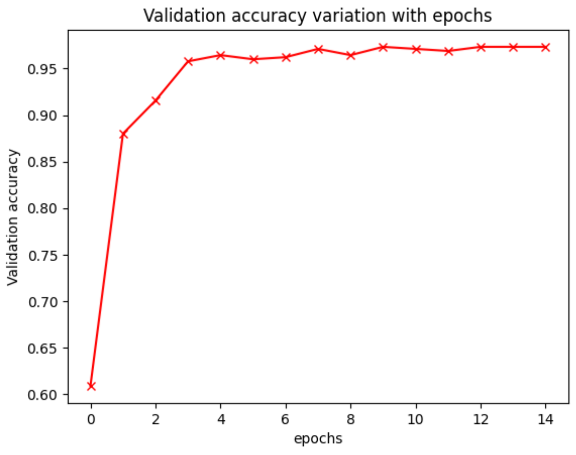


Figure 10: Validation Accuracy vs Epochs

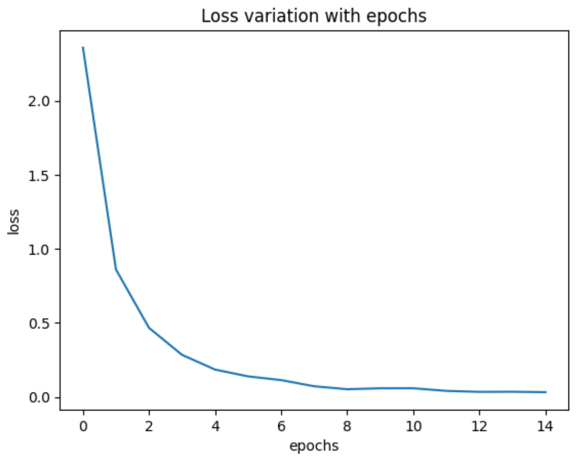


Figure 11: Loss vs Epochs

Model Name	Validation Accuracy	Validation Loss
VGG-19	0.995556	0.024584
ResNet 50	0.971111	0.298653
EfficientNet B3	1.0	0.000001
CNN (from paper)	0.973333	0.082041

Table 1: Comparative Analysis

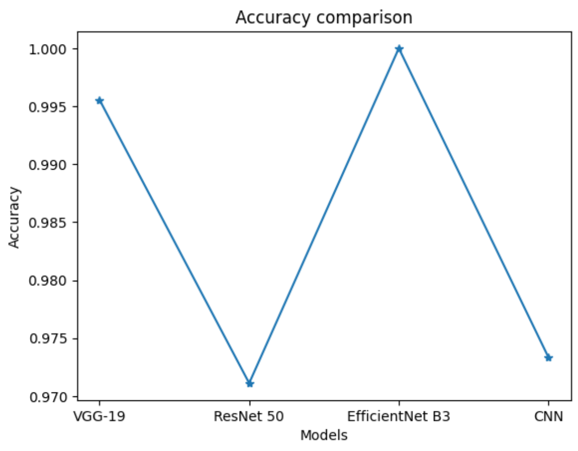


Figure 12: Validation Accuracy for different models

underfitting, and the layers were properly managed with the incorporation of dropout features. The decision to use this implementation was likely made because it demonstrated strong performance in testing and met the specific requirements of the project.
A multiplatform application was created that utilizes peripherals such as a camera and file explorer to allow users to select images. The application then uses saved weights to perform predictions. The project resulted in the creation of a Mobile Application, Desktop Application, and Web Application. The web application has been successfully deployed and is currently operational. The outcomes of the project can be viewed by visiting the deployed web application link. The use of multiple platforms allows for wider accessibility and ease of use for users across various devices.

8.1. *Mobile Application*

Flutter, a framework based on Dart, was utilized to develop mobile applications for both Android and iOS users. The trained model was saved using TensorFlow, specifically by using the command *model.save_weights('cnn_.h5')*, to an h5 format. Existing libraries within Flutter were used to access the camera and file explorer of the user's phone, perform the required preprocessing, and finally, *model.predict(image)* was used to detect the gesture and confidence. Images of the application

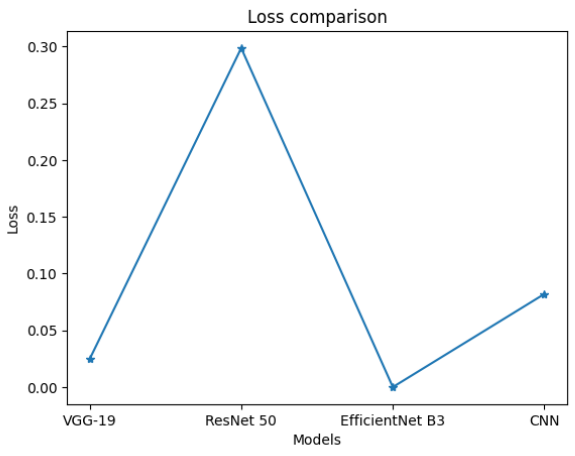


Figure 13: Validation loss for different models

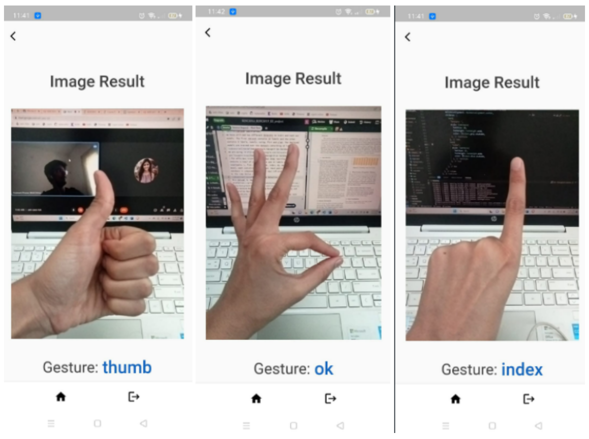
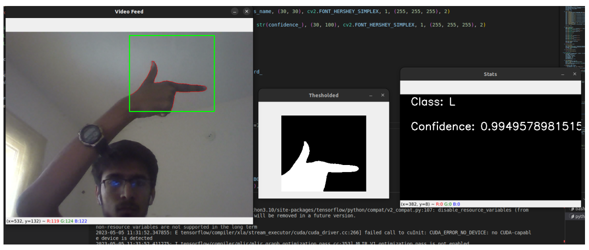


Figure 14: Detection of gestures through Mobile Application

can be found on the project's Github page under the account pranav_chiku@github. Approximately 25-30 image clicks were needed to predict all 10 classes, as shown in figure xx.

8.2. *Desktop Application*

CV2 and TensorFlow were utilized in the development of a desktop application using the Python programming language. When the application is launched, a prompt is opened and the camera is turned on, allowing the user to provide live input for continuous prediction. Ensuring low latency of prediction was crucial for the successful implementation of this feature, as any dropped frames could result in a poor user experience. The use of Python, CV2, and TensorFlow allowed for the development of a powerful desktop application capable of making real-time predictions based on live input from the user.
The same approach was followed to make this work, initially, we loaded weights of the CNN model in h5 format, and used .predict(frame) command on



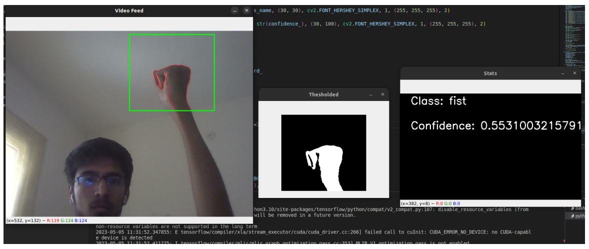
```
s_name, (30, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
 str(confidence_), (30, 100), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2)
```

Class: L

Confidence: 0.9949578981515

```
hon3.10/site-packages/tensorflow/python/compat/v2_compat.py:107: disable_resource_variables (from
will be removed in a future version.
non-resource variables are not supported in the long term
2023-05-05 11:31:52.347855: E tensorflow/compiler/xla/stream_executor/cuda/cuda_driver.cc:266] failed call to cuInit: CUDA_ERROR_NO_DEVICE: no CUDA-capabl
e device is detected
```

Figure 15: Detection of L through Desktop Application



Class: fist

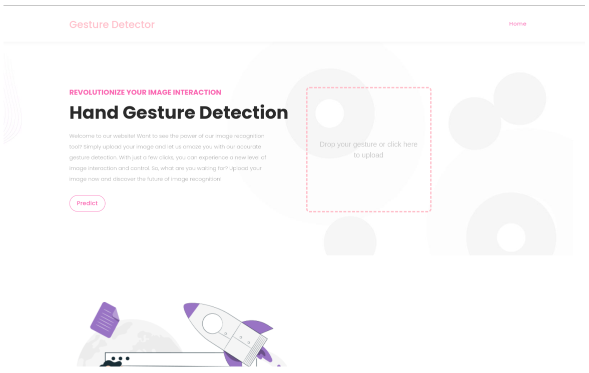
Confidence: 0.5531003215791

Figure 16: Detection of Fist through Desktop Application

every frame to perform prediction. More details about implementation can be found at pranav_chiku@github.

8.3. *Web Application*

To deploy a lightweight website, we utilized TensorFlow.js. The website is capable of directly loading model weights using JavaScript, and allows users to select images from their gallery on the web. Preprocessing similar to that done in Python is performed, and the gesture class is predicted. Additionally, users can download the APK file of the mobile application and obtain information about the desktop application via the website. The use of TensorFlow.js allowed for the successful deployment of a lightweight website with powerful gesture detection capabilities.

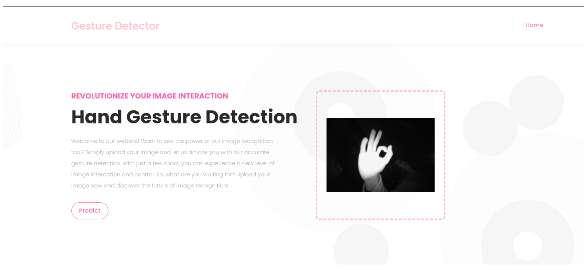


REVOLUTIONIZE YOUR IMAGE INTERACTION

Hand Gesture Detection

Welcome to our website! Want to see the power of our image recognition tool? Simply upload your image and let us amaze you with our accurate gesture detection. With just a few clicks, you can experience a new level of image interaction and control. So, what are you waiting for? Upload your image now and discover the future of image recognition!

Figure 17: Web landing page

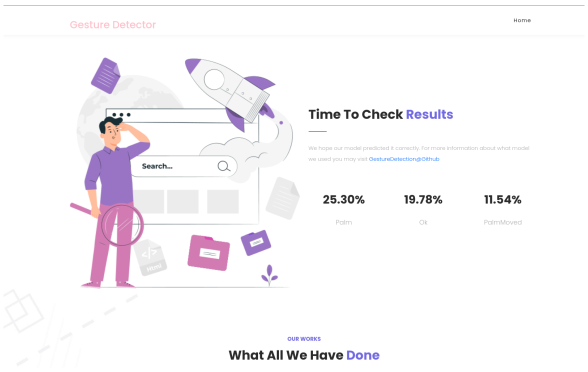


REVOLUTIONIZE YOUR IMAGE INTERACTION

Hand Gesture Detection

Welcome to our website! Want to see the power of our image recognition tool? Simply upload your image and let us amaze you with our accurate gesture detection. With just a few clicks, you can experience a new level of image interaction and control. So, what are you waiting for? Upload your image now and discover the future of image recognition!

Figure 18: Web uploaded image



Time To Check Results

We hope our model predicted it correctly. For more information about what model we used you may visit GestureDetection@Github

25.30%	19.78%	11.54%
Palm	Ok	PalmMoved

OUR WORKS

What All We Have Done

Figure 19: Web Prediction Using CNN Model

9. Contribution

Majority of the work is done by both of us together, it is very difficult to differentiate between it.

9.1. *Harshita Kalani (B20CS019)*

Data Loading, Data Processing, Mobile Application Pipeline, Web Application Pipeline, desktop application pipeline, VGG 19, EfficientNet B3, CNN scratch, etc.

9.2. *Pranav Goswami (B20CS016)*

Data loading and preprocessing, Mobile application ui/ux, web application pipeline, desktop application pipeling, ResNet 50, Auto Encoding Model, CNN scratch, etc.

References

1. Static Hand Gesture Recognition using Convolutional Neural Network with Data Augmentation
2. TensorFlow Documentation
3. Flutter Documentation
4. Hand Gesture Recognition Database
5. Hand Gesture Recognition Using Background Elimination and Convolution Neural Network