PROJECT ON HAND GESTURE DETECTION USING PYTHON.

Student Details:

Katha Bhattacharjee (University Roll: 193135-11-0028)

Gairik Baidya (University Roll: 193135-21-0037)

Paper Code: CMS-A-CC-6-14-P

Semester: VI, Department of Computer Science Year – 2022

Supervisor Name: Enakshi Ghosh

CERTIFICATE

This is to certify that the dissertation/project entitled ‘HAND GESTURE DETECTION USING PYTHON’ is done by Katha Bhattacharjee(193135-11-0028) and Gairik Baidya(193135-21-0037) is done under my supervision and guidance. This work has not been submitted elsewhere. This project is completed and submitted as a partial fulfillment of the last semester (6th semester) of their under graduate course B.Sc. in Computer Science-2022.

Date:

________________ (Name of the supervisor)

ACKNOWLEDGEMENT

I would like to express gratitude towards my supervisor “ENAKSHI GHOSH” for her able guidance and support in completing my project on the topic “HAND GESTURE DETECTION USING PYTHON”. This project has allowed me to explore an interesting side outside my regular curriculum. I am really thankful to her.

I would also like to express my gratitude to our honorable H.O.D Madam, “PARAMITA SIKDAR” for providing me with all the facilities that were required.

Lastly, I would like to thank my senior and alumnus “TANMAY DUTTA” for helping me out with the project. He has provided me with insights about the topic and guided me thoroughly.

Index

ABSTRACT

Hand gesture detection and recognition is a system which usually detects gestures of hand from a real video or an image. This project focuses on the later. The major issues that a developer can face during designing a gesture detection system are – detecting the hand itself and secondly recognizing and predicting the accurate result. Other challenging factors are orientation of hand, clarity of the image provided, location, variation in poses and scale of image. To overcome all these issues and perform accurately there was a need of various gestures showing numbers or signs. We took these data from several images which were taken from a dataset . The images along with their categories were stored methodically. They were resized and their color was converted as required. The data set was split and trained accordingly using a model. The model comprises of Pooling techniques, flattening, adding convolution layers and so on. Results are shown in the form of plotting based on the model's accuracy and lose in different cases. The accuracy of how well the model predicts an unknown image   is also provided as result.

1. INTRODUCTION

The usual method of communication between computers and humans is via contact. That might be through various input devices or screen touch technologies. In humans however, the method of communication is mainly verbal and is much more effortless. This has inspired researchers throughout years to develop some forms of non contact communication scheme between humans and computers. Gesture detection works with a similar approach. The computer tries to detect various gestures shown to it. In our project though, the system tries to detect the category of gesture provided through still images.

1.1 Domain Description

In this project we have tried to detect gestures using the object detection technique.

Our model is trained on a dataset containing 20,000 hand gestures (grouped accordingly).

Object detection:

In general, object detection is a technique which helps in identifying a particular class of image or video. The required subject in consideration is bounded and highlighted so that it can be recognized. In our case, we detect the class under which our gesture falls. The model is trained on the dataset and then used to predict results for unknown images. Object detection technique is similar to other techniques such as image recognition and segmentation that help us to study images and videos.

Object detection can be used for face detection, crowd counting, gesture detection, gesture detection etc.

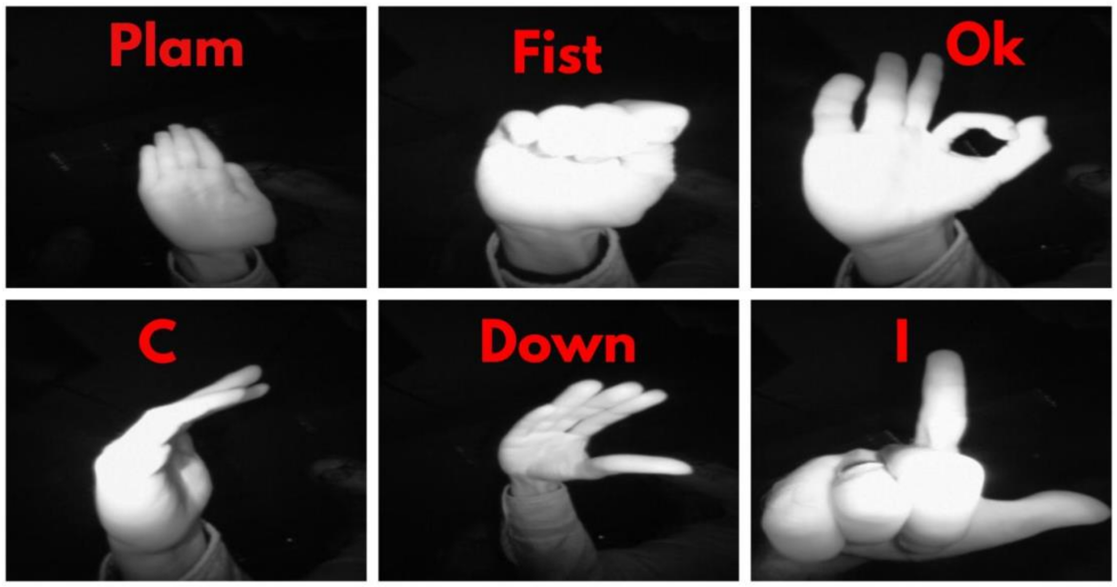
Dataset:

- Dataset used-(kaggle kernels output benenharrington/hand-gesture-recognition-database-with-cnn -p /path/to/dest)
- Our dataset contains 10 directories, with 10 sub directories(each one for one gesture) which contain 200 images each. Hence , we have worked with a total of 20,000 images. These were divided into training sets and testing sets and used accordingly.
- A constraint in the images is that they have a plain and empty background. Hence the model detects images with empty background.
- The images are converted to black-and-white before being used for detection.
- The images are resized to focus mainly on the hand gesture and kept small so that time consumption is lesser.

1.2 Motivation

As mentioned earlier, the primary motivation behind works related to gesture detection was to establish a non-contact communication between computers and humans. In this project of hand gesture detection, we have tried to cover a primary step towards that endeavor by developing a program that can detect the gesture shown in an image provided to it.

The gestures are categorized somewhat in the following manner:



Our aim is to identify these correctly using Python.

1.3 Scope

The field of gesture detection offers developers with a wide range of applications.

The various applications of gesture detection system in the real world are discussed below.

- <u>Talking to the computer</u>

Imagine a world in which a person putting together a presentation can add a quote or move an image with a flick of the wrist instead of a click of a mouse. A future in which we can easily interact in virtual reality much as we do in actual reality, using our hands for small, sophisticated movements like picking up a tool, pushing a button or squeezing a soft object in front of us.

- <u>Gesture based gaming control</u>

  Computer games are a particularly technologically promising and commercially rewarding arena for innovative interfaces due to the entertaining nature of the interaction. Users are eager to try new interface paradigms since they are likely immersed in a challenging game-like environment. In a multi-touch device, control is delivered through the user's fingertips.

- Home appliances

Most electronic devices focus on the hand gesture recognition algorithm and the corresponding user interface. Hand Gesture Based Remote is a device to replace all other remotes used in households and perform all their functions.

- Other applications

1. The project we have worked on can be further modified into a real time gesture detection device. It will then be able to capture and identify gestures from videos or CCTV footages or livestream cameras and so on.
2. This can be a way of interaction between disabled people and computers.
3. There are people who communicate purely through gestures to others. Sometimes even humans find it difficult to interpret the signs. A proper detection system can act as an interface between disabled people and others.
4. Psychology is based on gestures to a great extent. A thorough gesture detection system can interpret gestures and make psychological predictions. This can be helpful to study people in courtrooms, criminals during confessions and might test the truth of statements asserted.

- Future Scope

The project we have worked on can be further modified into a real time gesture detection device. It will then be able to capture and identify gestures from videos or CCTV footages or livestream cameras and so on.

2. Review of related work

Various works have been done on hand gesture recognition and some notable research in this topic are mentioned.

1. A hand gesture recognition system using an artificial neural network (ANN) based on shape fitting technique has been developed . In this system, a color segmentation technique on YbrCr color space was used after filtering to detect hand. Then the shape of the hand was approached by the hand morphology. The shape of hands and finger orientation features were extracted and passed to an ANN. They achieved 94.05% accuracy by using this method.

2. Gesture detection by using an ANN has also been developed . In this system, images were segmented based on skin colors. The selected features for ANN were changes of pixel through cross sections, boundary and the scalar description like aspect ratio and edge ratio. After establishing those feature vectors, they were fed to the ANN for training. The accuracy was around 98%.

3. A statistical method based on haar-like features to detect gestures was proposed . In this system, AdaBoost algorithm was used to learn the model. The whole work was divided into two levels. In higher level, a stochastic context-free grammar was used to detect gestures. In lower level, postures were detected. A terminal string was generated for each input according to the grammar. The probability associated with each rule was calculated and the rule with highest probability for the given string was selected as the rule. The gesture associated with this rule was returned as the gesture of the input.

4. An algorithm to recognize hand gestures using a 3D CNN was proposed . In this system, the basis of the recognition was challenging depth and intensity of the images. They also used data augmentation technique and achieved accuracy about 77.5% on VIVA challenge dataset .

5. Another system to detect gesture using CNN which is robust under five invariants scale, rotation, translation, illumination, noise and background was proposed . Sign Language of Peru (LSP) were used as dataset. They achieved 96.20% accuracy on the LSP Dataset.

6. The work in Anshal, Heidy & Emmanuel (2017) processed the hand gesture image by combining image segmentation and edge detection to extract morphological information, and frames were processed using the multi-modality technique used for processing individual characters. In Vedha Viyas et al. (2017) to improve recognition rate of hand gestures, various image processing techniques were used such as Histogram Equalization, Median filtering, Average filtering, and Morphological filtering, feature extraction image matching was done using cross-correlation co-efficient.

7. The classifier in Piotr, Tomasz & Jakub (2017) had five gestures (fist, pronation, supination, flexion, extension) and the feature vector consisted of 18 features: five representing muscle activity (RMS) and 13 parameters corresponding to relative sensor orientation. A feature vector which was composed of wavelet invariant moments and distance feature was generated by the work of Xi, Chen & Lihua (2017). The authors of Shunzhan et al. (2017) extract five eigenvalues in the time domain. Geometric features such as area and centroid were extracted from each video frame to capture the trajectory of the moving hand and compare it with the training gestures in the experiments of Atharva & Apurv (2017).

8. The proposed system (Oyndrila et al., 2016) identified hand-palm in video stream based on skin color and background subtraction scheme. Where in Rishabh et al. (2016) glove tracking was done and then fingertips were detected with respect to centroid of the hand. Features used in Ashish & Aarti (2016a) and Ashish & Aarti (2016b) were orientation, Centre of mass centroid, fingers status, thumb in positions of raised or folded fingers of hand. On the other hand, features extracted in Soumya & Muzameel (2017) were size, shape, color or texture.

3. Methodology

In research, methodology is defined as a systematic approach to resolving a study topic by collecting data using various approaches, interpreting the data, and deriving conclusions from the data. A research technique is essentially the plan for a research or study.

3.1 Problem Formulation

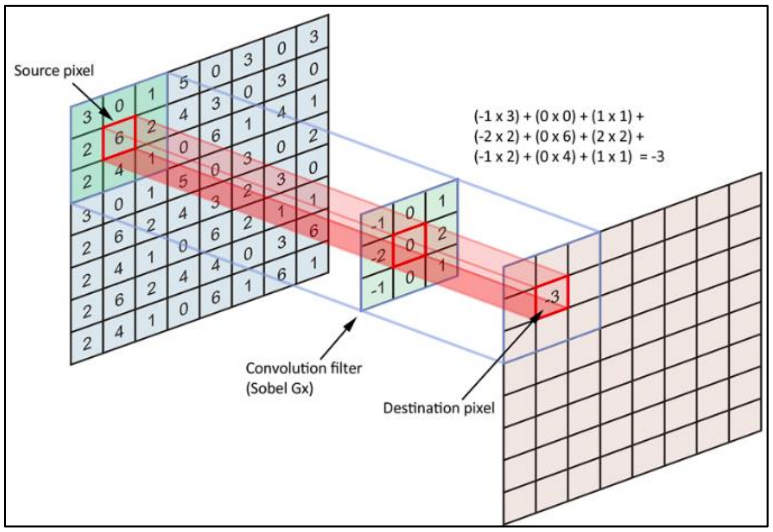
The step of problem formulation is to determine the user's characteristics and needs. The performance criteria for the chosen solvent will be defined in this step. For the developed solvents, performance requirements and goal attributes were established.

3.1.1 Convolution Layer

The Convolutional Neural Network's basic building element is this layer. Convolution is a mathematical term that refers to the mathematical combining of two functions to produce a third function. It employs a sliding window technique to aid in the extraction of characteristics from an image. This assists in the creation of feature maps. The output C is obtained by convolution two functional matrices, one of which is the input image matrix A and the other is the convolutional kernel B.

$$C(T)=(A*B)(x)= \int \infty-\infty A(T)\times B(T-x)\, dT$$

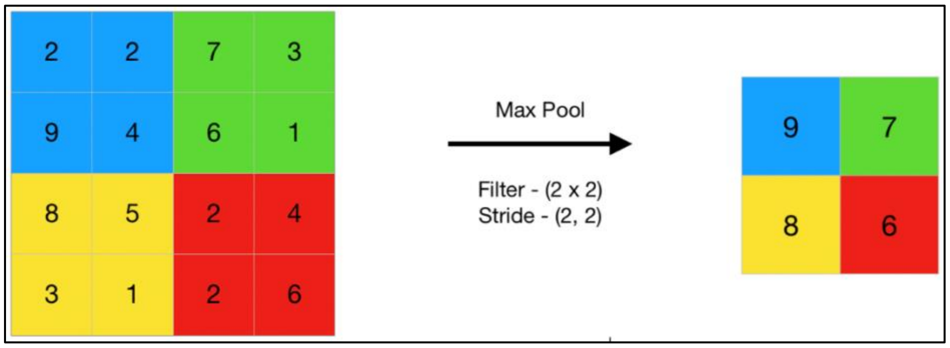
Convolution 2D depicted:


Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
(Sobel Gx)
Destination pixel

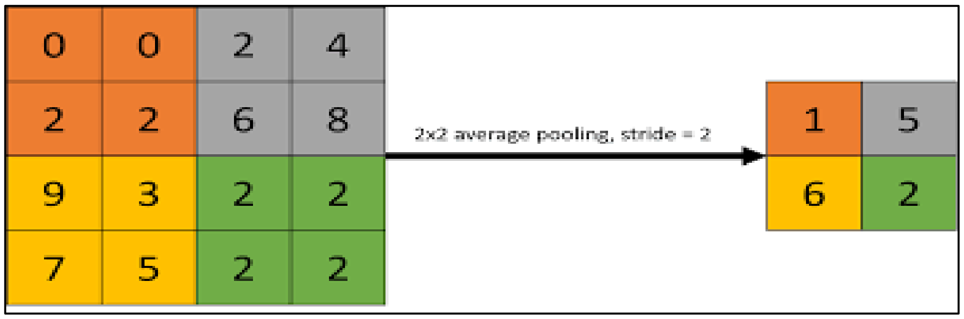
3.1.2 Pooling layer

The use of pooling techniques can speed up calculations by reducing the size of the input matrix without sacrificing many properties. There are several types of pooling operations that can be used, some of which are described below:

i.) Max Pooling: It uses the highest value in the designated region where the kernel is now located as the value for the output of the matrix value for that cell.

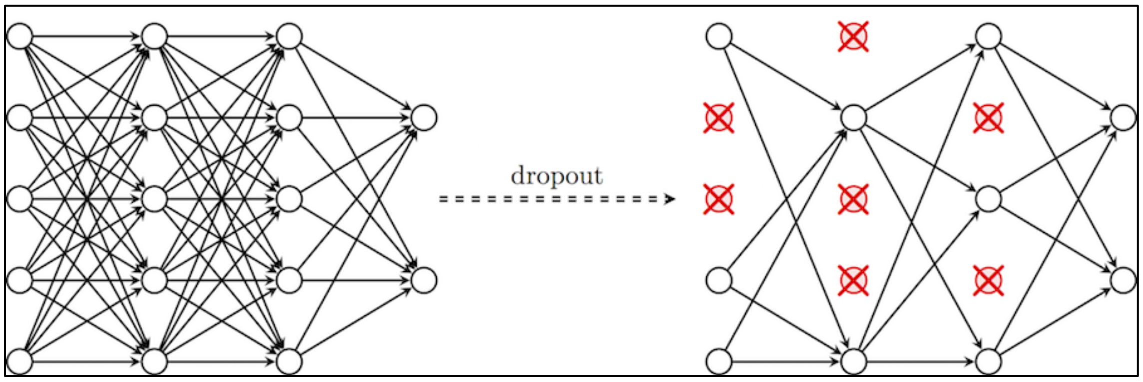


ii.) Average Pooling: It takes the average of all the values currently in the region where the kernel is located and uses that value as the matrix value for that cell's matrix value. The average-pooling operation is depicted.



3.1.3 **Dropout Layer:**

By removing random biased neurons from the model, this helps to reduce overfitting that can occur during training. These neurons can be found in both hidden and visible levels. The dropout ratio can be modified to alter the likelihood of a neuron being dropped.



3.1.4 Non-linear layer:

The convolutional layers are frequently followed by these layers. Rectified

Linear Units (ReLUs), such as Leaky ReLUs, Noisy ReLUs, Exponential ReLUs, and so on, as well as sigmoid and tanh functions, are among the most often utilised non-linear functions. Non-Linear Functions of various types and their equations are shown below:

Sigmoid: σ(x)= 1/1+e−x

Leaky Relu: f(x)=max(0.1x, x)

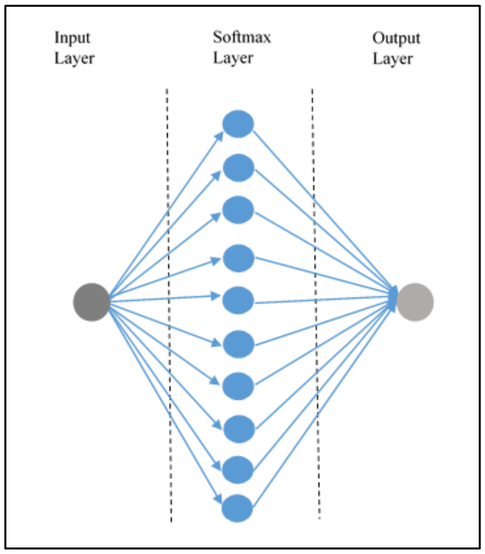
Tanh: f(x)=tanh(x)

Maxout: f(x)=max(wT1x+ b1 , wT2x+b2)

Relu: f(x)=max(0,x)
ELU: f(x)= {x x≥0 α(ex−1) x<0 }

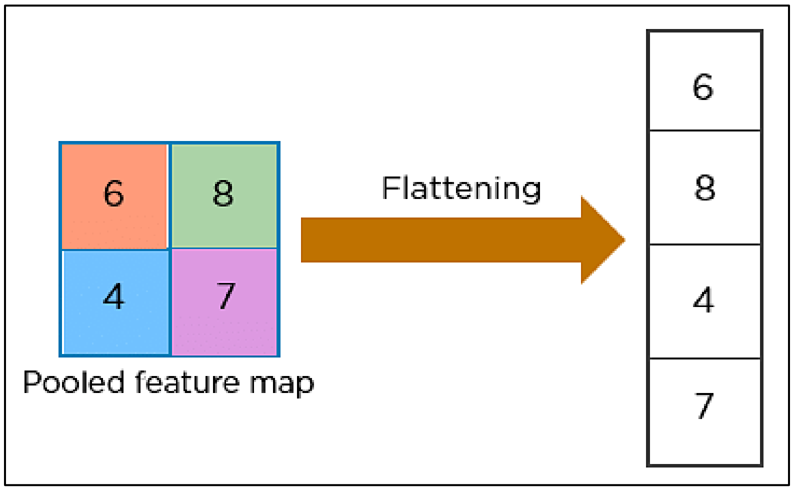
3.1.5 Fully Connected layer

These layers connect to the activation layers and append the model. These layers help in the multi-class or binary classification of pictures. SoftMax is an example of an activation function used in these layers, and it calculates the possibility of predicted output classes.



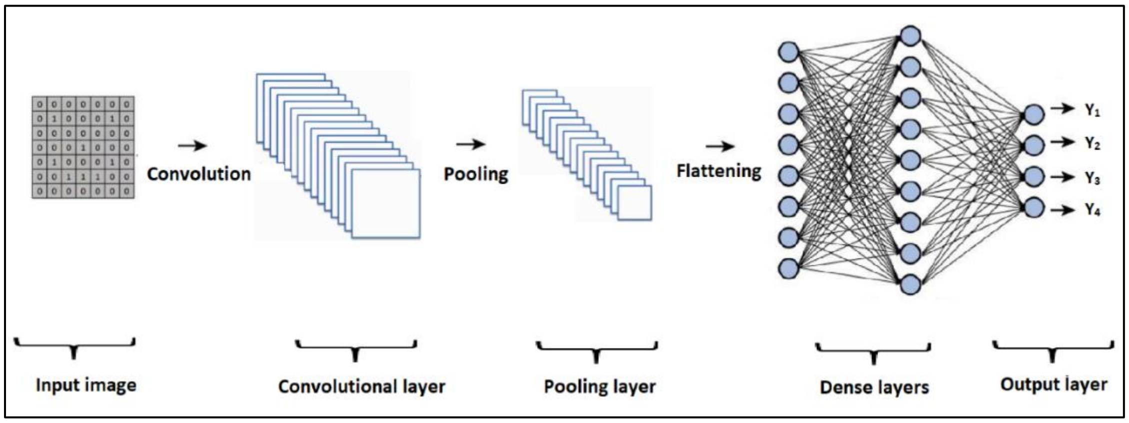
3.1.6 Flatten layer in CNN

Flattening is used to convert all the resultant 2-Dimensional arrays from pooled feature maps into a single long continuous linear vector. The flattened matrix is fed as input to the fully connected layer to classify the image.



3.1.7 Dense Layer in CNN

Dense Layer is simple layer of neurons in which each neuron receives input from all the neurons of previous layer, thus called as dense. Dense Layer is used to classify image based on output from convolutional layers.



3.2 Algorithm

Here are the steps which were followed to built our hand gesture detector.

INPUT: Images from the dataset. Images from user.

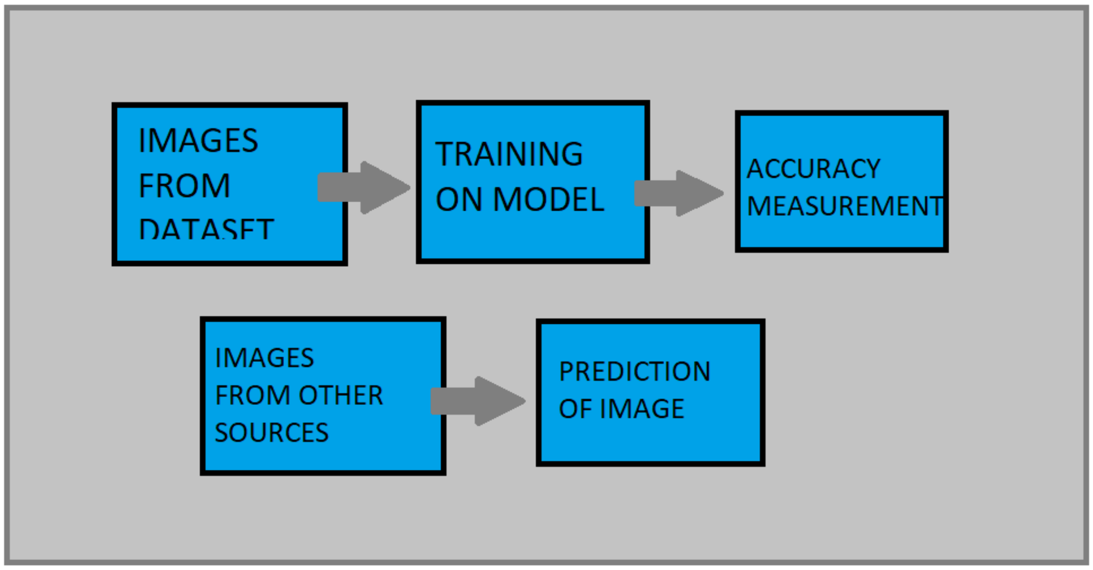
OUTPUT: 1. Accuracy vs Loss graph of the model.

2.Classification of gesture when an image is provided.

1. Mount google drive.
2. Unzip the folder.
3. Process each image(resize and grey scaling).
4. The images are put in an array and the categories are put in another array.
5. Split the folder into train and test data using split folder.
6. Perform Data generation using image data generation .
7. Create model using keras. Train it on training batches and compile it using RMSprop optimizer.
8. Fit the model then evaluate it.
9. Plot the graphs between accuracy and loss.
10. Check if the prediction is working correctly with the test images.
11. Then import images of your own and predict the gesture.

3.3 Design Description:

It depicts our suggested system design (image courtesy of the dataset). First the dataset is downloaded. Images are generated using image data generator. Model is created using various layers. Then the model that was created is fitted. Then image is taken from the testing images to predict the output. After that images from outside is added to predict the output.



4. Implementation

4.1Dataset:

Here we get dataset ‘Hand Gesture Recognisation Dataset’.

In our technique, we used 80% of the dataset for training and 20% for testing, resulting in a split ratio of 0.8:0.2 for train to test data. We used 20% of the training data to create a validation data set.

4.2 Pre-processing:

The pre-processing phase occurs before the data is trained and tested. The pre-processing consists of four steps: scaling the image size, turning the image to an array, pre-processing input , and changing the colour theme.

Due to the efficacy of training models, image scaling is a significant pre-processing step in computer vision. The model will perform better if the image is smaller.

The next step is to create an array from all of the photographs in the dataset. The image is transformed into an array so that the loop function can call it.

A lookup table containing names of categories is also made in this step.

4.3 Training:

At training time, we compare ground truth boxes with default bounding boxes of various sizes and aspect ratios, and then use the Intersection over Union (IoU) approach to select the best matching box for each pixel. IoU determines how much of our anticipated box corresponds to reality. The values range from 0 to 1, and increasing IoU values influence forecast accuracies; the best value is the highest IoU value. IoU's equation and pictorial representation are as follows:

$$IoU(B1, B2) = B1 \cap B2 / B1 \cup B2$$

4.4 Optimizer Hyperparameter:

We start the actual training process, we can adjust or optimise our model using optimizer parameters. Batch size is a hyperparameter that affects the training's resource requirements as well as the speed and number of repeats. Epochs are the hyperparameters that govern how often the model is performed. Learning rates are another parameter and the optimizer one uses may also vary.

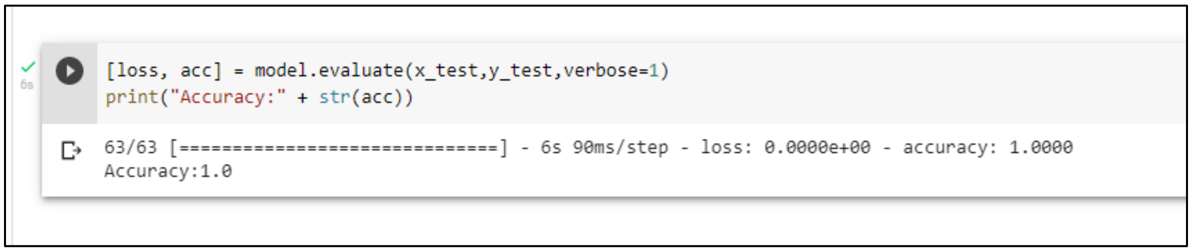
4.5 Loss and Accuracy Function:

The results of Accuracy and loss after the training is stored. Later the data is used to plot the results. These depict the loss vs accuracy graph and var-loss vs var-accuracy graph.

5.Result and Discussion:

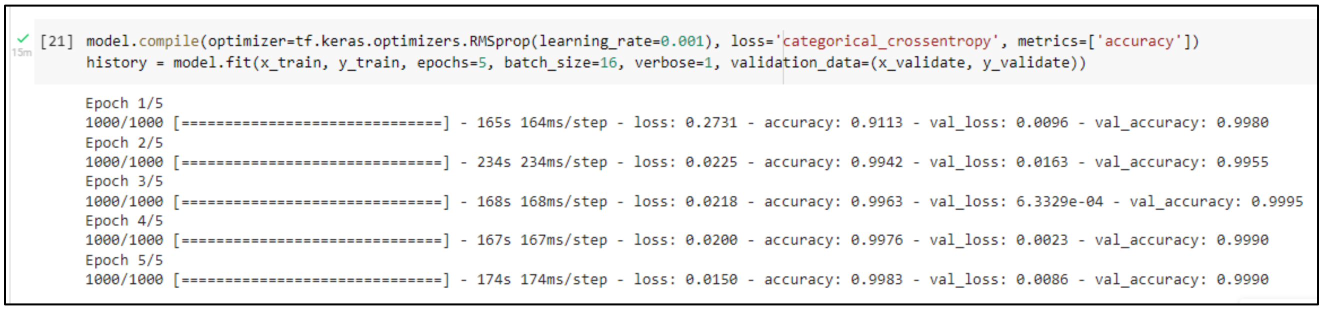
The outcomes of our experimental examination are presented in this section. The purpose of these tests is to gain insight into the performance of detection approaches , as well as to assess the performance of identification models that. As a result of our project, the accuracy level is 99.90%.

The accuracy when the model works on an image from test section.

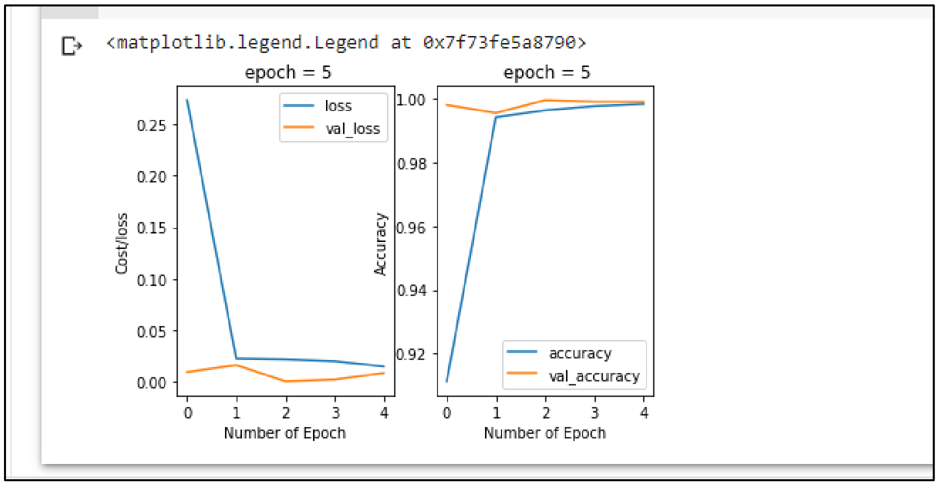


```
[loss, acc] = model.evaluate(x_test,y_test,verbose=1)
print("Accuracy:" + str(acc))

63/63 [==============================] - 6s 90ms/step - loss: 0.0000e+00 - accuracy: 1.0000
Accuracy:1.0
```

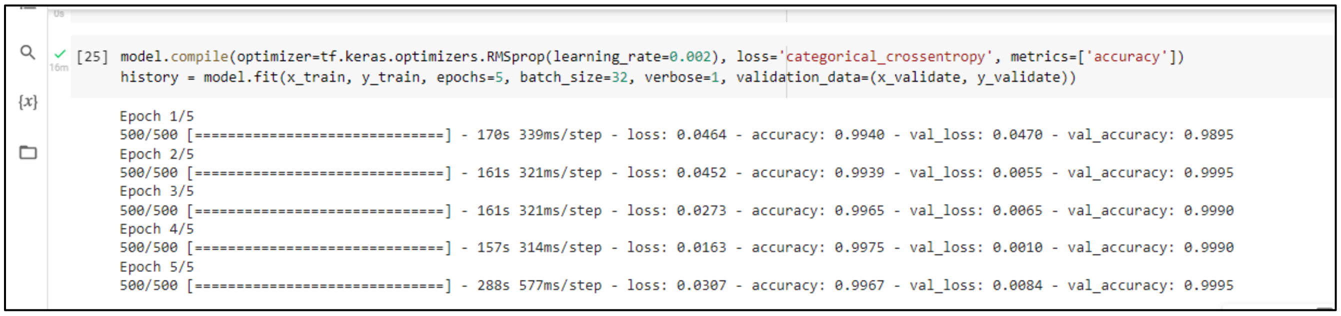
Accuracy vs Loss graph for different scenarios.



```
[21] model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
     history = model.fit(x_train, y_train, epochs=5, batch_size=16, verbose=1, validation_data=(x_validate, y_validate))

     Epoch 1/5
     1000/1000 [==============================] - 165s 164ms/step - loss: 0.2731 - accuracy: 0.9113 - val_loss: 0.0096 - val_accuracy: 0.9980
     Epoch 2/5
     1000/1000 [==============================] - 234s 234ms/step - loss: 0.0225 - accuracy: 0.9942 - val_loss: 0.0163 - val_accuracy: 0.9955
     Epoch 3/5
     1000/1000 [==============================] - 168s 168ms/step - loss: 0.0218 - accuracy: 0.9963 - val_loss: 6.3329e-04 - val_accuracy: 0.9995
     Epoch 4/5
     1000/1000 [==============================] - 167s 167ms/step - loss: 0.0200 - accuracy: 0.9976 - val_loss: 0.0023 - val_accuracy: 0.9990
     Epoch 5/5
     1000/1000 [==============================] - 174s 174ms/step - loss: 0.0150 - accuracy: 0.9983 - val_loss: 0.0086 - val_accuracy: 0.9990
```

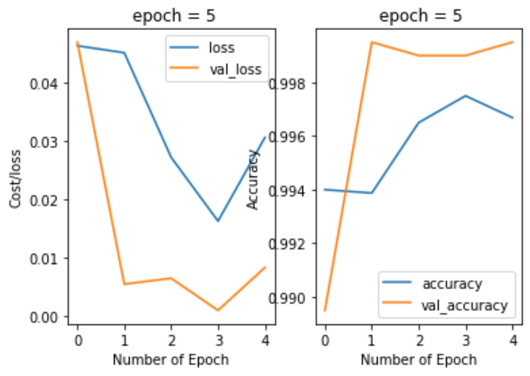


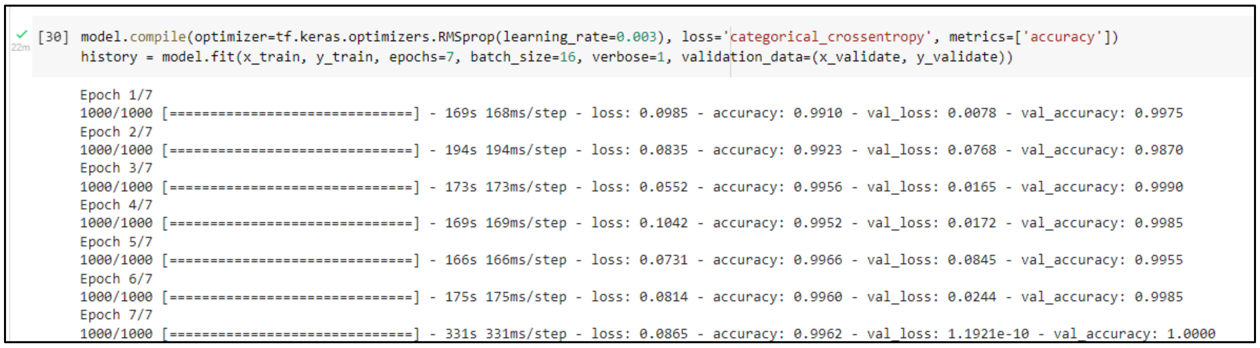
```
<matplotlib.legend.Legend at 0x7f73fe5a8790>
```



```
[25] model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.002), loss='categorical_crossentropy', metrics=['accuracy'])
     history = model.fit(x_train, y_train, epochs=5, batch_size=32, verbose=1, validation_data=(x_validate, y_validate))

     Epoch 1/5
     500/500 [==============================] - 170s 339ms/step - loss: 0.0464 - accuracy: 0.9940 - val_loss: 0.0470 - val_accuracy: 0.9895
     Epoch 2/5
     500/500 [==============================] - 161s 321ms/step - loss: 0.0452 - accuracy: 0.9939 - val_loss: 0.0055 - val_accuracy: 0.9995
     Epoch 3/5
     500/500 [==============================] - 161s 321ms/step - loss: 0.0273 - accuracy: 0.9965 - val_loss: 0.0065 - val_accuracy: 0.9990
     Epoch 4/5
     500/500 [==============================] - 157s 314ms/step - loss: 0.0163 - accuracy: 0.9975 - val_loss: 0.0010 - val_accuracy: 0.9990
     Epoch 5/5
     500/500 [==============================] - 288s 577ms/step - loss: 0.0307 - accuracy: 0.9967 - val_loss: 0.0084 - val_accuracy: 0.9995
```

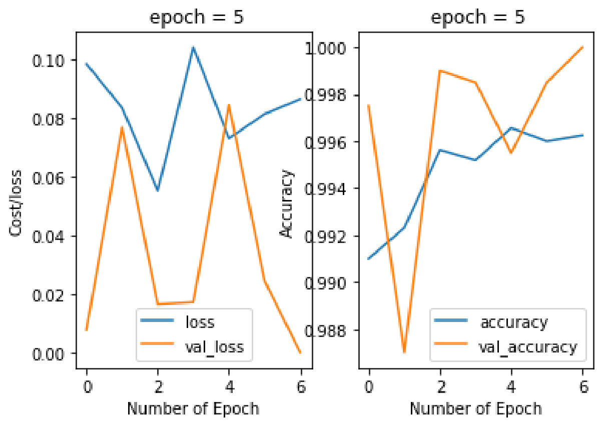
```
<matplotlib.legend.Legend at 0x7f74023ebdd0>
```

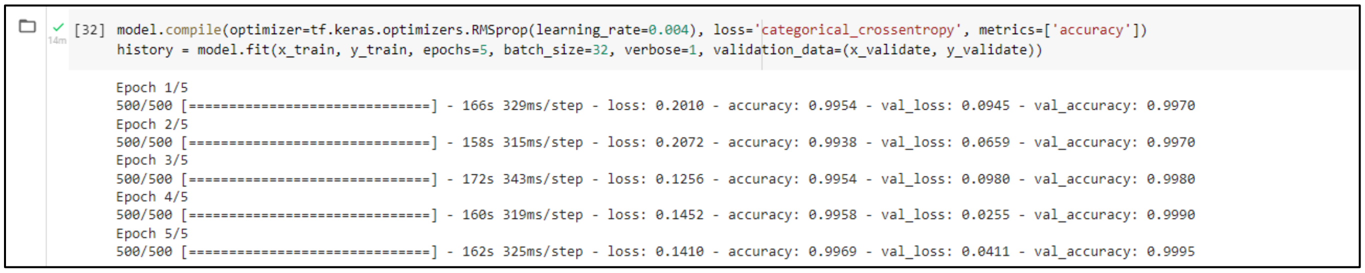




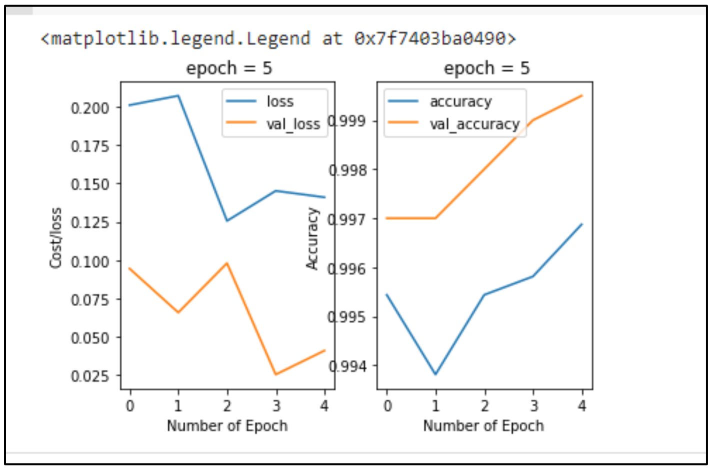
```
[30] model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.003), loss='categorical_crossentropy', metrics=['accuracy'])
     history = model.fit(x_train, y_train, epochs=7, batch_size=16, verbose=1, validation_data=(x_validate, y_validate))

     Epoch 1/7
     1000/1000 [==============================] - 169s 168ms/step - loss: 0.0985 - accuracy: 0.9910 - val_loss: 0.0078 - val_accuracy: 0.9975
     Epoch 2/7
     1000/1000 [==============================] - 194s 194ms/step - loss: 0.0835 - accuracy: 0.9923 - val_loss: 0.0768 - val_accuracy: 0.9870
     Epoch 3/7
     1000/1000 [==============================] - 173s 173ms/step - loss: 0.0552 - accuracy: 0.9956 - val_loss: 0.0165 - val_accuracy: 0.9990
     Epoch 4/7
     1000/1000 [==============================] - 169s 169ms/step - loss: 0.1042 - accuracy: 0.9952 - val_loss: 0.0172 - val_accuracy: 0.9985
     Epoch 5/7
     1000/1000 [==============================] - 166s 166ms/step - loss: 0.0731 - accuracy: 0.9966 - val_loss: 0.0845 - val_accuracy: 0.9955
     Epoch 6/7
     1000/1000 [==============================] - 175s 175ms/step - loss: 0.0814 - accuracy: 0.9960 - val_loss: 0.0244 - val_accuracy: 0.9985
     Epoch 7/7
     1000/1000 [==============================] - 331s 331ms/step - loss: 0.0865 - accuracy: 0.9962 - val_loss: 1.1921e-10 - val_accuracy: 1.0000
```

```
<matplotlib.legend.Legend at 0x7f73fd94d950>
```



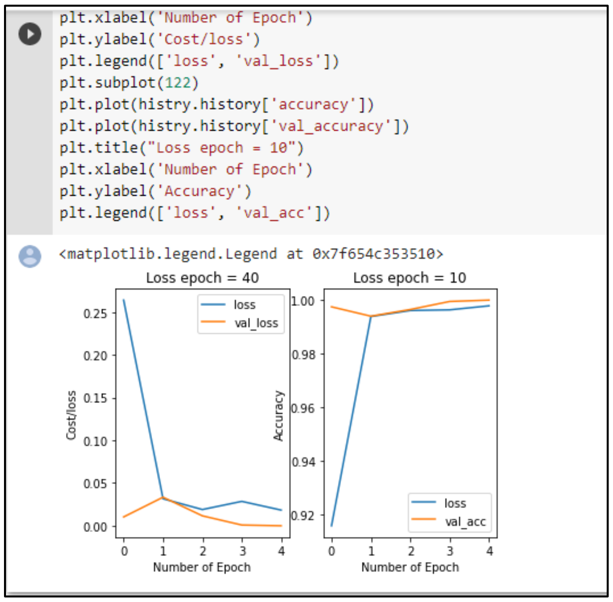


```
[32] model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.004), loss='categorical_crossentropy', metrics=['accuracy'])
     history = model.fit(x_train, y_train, epochs=5, batch_size=32, verbose=1, validation_data=(x_validate, y_validate))

     Epoch 1/5
     500/500 [==============================] - 166s 329ms/step - loss: 0.2010 - accuracy: 0.9954 - val_loss: 0.0945 - val_accuracy: 0.9970
     Epoch 2/5
     500/500 [==============================] - 158s 315ms/step - loss: 0.2072 - accuracy: 0.9938 - val_loss: 0.0659 - val_accuracy: 0.9970
     Epoch 3/5
     500/500 [==============================] - 172s 343ms/step - loss: 0.1256 - accuracy: 0.9954 - val_loss: 0.0980 - val_accuracy: 0.9980
     Epoch 4/5
     500/500 [==============================] - 160s 319ms/step - loss: 0.1452 - accuracy: 0.9958 - val_loss: 0.0255 - val_accuracy: 0.9990
     Epoch 5/5
     500/500 [==============================] - 162s 325ms/step - loss: 0.1410 - accuracy: 0.9969 - val_loss: 0.0411 - val_accuracy: 0.9995
```



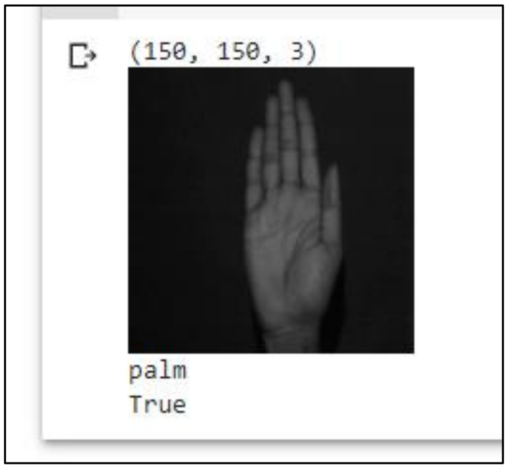
```
<matplotlib.legend.Legend at 0x7f7403ba0490>
```

Size= 200,200 epoch = 15, batch size= 16, learning rate= 0.001

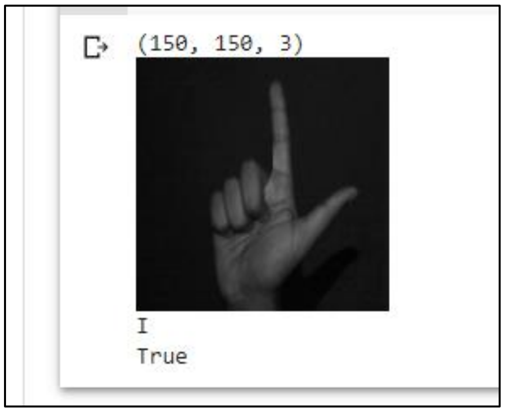


```
plt.xlabel('Number of Epoch')
plt.ylabel('Cost/loss')
plt.legend(['loss', 'val_loss'])
plt.subplot(122)
plt.plot(histry.history['accuracy'])
plt.plot(histry.history['val_accuracy'])
plt.title("Loss epoch = 10")
plt.xlabel('Number of Epoch')
plt.ylabel('Accuracy')
plt.legend(['loss', 'val_acc'])
```

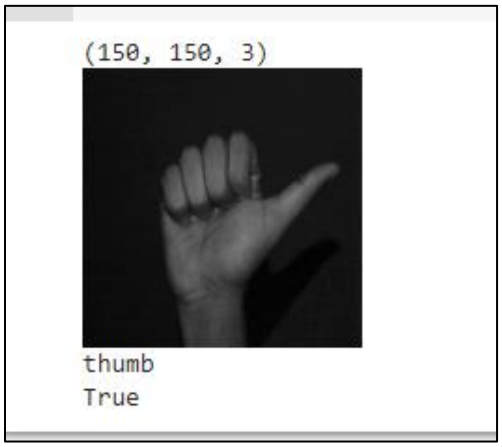
```
<matplotlib.legend.Legend at 0x7f654c353510>
```



```
(150, 150, 3)
palm
True
```



```
(150, 150, 3)
I
True
```



```
(150, 150, 3)
thumb
True
```

6.CONCLUSION

Gesture recognition technology is the turning point in the world of VR/AR development. It can allow seamless non-touchable control of computerized devices to create a highly interactive interface .
The inclusion of this technology in multiple applications across various sectors is further revolutionizing human-computer communication.
This project can successfully predict hand gestures. It can later be modified to implement better and more advanced features.

7. REFERENCES

- A. D. Wilson and A. F. Bobick, “Learning visual behavior for gesture analysis,” in *Proceedings of International Symposium on Computer Vision-ISCV*. IEEE, 1995, pp. 229–234.

- Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, no. 7553, p. 436, 2015.

- Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, “Object recognition with gradient-based learning,” in *Shape, contour and grouping in computer vision*. Springer, 1999, pp. 319–345.

- E. Stergiopoulou and N. Papamarkos, “Hand gesture recognition using a neural network shape fitting technique,” *Engineering Applications of Artificial Intelligence*, vol. 22, no. 8, pp. 1141–1158, 2009.

- T.-N. Nguyen, H.-H. Huynh, and J. Meunier, “Static hand gesture recognition using artificial neural network,” *Journal of Image and Graphics*, vol. 1, no. 1, pp. 34–38, 2013.

- Q. Chen, N. D. Georganas, and E. M. Petriu, “Hand gesture recognition using haar-like features and a stochastic context-free grammar,” *IEEE transactions on instrumentation and measurement*, vol. 57, no. 8, pp. 1562–1571, 2008.

- P. Molchanov, S. Gupta, K. Kim, and J. Kautz, “Hand gesture recognition with 3d convolutional neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2015, pp. 1–7.

- C. J. L. Flores, A. G. Cutipa, and R. L. Enciso, “Application of convolutional neural networks for static hand gestures recognition under different invariant features,” in *2017 IEEE XXIV International Conference on Electronics, Electrical Engineering and Computing (INTERCON)*. IEEE, 2017, pp. 1–4.

- Z. Zivkovic, “Improved adaptive gaussian mixture model for background subtraction,” in *null*. IEEE, 2004, pp. 28–31.

- Z. Zivkovic and F. Van Der Heijden, “Efficient adaptive density estimation per image pixel for the task of background subtraction,” *Pattern recognition letters*, vol. 27, no. 7, pp. 773–780, 2006.