



# HAND GESTURE RECOGNITION

Final Project

## ABSTRACT

Hand gestures recognition based on deep learning and computer vision.

Reda Mohsen Reda Gebril 18P5141

CSE485 – Deep Learning

[GitHub Link](#)

[Source Code](#)

## Introduction

With the rapid development of deep learning and computer vision, there is a growing demand for human-machine interaction. Hand gesture recognition, which can convey rich information, finds extensive applications in various fields such as facilitating communication for deaf-mute individuals, robot control, human-computer interaction (HCI), home automation, and medical applications. Different techniques, including instrumented sensor technology, deep learning, and computer vision, have been explored in research papers focusing on hand gestures. This report specifically delves into the application of deep learning techniques and computer vision for accurate hand gesture recognition.

## Problem Statement

The problem addressed in this report is the recognition of hand gestures using a combination of deep learning and computer vision. Camera vision-based sensors are commonly employed in this context due to their ability to enable contactless communication between humans and computers. However, this approach presents several challenges. These challenges include dealing with variations in lighting conditions, handling complex backgrounds, addressing occlusion effects, compensating for the presence of foreground or background objects that may resemble hands in terms of skin color tone, and managing the trade-offs between processing time, resolution, and frame rate. To overcome these challenges, the proposed solution involves leveraging computer vision techniques to preprocess the input image, followed by passing the processed image through a deep learning model to accurately predict the corresponding hand gesture class.

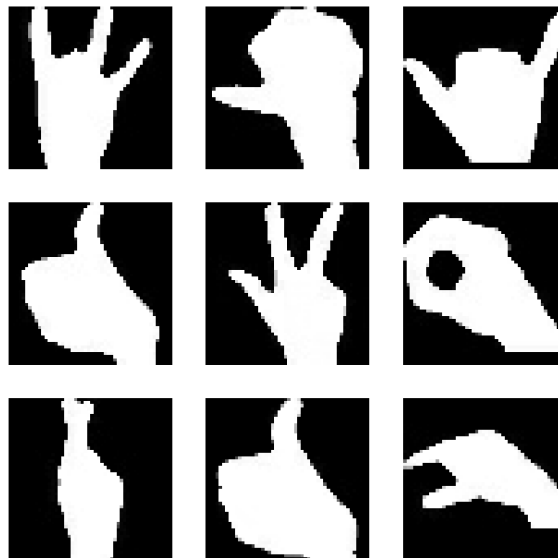
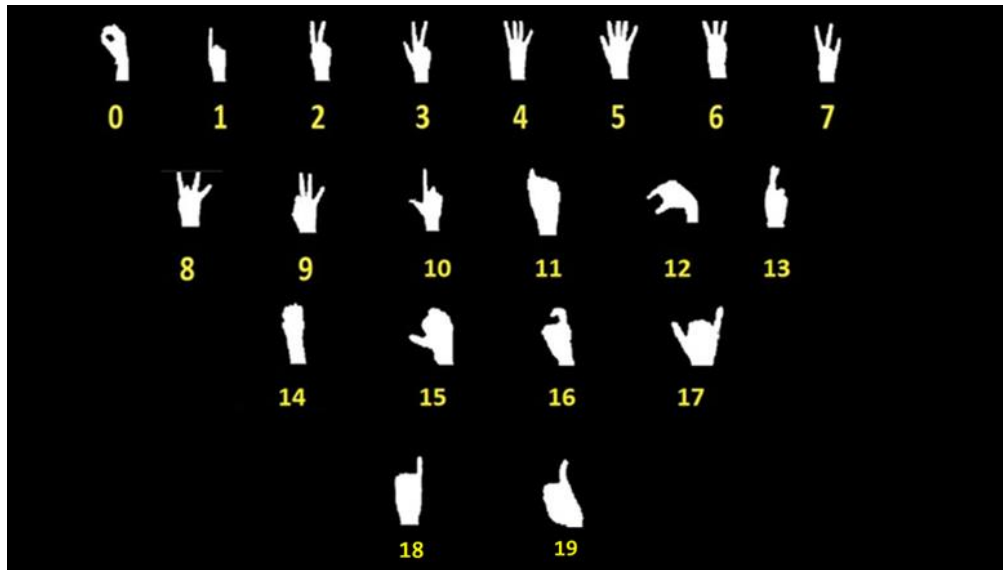
## Solution

The solution entails the implementation and testing of a hand gesture recognition system that combines deep learning and computer vision. The system utilizes camera input to capture images of hand gestures, which are then preprocessed using various computer vision techniques. These preprocessing steps serve to enhance the image quality and extract relevant features that aid in accurate recognition. The processed images are subsequently fed into a deep learning model specifically trained for gesture recognition. This model, having learned the patterns and features associated with different hand gestures from a labeled dataset, predicts the class of the hand gesture. The model is optimized using appropriate loss functions and optimization algorithms during the training process. Testing is performed using a separate set of images to evaluate the accuracy and overall performance of the system in recognizing hand gestures.

## Implementation

### About Dataset

hand-gesture-recognition-dataset. This dataset contains total 24000 images of 20 different gestures. For training purpose, there are 900 images in each directory and for testing purpose there are 300 images in each directory. Image shape is 50x50x3 and labeled.



This dataset contains images of hand gestures captured by a single camera and applying processing on the images using computer vision. They are captured in perfect conditions like there is no noise, all images are straight and perfectly rotated and light conditions are perfect. So, the dataset needs some preprocessing before training the deep learning model.

## Data Preprocessing

Applying data augmentation using albumentation library to add rotation, cutout, flipping, adjust the contrast and brightness, blur, and add noise to the dataset so that the model can predict better in practical.

```
[5]: # Define the data augmentation options you want to apply using Albumentations
transform = A.Compose([
    A.RandomRotate90(), # randomly rotate the image by 90 degrees
    A.HorizontalFlip(p=0.6), # randomly flip the image horizontally
    A.RandomBrightnessContrast(), # randomly adjust the brightness and contrast
    A.OneOf([
        A.CoarseDropout (p=0.6), # randomly cutout the images
        A.Blur(), # randomly blur the image
        A.GaussNoise(), # randomly add Gaussian noise to the image
    ]), # randomly apply one of the above augmentations with a probability of 0.5
])
```

Now for training purpose, there are 1800 images in each directory and for testing purpose there are 600 images in each directory after data augmentation.

## Loading the Dataset

Loading the dataset using TensorFlow (`tf.keras.utils.image_dataset_from_directory`). Class names are from 0 to 19, batch size is 16, input height and width is (50, 50). Color mode is grayscale to make the shape of the dataset is 50x50x1 as greyscale values is enough in our model no need to color images. Label mode is (int) as our data labels are integers not categories. Splitting our data into training, validation, testing datasets.

```
[6]: batch_size = 16
img_height = 50
img_width = 50
class_names = ['0','1','2','3','4','5','6','7','8','9',
               '10','11','12','13','14','15','16','17',
               '18','19']

train_dataset = tf.keras.utils.image_dataset_from_directory(
    'dataset/train',
    labels='inferred',
    # Label_mode='categorical',
    label_mode='int',
    color_mode='grayscale',
    class_names=class_names,
    batch_size=batch_size,
    image_size=(img_height, img_width),
    shuffle=True,
    seed=1,
    validation_split=0.33335,
    subset='training'
)
```

```
valid_dataset = tf.keras.utils.image_dataset_from_directory(
    'dataset/train',
    labels='inferred',
    # Label_mode='categorical',
    label_mode='int',
    color_mode='grayscale',
    class_names=class_names,
    batch_size=batch_size,
    image_size=(img_height, img_width),
    shuffle=True,
    seed=1,
    validation_split=0.33335,
    subset='validation'
)

test_dataset = tf.keras.utils.image_dataset_from_directory(
    'dataset/test',
    labels='inferred',
    # Label_mode='categorical',
    label_mode='int',
    color_mode='grayscale',
    class_names=class_names,
    batch_size=batch_size,
    image_size=(img_height, img_width),
    shuffle=True
)

Found 36000 files belonging to 20 classes.
Using 24000 files for training.
Found 36000 files belonging to 20 classes.
Using 12000 files for validation.
Found 12000 files belonging to 20 classes.
```

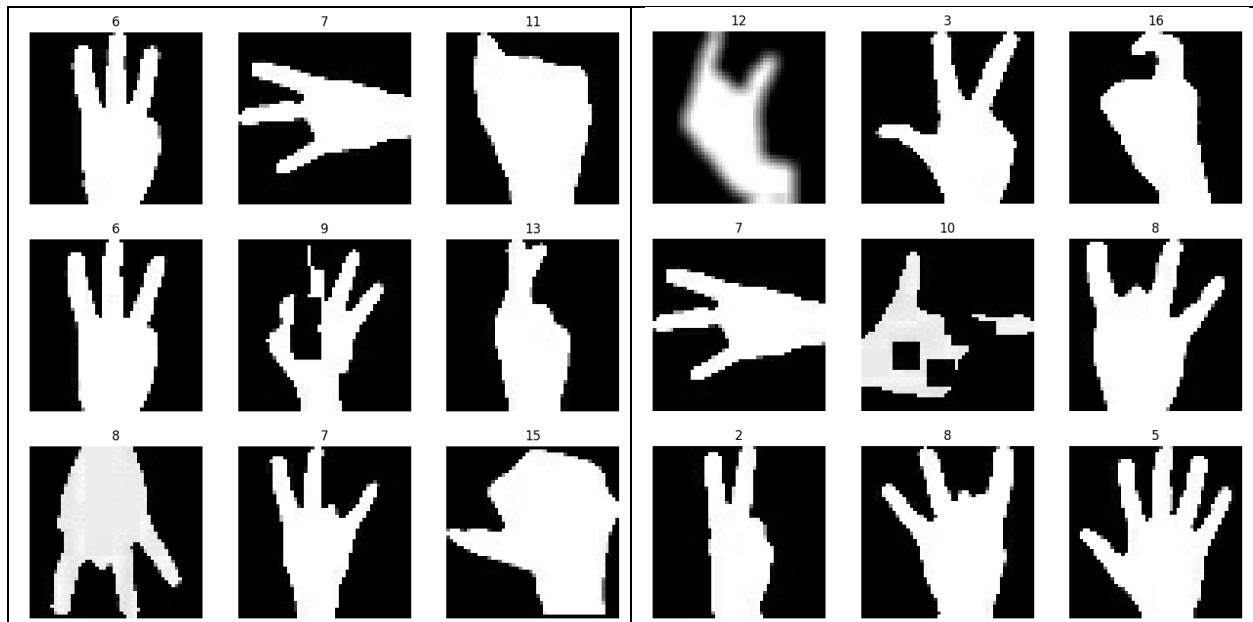
The class names are loaded successfully from the dataset and in order.

```
[7]: class_names = train_dataset.class_names
print(class_names)

['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19']
```

## Visualization of the Dataset After Preprocessing

Showing some of the preprocessing happened to the dataset like rotation, cutout, and flipping.



## Normalization of the Dataset

The dataset needs to be normalized to obtain a better performance in training and testing, so that the minimum value become 0 and the maximum value become 1.

```
[24]: normalization_layer = tf.keras.layers.Rescaling(1./255)

[25]: normalized_train_dataset = train_dataset.map(lambda x, y: (normalization_layer(x), y))
normalized_valid_dataset = valid_dataset.map(lambda x, y: (normalization_layer(x), y))
normalized_test_dataset = test_dataset.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_train_dataset))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))

0.0 1.0
```

## Configure the Dataset for Performance

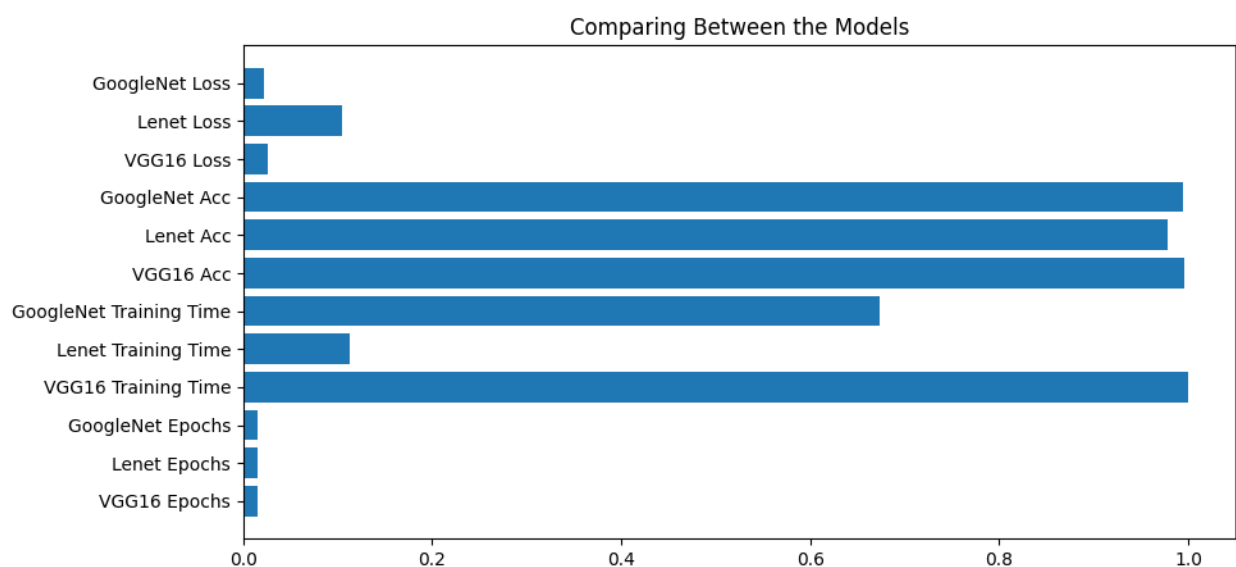
To optimize the performance, the dataset is cached, enabling faster training and improved performance during the training process.

```
[12]: AUTOTUNE = tf.data.AUTOTUNE

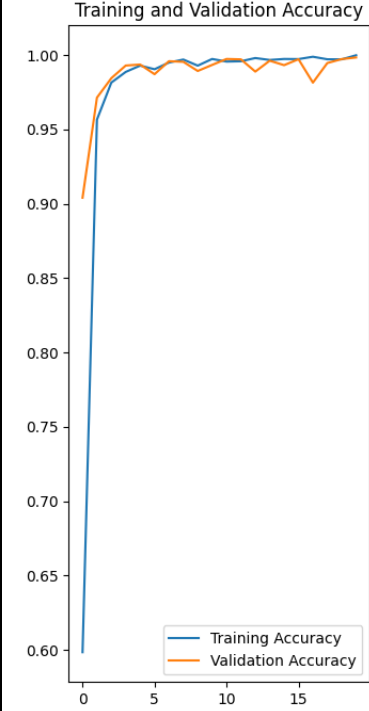
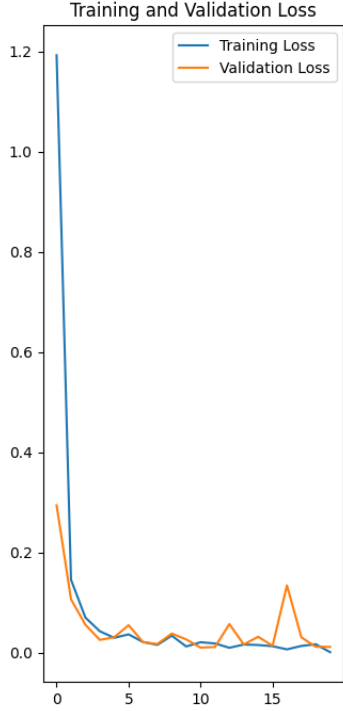
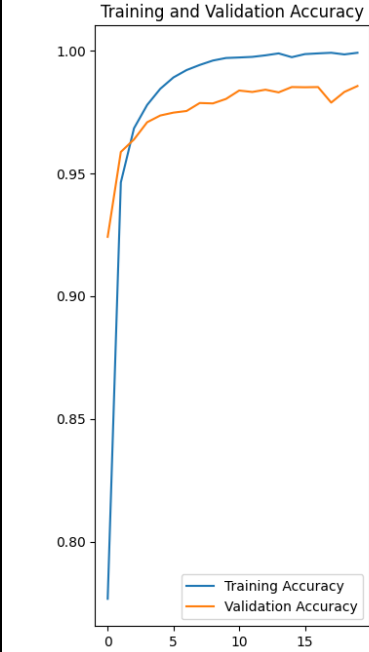
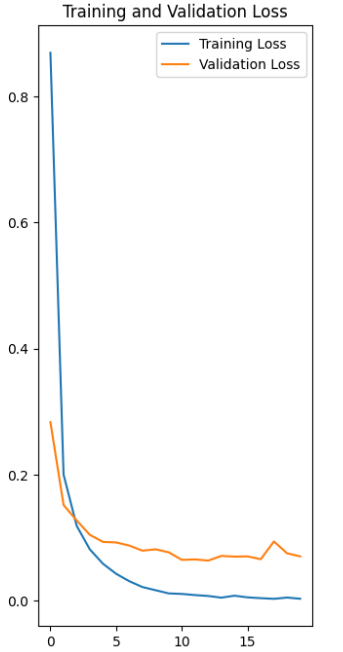
train_dataset = normalized_train_dataset.cache().prefetch(buffer_size=AUTOTUNE)
valid_dataset = normalized_valid_dataset.cache().prefetch(buffer_size=AUTOTUNE)
test_dataset = normalized_test_dataset.cache().prefetch(buffer_size=AUTOTUNE)
```

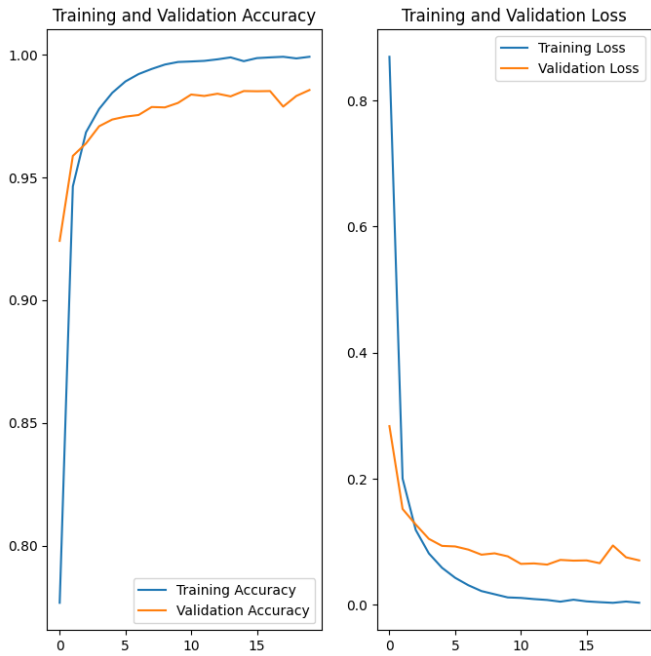
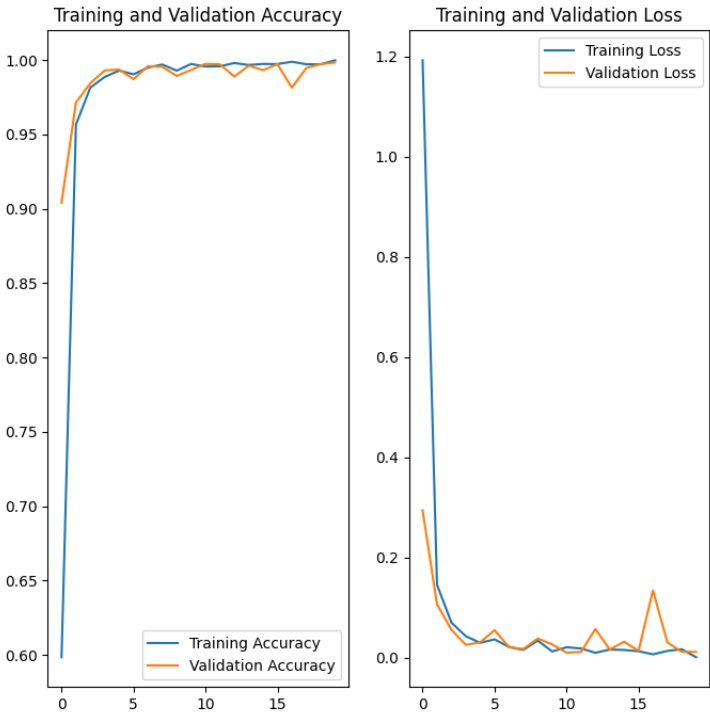
## Training and Comparing between Different Models

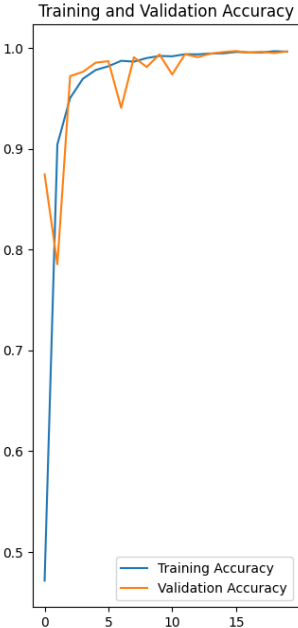
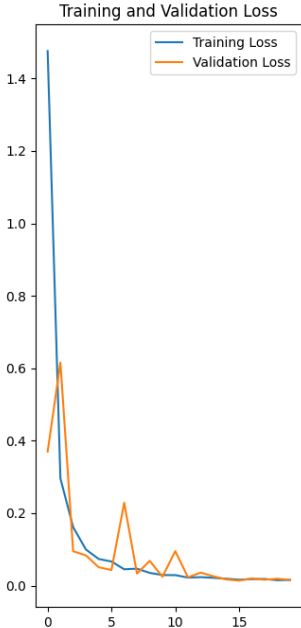
Multiple deep learning models, including VGG16, LeNet-5, and GoogleNet architectures, are trained and compared. The models are evaluated based on their training time, accuracy, and loss. The results indicate that the GoogleNet architecture achieves the highest accuracy with the lowest loss, making it the most suitable model for the hand gesture recognition task.



It is obvious that googleNet and vgg16 has the best accuracy but googleNet has the lowest loss. The Lenet-5 has the fastest training time but has the larger loss and less accuracy. So, the best model to pick in this case is googleNet.

	Accuracy / Loss		Time (Seconds)	Epochs
VGG16 Architecture	Accuracy = 99.6%		1382.9	20
	Loss = 0.026			
	<div><div>Training and Validation Accuracy</div><div>Training Accuracy Validation Accuracy</div></div> <div><div>Training and Validation Loss</div><div>Training Loss Validation Loss</div></div>			
Lenet-5 Architecture	Accuracy = 97.8%		155.4	20
	Loss = 0.105			
	<div><div>Training and Validation Accuracy</div><div>Training Accuracy Validation Accuracy</div></div> <div><div>Training and Validation Loss</div><div>Training Loss Validation Loss</div></div>			



GoogleNet Architecture	Accuracy = 99.5%	Loss = 0.022	931.3	20
	<div><div>Training and Validation Accuracy</div><div>Training and Validation Loss</div></div>			

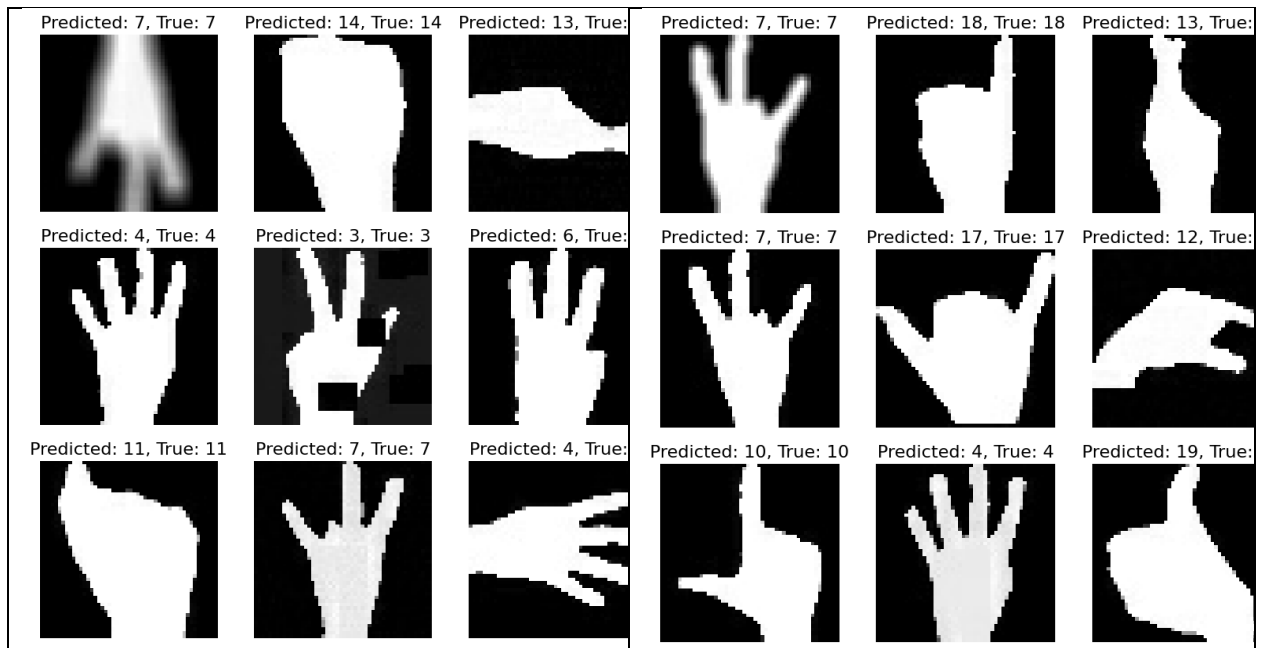
## Testing the GoogleNet Model

Test a random index image from the test dataset from the first batch of images.

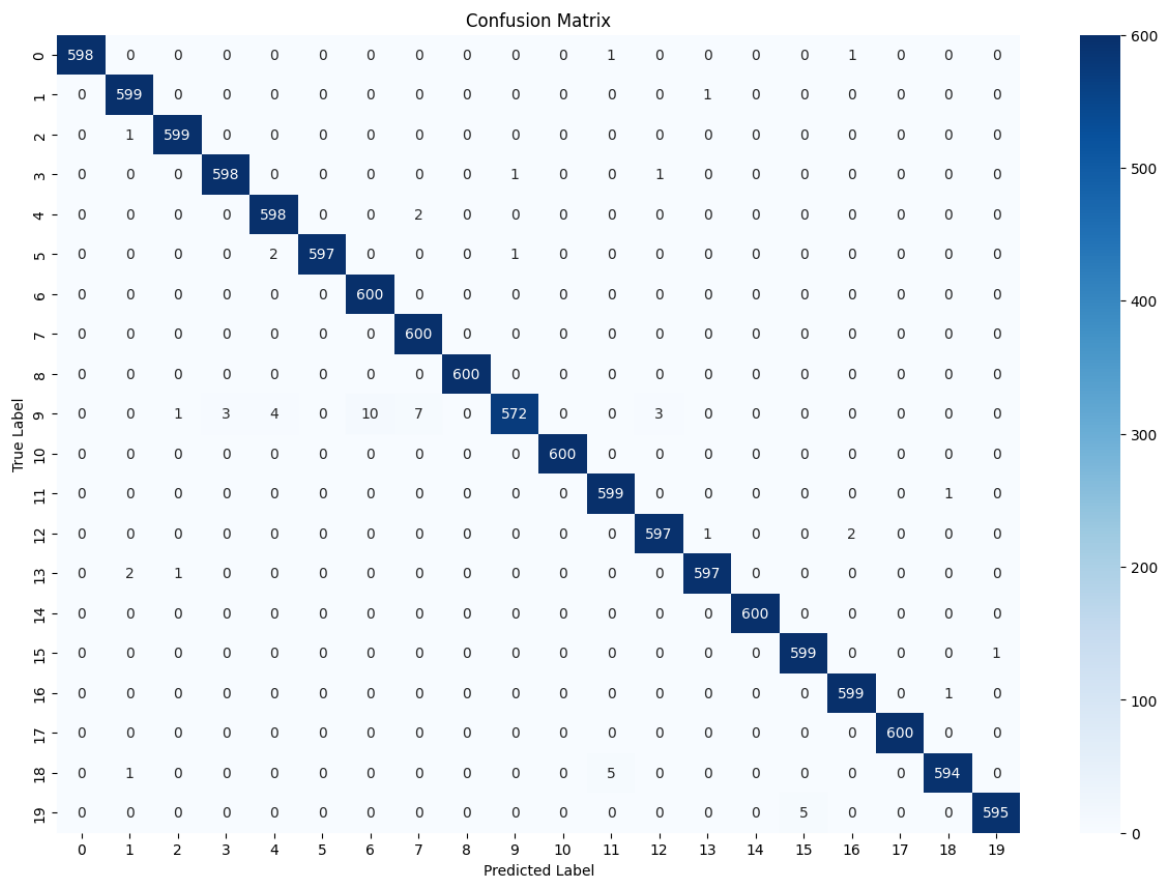




Testing multiple images from the test dataset provided.



## Confusion Matrix



## Testing the GoogleNet Model in Real Time

Testing the googleNet model in real time. Using computer vision library (OpenCV) to capture video and preprocessing the captured frames shown in green box below in real time, and the output image from the preprocessing is the mask image shown below.

```
[63]: cam = cv.VideoCapture(0)
cv.namedWindow("frame")

img_counter = 1

while True:
    ret, frame = cam.read()
    frame=cv.flip(frame,1)
    kernel = np.ones((3,3),np.uint8)

    # define region of interest
    roi=frame[100:350, 100:350]
    cv.rectangle(frame,(100,100),(350,350),(0,255,0),0)

    # convert region of interest into hsv color
    hsv = cv.cvtColor(roi, cv.COLOR_BGR2HSV)
    # define range of skin color in HSV
    lower_skin = np.array([0,20,70], dtype=np.uint8)
    upper_skin = np.array([20,255,255], dtype=np.uint8)

    # extract skin colour image
    mask = cv.inRange(hsv, lower_skin, upper_skin)

    # extrapolate the hand to fill dark spots within
    mask = cv.dilate(mask,kernel,iterations = 3)

    # blur the image
    mask = cv.GaussianBlur(mask,(3,3),10)

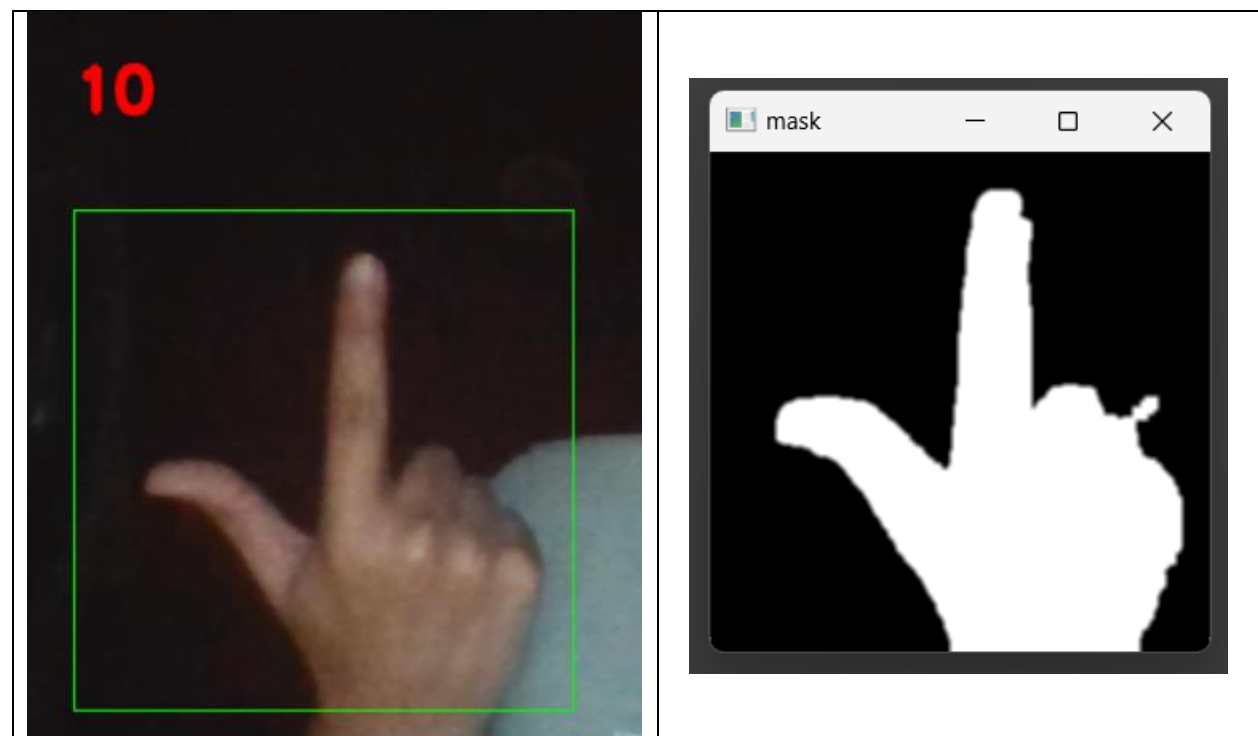
    test_mask=mask
    test_mask = cv.resize(test_mask, (50, 50))
    test_mask = tf.keras.utils.img_to_array(test_mask)
    test_mask = test_mask/255.
    test_mask = np.array([test_mask])
    test_mask_prediction = googleNet_model.predict(test_mask)
    test_mask_prediction_max=np.max(test_mask_prediction, axis=1)

    # print prediction on frame
    font = cv.FONT_HERSHEY_SIMPLEX
    if test_mask_prediction_max[0]<0.95:
        cv.putText(frame,'Put hand correctly in the box',(100,50), font, 1,
            (0,0,255), 3, cv.LINE_AA)
    else:
        test_mask_prediction_label = np.argmax(test_mask_prediction, axis=1)
        mask_label = np.array2string(test_mask_prediction_label[0])
        cv.putText(frame,mask_label,(100,50), font, 1, (0,0,255), 3, cv.LINE_AA)

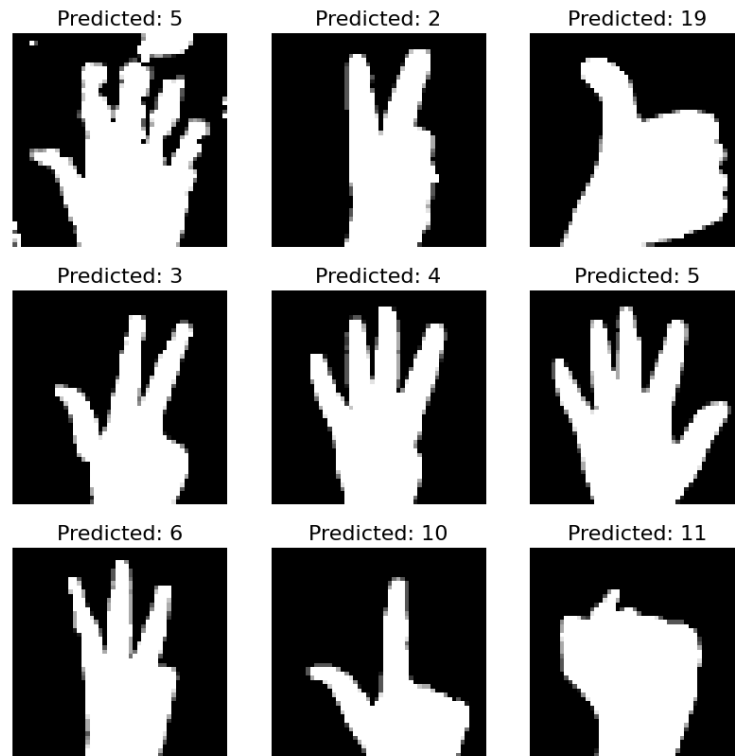
    if not ret:
        print("failed to grab frame")
        break

    cv.imshow("frame", frame)
    cv.imshow('mask', mask)
```

The model takes the mask and predict its label using model.predict(), if max prediction is above 0.95 (confidence) the result is shown in the frame in red above the green box shown below.



The trained GoogleNet model is tested on both individual test images and real-time video input. In the case of individual images, random images from the test dataset are selected for prediction. In real-time testing, the model uses computer vision techniques, specifically OpenCV, to capture video frames. The captured frames are preprocessed in real time, and the resulting mask image is fed into the trained model for gesture prediction. If the model's prediction confidence exceeds a certain threshold, the predicted gesture label is overlaid on the video frame, indicating the recognized hand gesture.



## Comparing with Others

For the signer dependent, the recognition accuracy ranges from 69% to 98%, with an average of 88.8% among the selected studies. On the other hand, the signer independent's recognition accuracy reported in the selected studies ranges from 48% to 97%, with an average recognition accuracy of 78.2%. The lack in the progress of continuous gesture recognition could indicate that more work is needed towards a practical vision-based gesture recognition system.

## References

1. [IEEE Xplore](#) [Published: 19 November 2021].
2. [Hand Gesture Recognition Based on Computer Vision: A Review of Techniques \(researchgate.net\)](#) [Published: 23 July 2020].