Deep Learning Extra Project : Rock Scissor Paper Machine

Course : 2023-1 DLIP Extra project

Student ID : 21800031

Name : Kook Jinho

Date : 2023-06-26

My github Link : JinhoKook (github.com)

File description	Root
main code	main.py
requirements	DLIP_GPU.yaml
OPENCV image data augmentation	data_augmentation.py
OPENCV image data generator	data_generator.py
trained model file	rsp_model.h5
model train code	training_code.ipynb
image raw datasets	raw_datasets folder
image processed datasets	rock_scissor_paper folder

I. Introduction

1. Purpose

(1) Familiar approach to deep learning with entertainment

- It is suitable for familiar approaches to deep learning. So I can show that deep learning can be sufficiently used for entertainment.

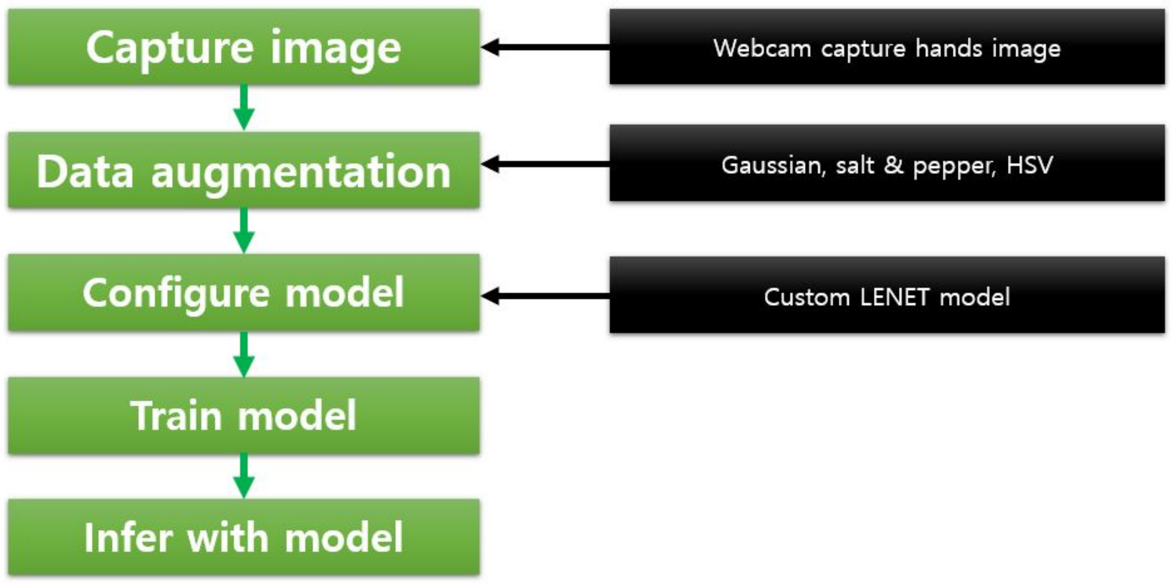
(2) Promotions to the Department of Mechanical Control Engineering

- It can give freshmen a familiar and interesting image of the Department of Mechanical Control Engineering.

2. Over View

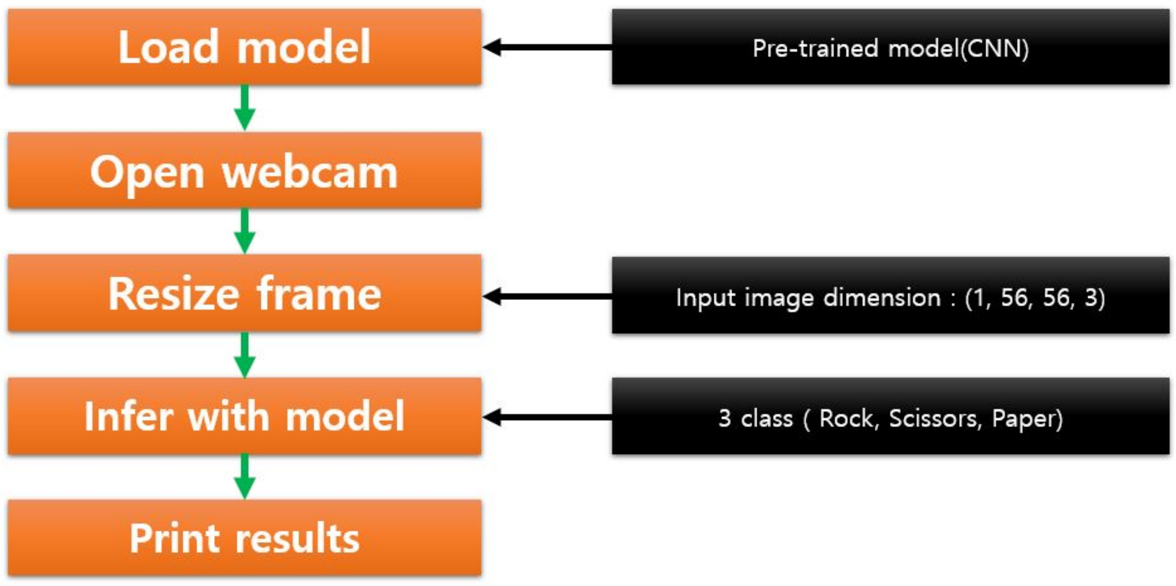
(a) Total flow chart

- This is a simple flow chart of the total process.



(b) Main code flow chart

- This is a simple flow chart of the main code.



3. Object

- Quick enough to play rock-paper-scissors(making light model).
- 90% or higher accuracy for custom test image.
- Whether the real-time model drive recognizes rock-paper-scissors.

II. Procedure

- **The order consists of data generation using a webcam, data augmentation, model training, and model testing.**

Procedure	File name	Contents
1. Data generation	data_generator.py	You take a picture of your hand using a webcam.
2. Data augmentation	data_augmentation.py	Image processing is performed to increase the number of data.
3. Train the model	training_code.ipynb	Construct and train the model.
4. Test the model	main.py	Use the webcam to play rock-paper-scissors in real time.

1. Environment setting

First, you have to set virtual enviroment that I provided and install requirements. But I provided yaml file that includes setting value. You just use it in ANACONDA.

Follow this:

```
conda env create -f DLIP_GPU.yaml
conda activate DLIP_GPU
```

- This format is optimized for CUDA 11.3 and PYTHON 3.9. Anything other than this format may cause errors.
- tensorflow 1.X is not recommand. And if you use tensorflow cpu version, train speed is very low.

My environment specification

- GPU : RTX 3060ti
- VRAM : 8GB
- GPU capability : 8.6
- cuda version : 11.3
- python version : 3.9.16
- pytorch version : 1.10.1
- cudnn : 8.0
- tensorflow-gpu version : 2.6.0

2. Generating datasets

- This is the distribution of datasets.
  - *Scissors can have various shapes, so more datasets are applied.*

	raw datasets number	**data augmentation number**	**total datasets number**
rock	200	300	500
scissor	340	197	537
paper	200	300	500
total	740	797	1537

(1) Generating datasets code

a. Directory

```
import cv2
import os

# Directory path to save the images
save_directory =
"C:/Users/ririk/Desktop/20230406_TEMP/DLIP_extra_project/raw_datasets/scissor"

...
```

- You can change directory where the image will be stored.
- Currently, it is set to the "scissor" directory, but if you need to create a new image, you need to create an additional image by changing the directory to "rock" and "paper".
- This is the raw data folder.

b. Video port and image number setting

```
...
# Open the webcam
cap = cv2.VideoCapture(0)

# Image counter
image_counter = 1
...
```

- Change the number of video ports, or determine from what point the image will be stored.
- If 'image_counter' set to 1, start saving from 1.jpg. If you want to change the image from the number you want, put the number of images you want here.

c. Save image

```
 ...
    # Press "s" key to save the current frame
    if cv2.waitKey(1) & 0xFF == ord("s"):
        # Set the file name as a number
        filename = str(image_counter) + ".jpg"
        filepath = os.path.join(save_directory, filename)
        ...
```

- 's' key set the capture webcam image.

(2) Data augmentation

a. Set directory and image number

```
# Folder path
folder_path = 
"C:/Users/ririk/Desktop/20230406_TEMP/DLIP_extra_project/raw_datasets/rock"

# Get the list of image files
file_list = os.listdir(folder_path)

# Number of expansions for each image
num_expansions = 300
...
```

- You can change how many images augmentation. Default number is 300.

b. Variation for images

```
...
# Select the type of transformation to apply
        transformation_type = random.choice(["gaussian", "salt_and_pepper", 
"hsv"])

        # Add Gaussian noise
        if transformation_type == "gaussian":
            mean = 0
            std_dev = 0.03  # Standard deviation of the noise
            noise = np.random.normal(mean, std_dev, 
image.shape).astype(np.uint8)
            noisy_image = cv2.add(image, noise)
            ...
```

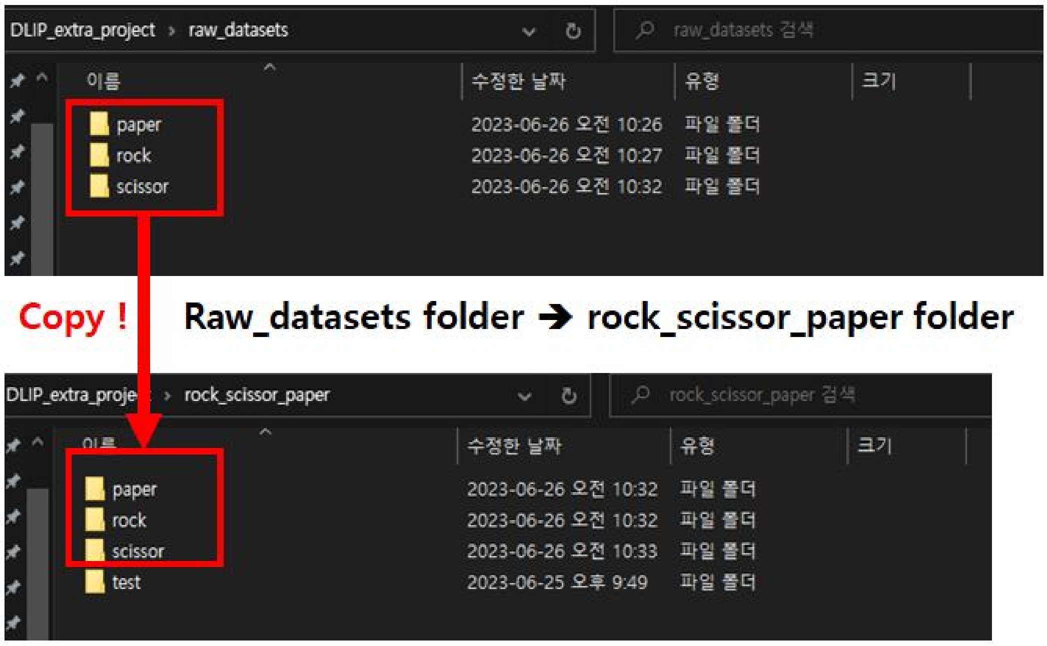
- You can add the desired noise among 'Gaussian blur',' salt and pepper', and 'hsv' values. The reason for doing this is to augment the data to make it more resilient to change.
- Each parameter can be modified as desired. For example, by changing the value of 'std_dev', the standard deviation for noise can be determined.
- However, it should be noted that if too much noise is given, learning can rather be degraded.

c. Image number setting

```
...
# Save the expanded image
expanded_file_path = os.path.join(folder_path, f"{i + 1001}.jpg")
cv2.imwrite(expanded_file_path, noisy_image)
```

- This code is the name code of the saved image.

(3) Copy raw data


DLIP_extra_project › raw_datasets
raw_datasets 검색
이름
수정한 날짜
유형
크기
Copy !
Raw_datasets folder ➔ rock_scissor_paper folder
DLIP_extra_proje › rock_scissor_paper
rock_scissor_paper 검색

```
paper     2023-06-26 오전 10:26   파일 폴더
rock      2023-06-26 오전 10:27   파일 폴더
scissor   2023-06-26 오전 10:32   파일 폴더
```

```
paper     2023-06-26 오전 10:32   파일 폴더
rock      2023-06-26 오전 10:32   파일 폴더
scissor   2023-06-26 오전 10:33   파일 폴더
test      2023-06-25 오후 9:49    파일 폴더
```

- The reason for doing this is to back up data just in case because it resizes the image to be used in the training model.

3. Train data

(1) Resize image

```
def resize_images(img_path):
    images=glob.glob(img_path + "/*.jpg")

    # save resize images
    target_size=(56,56)          # image size input !
    for img in images:
        old_img=Image.open(img)
        new_img=old_img.resize(target_size,Image.ANTIALIAS)
        new_img.save(img, "JPEG")

    print(len(images), " images resized.")
```

- You can change image size to fit model input. I selected 56 * 56 size.
- Too large image sizes weigh the model, and too small image sizes reduce the model's performance, requiring an appropriate compromise.

(2) Load data

```
def load_data(img_path, number_of_data=1537):       # Total data number
    # scissor : 0, rock : 1, paper : 2
    img_size=56     # Input image size
    color=3         # RGB : 3
    ...
```

- You have to fit the number of total datasets and input size.
- It will be printed that 'x_train shape: (1537, 56, 56, 3)'. It means that 1537 data number, 56* 56 image size, 3 channel for RGB.

(3) Configure CNN model

```
import tensorflow as tf
from tensorflow import keras
import numpy as np

model=keras.models.Sequential()
model.add(keras.layers.Conv2D(32, (3,3), activation='relu', input_shape=
(56,56,3)))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(64, (3,3), activation='relu'))
model.add(keras.layers.MaxPool2D(2,2))
model.add(keras.layers.Conv2D(128, (3,3), activation='relu'))
model.add(keras.layers.MaxPooling2D((2,2)))
model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(256, activation='relu'))
model.add(keras.layers.Dense(20, activation='softmax'))

model.summary()
```

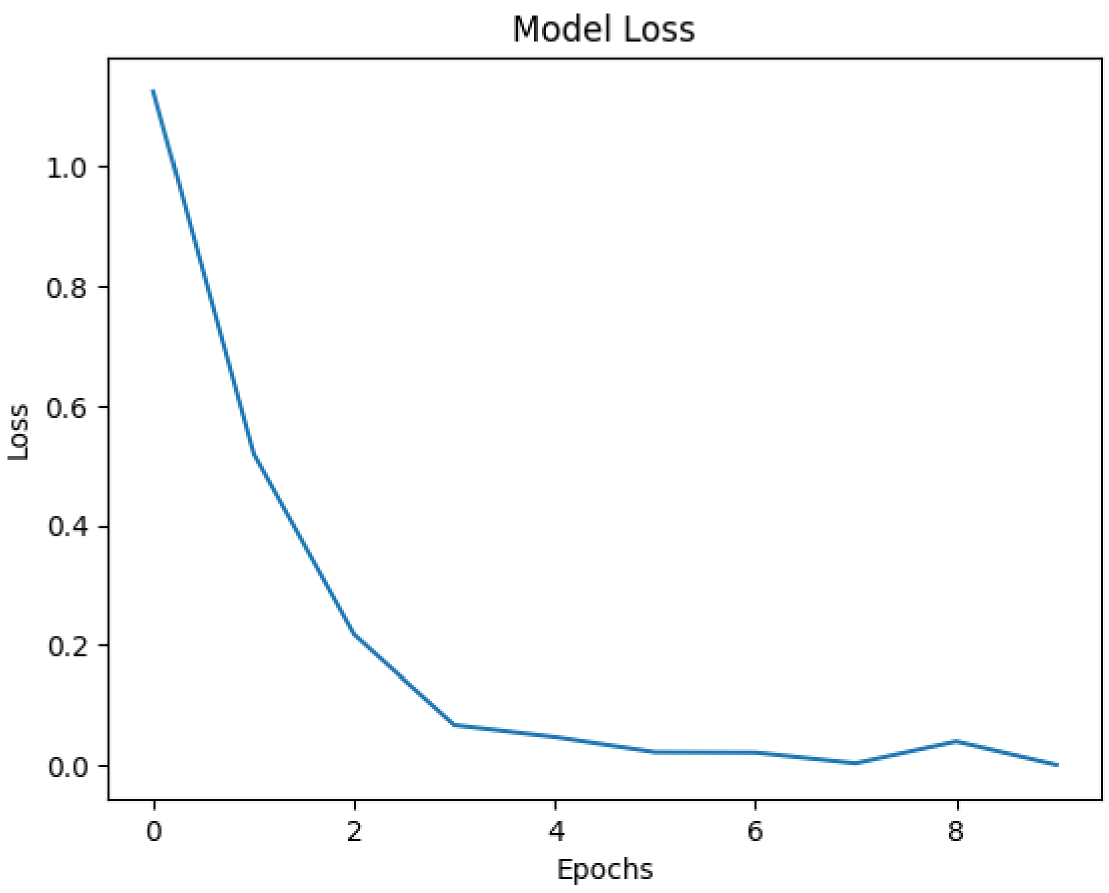
- I customized model a little bit.
- Based on the 'LENET' model used in the existing 'MNIST', the image input size was adjusted to 56×56, and a layer was added. Finally, 20 features could be extracted.
- The value of 56 was the image input size that was judged to be the most appropriate after several tests.

(4) Train model

```
import matplotlib.pyplot as plt
from tensorflow.keras.callbacks import EarlyStopping

# Early stopping
early_stopping = EarlyStopping(patience=2, monitor='loss',mode='min',
min_delta=0.01, verbose=1)
```

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

# Train
history = model.fit(x_train_norm, y_train,
                    epochs=100,
                    batch_size=4,
                    callbacks=[early_stopping])

# plot loss graph
plt.plot(history.history['loss'])
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.show()


model.save("./rsp_model.h5")
```

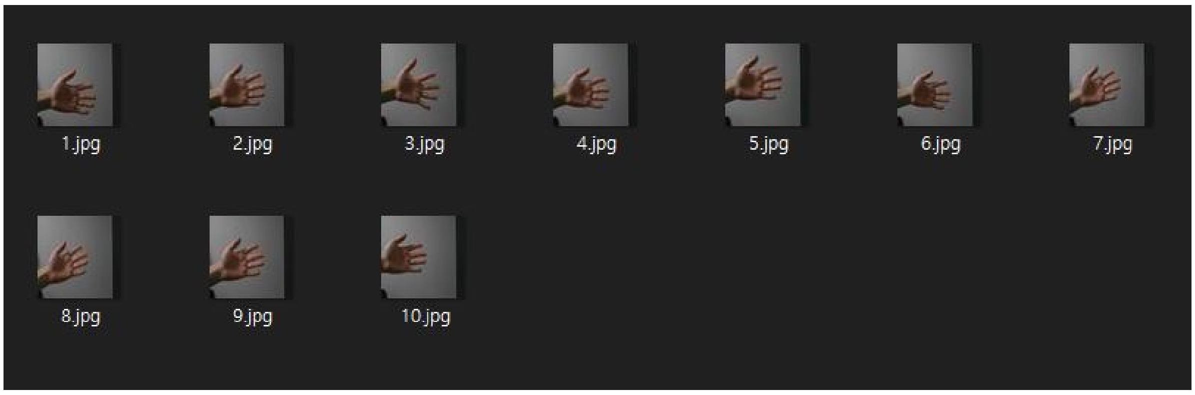
- Batch size, number of epochs, and early stopping conditions can be specified.
- The loss graph is output to the 'matplot library'.
- Finally, the model is stored in the same directory under the name 'rsp_model'.

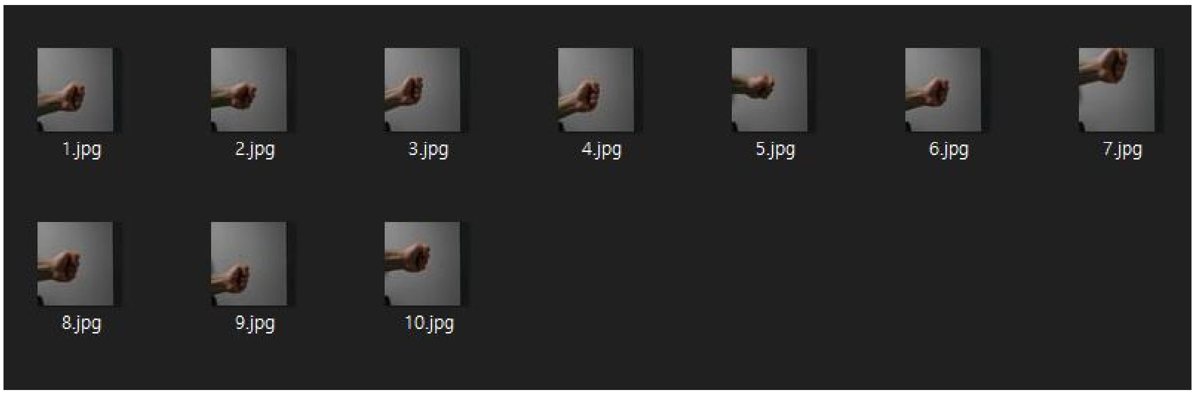


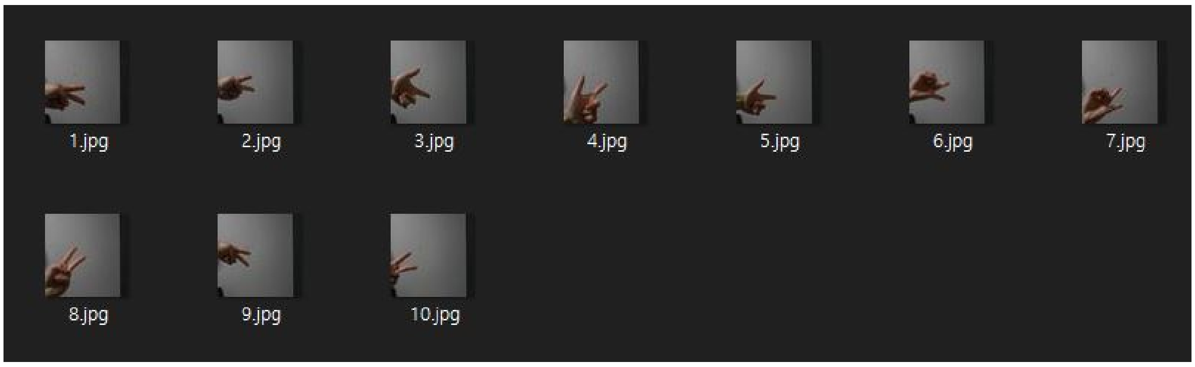
- This is the loss graph of the train.

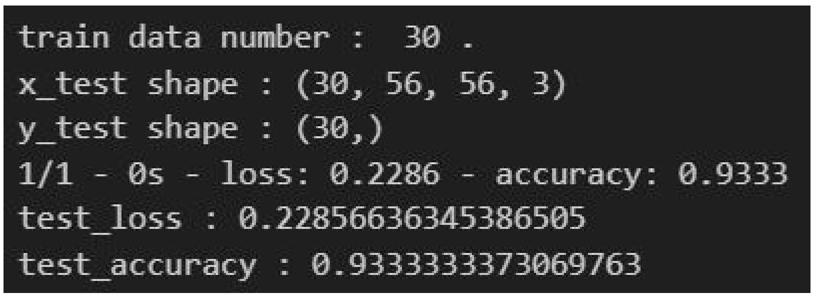
4. Test custom dataset

```
predicted_result = model.predict(x_test_norm)
predicted_labels = np.argmax(predicted_result, axis=1)

cor_num = 0
for idx in range(1,30):
    print(f'idx={idx}\n')
    #print('예측 결과 : ', predicted_result[idx])
    print('가장 가능성이 높은 결과 : ', predicted_labels[idx])
    print('실제 라벨 : ', y_test[idx])
    if (predicted_labels[idx] == y_test[idx]):
        cor_num += 1

print("맞힌 개수 : ", cor_num)
```









```
train data number :  30 .
x_test shape : (30, 56, 56, 3)
y_test shape : (30,)
1/1 - 0s - loss: 0.2286 - accuracy: 0.9333
test_loss : 0.22856636345386505
test_accuracy : 0.9333333373069763
```

- I newly made the following picture and used it as a test.
- As a result of testing a total of 30 data, 10 sheets for each class, the accuracy was 93.3%.

5. Real time test (webcam image)

You can run 'main.py' for the test. This code will call the trained model(rsp_model.h5).

(1) Load model and open webcam

```
...
# Load the trained model
model = load_model("rsp_model.h5")

# Open the webcam
cap = cv2.VideoCapture(0)
...
```

- The model has already been trained. Therefore, a model in the h5 format is imported.

(2) Resize input image

```
...
while True:
    # Read frame from the webcam
    ret, frame = cap.read()

    # Resize the frame for the model
    frame_resized = cv2.resize(frame, (56, 56))
    img = np.array(frame_resized, dtype=np.float32) / 255.0      # Normalize
image

    # Reshape the image dimension
    img = np.expand_dims(img, axis=0)  # Image dimension reshaped: (1, 56, 56,
3)

    # Predict the result using the model
    predicted_result = model.predict(img)
    predicted_labels = np.argmax(predicted_result, axis=1)
...
```

- Format the image received from the webcam and then normalize it.

III. Results

1. Object achievement

- The model is designed light enough to be fast enough for real-time recognition.
- The model's recognition rate exceeds 90% on the test dataset, so it can be said to be successful.
- As can be seen from the actual demonstration, it shows the accuracy of convergence at 100% on a real-time webcam camera.

2. Demo video

IV. Discussion

- **Selecting model**: There were concerns about the choice of models. Originally intended to use VGG-19, there was a problem that it took too long to run the model when the size of the input image was 224. Therefore, it was necessary to create a somewhat light model, and the model used in MNIST was customized to increase the recognition rate by adjusting the input image size to 56. As a result of this, it was possible to run a very fast model, allowing real-time rock-paper-scissors.
- **Various backgrounds for training**: I planned to make a model that is resistant to the environment by diversifying the backgrounds. However, it required too much data than expected. Even though I learned a total of 3,000 sheets of 1,000 sheets by myself through various backgrounds, I decided to fix the background because the accuracy was too low. As a result of fixing the background, it showed a much better recognition rate.

V. further works

- Create applicable models for different backgrounds with much more data and data augmentation.
- After floating rock-paper-scissors as an image, make actual working hand movements with MCU and servo motors.