```
### Hand Gesture Recognition Database
```

Context

Hand gesture recognition database is presented, composed by a set of near infrared images acquired by the Leap Motion sensor.

Content

The database is composed by 10 different hand-gestures (showed above) that were performed by 10 different subjects (5 men and 5 women).

The database is structured in different folders as:

/00 (subject with identifier 00)
/01_palm (images for palm gesture of subject 00)
/01palm/frame197957r.png,…,frame198136_l.png, … (images that corresponds to different samples obtained for the palm gesture performed by the subject with identifier 00)
/02_l (images for l gesture of subject 00)
/10_down
/01
/02
/09 (last subject with identifier 09)
Every root folder (00, 01,…) contains the infrared images of one subject. The folder name is the identifier of each different subject.

Citation

T. Mantecón, C.R. del Blanco, F. Jaureguizar, N. García, “Hand Gesture Recognition using Infrared Imagery Provided by Leap Motion Controller”, Int. Conf. on Advanced Concepts for Intelligent Vision Systems, ACIVS 2016, Lecce, Italy, pp. 47-57, 24-27 Oct. 2016. (doi: 10.1007/978-3-319-48680-2_5)

```
##!mkdir ~/.kaggle
```

```
##!cp /kaggle.json ~/.kaggle/
```

```
##!chmod 600 ~/.kaggle/kaggle.json
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
##! pip install kaggle
```

```
##!pip install keras-tuner
```

```
##!kaggle datasets download -d gti-upm/leapgestrecog
```

```
     Downloading leapgestrecog.zip to /content
      99% 2.11G/2.13G [00:22<00:00, 114MB/s]
     100% 2.13G/2.13G [00:22<00:00, 101MB/s]
```

```
#!unzip /content/leapgestrecog.zip
```

```
import tensorflow as tf
from tensorflow import keras
import numpy as np
from tensorflow.keras.applications.vgg19 import VGG19
from glob import glob
```

```
print(tf.__version__)
```

```
     2.7.0
```

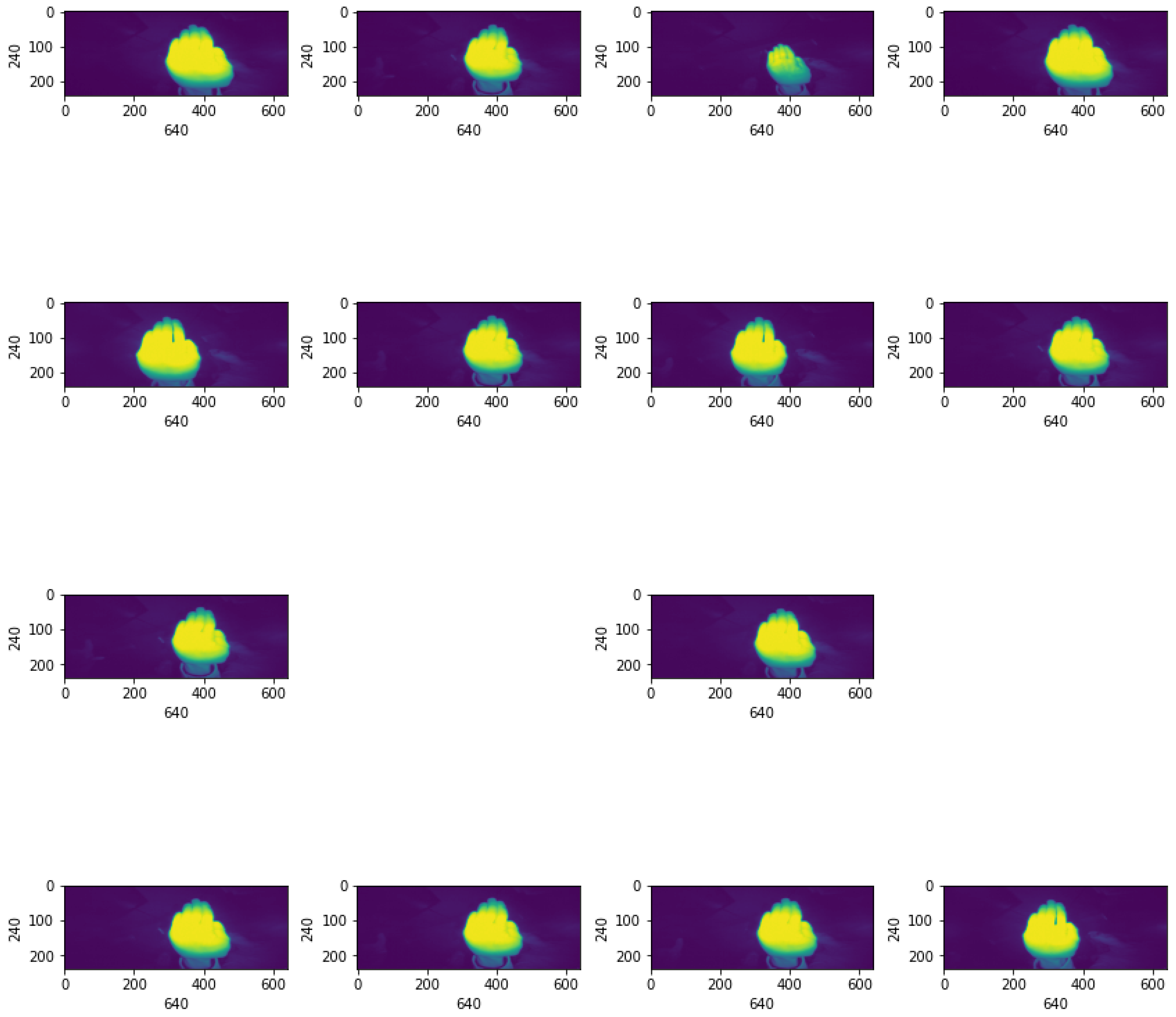
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.image import imread
import cv2
import random
import os
from os import listdir
from PIL import Image
from sklearn.preprocessing import label_binarize,  LabelBinarizer
from keras.preprocessing import image
from keras.preprocessing.image import img_to_array, array_to_img
from tensorflow.keras.optimizers import Adam # - Works
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Flatten, Dropout, Dense
from sklearn.model_selection import train_test_split
from keras.models import model_from_json
from tensorflow.keras.utils import to_categorical
```

```
from tensorflow.compat.v1 import ConfigProto
from tensorflow.compat.v1 import InteractiveSession

config = ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.5
config.gpu_options.allow_growth = True
session = InteractiveSession(config=config)
```

```
# Plotting 12 images to check dataset
```

```
plt.figure(figsize=(12,12))
path = "/content/leapGestRecog/00/01_palm"
for i in range(1,17):
    plt.subplot(4,4,i)
    plt.tight_layout()
    rand_img = imread(path +'/'+ random.choice(sorted(os.listdir(path))))
    plt.imshow(rand_img)
    plt.xlabel(rand_img.shape[1], fontsize = 10)#width of image
    plt.ylabel(rand_img.shape[0], fontsize = 10)#height of image
```



```
import os
import re
import glob
import hashlib
import argparse
import warnings

import six
import numpy as np
import tensorflow as tf
from tensorflow.python.platform import gfile
from keras.models import Model
from keras import backend as K
from keras.layers import Dense, GlobalAveragePooling2D, Input
from keras.applications.inception_v3 import InceptionV3
from keras.preprocessing.image import (ImageDataGenerator, Iterator,
                                       array_to_img, img_to_array, load_img)
from keras.callbacks import ModelCheckpoint, TensorBoard, EarlyStopping
```

```
image_generator = ImageDataGenerator(rescale=1/255, validation_split=0.2)

train_dataset = image_generator.flow_from_directory(batch_size=32,
                                                 directory='/content/leapGestRecog',
                                                 shuffle=True,
                                                 target_size=(224, 224),
                                                 subset="training",
                                                 class_mode='categorical')

validation_dataset = image_generator.flow_from_directory(batch_size=32,
                                                 directory='/content/leapgestrecog/leapGestRecog',
                                                 shuffle=True,
                                                 target_size=(224, 224),
                                                 subset="validation",
                                                 class_mode='categorical')
```

```
Found 16000 images belonging to 10 classes.
Found 4000 images belonging to 10 classes.
```

```
from tensorflow.keras.applications.vgg19 import VGG19
from glob import glob
```

```
# re-size all the images to this
IMAGE_SIZE = [224, 224]
```

```
# Import the Vgg 16 library as shown below and add preprocessing layer to the front of VGG
# Here we will be using imagenet weights

mobilnet = VGG19(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg19/vgg19_weights_tf_dim_ordering_tf_kernels_notop.h5
80142336/80134624 [==============================] - 1s 0us/step
80150528/80134624 [==============================] - 1s 0us/step
```

```
# don't train existing weights
for layer in mobilnet.layers:
    layer.trainable = False
```

```
  # useful for getting number of output classes
folders = glob('/content/leapGestRecog/*')
```

```
# our layers - you can add more if you want
x = Flatten()(mobilnet.output)
```

```
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=mobilnet.input, outputs=prediction)
```

```


# view the structure of the model
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0         
                                                                 
 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792      
                                                                 
 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928     
                                                                 
 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0         
                                                                 
 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856     
                                                                 
 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584    
                                                                 
 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0         
                                                                 
 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168    
                                                                 
 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080    
                                                                 
 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080    
 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv4 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv4 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 10)                250890

=================================================================
Total params: 20,275,274
Trainable params: 250,890
Non-trainable params: 20,024,384
_________________________________________________________________
```

```
model.compile(loss = 'categorical_crossentropy', optimizer = Adam(0.0001),metrics=['accuracy'])
```

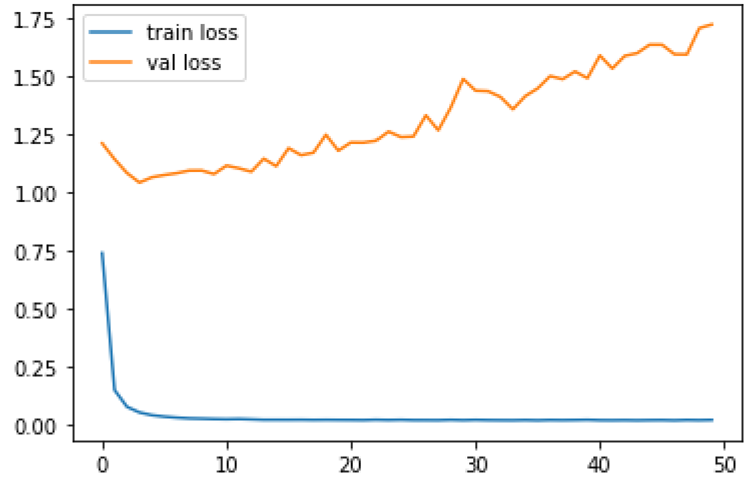
```
# fit the model
# Run the cell. It will take some time to execute
r = model.fit_generator(
  train_dataset,
  validation_data=validation_dataset,
  epochs=50,
  steps_per_epoch=len(train_dataset),
  validation_steps=len(validation_dataset)
)
```

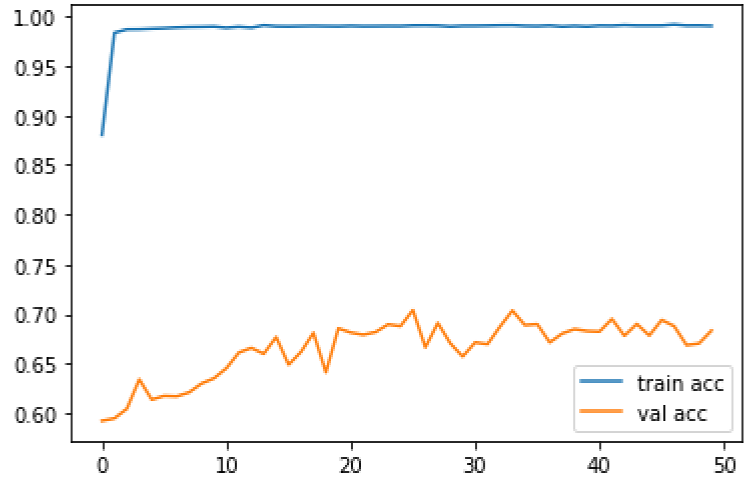
```
Epoch 22/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0170 - accuracy: 0.9901 - val_loss: 1.2128 - val_accuracy: 0.6793
Epoch 23/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0184 - accuracy: 0.9901 - val_loss: 1.2206 - val_accuracy: 0.6820
Epoch 24/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0173 - accuracy: 0.9902 - val_loss: 1.2605 - val_accuracy: 0.6898
Epoch 25/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0183 - accuracy: 0.9902 - val_loss: 1.2367 - val_accuracy: 0.6880
Epoch 26/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0169 - accuracy: 0.9908 - val_loss: 1.2392 - val_accuracy: 0.7042
```

```
Epoch 27/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0168 - accuracy: 0.9909 - val_loss: 1.3305 - val_accuracy: 0.6665
Epoch 28/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0166 - accuracy: 0.9906 - val_loss: 1.2660 - val_accuracy: 0.6913
Epoch 29/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0181 - accuracy: 0.9898 - val_loss: 1.3638 - val_accuracy: 0.6708
Epoch 30/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0167 - accuracy: 0.9904 - val_loss: 1.4876 - val_accuracy: 0.6572
Epoch 31/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0180 - accuracy: 0.9904 - val_loss: 1.4372 - val_accuracy: 0.6715
Epoch 32/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0169 - accuracy: 0.9906 - val_loss: 1.4349 - val_accuracy: 0.6697
Epoch 33/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0165 - accuracy: 0.9909 - val_loss: 1.4100 - val_accuracy: 0.6875
Epoch 34/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0162 - accuracy: 0.9910 - val_loss: 1.3567 - val_accuracy: 0.7038
Epoch 35/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0171 - accuracy: 0.9904 - val_loss: 1.4136 - val_accuracy: 0.6890
Epoch 36/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0161 - accuracy: 0.9902 - val_loss: 1.4471 - val_accuracy: 0.6900
Epoch 37/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0174 - accuracy: 0.9907 - val_loss: 1.5001 - val_accuracy: 0.6715
Epoch 38/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0168 - accuracy: 0.9898 - val_loss: 1.4865 - val_accuracy: 0.6805
Epoch 39/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0174 - accuracy: 0.9903 - val_loss: 1.5199 - val_accuracy: 0.6850
Epoch 40/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0182 - accuracy: 0.9898 - val_loss: 1.4909 - val_accuracy: 0.6830
Epoch 41/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0166 - accuracy: 0.9906 - val_loss: 1.5889 - val_accuracy: 0.6825
Epoch 42/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0163 - accuracy: 0.9905 - val_loss: 1.5326 - val_accuracy: 0.6952
Epoch 43/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0167 - accuracy: 0.9912 - val_loss: 1.5871 - val_accuracy: 0.6783
Epoch 44/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0160 - accuracy: 0.9907 - val_loss: 1.5989 - val_accuracy: 0.6902
Epoch 45/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0167 - accuracy: 0.9907 - val_loss: 1.6355 - val_accuracy: 0.6785
Epoch 46/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0170 - accuracy: 0.9908 - val_loss: 1.6354 - val_accuracy: 0.6942
Epoch 47/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0160 - accuracy: 0.9918 - val_loss: 1.5942 - val_accuracy: 0.6880
Epoch 48/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0174 - accuracy: 0.9907 - val_loss: 1.5936 - val_accuracy: 0.6687
Epoch 49/50
500/500 [==============================] - 171s 342ms/step - loss: 0.0167 - accuracy: 0.9908 - val_loss: 1.7069 - val_accuracy: 0.6705
Epoch 50/50
```

```
# plot the loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')
```

```
# plot the accuracy
plt.plot(r.history['accuracy'], label='train acc')
plt.plot(r.history['val_accuracy'], label='val acc')
plt.legend()
plt.show()
plt.savefig('AccVal_acc')
```





```
<Figure size 432x288 with 0 Axes>
```

```
# save it as a h5 file

from tensorflow.keras.models import load_model

model.save('model_vgg19_handgesture.h5')
```

```
y_pred = model.predict(train_dataset)
```

```
y_pred
```

```
array([[5.6797245e-10, 8.6474400e-10, 5.6291728e-11, ..., 9.8205660e-08,
        9.9999988e-01, 3.2789446e-10],
       [1.2027229e-10, 5.7065214e-11, 1.8392081e-11, ..., 1.1981915e-14,
        4.3243496e-08, 2.5218282e-11],
       [2.7785829e-09, 5.5848137e-10, 4.3264334e-09, ..., 7.2560645e-08,
        2.6487925e-07, 9.3228323e-09],
       ...,
       [1.9318423e-10, 8.6262119e-08, 6.1195720e-08, ..., 9.9999952e-01,
        9.5587872e-08, 9.5443853e-10],
       [6.9469621e-08, 1.4884543e-13, 8.7136992e-10, ..., 1.5852507e-08,
        9.9999976e-01, 6.9808888e-11],
       [6.8279276e-09, 2.8359232e-11, 3.0921584e-08, ..., 1.9838497e-07,
        1.2801022e-08, 7.7465685e-08]], dtype=float32)
```

```
import numpy as np
y_pred = np.argmax(y_pred, axis=1)
```

```
# Evaluating model on validation data
evaluate = model.evaluate(validation_dataset)
print(evaluate)
```

```
125/125 [==============================] - 34s 274ms/step - loss: 1.7227 - accuracy: 0.6835
[1.7226845026016235, 0.6834999918937683]
```

```
from sklearn.metrics import classification_report, confusion_matrix
def give_accuracy():
    p=model.predict(validation_dataset)
    cm=confusion_matrix(y_true=validation_dataset.classes,y_pred=np.argmax(p,axis=-1))
    acc=cm.trace()/cm.sum()
    print('The Classification Report \n', cm)
    print(f'Accuracy: {acc*100}')
```

```
give_accuracy()
```

```
import numpy as np
from tensorflow.keras.preprocessing import image
test_image = image.load_img('/content/leapGestRecog/01/03_fist/frame_01_03_0001.png', target_size = (224,224))
test_image = image.img_to_array(test_image)
test_image=test_image/255
test_image = np.expand_dims(test_image, axis = 0)
result = model.predict(test_image)
```

```
test = np.array(test_image)
```

```
# making predictions On the image
prediction = np.argmax(model.predict(test_image))
```

```
prediction
```

```
print("The prediction Of the Image is : ", prediction)
print("The prediction Of the Image is : ", prediction)
```

```
The prediction Of the Image is :  1
```

```
# show the Original image
import matplotlib.pyplot as plt
test_image = image.load_img('/content/leapGestRecog/01/03_fist/frame_01_03_0001.png', target_size = (224,224))
plt.axis('off')
plt.imshow(test_image)
plt.show()
```

```
import numpy as np
from tensorflow.keras.preprocessing import image
test_image = image.load_img('/content/leapGestRecog/06/05_thumb/frame_06_05_0003.png', target_size = (224,224))
test_image = image.img_to_array(test_image)
test_image=test_image/255
test_image = np.expand_dims(test_image, axis = 0)
result = model.predict(test_image)
```

```
test = np.array(test_image)
```

```
#·making·predictions·On·the·image
prediction = np.argmax(model.predict(test_image))
```

```
prediction
```

```
6
```

```
print("The prediction Of the Image is : ", prediction)
```

```
The prediction Of the Image is :  6
```

```
# show the Original image
import matplotlib.pyplot as plt
```

```
test_image = image.load_img('/content/leapGestRecog/06/05_thumb/frame_06_05_0003.png', target_size = (224,224))
plt.axis('off')
plt.imshow(test_image)
plt.show()
```

✓ 0s completed at 4:02 PM