# TASK 4

Develop a hand gesture recognition model that can accurately identify and classify different hand gestures from image or video data, enabling intuitive human-computer interaction and gesture-based control systems.

In [2]:
```
import warnings
warnings.filterwarnings('ignore')

import keras
import matplotlib.pyplot as plt # for plotting
import os # provides a way of using operating system dependent functio
import cv2 #Image handling library
import numpy as np

# Import of keras model and hidden layers for our convolutional networ
from keras.layers import Conv2D, Activation, MaxPool2D, Dense, Flatte

CATEGORIES = ["01_palm", '02_l','03_fist','04_fist_moved','05_thumb',
IMG_SIZE = 50

# paths for dataset
data_path = "/Users/manojt/Documents/leapGestRecog"
```

In [3]:
```
# Loading the images and their class(0 - 9)
image_data = []
for dr in os.listdir(data_path):
    for category in CATEGORIES:
        class_index = CATEGORIES.index(category)
        path = os.path.join(data_path, dr, category)
        for img in os.listdir(path):
            try:
                img_arr = cv2.imread(os.path.join(path, img), cv2.IMRE
                image_data.append([cv2.resize(img_arr, (IMG_SIZE, IMG_
            except Exception as e:
                pass
image_data[0]
```

Out[3]:
```
[array([[ 5,  6,  6, ...,  5,  4,  5],
        [ 5,  6,  7, ...,  3,  4,  3],
        [ 6,  6,  7, ...,  3,  3,  4],
        ...,
        [ 7, 10, 11, ...,  5,  5,  5],
        [ 8, 10, 12, ...,  3,  5,  5],
        [ 8,  9, 11, ...,  5,  5,  5]], dtype=uint8),
 0]
```

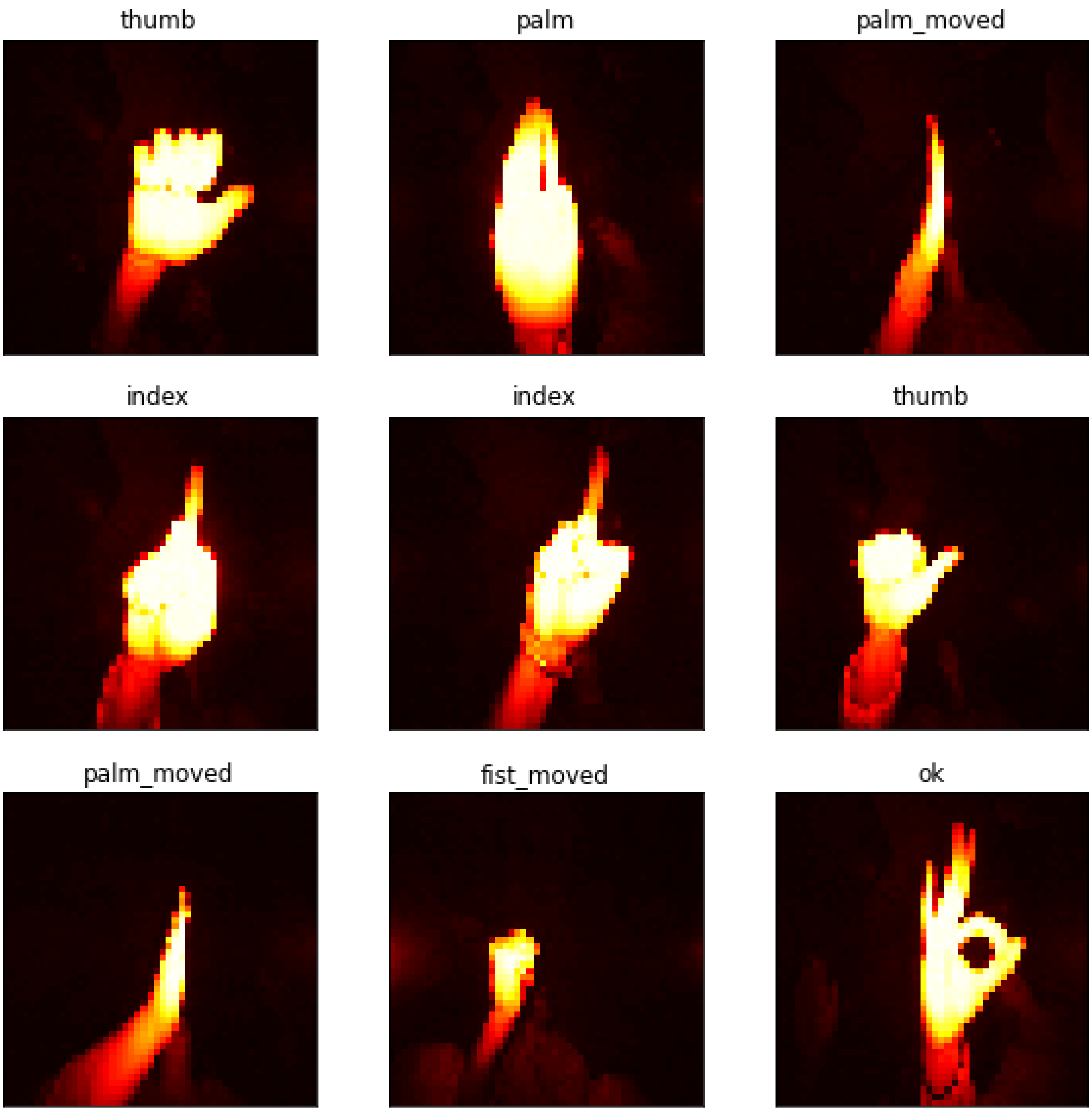
In [4]:
```
# shuffle the input data
import random
random.shuffle(image_data)
```

In [5]:
```
input_data = []
label = []
for X, y in image_data:
    input_data.append(X)
    label.append(y)
```

In [6]:
```
label[:10]
```

Out[6]:
```
[9, 4, 0, 7, 5, 5, 4, 7, 3, 6]
```

In [7]:
```
plt.figure(1, figsize=(10,10))
for i in range(1,10):
    plt.subplot(3,3,i)
    plt.imshow(image_data[i][0], cmap='hot')
    plt.xticks([])
    plt.yticks([])
    plt.title(CATEGORIES[label[i]][3:])
plt.show()
```




The code creates a visual representation of hand gestures by displaying grayscale images in a 3x3 grid. Each image corresponds to a specific finger gesture, allowing easy identification and understanding of various hand movements. The absence of axis labels enhances clarity, making it a straightforward tool for diagnosing finger-based hand gestures.

In [8]:
```
# Normalizing the data
input_data = np.array(input_data)
label = np.array(label)
input_data = input_data/255.0
input_data.shape
```

Out[8]:
```
(20000, 50, 50)
```

In [9]:
```
# one hot encoding
label = keras.utils.to_categorical(label, num_classes=10,dtype='i1')
label[0]
```

Out[9]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 1], dtype=int8)
```

In [10]:
```
# reshaping the data
input_data.shape = (-1, IMG_SIZE, IMG_SIZE, 1)
```

In [11]:
```
# splitting the input_data to train and test data
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(input_data, label,
```

In [12]:
```
model = keras.models.Sequential()

model.add(Conv2D(filters = 32, kernel_size = (3,3), input_shape = (IMG
model.add(Activation('relu'))


model.add(Conv2D(filters = 32, kernel_size = (3,3)))
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.3))

model.add(Conv2D(filters = 64, kernel_size = (3,3)))
model.add(Activation('relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.3))

model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(10, activation='softmax'))

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

This block defines the architecture of the Convolutional Neural Network (CNN) model for hand gesture recognition.

In [13]:
```
model.fit(X_train, y_train, epochs = 7, batch_size=32, validation_data
```

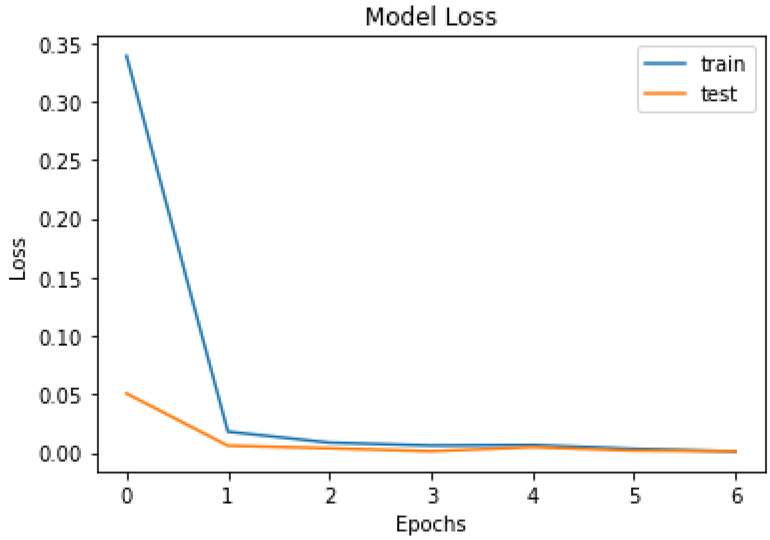
```
Train on 14000 samples, validate on 6000 samples
Epoch 1/7
14000/14000 [==============================] - 7s 521us/step - loss:
0.3391 - accuracy: 0.8910 - val_loss: 0.0507 - val_accuracy: 0.9820
Epoch 2/7
14000/14000 [==============================] - 3s 225us/step - loss:
0.0180 - accuracy: 0.9945 - val_loss: 0.0060 - val_accuracy: 0.9992
Epoch 3/7
14000/14000 [==============================] - 3s 233us/step - loss:
0.0086 - accuracy: 0.9974 - val_loss: 0.0038 - val_accuracy: 0.9993
Epoch 4/7
14000/14000 [==============================] - 4s 271us/step - loss:
0.0062 - accuracy: 0.9984 - val_loss: 0.0013 - val_accuracy: 0.9997
Epoch 5/7
14000/14000 [==============================] - 3s 228us/step - loss:
0.0066 - accuracy: 0.9985 - val_loss: 0.0047 - val_accuracy: 0.9990
Epoch 6/7
14000/14000 [==============================] - 3s 245us/step - loss:
0.0033 - accuracy: 0.9992 - val_loss: 0.0018 - val_accuracy: 0.9997
Epoch 7/7
14000/14000 [==============================] - 3s 231us/step - loss:
0.0011 - accuracy: 0.9998 - val_loss: 0.0013 - val_accuracy: 0.9997
```

Out[13]:
```
<keras.callbacks.callbacks.History at 0x7fac0c13ec10>
```

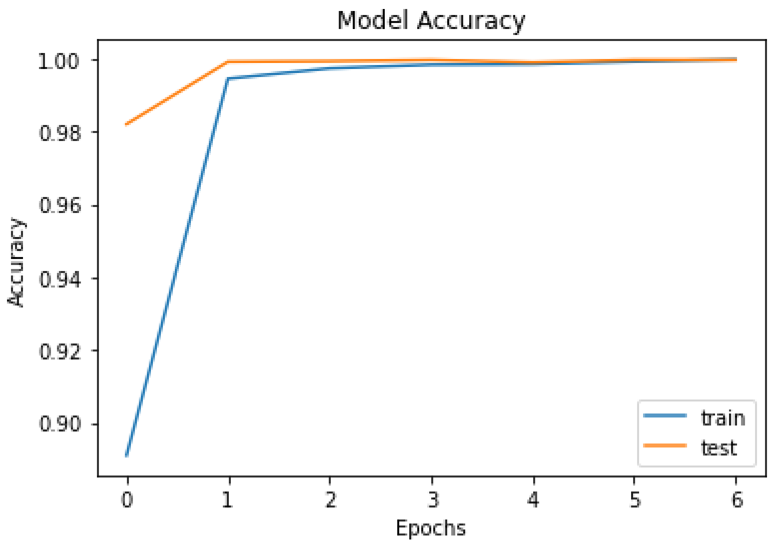
In [14]:
```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 48, 48, 32)        320
_________________________________________________________________
activation_1 (Activation)    (None, 48, 48, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 46, 46, 32)        9248
_________________________________________________________________
activation_2 (Activation)    (None, 46, 46, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 23, 23, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 21, 21, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 21, 21, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 10, 10, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 10, 10, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 6400)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               1638656
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2570
=================================================================
Total params: 1,669,290
Trainable params: 1,669,290
Non-trainable params: 0
_________________________________________________________________
```

In [15]:
```
plt.plot(model.history.history['loss'])
plt.plot(model.history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```




In [16]:
```
plt.plot(model.history.history['accuracy'])
plt.plot(model.history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epochs')
plt.legend(['train', 'test'])
plt.show()
```




In [17]:
```
#calculate loss and accuracy on test data

test_loss, test_accuracy = model.evaluate(X_test, y_test)

print('Test accuracy: {:2.2f}%'.format(test_accuracy*100))
```

```
6000/6000 [==============================] - 1s 98us/step
Test accuracy: 99.97%
```

*Test accuracy: 99.97%* highlights the model's exceptional performance. It correctly recognized hand gestures in 99.97% of the test samples, demonstrating its high accuracy and reliability for human-computer interaction and gesture-based control systems.

In [22]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sn
cat = [c[3:] for c in CATEGORIES]
plt.figure(figsize=(10,10))
cm = confusion_matrix(np.argmax(y_test, axis=1), np.argmax(model.pred:
sn.heatmap(cm, annot=True,xticklabels=cat, yticklabels=cat)
plt.plot()
```

Out[22]: []