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.

Importing libraries

In [4]:
```
import numpy as np
import pandas as pd
import os
import cv2

import tensorflow as tf
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import ImageDataGenerator

from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D

import matplotlib.pyplot as plt
```

Reading images

In [5]:
```
dir = r'C:\Users\Satoshi\Desktop\Data\prodigy-4\leapGestRecog'
```

In [6]:
```
images = []
labels = []
for directory in os.listdir(dir):
  for subDir in os.listdir(os.path.join(dir,directory)):
    for img in os.listdir(os.path.join(dir, directory, subDir)):
      img_path = os.path.join(dir, directory, subDir, img)
      images.append(img_path)
      labels.append(subDir)
```

In [7]:
```
# images = np.array(images)
# labels = np.array(labels)
# labels
```

Converting the data to DataFrame

In [8]:
```
Iseries = pd.Series(images, name="Images")
Lseries = pd.Series(labels, name="labels")
hand_gesture_data = pd.concat([Iseries, Lseries], axis=1)
hand_gesture_df = pd.DataFrame(hand_gesture_data)
```

counting the images in each class

In [10]:
```
pd.Series(labels).value_counts()
```

Out[10]:
```
05_thumb         2000
01_palm          2000
03_fist          2000
07_ok            2000
02_l             2000
09_c             2000
08_palm_moved    2000
10_down          2000
04_fist_moved    2000
06_index         2000
dtype: int64
```

Splitting the dataset into train and test

In [11]:
```
X_train, X_test = train_test_split(hand_gesture_df, test_size=0.2, random_state=42)
train_set, val_set = train_test_split(hand_gesture_df, test_size=0.3, random_state=42)
```

Data Preprocessing

In [12]:
```
image_gen = ImageDataGenerator(preprocessing_function= tf.keras.applications.mobilenet_v2.preprocess_input)
train = image_gen.flow_from_dataframe(dataframe= train_set,x_col="Images",y_col="labels",
                                      target_size=(244,244),
                                      color_mode='rgb',
                                      class_mode="categorical",
                                      batch_size=4,
                                      shuffle=False
                                     )
test = image_gen.flow_from_dataframe(dataframe= X_test,x_col="Images", y_col="labels",
                                     target_size=(244,244),
                                     color_mode='rgb',
                                     class_mode="categorical",
                                     batch_size=4,
                                     shuffle= False
                                    )
val = image_gen.flow_from_dataframe(dataframe= val_set,x_col="Images", y_col="labels",
                                    target_size=(244,244),
                                    color_mode= 'rgb',
                                    class_mode="categorical",
                                    batch_size=4,
                                    shuffle=False
                                   )
```

```
Found 14000 validated image filenames belonging to 10 classes.

Found 4000 validated image filenames belonging to 10 classes.

Found 6000 validated image filenames belonging to 10 classes.
```

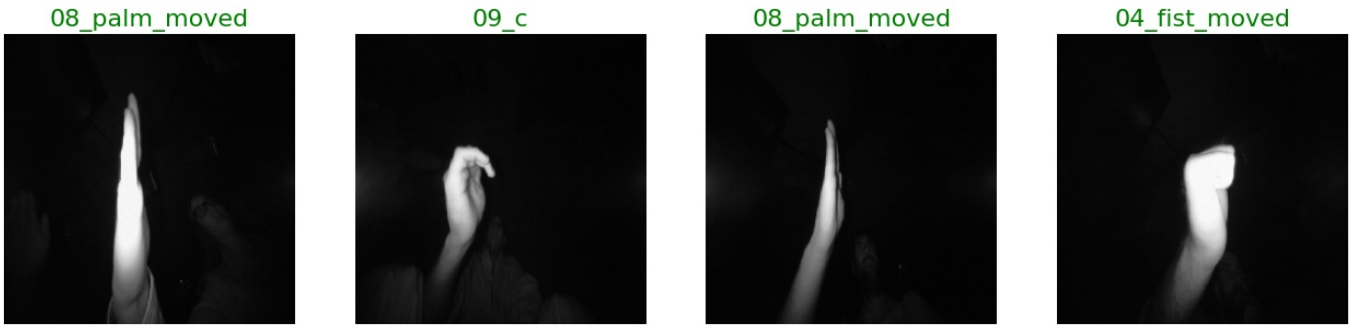
In [13]:
```
classes=list(train.class_indices.keys())
print (classes)
```

```
['01_palm', '02_l', '03_fist', '04_fist_moved', '05_thumb', '06_index', '07_ok', '08_palm_moved', '09_c', '10_do
wn']
```

Data visualization

In [14]:
```
def show_hand_gesture(image_gen):
    test_dict = test.class_indices
    classes = list(test_dict.keys())
    images, labels=next(image_gen)
    plt.figure(figsize=(20,20))
    length = len(labels)
    if length<25:
        r=length
    else:
        r=25
    for i in range(r):
        plt.subplot(5,5,i+1)
        image=(images[i]+1)/2
        plt.imshow(image)
        index=np.argmax(labels[i])
        class_name=classes[index]
        plt.title(class_name, color="green",fontsize=16)
        plt.axis('off')
    plt.show()
```

In [15]:
```
show_hand_gesture(train)
```



Building my model

In [20]:
```
model = keras.models.Sequential([
    keras.layers.Conv2D(filters=128, kernel_size=(8, 8), strides=(3, 3), activation='relu', input_shape=(224, 2
```

```
    keras.layers.BatchNormalization(),

    keras.layers.Conv2D(filters=256, kernel_size=(5, 5), strides=(1, 1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(3, 3)),

    keras.layers.Conv2D(filters=256, kernel_size=(3, 3), strides=(1, 1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(1, 1), strides=(1, 1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.Conv2D(filters=256, kernel_size=(1, 1), strides=(1, 1), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),

    keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),
    keras.layers.MaxPool2D(pool_size=(2, 2)),

    keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),

    keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),

    keras.layers.MaxPool2D(pool_size=(2, 2)),

    keras.layers.Conv2D(filters=512, kernel_size=(3, 3), activation='relu', padding="same"),
    keras.layers.BatchNormalization(),

    keras.layers.MaxPool2D(pool_size=(2, 2)),

    keras.layers.Flatten(),
    keras.layers.Dense(1024, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(1024, activation='relu'),
    keras.layers.Dropout(0.5),
    keras.layers.Dense(10, activation='softmax')
])

model.compile(
    loss='categorical_crossentropy',
    optimizer=tf.optimizers.SGD(learning_rate=0.001),
    metrics=['accuracy']
)

model.summary()
```

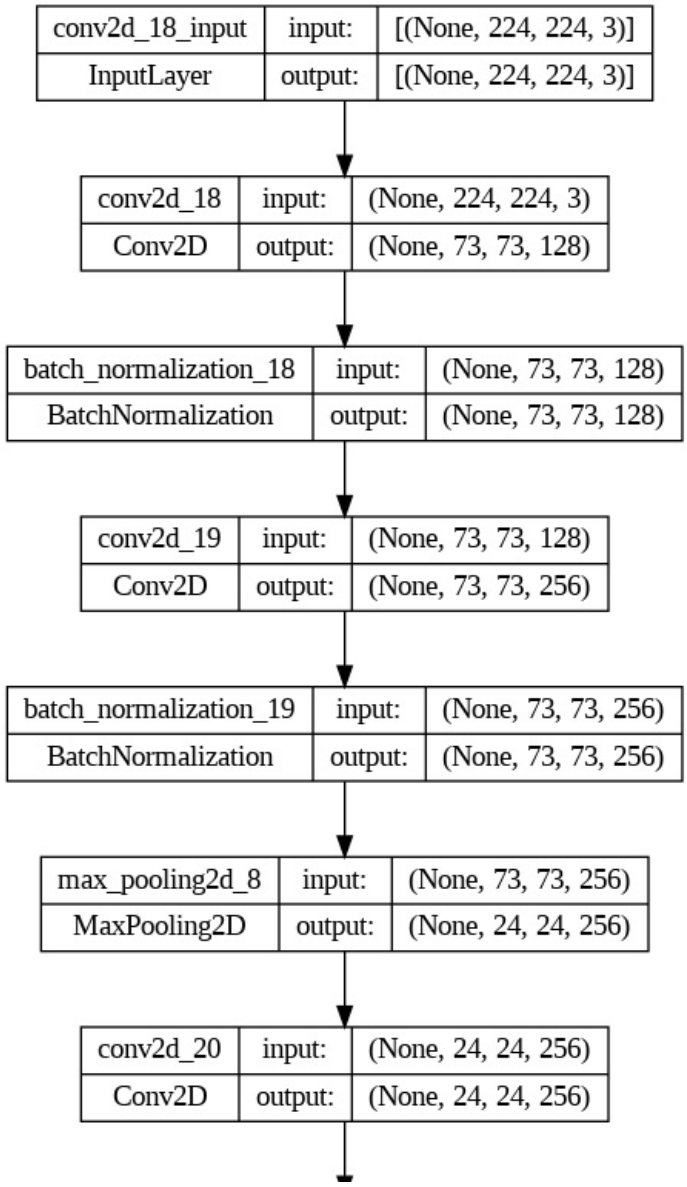
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_18 (Conv2D)          (None, 73, 73, 128)       24704


 batch_normalization_18 (Ba  (None, 73, 73, 128)       512
 tchNormalization)


 conv2d_19 (Conv2D)          (None, 73, 73, 256)       819456


 batch_normalization_19 (Ba  (None, 73, 73, 256)       1024
 tchNormalization)


 max_pooling2d_8 (MaxPoolin  (None, 24, 24, 256)       0
 g2D)


 conv2d_20 (Conv2D)          (None, 24, 24, 256)       590080
```

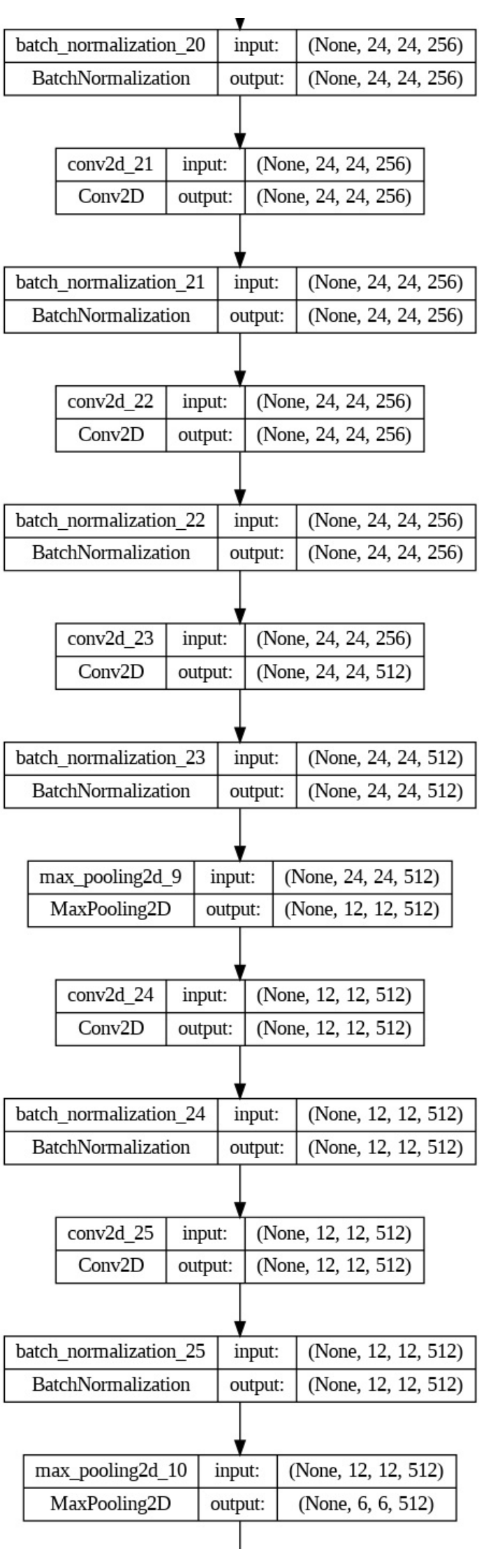
```
batch_normalization_20 (Ba  (None, 24, 24, 256)       1024
tchNormalization)

conv2d_21 (Conv2D)          (None, 24, 24, 256)       65792

batch_normalization_21 (Ba  (None, 24, 24, 256)       1024
tchNormalization)

conv2d_22 (Conv2D)          (None, 24, 24, 256)       65792

batch_normalization_22 (Ba  (None, 24, 24, 256)       1024
tchNormalization)

conv2d_23 (Conv2D)          (None, 24, 24, 512)       1180160

batch_normalization_23 (Ba  (None, 24, 24, 512)       2048
tchNormalization)

max_pooling2d_9 (MaxPoolin  (None, 12, 12, 512)       0
g2D)

conv2d_24 (Conv2D)          (None, 12, 12, 512)       2359808

batch_normalization_24 (Ba  (None, 12, 12, 512)       2048
tchNormalization)

conv2d_25 (Conv2D)          (None, 12, 12, 512)       2359808

batch_normalization_25 (Ba  (None, 12, 12, 512)       2048
tchNormalization)

max_pooling2d_10 (MaxPooli  (None, 6, 6, 512)         0
ng2D)

conv2d_26 (Conv2D)          (None, 6, 6, 512)         2359808

batch_normalization_26 (Ba  (None, 6, 6, 512)         2048
tchNormalization)

max_pooling2d_11 (MaxPooli  (None, 3, 3, 512)         0
ng2D)
```

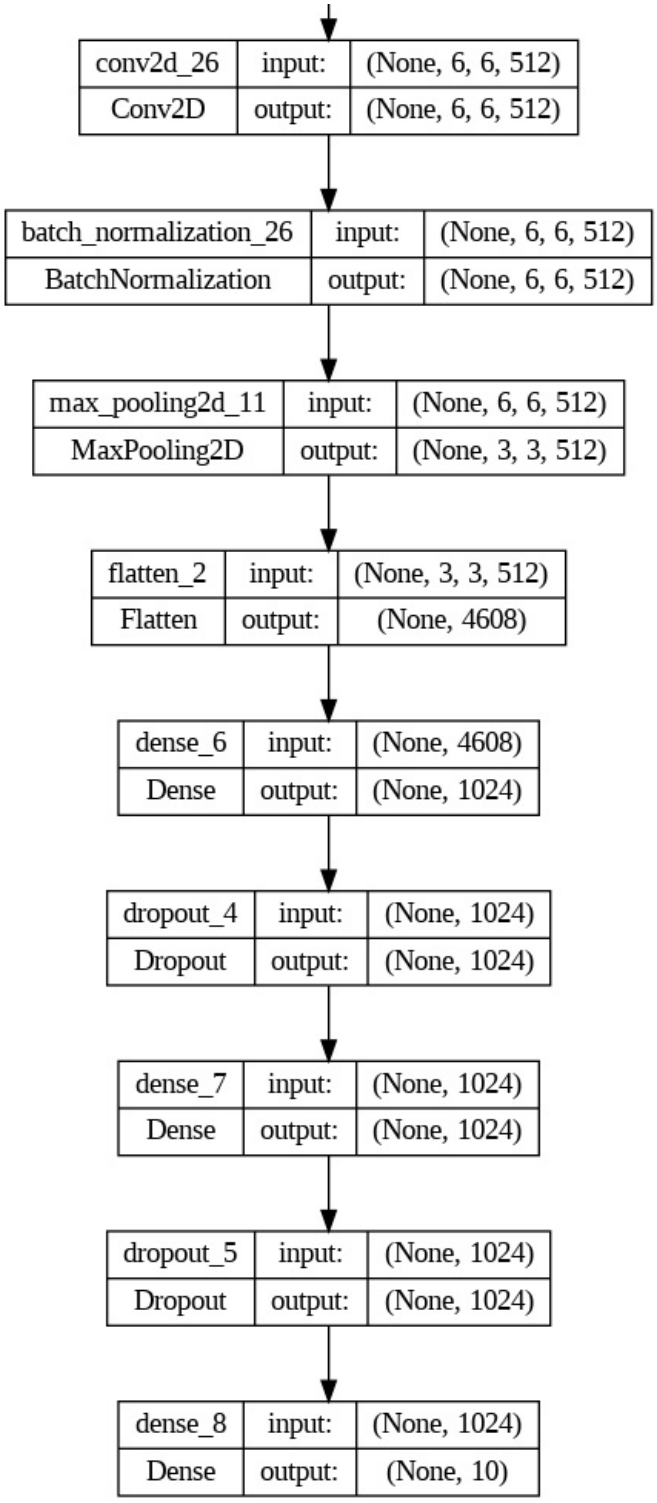
```
flatten_2 (Flatten)         (None, 4608)              0

dense_6 (Dense)             (None, 1024)              4719616

dropout_4 (Dropout)         (None, 1024)              0

dense_7 (Dense)             (None, 1024)              1049600

dropout_5 (Dropout)         (None, 1024)              0

dense_8 (Dense)             (None, 10)                10250

=================================================================

Total params: 15617674 (59.58 MB)

Trainable params: 15611274 (59.55 MB)

Non-trainable params: 6400 (25.00 KB)

_________________________________________________________________
```

In [38]:
```
from keras.utils import plot_model

plot_model(model, to_file='model_plot.png', show_shapes=True, show_layer_names=True)
```

Out[38]:




batch_normalization_20 | input: | (None, 24, 24, 256)
BatchNormalization | output: | (None, 24, 24, 256)
conv2d_21 | input: | (None, 24, 24, 256)
Conv2D | output: | (None, 24, 24, 256)
batch_normalization_21 | input: | (None, 24, 24, 256)
BatchNormalization | output: | (None, 24, 24, 256)
conv2d_22 | input: | (None, 24, 24, 256)
Conv2D | output: | (None, 24, 24, 256)
batch_normalization_22 | input: | (None, 24, 24, 256)
BatchNormalization | output: | (None, 24, 24, 256)
conv2d_23 | input: | (None, 24, 24, 256)
Conv2D | output: | (None, 24, 24, 512)
batch_normalization_23 | input: | (None, 24, 24, 512)
BatchNormalization | output: | (None, 24, 24, 512)
max_pooling2d_9 | input: | (None, 24, 24, 512)
MaxPooling2D | output: | (None, 12, 12, 512)
conv2d_24 | input: | (None, 12, 12, 512)
Conv2D | output: | (None, 12, 12, 512)
batch_normalization_24 | input: | (None, 12, 12, 512)
BatchNormalization | output: | (None, 12, 12, 512)
conv2d_25 | input: | (None, 12, 12, 512)
Conv2D | output: | (None, 12, 12, 512)
batch_normalization_25 | input: | (None, 12, 12, 512)
BatchNormalization | output: | (None, 12, 12, 512)
max_pooling2d_10 | input: | (None, 12, 12, 512)
MaxPooling2D | output: | (None, 6, 6, 512)



Training my mdoel

In [21]:
```
history = model.fit(train, epochs=3, validation_data=val, verbose=1)
```

```
Epoch 1/3

3500/3500 [==============================] - 197s 56ms/step - loss: 0.4896 - accuracy: 0.8483 - val_loss: 0.0043
- val_accuracy: 0.9993

Epoch 2/3

3500/3500 [==============================] - 194s 56ms/step - loss: 0.0190 - accuracy: 0.9944 - val_loss: 0.0011
- val_accuracy: 0.9997

Epoch 3/3

3500/3500 [==============================] - 204s 58ms/step - loss: 0.0077 - accuracy: 0.9981 - val_loss: 0.0012
- val_accuracy: 0.9995
```

Testing my model

In [22]:
```
model.evaluate(test, verbose=1)
```

```
1000/1000 [==============================] - 18s 18ms/step - loss: 0.0013 - accuracy: 0.9995
```

Out[22]:
```
[0.0013108871644362807, 0.9994999766349792]
```

Saving my model

In [23]:
```
model.save("hand_gesture_Model.h5")
```

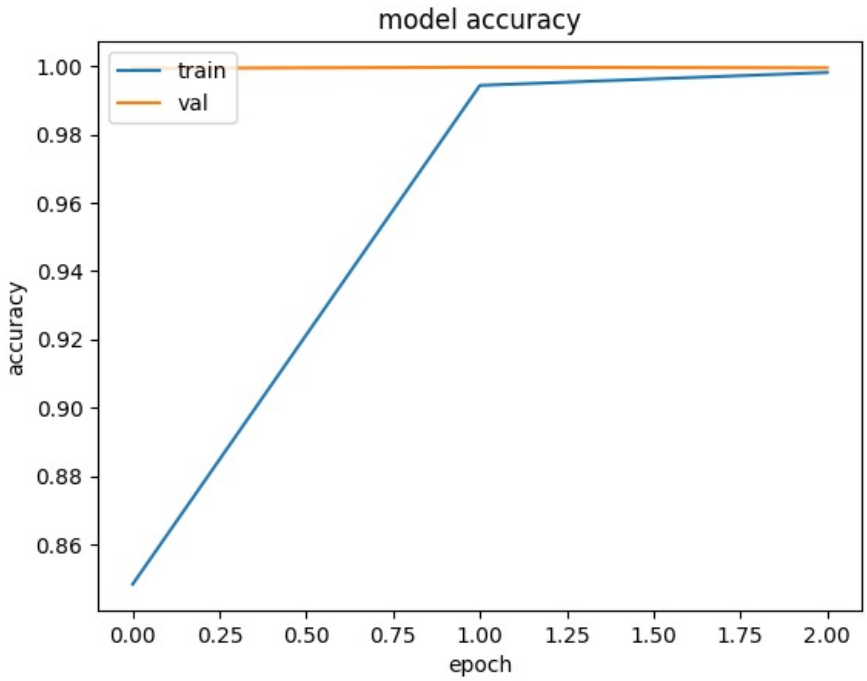
```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.

  saving_api.save_model(
```

Getting results

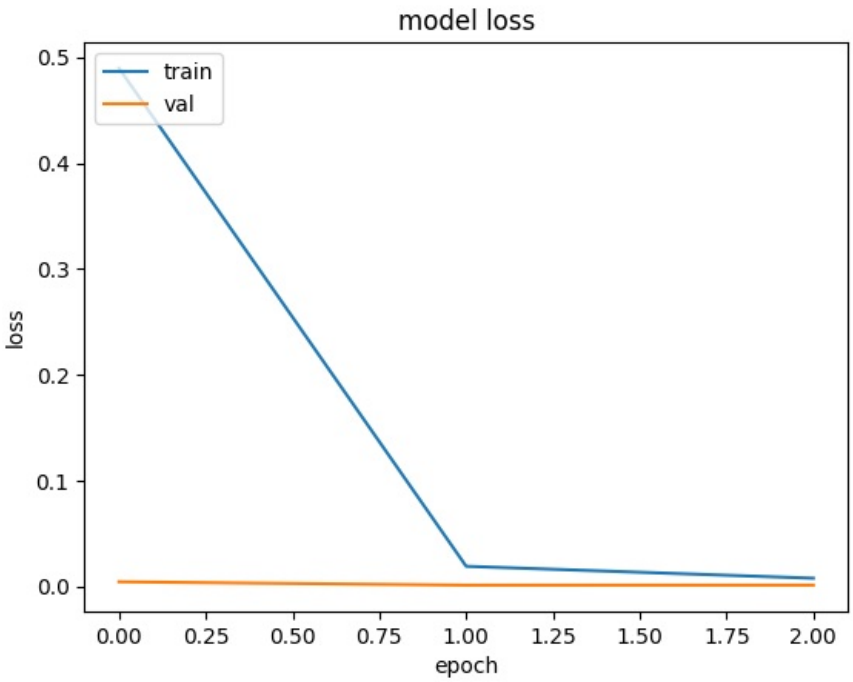
In [24]:
```
pred = model.predict(test)
pred = np.argmax(pred, axis=1)

labels = (train.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred2 = [labels[k] for k in pred]
```

```
1000/1000 [==============================] - 20s 20ms/step
```

In [25]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



In [26]:
```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



In [32]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

y_test = X_test.labels
print(classification_report(y_test, pred2))
print("Accuracy of the Model:","{:.1f}%".format(accuracy_score(y_test, pred2)*100))
```

```
              precision    recall  f1-score   support

      01_palm       1.00      1.00      1.00       404
         02_l       1.00      1.00      1.00       377
      03_fist       1.00      1.00      1.00       404
04_fist_moved       1.00      1.00      1.00       410
     05_thumb       1.00      1.00      1.00       417
     06_index       1.00      1.00      1.00       366
        07_ok       1.00      1.00      1.00       418
08_palm_moved       1.00      1.00      1.00       403
          09_c       1.00      1.00      1.00       392
      10_down       1.00      1.00      1.00       409

     accuracy                           1.00      4000
    macro avg       1.00      1.00      1.00      4000
 weighted avg       1.00      1.00      1.00      4000


Accuracy of the Model: 100.0%
```

In [37]:
```
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix

class_labels = ['Palm', 'l', 'Fist', 'Fist_moved', 'Thumb', 'Index', 'Ok', 'Palm_moved', 'C', 'Down']

cm = confusion_matrix(y_test, pred2)

plt.figure(figsize=(10, 5))
sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues')

plt.xticks(ticks=[0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5], labels=class_labels, rotation=25)
plt.yticks(ticks=[0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5], labels=class_labels, rotation=0)
plt.xlabel("Predicted")
```

```
plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()
```