Download and Prepare the Dataset

I will be using the Rock-Paper-Scissors dataset, a gallery of hands images in Rock, Paper, and Scissors poses.

In [1]:
```
import requests

# URL of the train set and test set
train_url = "https://storage.googleapis.com/tensorflow-1-public/course2/week4/rps.zip"
test_url = "https://storage.googleapis.com/tensorflow-1-public/course2/week4/rps-test-

# Download the train set
response = requests.get(train_url)
with open("rps.zip", "wb") as file:
    file.write(response.content)

# Download the test set
response = requests.get(test_url)
with open("rps-test-set.zip", "wb") as file:
    file.write(response.content)
```

In [2]:
```
import zipfile

# Extract the archive
local_zip = './rps.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('tmp/rps-train')
zip_ref.close()

local_zip = './rps-test-set.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('tmp/rps-test')
zip_ref.close()
```

Assigning the directory names into variables and look at the filenames as a sanity check.

In [3]:
```
import os

base_dir = 'tmp/rps-train/rps'

rock_dir = os.path.join(base_dir, 'rock')
paper_dir = os.path.join(base_dir, 'paper')
scissors_dir = os.path.join(base_dir, 'scissors')

print('total training rock images:', len(os.listdir(rock_dir)))
print('total training paper images:', len(os.listdir(paper_dir)))
print('total training scissors images:', len(os.listdir(scissors_dir)))

rock_files = os.listdir(rock_dir)
print(rock_files[:10])
```

```
paper_files = os.listdir(paper_dir)
print(paper_files[:10])

scissors_files = os.listdir(scissors_dir)
print(scissors_files[:10])
```

```
total training rock images: 840
total training paper images: 840
total training scissors images: 840
['rock01-000.png', 'rock01-001.png', 'rock01-002.png', 'rock01-003.png', 'rock01-004.png', 'rock01-005.png', 'rock01-006.png', 'rock01-007.png', 'rock01-008.png', 'rock01-009.png']
['paper01-000.png', 'paper01-001.png', 'paper01-002.png', 'paper01-003.png', 'paper01-004.png', 'paper01-005.png', 'paper01-006.png', 'paper01-007.png', 'paper01-008.png', 'paper01-009.png']
['scissors01-000.png', 'scissors01-001.png', 'scissors01-002.png', 'scissors01-003.png', 'scissors01-004.png', 'scissors01-005.png', 'scissors01-006.png', 'scissors01-007.png', 'scissors01-008.png', 'scissors01-009.png']
```

Inspect some of the images to see the variety in your model inputs.

In [4]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

pic_index = 2

next_rock = [os.path.join(rock_dir, fname)
                for fname in rock_files[pic_index-2:pic_index]]
next_paper = [os.path.join(paper_dir, fname)
                for fname in paper_files[pic_index-2:pic_index]]
next_scissors = [os.path.join(scissors_dir, fname)
                for fname in scissors_files[pic_index-2:pic_index]]

for i, img_path in enumerate(next_rock+next_paper+next_scissors):
  img = mpimg.imread(img_path)
  plt.imshow(img)
  plt.axis('Off')
  plt.show()
```

Build the model

In [5]:
```
import tensorflow as tf

model = tf.keras.models.Sequential([
    # Note the input shape is the desired size of the image 150x150 with 3 bytes color
    # This is the first convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu', input_shape=(150, 150, 3)),
    tf.keras.layers.MaxPooling2D(2, 2),
```

```python
    # The second convolution
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The third convolution
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # The fourth convolution
    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),
    # Flatten the results to feed into a DNN
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dropout(0.5),
    # 512 neuron hidden layer
    tf.keras.layers.Dense(512, activation='relu'),
    tf.keras.layers.Dense(3, activation='softmax')
])

# Print the model summary
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 148, 148, 64)      1792

 max_pooling2d (MaxPooling2  (None, 74, 74, 64)        0
 D)

 conv2d_1 (Conv2D)           (None, 72, 72, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 36, 36, 64)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 34, 34, 128)       73856

 max_pooling2d_2 (MaxPoolin  (None, 17, 17, 128)       0
 g2D)

 conv2d_3 (Conv2D)           (None, 15, 15, 128)       147584

 max_pooling2d_3 (MaxPoolin  (None, 7, 7, 128)         0
 g2D)

 flatten (Flatten)           (None, 6272)              0

 dropout (Dropout)           (None, 6272)              0

 dense (Dense)               (None, 512)               3211776

 dense_1 (Dense)             (None, 3)                 1539

=================================================================
Total params: 3473475 (13.25 MB)
Trainable params: 3473475 (13.25 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

You will then compile the model. The key change here is the `loss` function. Whereas before you were using `binary_crossentropy` for 2 classes, you will change it to categorical_crossentropy to extend it to more classes.

In [6]:
```
# Set the training parameters
model.compile(loss = 'categorical_crossentropy', optimizer='rmsprop', metrics=['accura
```

Prepare the ImageDataGenerator

In [7]:
```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

TRAINING_DIR = "tmp/rps-train/rps"
training_datagen = ImageDataGenerator(
      rescale = 1./255,
      rotation_range=40,
      width_shift_range=0.2,
      height_shift_range=0.2,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True,
      fill_mode='nearest')

VALIDATION_DIR = "tmp/rps-test/rps-test-set"
validation_datagen = ImageDataGenerator(rescale = 1./255)

train_generator = training_datagen.flow_from_directory(
    TRAINING_DIR,
    target_size=(150,150),
    class_mode='categorical',
    batch_size=126)

validation_generator = validation_datagen.flow_from_directory(
    VALIDATION_DIR,
    target_size=(150,150),
    class_mode='categorical',
    batch_size=126
)
```

```
Found 2520 images belonging to 3 classes.
Found 372 images belonging to 3 classes.
```

Train the model and evaluate the results

You will train for 25 epochs and evaludate the results afterwards. Observe how both the training and validation accuracy are trending upwards. This is a good indication that the model is not overfitting to only your training set.

In [8]:
```
# Train the model
history = model.fit(train_generator, epochs=25, steps_per_epoch=20, validation_data =
```

```
Epoch 1/25
20/20 [==============================] - 219s 10s/step - loss: 1.2251 - accuracy: 0.3
369 - val_loss: 1.0937 - val_accuracy: 0.3333
Epoch 2/25
20/20 [==============================] - 168s 8s/step - loss: 1.0915 - accuracy: 0.37
98 - val_loss: 1.0613 - val_accuracy: 0.5081
Epoch 3/25
20/20 [==============================] - 188s 9s/step - loss: 1.0775 - accuracy: 0.40
52 - val_loss: 1.0910 - val_accuracy: 0.3925
Epoch 4/25
20/20 [==============================] - 185s 9s/step - loss: 0.9861 - accuracy: 0.48
73 - val_loss: 0.6672 - val_accuracy: 0.8898
Epoch 5/25
20/20 [==============================] - 175s 9s/step - loss: 0.9081 - accuracy: 0.55
16 - val_loss: 0.6206 - val_accuracy: 0.6882
Epoch 6/25
20/20 [==============================] - 176s 9s/step - loss: 0.7460 - accuracy: 0.64
76 - val_loss: 0.3692 - val_accuracy: 0.8575
Epoch 7/25
20/20 [==============================] - 169s 8s/step - loss: 0.6719 - accuracy: 0.68
77 - val_loss: 0.4517 - val_accuracy: 0.8548
Epoch 8/25
20/20 [==============================] - 173s 9s/step - loss: 0.5814 - accuracy: 0.74
96 - val_loss: 0.3172 - val_accuracy: 0.8333
Epoch 9/25
20/20 [==============================] - 177s 9s/step - loss: 0.4755 - accuracy: 0.79
13 - val_loss: 0.1490 - val_accuracy: 0.9570
Epoch 10/25
20/20 [==============================] - 174s 9s/step - loss: 0.4509 - accuracy: 0.80
44 - val_loss: 0.1232 - val_accuracy: 0.9382
Epoch 11/25
20/20 [==============================] - 175s 9s/step - loss: 0.3543 - accuracy: 0.85
52 - val_loss: 0.0841 - val_accuracy: 0.9624
Epoch 12/25
20/20 [==============================] - 172s 9s/step - loss: 0.2700 - accuracy: 0.89
25 - val_loss: 0.0820 - val_accuracy: 0.9866
Epoch 13/25
20/20 [==============================] - 172s 8s/step - loss: 0.2559 - accuracy: 0.90
99 - val_loss: 0.0838 - val_accuracy: 1.0000
Epoch 14/25
20/20 [==============================] - 171s 8s/step - loss: 0.2041 - accuracy: 0.92
26 - val_loss: 0.0573 - val_accuracy: 1.0000
Epoch 15/25
20/20 [==============================] - 171s 8s/step - loss: 0.1669 - accuracy: 0.94
33 - val_loss: 0.2124 - val_accuracy: 0.9167
Epoch 16/25
20/20 [==============================] - 175s 9s/step - loss: 0.1787 - accuracy: 0.92
74 - val_loss: 0.1294 - val_accuracy: 0.9543
Epoch 17/25
20/20 [==============================] - 175s 9s/step - loss: 0.1295 - accuracy: 0.95
00 - val_loss: 0.0340 - val_accuracy: 0.9892
Epoch 18/25
20/20 [==============================] - 176s 9s/step - loss: 0.1595 - accuracy: 0.94
21 - val_loss: 0.0617 - val_accuracy: 0.9677
Epoch 19/25
20/20 [==============================] - 176s 9s/step - loss: 0.1459 - accuracy: 0.95
12 - val_loss: 0.4478 - val_accuracy: 0.7500
Epoch 20/25
20/20 [==============================] - 177s 9s/step - loss: 0.1372 - accuracy: 0.95
08 - val_loss: 0.1145 - val_accuracy: 0.9651
```

```
Epoch 21/25
20/20 [==============================] - 176s 9s/step - loss: 0.0901 - accuracy: 0.96
87 - val_loss: 0.0483 - val_accuracy: 0.9785
Epoch 22/25
20/20 [==============================] - 176s 9s/step - loss: 0.0884 - accuracy: 0.96
79 - val_loss: 0.1204 - val_accuracy: 0.9435
Epoch 23/25
20/20 [==============================] - 176s 9s/step - loss: 0.1225 - accuracy: 0.95
67 - val_loss: 0.0160 - val_accuracy: 0.9946
Epoch 24/25
20/20 [==============================] - 176s 9s/step - loss: 0.0964 - accuracy: 0.97
14 - val_loss: 0.3902 - val_accuracy: 0.8280
Epoch 25/25
20/20 [==============================] - 175s 9s/step - loss: 0.0802 - accuracy: 0.97
46 - val_loss: 0.0439 - val_accuracy: 0.9704
```

In [9]:

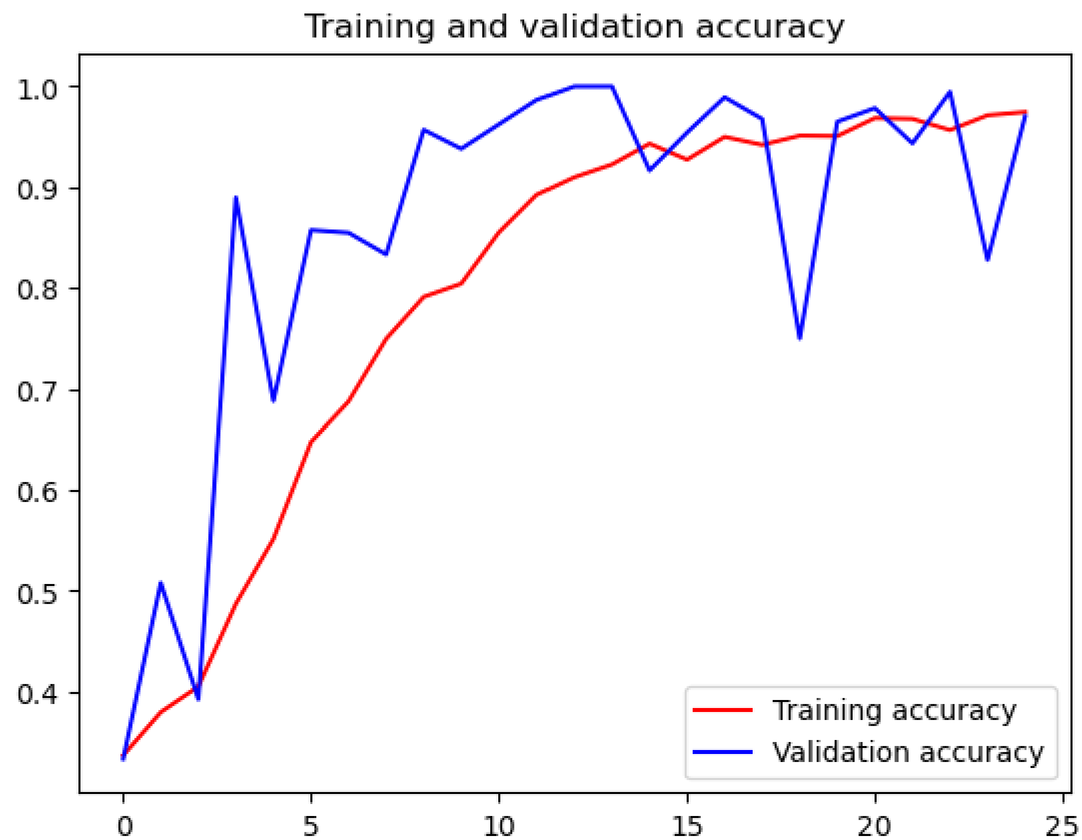
```python
import matplotlib.pyplot as plt

# Plot the results
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'r', label='Training accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend(loc=0)
plt.figure()

plt.show()
```




```
<Figure size 640x480 with 0 Axes>
```

Model Prediction

In [23]:
```
import os
import numpy as np
from tensorflow.keras.utils import load_img, img_to_array
from tensorflow.keras.models import load_model  # Load your pre-trained model

# Create a dictionary that maps class indices to class names
class_names = {0: 'Rock', 1: 'Paper', 2: 'Scissor'}  # Add your class names here

# Specify the folder where your images are located
image_folder = r"C:\Users\Admin\Downloads\hi"

# List all files in the specified folder
image_files = os.listdir(image_folder)

# Load your pre-trained model
model = model

# Iterate through the image files
for image_file in image_files:
    if image_file.endswith((".jpg", ".jpeg", ".png")):
        # Load the image
        image_path = os.path.join(image_folder, image_file)
        img = load_img(image_path, target_size=(150, 150))
        x = img_to_array(img)
        x = np.expand_dims(x, axis=0)

        # Make predictions using your loaded model
```

```python
        predictions = model.predict(x, batch_size=10)

        # Find the predicted class index
        predicted_class_index = np.argmax(predictions)

        # Get the class name based on the predicted class index
        predicted_class_name = class_names.get(predicted_class_index, "Unknown")

        # Print the image file name, class name, and the corresponding prediction resu
        print(f"Image: {image_file}")
        print(f"Predicted class: {predicted_class_name}")
        print("Prediction probabilities:", predictions)
```

```
1/1 [==============================] - 1s 837ms/step
Image: depositphotos_112684400-stock-illustration-hand-with-clenched-fist-icon.jpg
Predicted class: Rock
Prediction probabilities: [[1. 0. 0.]]
1/1 [==============================] - 0s 132ms/step
Image: OIP (1).jpg
Predicted class: Scissor
Prediction probabilities: [[0. 0. 1.]]
1/1 [==============================] - 0s 52ms/step
Image: OIP.jpg
Predicted class: Paper
Prediction probabilities: [[0. 1. 0.]]
```

In [ ]: