Final_Pranav_Vyas_TensorFlow_Final_Project

[ ]:
```
# Pranav Vyas - TensorFlow Final Project

# Import TensorFlow and tensorflow_datasets to load the data
import tensorflow as tf
import tensorflow_datasets as tfds

# Import matplotlib to visualize the images
import matplotlib.pyplot as plt

# Import numpy for its methods involving arrays
import numpy as np

# Import random to generate random values
import random

# Load TensorBoard in order to visualize the model's statistics
%load_ext tensorboard
```

[ ]:
```
# Unzip the folder that contains the information for the saved model
!unzip Final_Rock_Paper_Scissors_CNN.zip
```

```
Archive:  Final_Rock_Paper_Scissors_CNN.zip
   creating: Rock_Paper_Scissors_CNN/
  inflating: Rock_Paper_Scissors_CNN/.DS_Store
  inflating: __MACOSX/Rock_Paper_Scissors_CNN/._.DS_Store
  inflating: Rock_Paper_Scissors_CNN/keras_metadata.pb
  inflating: __MACOSX/Rock_Paper_Scissors_CNN/._keras_metadata.pb
   creating: Rock_Paper_Scissors_CNN/variables/
  inflating: Rock_Paper_Scissors_CNN/saved_model.pb
  inflating: __MACOSX/Rock_Paper_Scissors_CNN/._saved_model.pb
   creating: Rock_Paper_Scissors_CNN/assets/
  inflating: Rock_Paper_Scissors_CNN/variables/variables.data-00000-of-00001
  inflating:
__MACOSX/Rock_Paper_Scissors_CNN/variables/._variables.data-00000-of-00001
  inflating: Rock_Paper_Scissors_CNN/variables/variables.index
  inflating: __MACOSX/Rock_Paper_Scissors_CNN/variables/._variables.index
```

[ ]:
```
# Load the dataset using the tensorflow dataset module
(train_dataset_color, test_dataset_color) = tfds.load("rock_paper_scissors",␣
 ↪split=["train", "test"], shuffle_files=True, as_supervised=True)
```

```
Downloading and preparing dataset 219.53 MiB (download: 219.53 MiB,
generated: Unknown size, total: 219.53 MiB) to
~/tensorflow_datasets/rock_paper_scissors/3.0.0…
Dl Completed…: 0 url [00:00, ? url/s]
Dl Size…: 0 MiB [00:00, ? MiB/s]
Generating splits…:   0%|          | 0/2 [00:00<?, ? splits/s]
Generating train examples…:   0%|          | 0/2520 [00:00<?, ? examples/s]
Shuffling ~/tensorflow_datasets/rock_paper_scissors/3.0.0.incompleteWF1MD0/
 ↪rock_paper_scissors-train.tfrecord*…
Generating test examples…:   0%|          | 0/372 [00:00<?, ? examples/s]
Shuffling ~/tensorflow_datasets/rock_paper_scissors/3.0.0.incompleteWF1MD0/
 ↪rock_paper_scissors-test.tfrecord*.…
Dataset rock_paper_scissors downloaded and prepared to
~/tensorflow_datasets/rock_paper_scissors/3.0.0. Subsequent calls will reuse
this data.
```

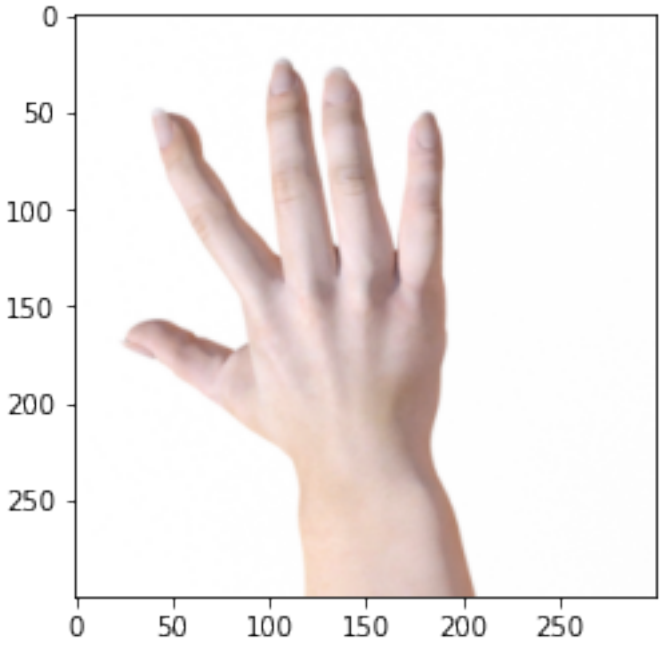
[ ]:
```
# Create methods to normalize the data

# Convert the images to grayscale to simplify the data to 1 color channel,␣
 ↪making the data easier to process
def normalize_images(images, label):
  grayscale_images = tf.image.rgb_to_grayscale(images)
  return (grayscale_images, label)

# Normalize the pixel values from 0 to 1
def normalize_pixel_values(images, label):
  return (images / 255, label)
```

[ ]:
```
# Use the normalize_images function to change all of the rgb images into␣
 ↪grayscale images
# Use the normalize_pixel_values to keep the value of each pixel in the 0 to 1␣
 ↪range
train_dataset = train_dataset_color.map(normalize_images)
train_dataset = train_dataset.map(normalize_pixel_values)

test_dataset = test_dataset_color.map(normalize_images)
test_dataset = test_dataset.map(normalize_pixel_values)
```

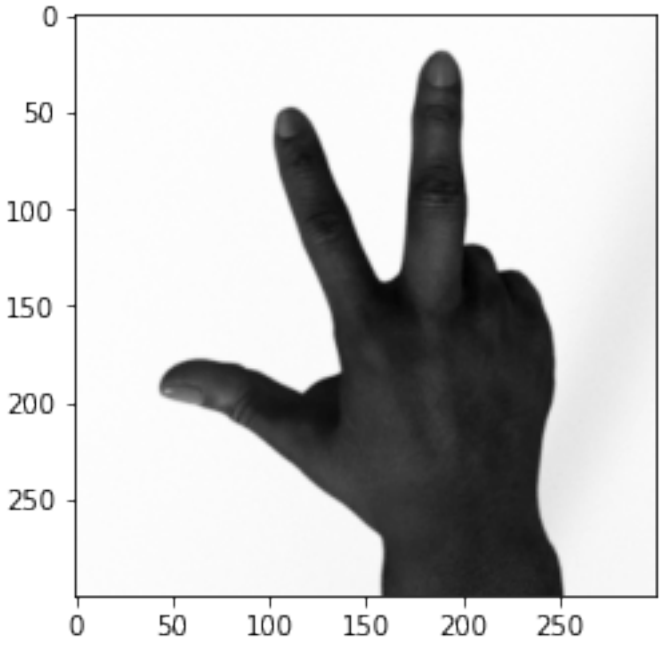
[ ]:
```
# Seperating the training images and labels from the training dataset
train_images = [image for image, _ in train_dataset]
train_labels = [label for _, label in train_dataset]
train_images_color = [image for image, _ in train_dataset_color]

# Converting the images and labels to tensors in order to be used in the model
train_images = tf.convert_to_tensor(train_images)
train_labels = tf.convert_to_tensor(train_labels)
train_images_color = tf.convert_to_tensor(train_images_color)

# Seperating the testing images and labels from the testing dataset
test_images = [image for image, _ in test_dataset]
test_labels = [label for _, label in test_dataset]
test_images_color = [image for image, _ in test_dataset_color]

test_images = tf.convert_to_tensor(test_images)
test_labels = tf.convert_to_tensor(test_labels)
test_images_color = tf.convert_to_tensor(test_images_color)
```

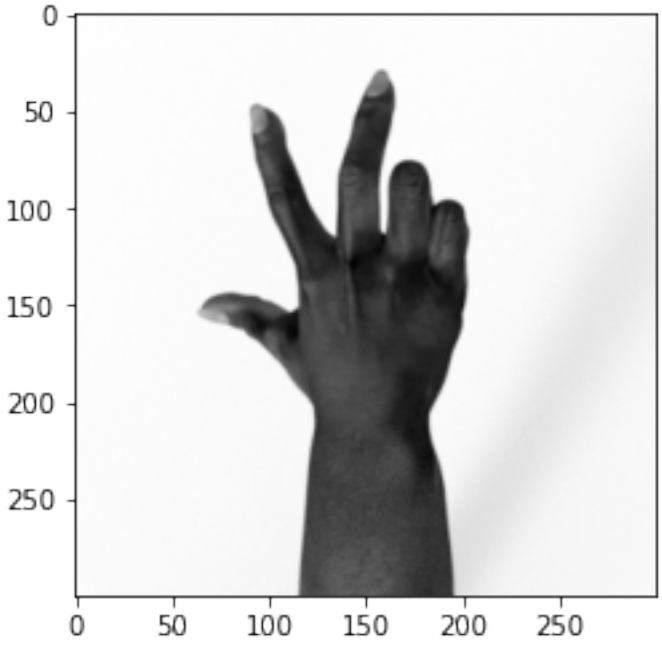
[ ]:
```
# Load the images from the train_dataset
index = int(input("Please provide an index to access one of the training images:
 ↪ "))
train_image = train_images_color[index]

# Use the matplotlib module to display the image
plt.imshow(train_image)
plt.show()
```

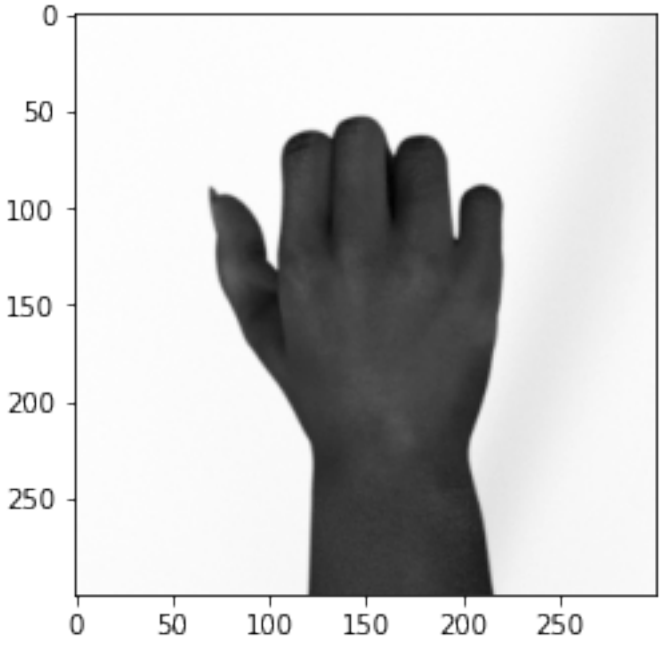
```
Please provide an index to access one of the training images: 10
```

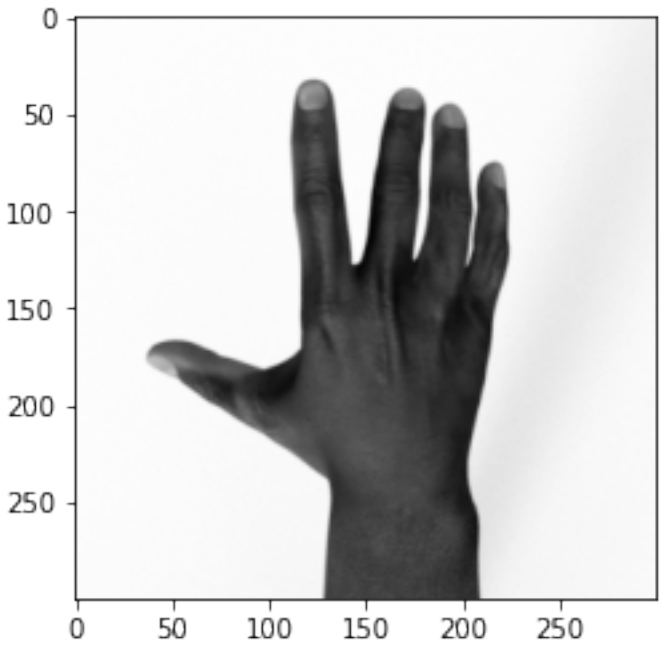


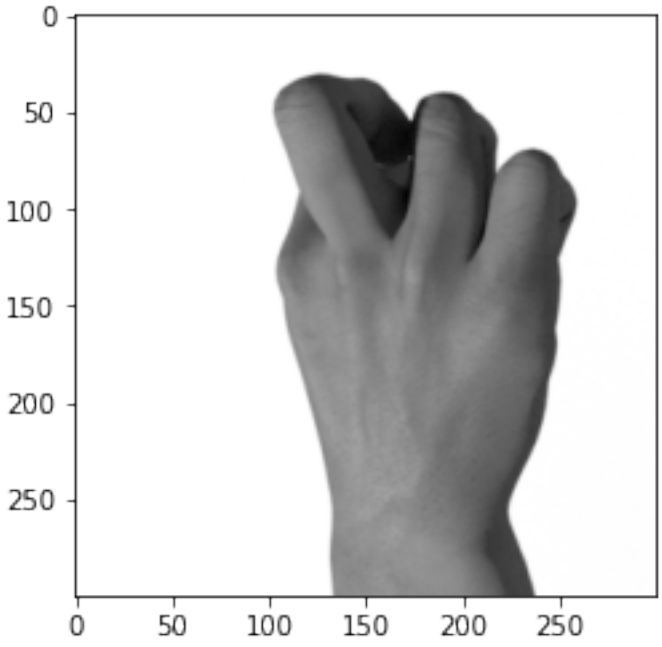
[ ]:
```
# Display the grayscale images
for image, _ in train_dataset.take(5):
  image = tf.squeeze(image)
  plt.imshow(image, cmap='gray')
  plt.show()
```




0
50
100
150
200
250
0
50
100
150
200
250


0
50
100
150
200
250
0
50
100
150
200
250





[ ]:
```
# Create a neural network that is capable of accurately classifying each image␣
↪as rock, paper, or scissors
model = tf.keras.models.Sequential([
  # Creating a convolutional layer with 32 filters
```

```
  tf.keras.layers.Conv2D(filters=32, kernel_size=(4, 4), strides=(1, 1),␣
 ↪input_shape=(300, 300, 1)),
  tf.keras.layers.Activation('relu'),
  tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
  tf.keras.layers.Dropout(rate=0.6),

  # Creating a convolutional layer with 64 filters
  tf.keras.layers.Conv2D(filters=64, kernel_size=(4, 4), strides=(1, 1)),
  tf.keras.layers.Activation('relu'),
  tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
  tf.keras.layers.Dropout(rate=0.6),

  # Creating a convolutional layer with 64 filters
  tf.keras.layers.Conv2D(filters=64, kernel_size=(4, 4), strides=(1, 1)),
  tf.keras.layers.Activation('relu'),
  tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
  tf.keras.layers.Dropout(rate=0.6),

  # Creating a convolutional layer with 128 filters
  tf.keras.layers.Conv2D(filters=128, kernel_size=(4, 4), strides=(1, 1)),
  tf.keras.layers.Activation('relu'),
  tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
  tf.keras.layers.Dropout(rate=0.6),

  # Flattening the filters and creating a densely connected layer in order to␣
 ↪classify the images
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(units=128),
  tf.keras.layers.Activation('relu'),
  tf.keras.layers.Dense(units=3),
])

# Compiling the model with the adam optimizer, the␣
 ↪SparseCategoricalCrossentropy loss function from keras and log the accuracy␣
 ↪of the model
model.compile(optimizer=tf.keras.optimizers.Adam(),
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# This function will print a summary of the model
model.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
```

```
conv2d_16 (Conv2D)           (None, 297, 297, 32)      544

activation_20 (Activation)   (None, 297, 297, 32)      0

max_pooling2d_16 (MaxPoolin  (None, 148, 148, 32)      0
g2D)

dropout_16 (Dropout)         (None, 148, 148, 32)      0

conv2d_17 (Conv2D)           (None, 145, 145, 64)      32832

activation_21 (Activation)   (None, 145, 145, 64)      0

max_pooling2d_17 (MaxPoolin  (None, 72, 72, 64)       0
g2D)

dropout_17 (Dropout)         (None, 72, 72, 64)        0

conv2d_18 (Conv2D)           (None, 69, 69, 64)        65600

activation_22 (Activation)   (None, 69, 69, 64)        0

max_pooling2d_18 (MaxPoolin  (None, 34, 34, 64)       0
g2D)

dropout_18 (Dropout)         (None, 34, 34, 64)        0

conv2d_19 (Conv2D)           (None, 31, 31, 128)       131200

activation_23 (Activation)   (None, 31, 31, 128)       0

max_pooling2d_19 (MaxPoolin  (None, 15, 15, 128)      0
g2D)

dropout_19 (Dropout)         (None, 15, 15, 128)       0

flatten_4 (Flatten)          (None, 28800)             0

dense_8 (Dense)              (None, 128)               3686528

activation_24 (Activation)   (None, 128)               0

dense_9 (Dense)              (None, 3)                 387

=================================================================
Total params: 3,917,091
Trainable params: 3,917,091
Non-trainable params: 0
```

[ ]:
```
print("Training:")
print()

# Create a tensorboard callback to track the statistics of the model
tensorboard = tf.keras.callbacks.TensorBoard(log_dir='./logs')

# Create a checkpoint callback to save the model
checkpoint = tf.keras.callbacks.ModelCheckpoint('checkpoints/
 ↪rock_paper_scissors.cpkt', monitor='loss', verbose=1, save_best_only=True,␣
 ↪mode='min')

# Using the model.fit() function to run the model over multiple epochs
model.fit(train_images, train_labels, batch_size=16, epochs=30,␣
 ↪callbacks=[tensorboard, checkpoint])

print()
print("_" * 100)
print()
print("Evaluation:")
# Evaluate the performance of the model using the test dataset
model.evaluate(test_images, test_labels, batch_size=16, callbacks=[tensorboard])
model.save("Rock_Paper_Scissors_CNN_new")
```

```
Training:

Epoch 1/30
157/158 [============================>.] - ETA: 0s - loss: 1.1605 - accuracy:
0.3973
Epoch 1: loss improved from inf to 1.16023, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 12s 71ms/step - loss: 1.1602 -
accuracy: 0.3972
Epoch 2/30
157/158 [============================>.] - ETA: 0s - loss: 0.8046 - accuracy:
0.6612
Epoch 2: loss improved from 1.16023 to 0.80332, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 11s 69ms/step - loss: 0.8033 -
accuracy: 0.6619
Epoch 3/30
157/158 [============================>.] - ETA: 0s - loss: 0.2756 - accuracy:
0.9064
Epoch 3: loss improved from 0.80332 to 0.27471, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 11s 69ms/step - loss: 0.2747 -
accuracy: 0.9067
```

```
Epoch 4/30
157/158 [============================>.] - ETA: 0s - loss: 0.0453 - accuracy:
0.9861
Epoch 4: loss improved from 0.27471 to 0.04520, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 11s 71ms/step - loss: 0.0452 -
accuracy: 0.9861
Epoch 5/30
157/158 [============================>.] - ETA: 0s - loss: 0.0253 - accuracy:
0.9916
Epoch 5: loss improved from 0.04520 to 0.02520, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 11s 69ms/step - loss: 0.0252 -
accuracy: 0.9917
Epoch 6/30
157/158 [============================>.] - ETA: 0s - loss: 0.0159 - accuracy:
0.9948
Epoch 6: loss improved from 0.02520 to 0.01602, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 11s 69ms/step - loss: 0.0160 -
accuracy: 0.9948
Epoch 7/30
157/158 [============================>.] - ETA: 0s - loss: 0.0225 - accuracy:
0.9920
Epoch 7: loss did not improve from 0.01602
158/158 [==============================] - 10s 60ms/step - loss: 0.0225 -
accuracy: 0.9921
Epoch 8/30
157/158 [============================>.] - ETA: 0s - loss: 0.0104 - accuracy:
0.9976
Epoch 8: loss improved from 0.01602 to 0.01033, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 11s 71ms/step - loss: 0.0103 -
accuracy: 0.9976
Epoch 9/30
157/158 [============================>.] - ETA: 0s - loss: 0.0239 - accuracy:
0.9936
Epoch 9: loss did not improve from 0.01033
158/158 [==============================] - 10s 60ms/step - loss: 0.0238 -
accuracy: 0.9937
Epoch 10/30
157/158 [============================>.] - ETA: 0s - loss: 0.0050 - accuracy:
0.9988
Epoch 10: loss improved from 0.01033 to 0.00494, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 11s 70ms/step - loss: 0.0049 -
accuracy: 0.9988
Epoch 11/30
```

```
157/158 [============================>.] - ETA: 0s - loss: 0.0012 - accuracy:
0.9996
Epoch 11: loss improved from 0.00494 to 0.00118, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 11s 71ms/step - loss: 0.0012 -
accuracy: 0.9996
Epoch 12/30
157/158 [============================>.] - ETA: 0s - loss: 0.0206 - accuracy:
0.9924
Epoch 12: loss did not improve from 0.00118
158/158 [==============================] - 10s 61ms/step - loss: 0.0205 -
accuracy: 0.9925
Epoch 13/30
157/158 [============================>.] - ETA: 0s - loss: 0.0215 - accuracy:
0.9944
Epoch 13: loss did not improve from 0.00118
158/158 [==============================] - 10s 60ms/step - loss: 0.0214 -
accuracy: 0.9944
Epoch 14/30
157/158 [============================>.] - ETA: 0s - loss: 7.1985e-04 -
accuracy: 1.0000
Epoch 14: loss improved from 0.00118 to 0.00072, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 11s 70ms/step - loss: 7.2013e-04 -
accuracy: 1.0000
Epoch 15/30
157/158 [============================>.] - ETA: 0s - loss: 0.0012 - accuracy:
1.0000
Epoch 15: loss did not improve from 0.00072
158/158 [==============================] - 10s 60ms/step - loss: 0.0012 -
accuracy: 1.0000
Epoch 16/30
157/158 [============================>.] - ETA: 0s - loss: 0.0112 - accuracy:
0.9968
Epoch 16: loss did not improve from 0.00072
158/158 [==============================] - 10s 61ms/step - loss: 0.0112 -
accuracy: 0.9968
Epoch 17/30
157/158 [============================>.] - ETA: 0s - loss: 0.0058 - accuracy:
0.9988
Epoch 17: loss did not improve from 0.00072
158/158 [==============================] - 10s 61ms/step - loss: 0.0058 -
accuracy: 0.9988
Epoch 18/30
157/158 [============================>.] - ETA: 0s - loss: 0.0018 - accuracy:
0.9996
Epoch 18: loss did not improve from 0.00072
158/158 [==============================] - 10s 60ms/step - loss: 0.0018 -
```

```
accuracy: 0.9996
Epoch 19/30
157/158 [============================>.] - ETA: 0s - loss: 0.0045 - accuracy:
0.9988
Epoch 19: loss did not improve from 0.00072
158/158 [==============================] - 10s 61ms/step - loss: 0.0045 -
accuracy: 0.9988
Epoch 20/30
157/158 [============================>.] - ETA: 0s - loss: 0.0083 - accuracy:
0.9976
Epoch 20: loss did not improve from 0.00072
158/158 [==============================] - 10s 60ms/step - loss: 0.0083 -
accuracy: 0.9976
Epoch 21/30
157/158 [============================>.] - ETA: 0s - loss: 5.6472e-04 -
accuracy: 1.0000
Epoch 21: loss improved from 0.00072 to 0.00056, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 11s 70ms/step - loss: 5.6333e-04 -
accuracy: 1.0000
Epoch 22/30
157/158 [============================>.] - ETA: 0s - loss: 0.0236 - accuracy:
0.9948
Epoch 22: loss did not improve from 0.00056
158/158 [==============================] - 10s 61ms/step - loss: 0.0235 -
accuracy: 0.9948
Epoch 23/30
157/158 [============================>.] - ETA: 0s - loss: 0.0040 - accuracy:
0.9992
Epoch 23: loss did not improve from 0.00056
158/158 [==============================] - 10s 61ms/step - loss: 0.0040 -
accuracy: 0.9992
Epoch 24/30
157/158 [============================>.] - ETA: 0s - loss: 0.0176 - accuracy:
0.9980
Epoch 24: loss did not improve from 0.00056
158/158 [==============================] - 10s 61ms/step - loss: 0.0175 -
accuracy: 0.9980
Epoch 25/30
157/158 [============================>.] - ETA: 0s - loss: 0.0049 - accuracy:
0.9980
Epoch 25: loss did not improve from 0.00056
158/158 [==============================] - 10s 60ms/step - loss: 0.0049 -
accuracy: 0.9980
Epoch 26/30
157/158 [============================>.] - ETA: 0s - loss: 0.0267 - accuracy:
0.9924
Epoch 26: loss did not improve from 0.00056
```

```
158/158 [==============================] - 10s 61ms/step - loss: 0.0266 -
accuracy: 0.9925
Epoch 27/30
157/158 [============================>.] - ETA: 0s - loss: 0.0030 - accuracy:
0.9988
Epoch 27: loss did not improve from 0.00056
158/158 [==============================] - 10s 61ms/step - loss: 0.0030 -
accuracy: 0.9988
Epoch 28/30
157/158 [============================>.] - ETA: 0s - loss: 3.6465e-05 -
accuracy: 1.0000
Epoch 28: loss improved from 0.00056 to 0.00004, saving model to
checkpoints/rock_paper_scissors.cpkt
158/158 [==============================] - 11s 72ms/step - loss: 3.6351e-05 -
accuracy: 1.0000
Epoch 29/30
157/158 [============================>.] - ETA: 0s - loss: 6.5116e-04 -
accuracy: 0.9996
Epoch 29: loss did not improve from 0.00004
158/158 [==============================] - 10s 61ms/step - loss: 6.4909e-04 -
accuracy: 0.9996
Epoch 30/30
157/158 [============================>.] - ETA: 0s - loss: 4.2024e-04 -
accuracy: 1.0000
Epoch 30: loss did not improve from 0.00004
158/158 [==============================] - 10s 61ms/step - loss: 4.1891e-04 -
accuracy: 1.0000

--------------------------------------------------------------------------------
--------------------

Evaluation:
24/24 [==============================] - 1s 29ms/step - loss: 1.1075 - accuracy:
0.8360
```

[ ]:
```
%tensorboard --logdir ./logs
```

```
<IPython.core.display.Javascript object>
```

[ ]:
```
model_checkpoint = tf.keras.models.load_model("checkpoints/rock_paper_scissors.
  ↪cpkt")
model_checkpoint.evaluate(test_images, test_labels, batch_size=16)
```

```
24/24 [==============================] - 1s 23ms/step - loss: 1.4468 - accuracy:
0.8199
```

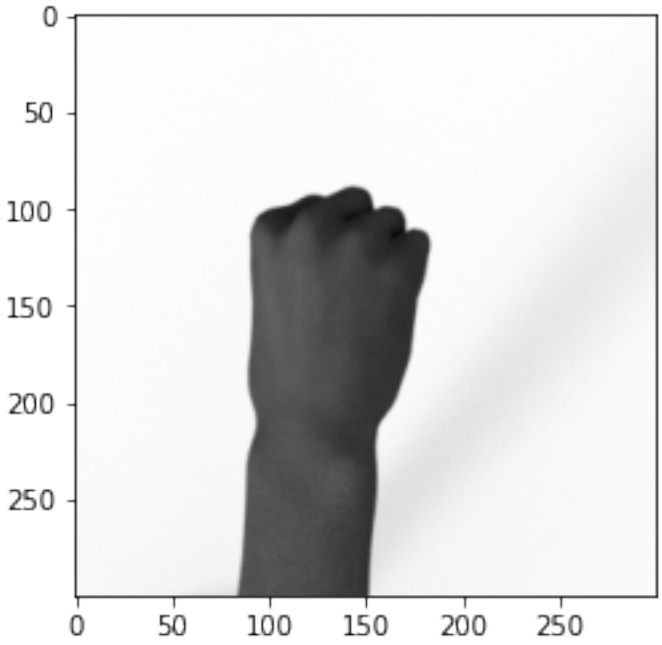
[ ]:
```
[1.4468382596969604, 0.8198924660682678]
```

[ ]:
```
model_1 = tf.keras.models.load_model("Rock_Paper_Scissors_CNN")
model_1.evaluate(test_images, test_labels, batch_size=16)
```

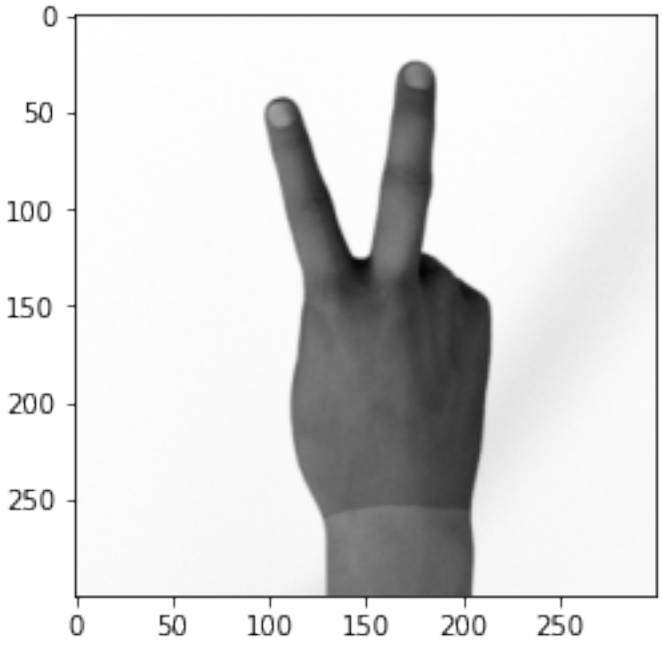
```
24/24 [==============================] - 10s 27ms/step - loss: 0.1967 -
accuracy: 0.9113
```

[ ]:
```
[0.19674678146839142, 0.9112903475761414]
```

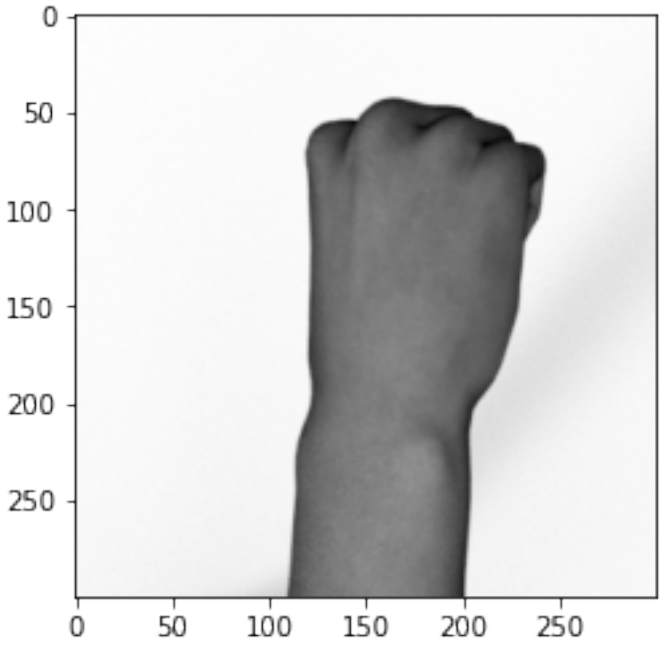
[ ]:
```
# This is a demonstration of the model's classification capabilites, using the␣
 ↪model.predict() function
# The model is classifying images from the testing dataset, images that have␣
 ↪not been seen during training
index = 50
for num in range(5):
  image = test_images[index]
  image = tf.squeeze(image)
  plt.imshow(image, cmap='gray')
  plt.show()

  label_pred = model_1.predict(tf.expand_dims(test_images[index], axis=0))
  real_label = test_labels[index]
  print("Predicted Label: {}".format(np.argmax(label_pred)))
  print("Actual Label: {}".format(real_label.numpy()))
  print()
  print()

  index = index + 1
```



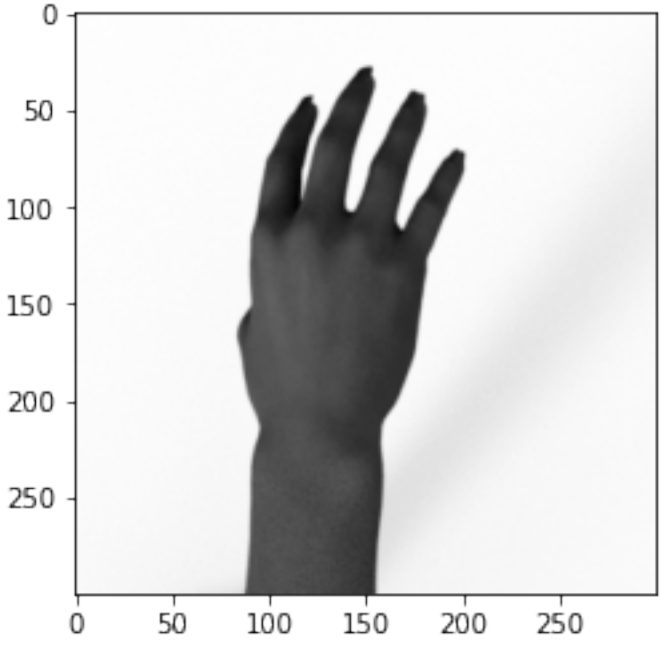
```
Predicted Label: 0
Actual Label: 0
```



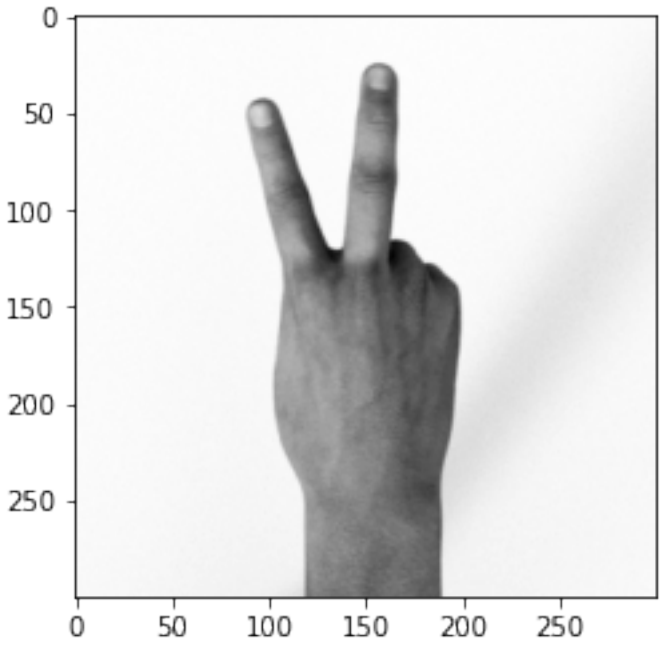

```
Predicted Label: 2
Actual Label: 2
```




```
Predicted Label: 0
Actual Label: 0
```



```
Predicted Label: 1
Actual Label: 1
```




```
Predicted Label: 2
Actual Label: 2
```

[ ]:
```
mode = ""
# Ask the user which mode they would like to select
while(mode != "Gesture Mode" and mode != "Index Mode"):
  mode = input("Would you like to select a gesture and have the model generate␣
 ↪an image representing your choice from the testing dataset?\nor\nWould you␣
 ↪like to select an index from the testing dataset for the model to predict␣
 ↪the gesture from the image?\n\nPlease enter Gesture Mode or Index Mode\n")

print()

if (mode == "Gesture Mode"):
  # Take the user choice and convert it to the label value in the dataset
  user_choice = ""
  while(user_choice != "rock" and user_choice != "paper" and user_choice !=␣
 ↪"scissors"):
    user_choice = input("Please select rock, paper, or scissors: ")
```

```
  print()

if (user_choice == 'rock'):
  input_val = 0
if (user_choice == 'paper'):
  input_val = 1
if (user_choice == 'scissors'):
  input_val = 2

# Generate a random number as an index for the dataset that matches the␣
↪selected gesture using the model's prediction capabilities
def generate_image_index(input_val):
  index = random.randint(0, 360)
  match=False
  while(match==False):
    prediction_arr = model_1.predict(tf.expand_dims(test_images[index],␣
↪axis=0))
    prediction = np.argmax(prediction_arr)
    if (prediction == input_val):
      match=True
    else:
      index = index + 1
  return index

answer = ""

# Generate the image to user, and ask them if their choice matches the image
while (answer!="yes"):
  answer = ""
  index = generate_image_index(input_val)
  image = test_images_color[index]
  plt.imshow(image)
  plt.show()
  while(answer != "no" and answer != "yes"):
    print("\nThe image above was selected by the machine learning model to␣
↪represent your choice ({}).".format(user_choice))
    answer = input("Is the model's prediction correct? (yes/no)\n")

print("_" * 100)
print()

# Create a computer opponent for the user to play against

answer = ""
while (answer!="yes"):
  answer = ""
  # Display and random image and the model's prediction to the user
```

```
  print("Your opponent randomly selected this image:\n")
  random_index = random.randint(0, 360)
  opp_image = test_images_color[random_index]
  plt.imshow(opp_image)
  plt.show()

  # Identify the model's prediction of the random image
  opp_prediction_arr = model_1.predict(tf.
↪expand_dims(test_images[random_index], axis=0))
  opp_prediction = np.argmax(opp_prediction_arr)

  if (opp_prediction == 0):
    opp_choice = 'rock'
  if (opp_prediction == 1):
    opp_choice = 'paper'
  if (opp_prediction == 2):
    opp_choice = "scissors"

  # Ask the user if the prediction matches the gesture displayed in the image
  while (answer != "no" and answer != "yes"):
    print("\nThe model classifies this image as a gesture representing the␣
↪choice of {}".format(opp_choice))
    answer = input("Is the model's prediction correct? (yes/no)\n")
    print()

# Determine the winner of the game
if (user_choice == opp_choice):
  print("Since both players chose the same option, it's a draw!")

if (user_choice == "rock" and opp_choice == "paper"):
  print("Since you chose {}, and your opponent chose {}, your opponent wins.".
↪format(user_choice, opp_choice))
if (user_choice == "rock" and opp_choice == "scissors"):
  print("Since you chose {}, and your opponent chose {}, you win!".
↪format(user_choice, opp_choice))

if (user_choice == "paper" and opp_choice == "scissors"):
  print("Since you chose {}, and your opponent chose {}, your opponent wins.".
↪format(user_choice, opp_choice))
if (user_choice == "paper" and opp_choice == "rock"):
  print("Since you chose {}, and your opponent chose {}, you win!".
↪format(user_choice, opp_choice))

if (user_choice == "scissors" and opp_choice == "rock"):
  print("Since you chose {}, and your opponent chose {}, your opponent wins.".
↪format(user_choice, opp_choice))
```

```
  if (user_choice == "scissors" and opp_choice == "paper"):
    print("Since you chose {}, and your opponent chose {}, you win!".
 ↪format(user_choice, opp_choice))

# In "Index Mode" users select an index and the model predicts the gesture␣
 ↪displayed in the image at that index
elif(mode == "Index Mode"):
  # Take the user choice's and convert it to the label values in the dataset
  index_choice = -1

  answer = ""
  while (answer!="yes"):
    answer = ""
    index_choice = -1
    # Ask the user for an index value
    while(index_choice < 0 or index_choice > 371):
      index_choice = -1
      index_choice = int(input("Please select an index from the testing dataset:
 ↪ "))
      print()

    # Show the image at that index to the user
    image = test_images_color[index_choice]
    plt.imshow(image)
    plt.show()

    # Identify the model's prediction for the image
    user_index_prediction_arr = model_1.predict(tf.
 ↪expand_dims(test_images[index_choice], axis=0))
    user_index_prediction = np.argmax(user_index_prediction_arr)

    if (user_index_prediction == 0):
      user_choice = 'rock'
    if (user_index_prediction == 1):
      user_choice = 'paper'
    if (user_index_prediction == 2):
      user_choice = "scissors"

    # Ask the user if the model's prediction matches the gesture displayed in␣
 ↪the image
    while (answer != "no" and answer != "yes"):
      print("\nThe model classifies this image as a gesture representing the␣
 ↪choice of {}\n".format(user_choice))
      answer = input("Is the model's prediction correct? (yes/no)\n")

  print("_" * 100)
```

```
print()

answer = ""
while (answer!="yes"):
  # Randomly select an image from the testing dataset
  answer = ""
  print("Your opponent randomly selected this image:\n")
  random_index = random.randint(0, 360)
  opp_image = test_images_color[random_index]
  plt.imshow(opp_image)
  plt.show()

  # Identify the model's prediction for the random image
  opp_prediction_arr = model_1.predict(tf.
↪expand_dims(test_images[random_index], axis=0))
  opp_prediction = np.argmax(opp_prediction_arr)

  if (opp_prediction == 0):
    opp_choice = 'rock'
  if (opp_prediction == 1):
    opp_choice = 'paper'
  if (opp_prediction == 2):
    opp_choice = "scissors"

  # Ask the user if the model's prediction matches the gesture displayed in␣
↪the image
  while (answer != "no" and answer != "yes"):
    print("\nThe model classifies this image as a gesture representing the␣
↪choice of {}".format(opp_choice))
    answer = input("Is the model's prediction correct? (yes/no)\n")
    print()

# Determine the winner of the game

if (user_choice == opp_choice):
  print("Since both players chose the same option, it's a draw!")

# Conditions for if the user chooses rock
if (user_choice == "rock" and opp_choice == "paper"):
  print("Since you chose {}, and your opponent chose {}, your opponent wins.".
↪format(user_choice, opp_choice))
if (user_choice == "rock" and opp_choice == "scissors"):
  print("Since you chose {}, and your opponent chose {}, you win!".
↪format(user_choice, opp_choice))

# Conditions for if the user chooses paper
if (user_choice == "paper" and opp_choice == "scissors"):
```

```
    print("Since you chose {}, and your opponent chose {}, your opponent wins.".
↪format(user_choice, opp_choice))
  if (user_choice == "paper" and opp_choice == "rock"):
    print("Since you chose {}, and your opponent chose {}, you win!".
↪format(user_choice, opp_choice))

  # Conditions for if the user chooses scissors
  if (user_choice == "scissors" and opp_choice == "rock"):
    print("Since you chose {}, and your opponent chose {}, your opponent wins.".
↪format(user_choice, opp_choice))
  if (user_choice == "scissors" and opp_choice == "paper"):
    print("Since you chose {}, and your opponent chose {}, you win!".
↪format(user_choice, opp_choice))
```