Kelompok 5

> Raisyad Jullfikar 2106238
>
> Nadhief Athallah Isya 2106413
>
> Azzahra Fahriza 2102296
>
> M. Azka Atqiya 2100812
>
> Afina Rachmani 1901377

-- Mengimport Library yang di Perlukan

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import PIL
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow import keras
from keras import layers
from keras.models import Sequential
import pathlib
from google.colab import drive
import os
import matplotlib.pyplot as plt
import zipfile, os
import shutil
```

Mengekstrak folder tujuan dari Google Drive serta Membaca Folder Konten (folder tujuan) yang ada di Google Drive

In [2]:
```
drive.mount('/content/drive')
```

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

In [3]:
```
# ekstrak

local_zip = '/content/drive/MyDrive/DATAMINING/datasets.zip'
zip_ref = zipfile.ZipFile(local_zip, 'r')
zip_ref.extractall('/content')
zip_ref.close()
```

Membaca data path folder dan file tujuan kedalam var base_dir (base directory)

In [4]:
```
base_dir = '/content/datasets'
```

Membaca path directory dari var sebelumnya menggunakan syntax pathlib dan dimasukkan kedalam var data_dir (data directory)

In [5]:
```
data_dir = pathlib.Path(base_dir)
```

Menghitung banyaknya gambar berekstensi jpg pada folder datasets

In [6]:
```
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
```

```
3000
```

Mengklasifikasikan berbagai kelas yang terdapat pada folder datasets

In [7]:
```
list_dir = [os.path.basename(x) for x in data_dir.iterdir() if x.is_dir()]
print("Jumlah class: {} ".format(len(list_dir)))
print("Jumlah instance per class")
for x in list_dir:
 print("{} = {} ".format(x,len(list(data_dir.glob('{}/*.jpg'.format(x))))))
```

```
Jumlah class: 6
Jumlah instance per class
Ruby = 500
Fake_Emerald = 500
Emerald = 500
Fake_Ruby = 500
Fake_Turquoise = 500
Turquoise = 500
```

Menampilkan data dalam bentuk image pada indeks pertama[0]

In [8]:
```
visual_img = list(data_dir.glob('Ruby/*'))
PIL.Image.open(str(visual_img[0]))
```

Out[8]:

Menentukan Ukuran tinggi dan lebar gambar serta ukuran batch

In [9]:
```
batch_size = 34
img_height = 200
img_width = 200
# 32 - 180 34 200 36 220
```

Menyiapkan data training dengan mengambil 0.7 atau 70% dari data asli

In [10]:
```
train_ds = tf.keras.utils.image_dataset_from_directory(
 data_dir,
 validation_split=0.7,
 subset="training",
 seed=123,
 image_size=(img_height, img_width),
 batch_size=batch_size)
```

```
Found 3000 files belonging to 6 classes.
Using 900 files for training.
```

Menyiapkan data validasi dengan mengambil 0.1 atau 10% dari data validasi

In [11]:
```
val_ds = tf.keras.utils.image_dataset_from_directory(
 data_dir,
 validation_split=0.2,
 subset="validation",
 seed=123,
 image_size=(img_height, img_width),
 batch_size=batch_size)
```

```
Found 3000 files belonging to 6 classes.
Using 600 files for validation.
```

Kami mengambil 70% data untuk train dan 10% data untuk validasi, dikarenakan kebutuhan data untuk train harus lebih banyak dibanding data untuk validasi

---

Memasukkan nama - nama kelas kedalam var serta Menampilkan nama nama kelas

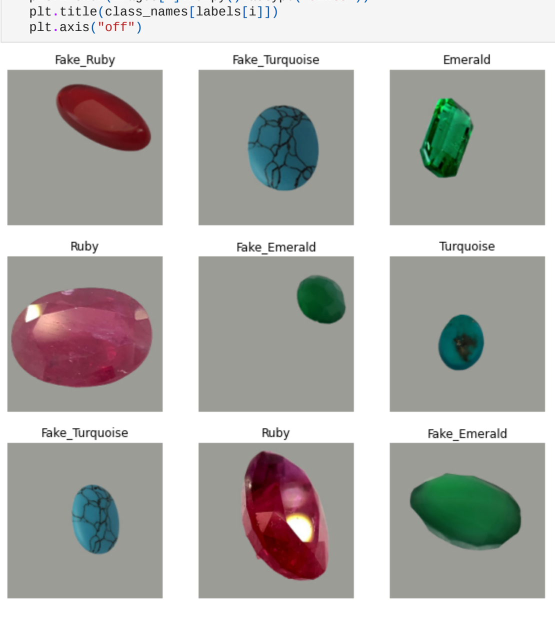
In [12]:
```
class_names = train_ds.class_names
print(class_names)
```

```
['Emerald', 'Fake_Emerald', 'Fake_Ruby', 'Fake_Turquoise', 'Ruby', 'Turquoise']
```

Menampilkan preview dataset training

In [13]:
```
# lihat dataset training

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
 for i in range(9):
   ax = plt.subplot(3, 3, i + 1) # 3 baris, 3 kolom
   plt.imshow(images[i].numpy().astype("uint8"))
   plt.title(class_names[labels[i]])
   plt.axis("off")
```



Memperlihatkan shape untuk data train

In [14]:
```
for image_batch, labels_batch in train_ds:
 print(image_batch.shape)
 print(labels_batch.shape)
 break
```

```
(34, 200, 200, 3)
(34,)
```

Set cache buffer untuk meningkatkan efisiensi training

In [15]:
```
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
```

Normalisasi nilai rgb dari 0-255 menjadi 0-1

In [16]:
```
# normalisasi nilai RGB
normalization_layer = layers.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# nilai dari [0 sd 255] menjadi [0 sd 1]
print(np.min(first_image), np.max(first_image))
```

```
0.0 1.0
```

Membuat arsitektur deep learning

In [17]:
```
num_classes = len(class_names)

model = Sequential([
 layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
 layers.Conv2D(16, 3, padding='same', activation='relu'),
 layers.MaxPooling2D(),
 layers.Conv2D(32, 3, padding='same', activation='relu'),
 layers.MaxPooling2D(),
 layers.Conv2D(64, 3, padding='same', activation='relu'),
 layers.MaxPooling2D(),
 layers.Flatten(),
 layers.Dense(128, activation='relu'),
 layers.Dense(num_classes)
])
```

Compile model

In [18]:
```
model.compile(optimizer='adam',
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
             metrics=['accuracy'])
```

Memperlihatkan arsitektur neural network

In [19]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 200, 200, 3)       0

 conv2d (Conv2D)             (None, 200, 200, 16)      448

 max_pooling2d (MaxPooling2D  (None, 100, 100, 16)     0
 )

 conv2d_1 (Conv2D)           (None, 100, 100, 32)      4640

 max_pooling2d_1 (MaxPooling  (None, 50, 50, 32)       0
 2D)

 conv2d_2 (Conv2D)           (None, 50, 50, 64)        18496

 max_pooling2d_2 (MaxPooling  (None, 25, 25, 64)       0
 2D)

 flatten (Flatten)           (None, 40000)             0

 dense (Dense)               (None, 128)               5120128

 dense_1 (Dense)             (None, 6)                 774

=================================================================
Total params: 5,144,486
Trainable params: 5,144,486
Non-trainable params: 0
_________________________________________________________________
```

Memproses data train

In [20]:
```
# Memproses data train yang sudah kita model sebelumnya
epochs= 5
history = model.fit(
 train_ds,
 validation_data=val_ds,
 epochs=epochs
)
```

```
Epoch 1/5
27/27 [==============================] - 8s 93ms/step - loss: 1.2054 - accuracy: 0.5167 - val_loss: 0.5726 - val_accuracy: 0.7833
Epoch 2/5
27/27 [==============================] - 1s 37ms/step - loss: 0.4667 - accuracy: 0.8489 - val_loss: 0.5554 - val_accuracy: 0.7900
Epoch 3/5
27/27 [==============================] - 1s 35ms/step - loss: 0.3392 - accuracy: 0.8722 - val_loss: 0.3413 - val_accuracy: 0.8833
Epoch 4/5
27/27 [==============================] - 1s 35ms/step - loss: 0.2284 - accuracy: 0.9289 - val_loss: 0.2858 - val_accuracy: 0.8933
Epoch 5/5
27/27 [==============================] - 1s 36ms/step - loss: 0.2625 - accuracy: 0.8944 - val_loss: 0.3384 - val_accuracy: 0.8783
```

Kami hanya memproses epoch hingga 5 saja pada data train berikut, dikarenakan ketika memproses hingga epoch 10, maka data tersebut akan semakin overfitting yang mana gap antara train dan validasi semakin besar

---

Membuat plot dari hasil proses train sebelumnya, untuk memperjelas

In [21]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs_range = range(epochs)
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
# Plot dibawah ini terlihat masih buruk untuk bagian validation
```

Analisis yang kami dapatkan dari plot diatas adalah, plot tersebut masih termasuk kedalam plot yang overfitting dikarenakan plot atau garis train dan validasi yang masih belum stabil serta gap antara train dengan validasi yang lumayan agak jauh, maka dari itu, kami mencoba cara untuk menghilangkan data overfitting diatas dengan cara yang sesuai dengan modul

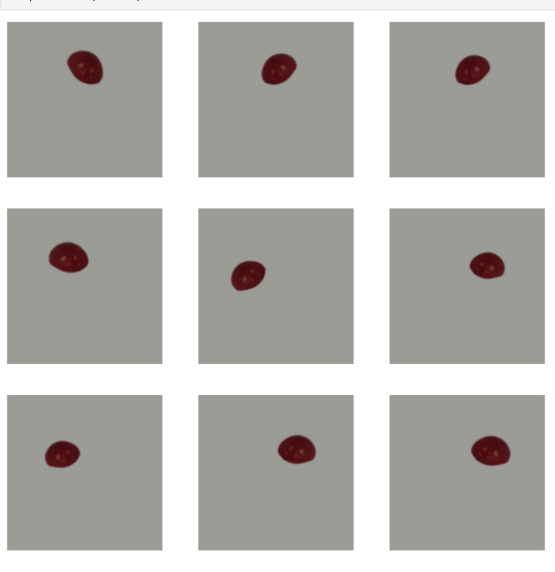
---

Mengatasi data overfitting yaitu dengan mengaugmentasi data training

In [22]:
```
data_augmentation = keras.Sequential(
 [
   layers.RandomFlip("horizontal",
                     input_shape=(img_height,
                                 img_width,
                                 3)),
   layers.RandomRotation(0.1),
   layers.RandomZoom(0.1),
 ]
)
```

Test data sebelumnya berupa visual gambar

In [23]:
```
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
 for i in range(9):
   augmented_images = data_augmentation(images)
   ax = plt.subplot(3, 3, i + 1)
   plt.imshow(augmented_images[0].numpy().astype("uint8"))
   plt.axis("off")
```



Menambahkan dropout, salah satu teknik untuk mengurangi overfitting dalam sebuah data

In [24]:
```
model = Sequential([
 data_augmentation,
 layers.Rescaling(1./255),
 layers.Conv2D(16, 3, padding='same', activation='relu'),
 layers.MaxPooling2D(),
 layers.Conv2D(32, 3, padding='same', activation='relu'),
 layers.MaxPooling2D(),
 layers.Conv2D(64, 3, padding='same', activation='relu'),
 layers.MaxPooling2D(),
 layers.Dropout(0.2),
 layers.Flatten(),
 layers.Dense(128, activation='relu'),
 layers.Dense(num_classes, name="outputs")
])
```

Compile kembali model Arsitektur CNN yang sudah dibuat

In [25]:
```
model.compile(optimizer='adam',
             loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
             metrics=['accuracy'])
```

Lakukan kembali proses train

In [26]:
```
epochs = 10
history = model.fit(
 train_ds,
 validation_data=val_ds,
 epochs=epochs
)
```

```
Epoch 1/10
27/27 [==============================] - 3s 61ms/step - loss: 1.5016 - accuracy: 0.4189 - val_loss: 0.9873 - val_accuracy: 0.6033
Epoch 2/10
27/27 [==============================] - 1s 54ms/step - loss: 0.8112 - accuracy: 0.6822 - val_loss: 0.5733 - val_accuracy: 0.7983
Epoch 3/10
27/27 [==============================] - 1s 53ms/step - loss: 0.5277 - accuracy: 0.8122 - val_loss: 0.4226 - val_accuracy: 0.8483
Epoch 4/10
27/27 [==============================] - 1s 51ms/step - loss: 0.4244 - accuracy: 0.8500 - val_loss: 0.3602 - val_accuracy: 0.8817
Epoch 5/10
27/27 [==============================] - 1s 53ms/step - loss: 0.3392 - accuracy: 0.8711 - val_loss: 0.3449 - val_accuracy: 0.8467
Epoch 6/10
27/27 [==============================] - 1s 53ms/step - loss: 0.3296 - accuracy: 0.8789 - val_loss: 0.2787 - val_accuracy: 0.8917
Epoch 7/10
27/27 [==============================] - 1s 52ms/step - loss: 0.2674 - accuracy: 0.8978 - val_loss: 0.2502 - val_accuracy: 0.9100
Epoch 8/10
27/27 [==============================] - 1s 54ms/step - loss: 0.2713 - accuracy: 0.9033 - val_loss: 0.2660 - val_accuracy: 0.8800
Epoch 9/10
27/27 [==============================] - 1s 54ms/step - loss: 0.2263 - accuracy: 0.9222 - val_loss: 0.1904 - val_accuracy: 0.9317
Epoch 10/10
27/27 [==============================] - 1s 54ms/step - loss: 0.1936 - accuracy: 0.9367 - val_loss: 0.2115 - val_accuracy: 0.9300
```

Model yang telah dibuat kembali untuk mengurangi overfitting diproses kembali dengan 10 epoch, epoch kali ini agak berbeda dengan epoch sebelumnya dikarenakan agar plot dan val_accuracy semakin jelas angka dan grafik yang ditunjukkan, epoch ini menurut saya lebih bagus ketika dipasangkan 10 dibanding lebih atau kurang, dikarenakan ketika epoch tersebut lebih banyak, maka grafik yang ditampilkan dan val yang dihasilkan juga akan semakin tidak bagus atau yang disebut overfitting tadi, dan ketika epoch tersebut hanya 10, val dan grafik terlihat lumayan bagus

---

Tampilkan kembali plot dari hasil proses train sebelumnya untuk mengetahui perbedaan dari model awal dan model sekarang

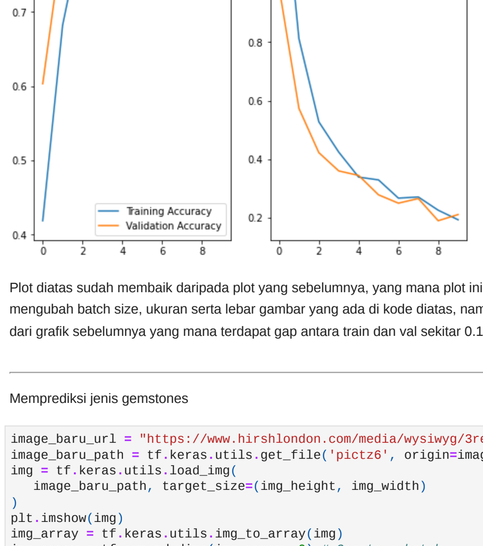
In [27]:
```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
```



Plot diatas sudah membaik daripada plot yang sebelumnya, yang mana plot ini sudah melewati proses pengatasan data overfitting, namun jika dilihat, bahwa grafik yang sebelah kanan yaitu accuracy, masih terlihat naik turun 'agak' drastis dibanding lossnya, Kami sudah mencoba mengubah batch size, ukuran serta lebar gambar yang ada di kode diatas, namun ketika kami ubah kode tersebut yang asalnya batch_size : 34, img_height : 200, img_width : 200, menjadi 36, 220, 220 atau menjadi 38, 240, 240, maka grafik tersebut akan semakin buruk, lebih buruk dari grafik sebelumnya yang mana terdapat gap antara train dan val sekitar 0.1 - 0.3

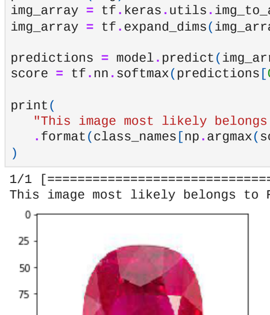
Memprediksi jenis gemstones

In [28]:
```
image_baru_url = "https://www.hirshlondon.com/media/wysiwyg/3reasons/emerald-largercopy.jpg"
image_baru_path = tf.keras.utils.get_file('pictz6', origin=image_baru_url)
img = tf.keras.utils.load_img(
   image_baru_path, target_size=(img_height, img_width)
)
plt.imshow(img)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
   "This image most likely belongs to {} with a {:.2f} percent confidence."
   .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
1/1 [==============================] - 0s 136ms/step
This image most likely belongs to Emerald with a 100.00 percent confidence.
```



In [29]:
```
image_baru_url = "data:image/jpeg;base64,/9j/4AAQSkZJRgABAQAAAQABAAD/2wCEAAoGBxIRExYUFBMXFBQWFhcYFxkWFhYWFhgZGRYZGBgWGRgZHyoiGRwnHRYWIzQkJysuMTExGCE2OzYwOiowMS4BCwsLDw4PHRERHTAoIicwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwMDAwM
image_baru_path = tf.keras.utils.get_file('pictr1', origin=image_baru_url)
img = tf.keras.utils.load_img(
   image_baru_path, target_size=(img_height, img_width)
)
plt.imshow(img)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
   "This image most likely belongs to {} with a {:.2f} percent confidence."
   .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

```
1/1 [==============================] - 0s 16ms/step
This image most likely belongs to Ruby with a 100.00 percent confidence.
```

Gambar - gambar diatas merupakan sample gambar beberapa kelas yang belum ada sebelumnya di datasets, dan berhasil dipredict oleh model dengan persentase confidence yang sangat tinggi