Name: Niraj Lamichhane

Group Leader: Niraj Lamichhane

Group Members:Rojan Shrestha,Saugat Karki, Niraj Lammichane, Aayush Niraula

Group: L6CG5

University_id: 2059514

Module Leader: Mr.Siman Giri

Tutor: Mr.Akash Adhikari

Flower Image Classification with Convolutional Neural Network.

```
from google.colab import drive
drive.mount('/content/drive')    # Connecting the drive
```

```
Mounted at /content/drive
```

```
data_path="/content/drive/MyDrive/Flower Classification/Train"   # path set
```

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow.keras.optimizers import Adam
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
# Define hyperparameters
batch_size = 40
img_height = 256
img_width = 256
dropout_rate = 0.2
```


```
# Define hyperparameters
batch_size = 40
img_height = 256
img_width = 256
dropout_rate = 0.2
```


Tasks and Marks Division-CNN

Data Understanding, Analysis, Visualization and Cleaning[5]:

How many total images are in the dataset?

```
# Initialize a variable to store the total number of images
total_images = 0

# Iterate over the directory and its subdirectories
for root, dirs, files in os.walk(data_path):
    # Count the number of files in each directory
    num_files = len(files)
    # Add the number of files to the total_images variable
    total_images += num_files

# Print the total number of images
print("Total number of images:", total_images)
```

How many images per class?

```
# Count images per class
classes = os.listdir(data_path)   # list of classes in datset directory
images_per_class = {}
for class_name in classes:  # loop through each class and count
    class_path = os.path.join(data_path, class_name) # path of class directory
    if os.path.isdir(class_path):
        images_per_class[class_name] = len(os.listdir(class_path))
print("Total number of images per classes:", images_per_class) # total number of images per class
```

```
Total number of images per classes: {'sunflower': 732, 'daisy': 763, 'tulip': 983, 'rose': 783, 'dandelion': 1051}
```

How do you split between validation and train set?

```
# Creating an image dataset from a directory of images
# Define the path to the directory containing the images
train_ds = tf.keras.utils.image_dataset_from_directory(
    # Your code Here.
    data_path,batch_size=32, # batch size
    image_size=(256, 256),  # image size
    shuffle=True,           # shuffle the dataset randomly
    seed=100,               # set random seed
    validation_split=0.1,   #split the datasets into training and valisate subsets
    subset="training",      # use the training subset for this dataset

  )
```

```
Found 4312 files belonging to 5 classes.
Using 3881 files for training.
```

```
val_ds = tf.keras.utils.image_dataset_from_directory(
    # Your code Here.
    data_path,batch_size=32,   # batch size and path to the directory
    image_size=(256, 256),     # target image size of the datset
    shuffle=True,              # shuffle the dataset randomly
    seed=100,                  #Set the random seed during suffle
    validation_split=0.3,      # split the datsets inot training and validate subset
    subset="validation",       # use the vallidation subset for the dataset

  )
```

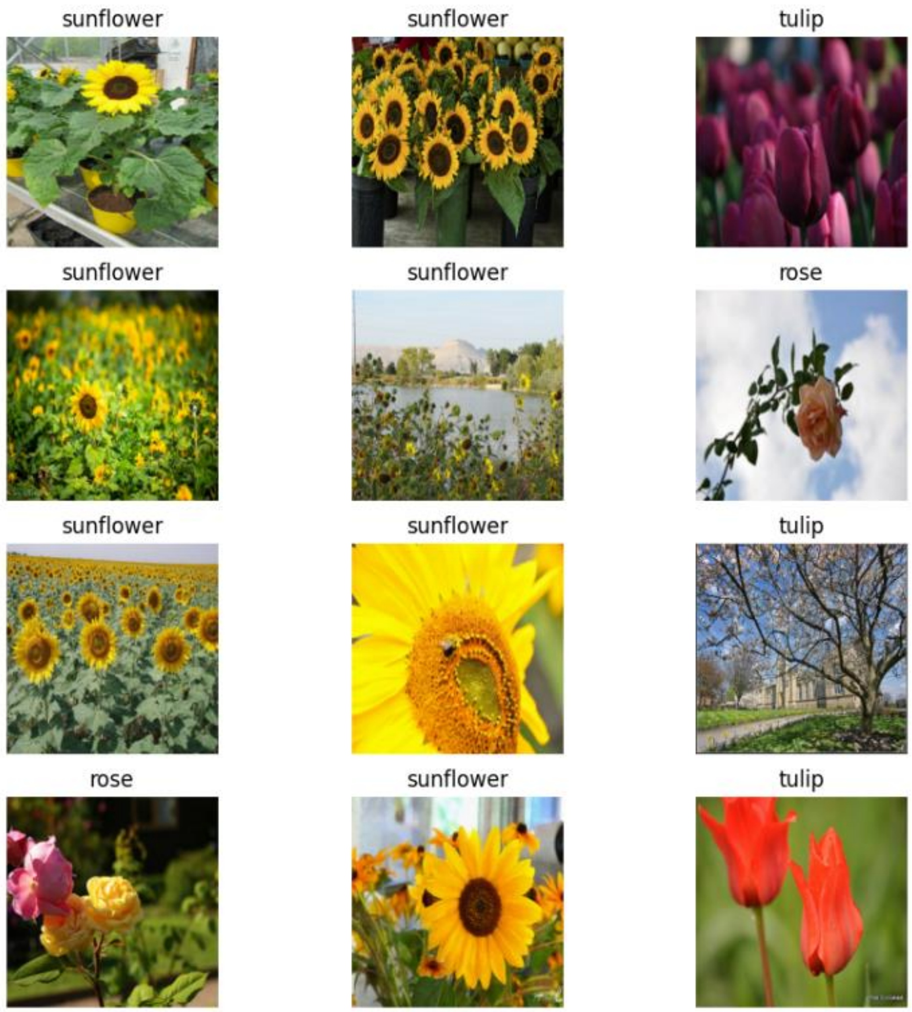
```
Found 4312 files belonging to 5 classes.
Using 1293 files for validation.
```

Visualization

```
# Printing out number of Classes
class_names = train_ds.class_names  # get the class name from the training dataset
print(class_names) # print the class name
```

```
['daisy', 'dandelion', 'rose', 'sunflower', 'tulip']
```

```
plt.figure(figsize=(10, 10))  # Set the size of the figure to 10x10 inches
for images, labels in train_ds.take(1): # Take the first batch of images and labels from the training dataset
  for i in range(12):  # plot 12 images from the batch
    ax = plt.subplot(4, 3, i + 1)  #set the subplot position to be 4*3
    plt.imshow(images[i].numpy().astype("uint8")) # # Show the image in the subplot
    plt.title(class_names[labels[i]]) # # Set the title of the subplot to be the corresponding class name
    plt.axis("off")  # turn off the axis label
```


sunflower
sunflower
tulip
sunflower
sunflower
rose
sunflower
sunflower
tulip
rose
sunflower
tulip

Build Model

Based on the size of your input image, design and build your CNN model. You can have as many layers you think is required for your task.

```
def generate_model(image_height, image_width, nchannels, num_classes):
  """
  This function will generate a model with set of hyperparameters defined above.
  Input Args:
  image_height[int] = Height of an image.
  image_width[int] = Width of an image.
  nchannels[int] = Number of channels in image.
  num_classes[int] = Number of classes in dataset.
  Output Args:
  model-> A CNN model.
  """
  model = tf.keras.Sequential([
      # Rescaling and input layer, [For keras the input shape must be(image height, image width, channels)]
      layers.Rescaling(1./255, input_shape=(image_height,image_width, nchannels)),
      # First Block of Convolution and Pooling Operations.
      layers.Conv2D(filters=16, kernel_size=3, padding="same", activation="relu"),
      layers.MaxPooling2D(),
      # Second Block of Convolution and Pooling Operations.
      layers.Conv2D(filters=32, kernel_size=3, padding="same", activation="relu"),
      layers.MaxPooling2D(),

      # Fully connected classifier.
      layers.Flatten(),
      layers.Dense(128, activation="relu"),
      layers.Dropout(0.5),
      layers.BatchNormalization(),
      layers.Dense(num_classes, activation='softmax')
  ])
  return model
```

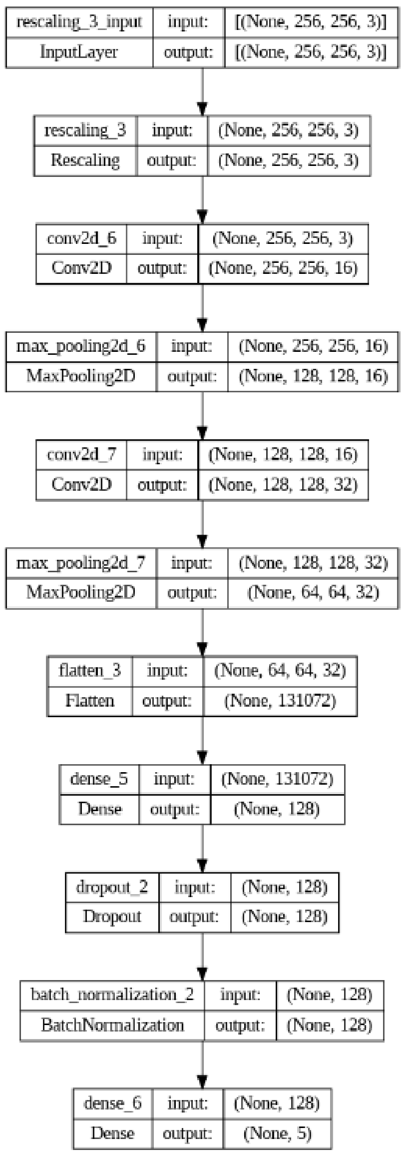
```
num_classes = len(class_names) # numbe of classes
model = generate_model(img_height, img_width, 3, num_classes) # generate a model using the geenrate model function
```



```
model.summary() # plot the model's structure with the shapes of each layer shown
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_3 (Rescaling)     (None, 256, 256, 3)       0

 conv2d_6 (Conv2D)           (None, 256, 256, 16)      448

 max_pooling2d_6 (MaxPooling  (None, 128, 128, 16)     0
 2D)

 conv2d_7 (Conv2D)           (None, 128, 128, 32)      4640

 max_pooling2d_7 (MaxPooling  (None, 64, 64, 32)       0
 2D)

 flatten_3 (Flatten)         (None, 131072)            0

 dense_5 (Dense)             (None, 128)               16777344

 dropout_2 (Dropout)         (None, 128)               0

 batch_normalization_2 (Batc  (None, 128)              512
 hNormalization)

 dense_6 (Dense)             (None, 5)                 645

=================================================================
Total params: 16,783,589
Trainable params: 16,783,333
Non-trainable params: 256
_________________________________________________________________
```

```
keras.utils.plot_model(model, show_shapes=True) # plot the model's structure with shapes of each layer shown
```


rescaling_3_input input: [(None, 256, 256, 3)]
InputLayer output: [(None, 256, 256, 3)]
rescaling_3 input: (None, 256, 256, 3)
Rescaling output: (None, 256, 256, 3)
conv2d_6 input: (None, 256, 256, 3)
Conv2D output: (None, 256, 256, 16)
max_pooling2d_6 input: (None, 256, 256, 16)
MaxPooling2D output: (None, 128, 128, 16)
conv2d_7 input: (None, 128, 128, 16)
Conv2D output: (None, 128, 128, 32)
max_pooling2d_7 input: (None, 128, 128, 32)
MaxPooling2D output: (None, 64, 64, 32)
flatten_3 input: (None, 64, 64, 32)
Flatten output: (None, 131072)
dense_5 input: (None, 131072)
Dense output: (None, 128)
dropout_2 input: (None, 128)
Dropout output: (None, 128)
batch_normalization_2 input: (None, 128)
BatchNormalization output: (None, 128)
dense_6 input: (None, 128)
Dense output: (None, 5)

Training of the Model

```
# Compile the model by specifying the loss function, optimizer, and metrics to be used during training
model.compile
(loss="sparse_categorical_crossentropy",
 optimizer="adam",
 metrics=["accuracy"])
```

```
# Define a custom callback named Mycallback that will be used during training
class Mycallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}): # checking if the accuracy of the model exceeds 0.95
    if(logs.get("accuracy")>0.95):
      print('\nLoss is low so stop training') # print it if the accuracy is high enough
      self.model.stop_training =True
```

```
callbacks=Mycallback() # Create an instance of the Mycallback class to monitor the training process
```

| Dense | output: | (None, 128) |

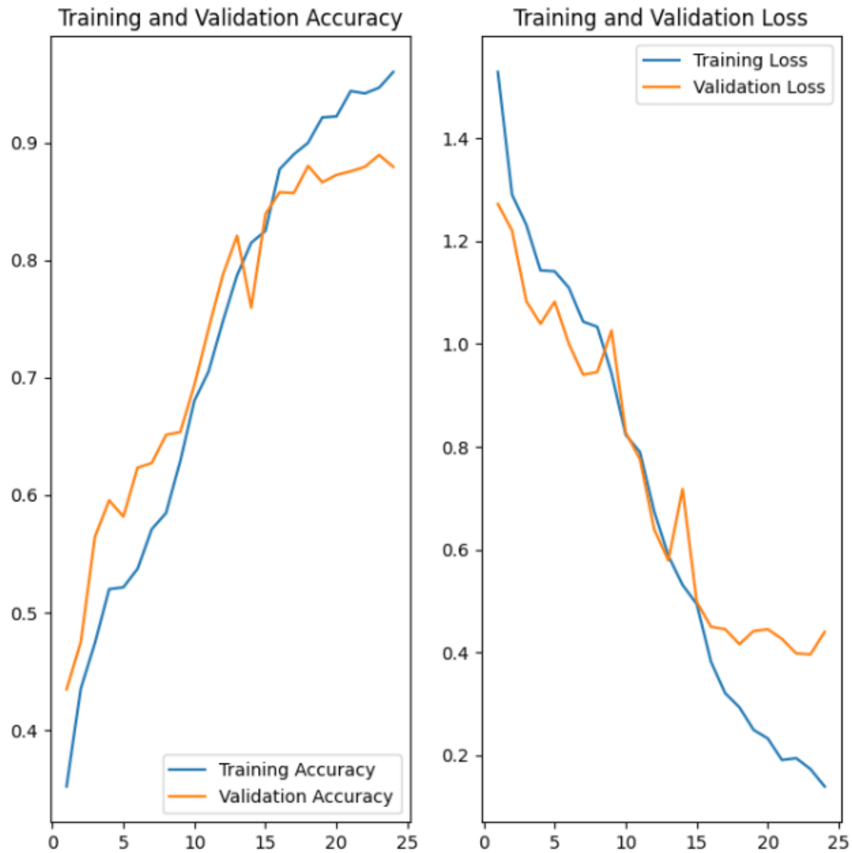
```
# set epochs of 30

epochs = 30
history = model.fit(  Train the model on the training data
  train_ds,    # Use the training dataset to train the model
  validation_data=val_ds,  # Use the validation dataset to evaluate the model
  epochs=epochs,   # train model for the speicified number
  callbacks=[callbacks] # use it for monitor train process
)
```

```
Epoch 1/30
122/122 [==============================] - 28s 192ms/step - loss: 1.5293 - accuracy: 0.3520 - val_loss: 1.2723 - val_accuracy: 0.4346
Epoch 2/30
122/122 [==============================] - 16s 130ms/step - loss: 1.2905 - accuracy: 0.4352 - val_loss: 1.2206 - val_accuracy: 0.4749
Epoch 3/30
122/122 [==============================] - 16s 128ms/step - loss: 1.2318 - accuracy: 0.4741 - val_loss: 1.0832 - val_accuracy: 0.5646
Epoch 4/30
122/122 [==============================] - 19s 148ms/step - loss: 1.1431 - accuracy: 0.5200 - val_loss: 1.0394 - val_accuracy: 0.5955
Epoch 5/30
122/122 [==============================] - 16s 130ms/step - loss: 1.1413 - accuracy: 0.5215 - val_loss: 1.0823 - val_accuracy: 0.5816
Epoch 6/30
122/122 [==============================] - 16s 128ms/step - loss: 1.1093 - accuracy: 0.5372 - val_loss: 1.0000 - val_accuracy: 0.6234
Epoch 7/30
122/122 [==============================] - 16s 129ms/step - loss: 1.0435 - accuracy: 0.5710 - val_loss: 0.9402 - val_accuracy: 0.6272
Epoch 8/30
122/122 [==============================] - 18s 142ms/step - loss: 1.0333 - accuracy: 0.5846 - val_loss: 0.9453 - val_accuracy: 0.6512
Epoch 9/30
122/122 [==============================] - 21s 164ms/step - loss: 0.9431 - accuracy: 0.6287 - val_loss: 1.0263 - val_accuracy: 0.6535
Epoch 10/30
122/122 [==============================] - 18s 144ms/step - loss: 0.8230 - accuracy: 0.6802 - val_loss: 0.8277 - val_accuracy: 0.6937
Epoch 11/30
122/122 [==============================] - 16s 128ms/step - loss: 0.7897 - accuracy: 0.7052 - val_loss: 0.7757 - val_accuracy: 0.7417
Epoch 12/30
```

```
122/122 [==============================] - 18s 141ms/step - loss: 0.6746 - accuracy: 0.7477 - val_loss: 0.6396 - val_accuracy: 0.7873
Epoch 13/30
122/122 [==============================] - 17s 132ms/step - loss: 0.5879 - accuracy: 0.7869 - val_loss: 0.5787 - val_accuracy: 0.8206
Epoch 14/30
122/122 [==============================] - 16s 129ms/step - loss: 0.5311 - accuracy: 0.8147 - val_loss: 0.7176 - val_accuracy: 0.7595
Epoch 15/30
122/122 [==============================] - 16s 129ms/step - loss: 0.4939 - accuracy: 0.8248 - val_loss: 0.4979 - val_accuracy: 0.8391
Epoch 16/30
122/122 [==============================] - 16s 129ms/step - loss: 0.3817 - accuracy: 0.8774 - val_loss: 0.4498 - val_accuracy: 0.8577
Epoch 17/30
122/122 [==============================] - 17s 131ms/step - loss: 0.3204 - accuracy: 0.8900 - val_loss: 0.4449 - val_accuracy: 0.8569
Epoch 18/30
122/122 [==============================] - 16s 128ms/step - loss: 0.2931 - accuracy: 0.8998 - val_loss: 0.4158 - val_accuracy: 0.8801
Epoch 19/30
122/122 [==============================] - 16s 129ms/step - loss: 0.2490 - accuracy: 0.9214 - val_loss: 0.4414 - val_accuracy: 0.8662
Epoch 20/30
122/122 [==============================] - 18s 144ms/step - loss: 0.2322 - accuracy: 0.9222 - val_loss: 0.4450 - val_accuracy: 0.8724
Epoch 21/30
122/122 [==============================] - 16s 128ms/step - loss: 0.1904 - accuracy: 0.9438 - val_loss: 0.4258 - val_accuracy: 0.8755
Epoch 22/30
122/122 [==============================] - 16s 130ms/step - loss: 0.1939 - accuracy: 0.9418 - val_loss: 0.3980 - val_accuracy: 0.8794
Epoch 23/30
122/122 [==============================] - 22s 174ms/step - loss: 0.1729 - accuracy: 0.9467 - val_loss: 0.3959 - val_accuracy: 0.8894
Epoch 24/30
120/122 [============================>.] - ETA: 0s - loss: 0.1391 - accuracy: 0.9599
Loss is low so stop training
122/122 [==============================] - 16s 130ms/step - loss: 0.1389 - accuracy: 0.9601 - val_loss: 0.4397 - val_accuracy: 0.8794
```

Evaluate the model:

```
# Extract the training and validation accuracy and loss from the history object returned by model.fit()
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

# Adjust the lengths of the arrays to match the actual number of epochs executed
epochs_range = range(1, len(acc) + 1)
# Create a figure with two subplots
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)

# Plot the training and validation accuracy for each epoch
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)
# Plot the training and validation loss for each epoch
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()
```



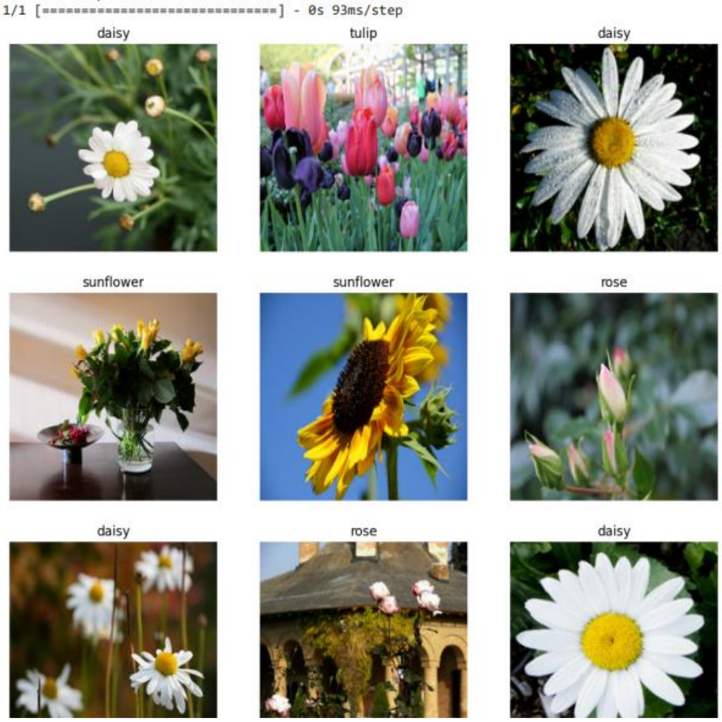

```
[ ] test_loss, test_accuracy = model.evaluate(val_ds)
    print("Test Loss:", test_loss)
    print("Test Accuracy:", test_accuracy)
```

```
41/41 [==============================] - 3s 71ms/step - loss: 0.4397 - accuracy: 0.8794
Test Loss: 0.43968042731285095
Test Accuracy: 0.8793503642082214
```

Results and Prediction:

```python
# Function to predict input examples and plot the results
def predict_and_plot(model, dataset, class_names):
    plt.figure(figsize=(12, 12))
    for images, labels in dataset.take(1):
        predictions = model.predict(images)
        for i in range(9):
            ax = plt.subplot(3, 3, i + 1)
            plt.imshow(images[i].numpy().astype("uint8"))
            plt.title(class_names[np.argmax(predictions[i])])
            plt.axis("off")

# Create test dataset
test_dir = "/content/drive/MyDrive/Flower Classification/test"
test_ds = tf.keras.utils.image_dataset_from_directory(
    test_dir,
    batch_size=32,
    image_size=(256, 256),
    shuffle=True
)

# Evaluate the model on the test dataset
test_loss, test_accuracy = model.evaluate(test_ds)
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)

# Predict and plot examples from the test set
predict_and_plot(model, test_ds, class_names)
```

```
Found 50 files belonging to 5 classes.
2/2 [==============================] - 0s 20ms/step - loss: 0.2566 - accuracy: 0.9200
Test Loss: 0.2565712034702301
Test Accuracy: 0.9200000166893005
1/1 [==============================] - 0s 93ms/step
```



```
sunflower_url = "/content/drive/MyDrive/Flower Classification/test/dandelion/13910677675_4900fa3dbf_n.jpg"
# path of image

img = tf.keras.utils.load_img(
    sunflower_url, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)  # converting image to array
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0]) # use softmax to convert the predictions

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 18ms/step
This image most likely belongs to dandelion with a 40.25 percent confidence.
```

Explaing our result, and also make predictions of test cases given in the dataset.

After evaluating the model to determine its accuracy on the validation set and test set, we obtained a Test Accuracy of o. 8793503642082214, Test Loss of o. 0.43968042731285095 on the validation set, and Test Loss of o. 0.2565712034702301 along with the Test Accuracy of o. 0.9200000166893005, indicating that the model has both a very good accuracy and good generalization ability on the given dataset.

The model, test datasets, class names, and plotting the images for the predicted class are the actual model requirements, so we must use the prediction and plot functions to carry out the prediction. We can infer from the model's output that it had a loss of 0.4397 and could predict the test accuracy of 87.94% for predicting the right value of the flower images. We can conclude that the model is performing well on the validation datasets after looking at the model as a whole.

The function we used has plotted 9 images from the test set based on the prediction done and plotted it visually to inspect the performance of the model on the test set, and finally, we can see that the model can achieve the test accuracy of 92% on the test. In the test datasets, we have used the evaluate methods to compute the overall accuracy and loss from the model of the given datasets where we have given 50 images with five different classes of the flower.

Fine-tuning a pre-trained model(Transfer Learning):

```
# Apply one-hot encoding to the labels of the training and validation dataset
train_ds = train_ds.map(lambda x, y: (x, tf.one_hot(y, depth=5)))
val_ds = val_ds.map(lambda x, y: (x, tf.one_hot(y, depth=5)))
```

```
# resnet model
resnet_model = Sequential()

pretrained_model= tf.keras.applications.ResNet50(include_top=False, # ResNet50 models
                   input_shape=(256,256,3),
                   pooling='avg',classes=5,
                   weights='imagenet')
# freeze tyhe layer in the pre-trained model
for layer in pretrained_model.layers:
        layer.trainable=False
```

```
resnet_model.add(pretrained_model)
resnet_model.add(Flatten())  # fatten the output of the pre-trained model
resnet_model.add(Dense(512, activation='relu')) # add a fully connected dense layer with 512
resnet_model.add(Dropout(0.5)),# adda adroupot layer to prevent overfitting
resnet_model.add(BatchNormalization()),  # add batch normalization layer to normalize the activations of the previous layer
resnet_model.add(Dense(5, activation='softmax')) #Add a dense layer with 5 units and softmax activation function for classification
```

```
resnet_model.summary()  # summary for resnet model
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 resnet50 (Functional)       (None, 2048)              23587712  
                                                                 
 module_wrapper (ModuleWrapp  (None, 2048)             0         
 er)                                                             
                                                                 
 module_wrapper_1 (ModuleWra  (None, 512)              1049088   
 pper)                                                           
                                                                 
 dropout (Dropout)           (None, 512)               0         
                                                                 
 batch_normalization (BatchN  (None, 512)              2048      
 ormalization)                                                   
                                                                 
 module_wrapper_2 (ModuleWra  (None, 5)                2565      
 pper)                                                           
                                                                 
=================================================================
Total params: 24,641,413
Trainable params: 1,052,677
Non-trainable params: 23,588,736
_________________________________________________________________
```

```
resnet_model.compile(optimizer=Adam(lr=0.001),loss='categorical_crossentropy',metrics=['accuracy'])
```

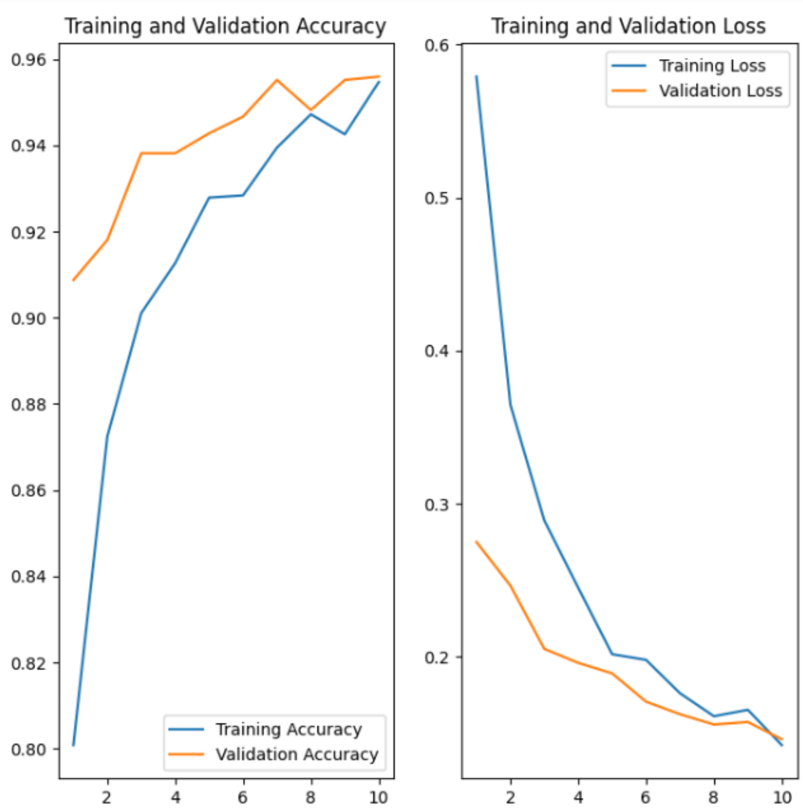
```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras.optimizers.leg
```

```
# running the epoch
epochs=10
history = resnet_model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

```
Epoch 1/10
122/122 [==============================] - 670s 5s/step - loss: 0.5790 - accuracy: 0.8008 - val_loss: 0.2749 - val_accuracy: 0.9087
Epoch 2/10
122/122 [==============================] - 22s 178ms/step - loss: 0.3647 - accuracy: 0.8725 - val_loss: 0.2464 - val_accuracy: 0.9180
Epoch 3/10
122/122 [==============================] - 23s 187ms/step - loss: 0.2890 - accuracy: 0.9011 - val_loss: 0.2051 - val_accuracy: 0.9381
Epoch 4/10
122/122 [==============================] - 27s 219ms/step - loss: 0.2450 - accuracy: 0.9127 - val_loss: 0.1959 - val_accuracy: 0.9381
Epoch 5/10
122/122 [==============================] - 23s 183ms/step - loss: 0.2016 - accuracy: 0.9279 - val_loss: 0.1890 - val_accuracy: 0.9428
Epoch 6/10
122/122 [==============================] - 28s 217ms/step - loss: 0.1979 - accuracy: 0.9284 - val_loss: 0.1705 - val_accuracy: 0.9466
Epoch 7/10
122/122 [==============================] - 27s 216ms/step - loss: 0.1761 - accuracy: 0.9394 - val_loss: 0.1623 - val_accuracy: 0.9551
Epoch 8/10
122/122 [==============================] - 27s 217ms/step - loss: 0.1610 - accuracy: 0.9472 - val_loss: 0.1556 - val_accuracy: 0.9482
Epoch 9/10
122/122 [==============================] - 27s 216ms/step - loss: 0.1652 - accuracy: 0.9425 - val_loss: 0.1573 - val_accuracy: 0.9551
Epoch 10/10
122/122 [==============================] - 27s 216ms/step - loss: 0.1422 - accuracy: 0.9547 - val_loss: 0.1460 - val_accuracy: 0.9559
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

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

# Adjust the lengths of the arrays to match the actual number of epochs executed
epochs_range = range(1, len(acc) + 1)

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()
```

[ ]



```python
sunflower_url = "/content/drive/MyDrive/Flower Classification/test/roses/21413573151_e681c6a97a.jpg"
# load image

img = tf.keras.utils.load_img(
    sunflower_url, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img) #Convert the image to an array
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = resnet_model.predict(img_array) #Use the ResNet model to predict the class of the image
score = tf.nn.softmax(predictions[0])  #Compute the softmax score for the predicted probabilities

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

```
1/1 [==============================] - 0s 25ms/step
This image most likely belongs to rose with a 36.34 percent confidence.
```