

Lab Report – 6

Transfer Learning

Goutham Krishnan

21BAI1007

Aim

1. Sample Network – Classifying the Cats v/s Dogs dataset using MobilenetV2 pretrained model.
 2. Exploring the model structure and analysis for some common pretrained models in PyTorch – AlexNet, ConvNeXt, VGG16 and ResNet.
 3. Implementation of transfer learning on classification of Flowers dataset using pretrained models VGG16, ResNet50 and ResNet152 and TensorFlow.

Code and Output for Aim 1 – Sample Network

Transfer Learning - 21BAI1007 - Sample

```
[ ] import matplotlib.pyplot as plt
import numpy as np
import os
import tensorflow as tf

_URL = 'https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip'
path_to_zip = tf.keras.utils.get_file('cats_and_dogs.zip', origin=_URL, extract=True)
PATH = os.path.join(os.path.dirname(path_to_zip), 'cats_and_dogs_filtered')

train_dir = os.path.join(PATH, 'train')
validation_dir = os.path.join(PATH, 'validation')

BATCH_SIZE = 32
IMG_SIZE = (160, 160)

train_dataset = tf.keras.utils.image_dataset_from_directory(train_dir,
                                                        shuffle=True,
                                                        batch_size=BATCH_SIZE,
                                                        image_size=IMG_SIZE)
```

Downloading data from https://storage.googleapis.com/mledu-datasets/cats_and_dogs_filtered.zip
68606236/68606236 [=====] - 1s 0us/step
Found 2000 files belonging to 2 classes.

21BAI1007-lab6-sample.ipynb

```
[ ] val_batches = tf.data.experimental.cardinality(validation_dataset)
test_dataset = validation_dataset.take(val_batches // 5)
validation_dataset = validation_dataset.skip(val_batches // 5)

[ ] print(
    "Number of validation batches: %d" %
    tf.data.experimental.cardinality(validation_dataset))
print("Number of test batches: %d" % tf.data.experimental.cardinality(test_dataset))

@ Number of validation batches: 26
Number of test batches: 6
```

21BAI1007-lab6-sample.ipynb

```
[ ] AUTOTUNE = tf.data.AUTOTUNE
train_dataset = train_dataset.prefetch(buffer_size=AUTOTUNE)
validation_dataset = validation_dataset.prefetch(buffer_size=AUTOTUNE)
test_dataset = test_dataset.prefetch(buffer_size=AUTOTUNE)

[ ] data_augmentation = tf.keras.Sequential([
    tf.keras.layers.RandomFlip("horizontal"),
    tf.keras.layers.RandomRotation(0.2),
])

[ ] for image, _ in train_dataset.take(1):
    plt.figure(figsize=(10, 10))
    first_image = image[0]
    for i in range(3):
        ax = plt.subplot(3, 3, i + 1)
        augmented_image = data_augmentation(tf.expand_dims(first_image, 0))
        plt.imshow(augmented_image[0] / 255)
        plt.axis("off")
```

21BAI1007-lab6-sample.ipynb

```
[ ] preprocess_input = tf.keras.applications.mobilenet_v2.preprocess_input
[ ] rescale = tf.keras.layers.Rescaling(1.0 / 127.5, offset=-1)

[ ] # Create the base model from the pre-trained model MobileNet V2
IMG_SHAPE = IMG_SIZE + (3,)
base_model = tf.keras.applications.MobileNetV2(
    input_shape=IMG_SHAPE, include_top=False, weights="imagenet"
)

@ Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_160_no_top.h5
9406464/9406464 [=====] - 0s 0us/step

[ ] image_batch, label_batch = next(iter(train_dataset))
feature_batch = base_model(image_batch)
print(feature_batch.shape)
```

(32, 5, 5, 1280)

21BAI1007-lab6-sample.ipynb

```
[ ] base_model.trainable = False

{x}
# Let's take a look at the base model architecture
base_model.summary()

Model: "mobilenetv2_1.00_160"
+-----+-----+-----+-----+
| Layer (Type) | Output Shape | Param # | Connected to |
+-----+-----+-----+-----+
| input_1 (Inputlayer) | [(None, 160, 160, 3)] | 0 | [] |
| Conv1 (Conv2D) | (None, 80, 80, 32) | 864 | ['input_1[0][0]'] |
| bn_Conv1 (BatchNormalizati on) | (None, 80, 80, 32) | 128 | ['Conv1[0][0]'] |
| Conv1_relu (ReLU) | (None, 80, 80, 32) | 0 | ['bn_Conv1[0][0]'] |
| expanded_conv_depthwise (D eepwiseConv2D) | (None, 80, 80, 32) | 288 | ['Conv1_relu[0][0]'] |
| expanded_conv_depthwise_EN (BatchNormalizati on) | (None, 80, 80, 32) | 128 | ['expanded_conv_depthwise[0][0]'] |
| expanded_conv_depthwise_re lu (ReLU) | (None, 80, 80, 32) | 0 | ['expanded_conv_depthwise_EN[0][0]'] |
| expanded_conv_project (Con v2D) | (None, 80, 80, 16) | 512 | ['expanded_conv_depthwise_relu[0][0]'] |
| expanded_conv_project_BN (BatchNormalization) | (None, 80, 80, 16) | 64 | ['expanded_conv_project[0][0]'] |
| block_1_expand (Conv2D) | (None, 80, 80, 96) | 1536 | ['expanded_conv_project_BN[0][0]'] |
| block_1_expand_BN (BatchN ormalization) | (None, 80, 80, 96) | 384 | ['block_1_expand[0][0]'] |
```

21BAI1007-lab6-sample.ipynb

```
[ ] global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
feature_batch_average = global_average_layer(feature_batch)
print(feature_batch_average.shape)

(32, 1280)

{ }
prediction_layer = tf.keras.layers.Dense(1)
prediction_batch = prediction_layer(feature_batch_average)
print(prediction_batch.shape)

(32, 1)

{ }
inputs = tf.keras.Input(shape=(160, 160, 3))
x = data_augmentation(inputs)
x = preprocess_input(x)
x = base_model(x, training=False)
x = global_average_layer(x)
x = tf.keras.layers.Dropout(0.2)(x)
outputs = prediction_layer(x)
model = tf.keras.Model(inputs, outputs)

base_learning_rate = 0.0001
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=base_learning_rate),
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    metrics=['accuracy'],
)

len(model.trainable_variables)
```

21BAI1007-lab6-sample.ipynb

```
[ ]
initial_epochs = 10
loss0, accuracy0 = model.evaluate(validation_dataset)
26/26 [=====] - 17s 494ms/step - loss: 0.9386 - accuracy: 0.3651

{ }
print("initial loss: {:.2f}".format(loss0))
print("initial accuracy: {:.2f}".format(accuracy0))

initial loss: 0.94
initial accuracy: 0.37

{ }
history = model.fit(
    train_dataset, epochs=initial_epochs, validation_data=validation_dataset
)

Epoch 1/10
63/63 [=====] - 63s 919ms/step - loss: 0.7915 - accuracy: 0.5880 - val_loss: 0.6452 - val_accuracy: 0.5990
Epoch 2/10
63/63 [=====] - 58s 917ms/step - loss: 0.5944 - accuracy: 0.6505 - val_loss: 0.4584 - val_accuracy: 0.7450
Epoch 3/10
63/63 [=====] - 59s 925ms/step - loss: 0.4696 - accuracy: 0.7548 - val_loss: 0.3505 - val_accuracy: 0.8379
Epoch 4/10
63/63 [=====] - 57s 910ms/step - loss: 0.3793 - accuracy: 0.8115 - val_loss: 0.2818 - val_accuracy: 0.8688
Epoch 5/10
63/63 [=====] - 58s 921ms/step - loss: 0.3295 - accuracy: 0.8415 - val_loss: 0.2358 - val_accuracy: 0.9010
Epoch 6/10
63/63 [=====] - 50s 796ms/step - loss: 0.3057 - accuracy: 0.8610 - val_loss: 0.1988 - val_accuracy: 0.9295
Epoch 7/10
63/63 [=====] - 57s 903ms/step - loss: 0.2687 - accuracy: 0.8805 - val_loss: 0.1819 - val_accuracy: 0.9332
Epoch 8/10
63/63 [=====] - 50s 786ms/step - loss: 0.2498 - accuracy: 0.8900 - val_loss: 0.1609 - val_accuracy: 0.9493
Epoch 9/10
63/63 [=====] - 52s 819ms/step - loss: 0.2274 - accuracy: 0.9005 - val_loss: 0.1472 - val_accuracy: 0.9493
Epoch 10/10
63/63 [=====] - 58s 929ms/step - loss: 0.2207 - accuracy: 0.9045 - val_loss: 0.1296 - val_accuracy: 0.9616
```

21BAI007-lab6-sample.ipynb

```
[ ] base_model.trainable = True

{x} model.compile(
    loss=tf.keras.losses.BinaryCrossentropy(from_logits=True),
    optimizer=tf.keras.optimizers.RMSprop(learning_rate=base_learning_rate / 10),
    metrics=["accuracy"],
)

[ ] len(model.trainable_variables)
158

@finetune_epochs = initial_epochs + fine_tune_epochs

history_fine = model.fit(
    train_dataset,
    epochs=total_epochs,
    initial_epoch=history.epoch[-1],
    validation_data=validation_dataset,
)

Epoch 10/20
63/63 [=====] - 160s 2s/step - loss: 0.1525 - accuracy: 0.9355 - val_loss: 0.0495 - val_accuracy: 0.9864
Epoch 11/20
63/63 [=====] - 145s 2s/step - loss: 0.1063 - accuracy: 0.9560 - val_loss: 0.0507 - val_accuracy: 0.9777
Epoch 12/20
63/63 [=====] - 144s 2s/step - loss: 0.0952 - accuracy: 0.9650 - val_loss: 0.0491 - val_accuracy: 0.9802
Epoch 13/20
63/63 [=====] - 145s 2s/step - loss: 0.0925 - accuracy: 0.9585 - val_loss: 0.0395 - val_accuracy: 0.9839
Epoch 14/20
63/63 [=====] - 152s 2s/step - loss: 0.0741 - accuracy: 0.9745 - val_loss: 0.0415 - val_accuracy: 0.9851
Epoch 15/20
63/63 [=====] - 143s 2s/step - loss: 0.0774 - accuracy: 0.9690 - val_loss: 0.0418 - val_accuracy: 0.9827
Epoch 16/20
63/63 [=====] - 136s 2s/step - loss: 0.0749 - accuracy: 0.9680 - val_loss: 0.0378 - val_accuracy: 0.9814
Epoch 17/20
```

21BAI007-lab6-sample.ipynb

```
[ ] # Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()

{x} predictions = model.predict_on_batch(image_batch).flatten()

# Apply a sigmoid since our model returns logits
predictions = tf.where(predictions < 0.5, 0, 1)

print("Predictions:\n", predictions.numpy())
print("Labels:\n", label_batch)

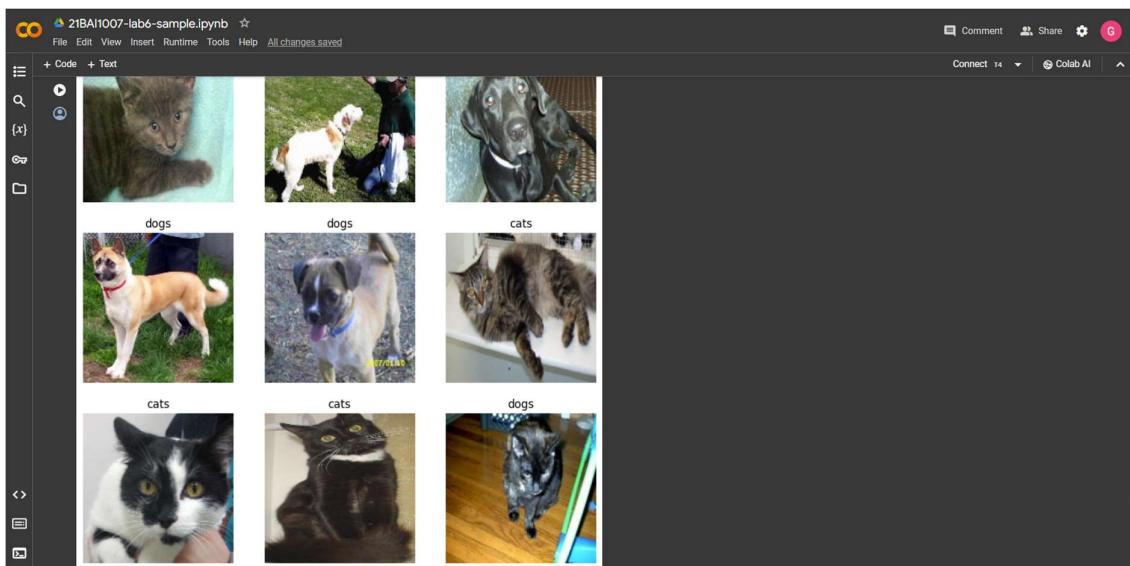
plt.figure(figsize=(10, 10))
for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(image_batch[i].astype("uint8"))
    plt.title(class_names[predictions[i]])
    plt.axis("off")

Predictions:
[0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 0 1 0 1 1 0 1 0 1 1 1 0 0 1]
Labels:
[0 1 1 1 0 0 0 1 1 0 0 0 0 0 0 1 1 0 1 0 0 0 1 1 1 0 0 1]

cats      dogs      dogs



```



Code and Output for Aim 2 – Model Structure Analysis

Exploratory model structure analysis - 21BAI1007

```
[ ] pip install torchinfo
Collecting torchinfo
  Downloading torchinfo-1.8.0-py3-none-any.whl (23 kB)
Installing collected packages: torchinfo
Successfully installed torchinfo-1.8.0

[ ] import torch
from torch import nn
import torchvision
import os
from typing import List, Any, Tuple
from torchinfo import summary
from torchvision.datasets import VisionDataset
```

View the alexnet model structure

```
[ ] alexnet = torchvision.models.alexnet(weights=None)
alexnet
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(5, 5))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

View the ConvNeXt Base model

```
[ ] convnext_base = torchvision.models.convnext_base(weights=None)
convnext_base
```

```
(stochastic_depth): StochasticDepth(p=0.44285714285714284, mode=row)
  (0): CMBlock(
    (block): Sequential(
      (0): Conv2d(3, 512, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=512)
      (1): LayerNorm(512), eps=1e-06, elementwise_affine=True
      (2): LayerNorm(512), eps=1e-06, elementwise_affine=True
      (3): Linear(in_features=512, out_features=2048, bias=True)
      (4): GELU(approximate='none')
      (5): Linear(in_features=2048, out_features=512, bias=True)
      (6): Permute()
    )
    (stochastic_depth): StochasticDepth(p=0.45714285714285713, mode=row)
  )
  (1): Sequential(
    (0): LayerNorm2d(512,), eps=1e-06, elementwise_affine=True
    (1): Conv2d(512, 1024, kernel_size=(2, 2), stride=(2, 2))
  )
  (2): Sequential(

```

ConvNeXt Large model

```
[ ] convnext_large = torchvision.models.convnext_large(weights=None)
convnext_large
```

```
(stochastic_depth): StochasticDepth(p=0.4714285714285714, mode=row)
  (0): CMBlock(
    (block): Sequential(
      (0): Conv2d(1024, 1024, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=1024)
      (1): Permute()
      (2): LayerNorm(1024), eps=1e-06, elementwise_affine=True
      (3): Linear(in_features=1024, out_features=4096, bias=True)
      (4): GELU(approximate='none')
      (5): Linear(in_features=4096, out_features=1024, bias=True)
      (6): Permute()
    )
    (stochastic_depth): StochasticDepth(p=0.4714285714285714, mode=row)
  )
  (1): CMBlock(
    (block): Sequential(
      (0): Conv2d(1024, 1024, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=1024)
      (1): Permute()
      (2): LayerNorm(1024), eps=1e-06, elementwise_affine=True
      (3): Linear(in_features=1024, out_features=4096, bias=True)
      (4): GELU(approximate='none')
      (5): Linear(in_features=4096, out_features=1024, bias=True)
      (6): Permute()
    )
    (stochastic_depth): StochasticDepth(p=0.4857142857142857, mode=row)
  )
  (2): CMBlock(
    (block): Sequential(
      (0): Conv2d(1024, 1024, kernel_size=(7, 7), stride=(1, 1), padding=(3, 3), groups=1024)
      (1): Permute()
      (2): LayerNorm(1024), eps=1e-06, elementwise_affine=True
      (3): Linear(in_features=1024, out_features=4096, bias=True)
      (4): GELU(approximate='none')
      (5): Linear(in_features=4096, out_features=1024, bias=True)
      (6): Permute()
    )
    (stochastic_depth): StochasticDepth(p=0.5, mode=row)
  )
  (3): AdaptiveAvgPool2d(output_size=1)
  (4): Flatten(start_dim=1, end_dim=-1)
  (5): LayerNorm2d(1024,), eps=1e-06, elementwise_affine=True
  (6): Linear(in_features=1024, out_features=1000, bias=True)
```

2IBAI007-modelStructure ☆

File Edit View Insert Runtime Tools Help All changes saved

Comment Share Colab AI

+ Code + Text Connect Colab AI

Explore the children modules for alexnet

```
[ ] dict(alexnet.named_children()).keys()  
dict_keys(['features', 'avgpool', 'classifier'])
```

Explore child layers for all pre-trained classification models in torchvision

```
layer_count = dict()  
  
for model_name in classification_models:  
    m = getattr(torchvision.models, model_name)(weights=None)  
    layer_count = dict(enumerate(m.named_children()).keys())  
    for l in layer_names:  
        layer_count[l] = layer_count.get(l, 0) + 1  
    # end for  
# end for  
  
print(layer_count)
```

/usr/local/lib/python3.10/dist-packages/torchvision/models/googlenet.py:47: FutureWarning: The default weight initialization of GoogleNet will be changed in future releases of torchvision. If you wish to keep warnings.warn()
/usr/local/lib/python3.10/dist-packages/torchvision/models/inception_v3.py:43: FutureWarning: The default weight initialization of inception_v3 will be changed in future releases of torchvision. If you wish to keep warnings.warn()
/usr/local/lib/python3.10/dist-packages/torch/functional.py:507: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered internally at .../aten/src/ATen/native/TensorUtils.cpp:101)
{'features': 39, 'avgpool': 59, 'classifier': 38, 'conv1': 15, 'maxpool1': 2, 'conv2': 3, 'conv3': 1, 'maxpool2': 2, 'inception3a': 1, 'inception3b': 1, 'maxpool3': 1, 'inception4a': 1, 'inception4b': 1, 'maxpool4': 1, 'inception5a': 1, 'inception5b': 1}

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** 2IBAH007-modelStructure
- File Menu:** File, Edit, View, Insert, Runtime, Tools, Help, All changes saved
- Toolbar:** Comment, Share, Colab AI
- Code Cell:** Focus on classification layer for vgg16, resnet50, resnet152
- Code Content:**

```
vgg16 = torchvision.models.vgg16_bn(weights=None)
resnet50 = torchvision.models.resnet50(weights=None)
resnet152 = torchvision.models.resnet152(weights=None)
print("vgg16\n", vgg16.classifier)
print("resnet50\n", resnet50.fc)
print("resnet152\n", resnet152.fc)

vgg16
Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
)
resnet50
Linear(in_features=2048, out_features=1000, bias=True)
resnet152
Linear(in_features=2048, out_features=1000, bias=True)
```

Code and Output for Aim 3 – VGG16, ResNet50, ResNet152

2IBA1007_HOTS.ipynb

File Edit View Insert Runtime Tools Help Last saved at 7:53PM

Comment Share

+ Code + Text

Connect 14 Colab AI

Classifying flowers with pretrained mobilenetV2

```
[x] import numpy as np
import cv2

import PIL.Image as Image
import os

import matplotlib.pyplot as plt

import tensorflow_hub as hub
import tensorflow as tf

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

from keras.optimizers import Adam
from keras.utils import plot_model

import torch

[ ] dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, cache_dir='.', untar=True)
# cache_dir indicates where to download data. I specified . which means current directory
# untar True will unzip it

[ ] data_dir
'./datasets/flower_photos'
```

21BAI1007_HOTS.ipynb

```
[ ] import pathlib  
[x] data_dir = pathlib.Path(data_dir)  
data_dir  
PosixPath('datasets/flower_photos')  
  
◎ img_height , img_width = 180, 180  
batch_size = 32  
  
train_ds = tf.keras.preprocessing.image_dataset_from_directory(  
    data_dir,  
    label_mode = 'categorical',  
    validation_split = 0.2,  
    subset = 'training',  
    seed = 123,  
    image_size = (img_height, img_width),  
    batch_size = batch_size  
)  
  
Found 3670 files belonging to 5 classes.  
Using 2936 files for training.  
  
[ ] # Preprocessing and load the validation data  
val_ds = tf.keras.preprocessing.image_dataset_from_directory(  
    data_dir,  
    label_mode = 'categorical',  
    validation_split = 0.2,  
    subset = 'validation',  
    seed = 123,  
    image_size=(img_height,img_width),  
    batch_size=batch_size  
)  
  
Found 3670 files belonging to 5 classes.  
Using 734 files for validation.
```

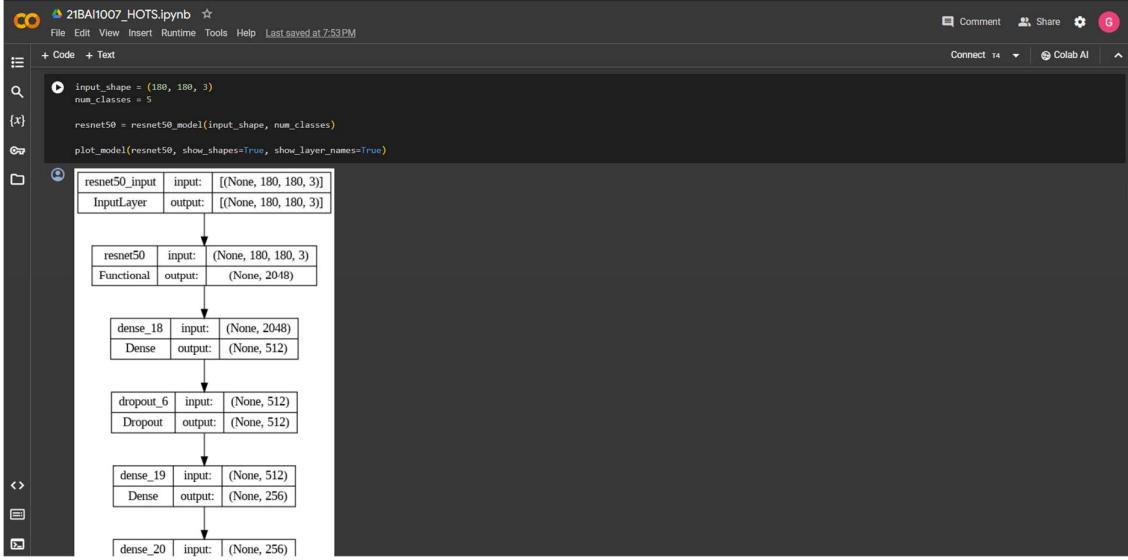
21BAI1007_HOTS.ipynb

```
[ ] ## Class names list that stores the class names of the flowers dataset  
class_names = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']  
  
print(class_names)  
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']  
  
[ ] import matplotlib.pyplot as plt  
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)  
        plt.imshow(images[i].numpy().astype('uint8'))  
        plt.axis('off')  
  

```

21BAI1007_HOTS.ipynb

```
[ ]  
  
[ ] Resnet50 Model  
  
[ ] def resnet50_model(in_shape, num_classes):  
    #####Importing the model  
    resnet_model = Sequential()  
    pretrained_model = keras.applications.ResNet50(include_top=False, input_shape=in_shape, pooling='avg', classes=num_classes, weights='imagenet')  
  
    for layer in pretrained_model.layers:  
        layer.trainable = False  
        resnet_model.add(pretrained_model)  
  
    # Add dense layers for classification  
    resnet_model.add(Dense(512, activation='relu'))  
    resnet_model.add(Dropout(0.6))  
    resnet_model.add(Dense(256, activation='relu'))  
    resnet_model.add(Dense(5, activation='softmax'))  
  
    resnet_model.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001), metrics='accuracy')  
  
    return resnet_model
```

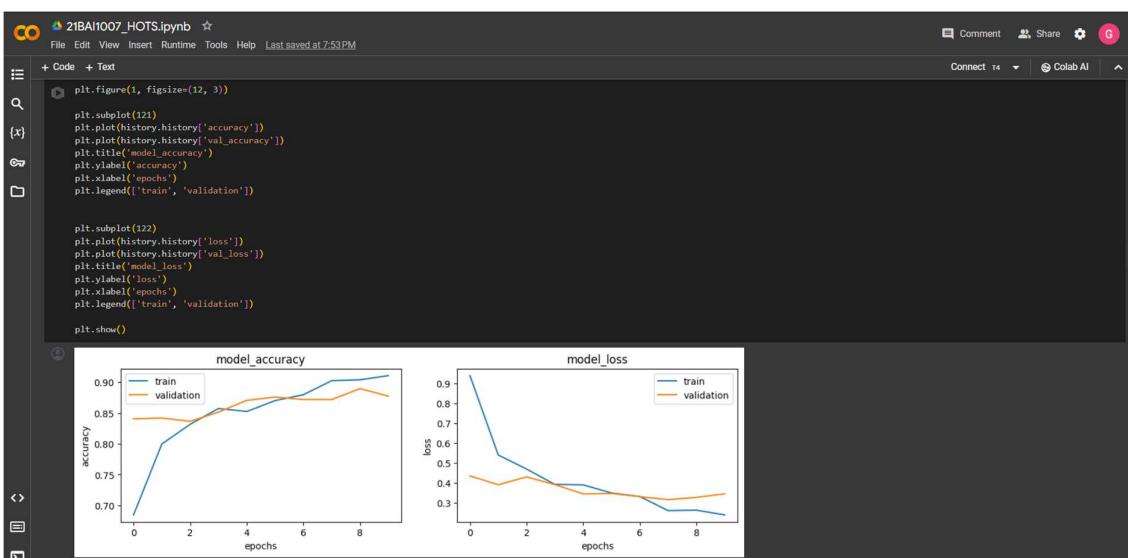


```

[ ] # Train upto 10 epochs
# Fit the model in a variable called history for further analysis
history = resnet50.fit(train_ds, validation_data=val_ds, epochs=10)

Epoch 1/10
92/92 [=====] - 14s 11ms/step - loss: 0.9398 - accuracy: 0.6846 - val_loss: 0.4361 - val_accuracy: 0.8406
Epoch 2/10
92/92 [=====] - 9s 96ms/step - loss: 0.5499 - accuracy: 0.7997 - val_loss: 0.3913 - val_accuracy: 0.8420
Epoch 3/10
92/92 [=====] - 12s 129ms/step - loss: 0.4702 - accuracy: 0.8317 - val_loss: 0.4088 - val_accuracy: 0.8365
Epoch 4/10
92/92 [=====] - 9s 98ms/step - loss: 0.3930 - accuracy: 0.8573 - val_loss: 0.3921 - val_accuracy: 0.8515
Epoch 5/10
92/92 [=====] - 10s 101ms/step - loss: 0.3908 - accuracy: 0.8525 - val_loss: 0.3453 - val_accuracy: 0.8706
Epoch 6/10
92/92 [=====] - 9s 93ms/step - loss: 0.3499 - accuracy: 0.8702 - val_loss: 0.3476 - val_accuracy: 0.8760
Epoch 7/10
92/92 [=====] - 9s 98ms/step - loss: 0.3322 - accuracy: 0.8798 - val_loss: 0.3315 - val_accuracy: 0.8719
Epoch 8/10
92/92 [=====] - 9s 97ms/step - loss: 0.2613 - accuracy: 0.9026 - val_loss: 0.3165 - val_accuracy: 0.8719
Epoch 9/10
92/92 [=====] - 9s 99ms/step - loss: 0.2636 - accuracy: 0.9046 - val_loss: 0.3281 - val_accuracy: 0.8896
Epoch 10/10
92/92 [=====] - 9s 91ms/step - loss: 0.2397 - accuracy: 0.9198 - val_loss: 0.3458 - val_accuracy: 0.8774

```



21BAI007_HOTS.ipynb

```
+ Code + Text
```

VGG16 Model

```
[x] def vgg16_model(in_shape, num_classes):
    # Importing the model
    vgg16 = Sequential()
    pretrained_model = keras.applications.vgg16.VGG16(include_top=False, input_shape=in_shape, pooling='avg', classes=num_classes, weights='imagenet')

    for layer in pretrained_model.layers:
        layer.trainable = False
        vgg16.add(pretrained_model)

    # Add dense layers for classification
    vgg16.add(Dense(512, activation='relu'))
    vgg16.add(Dropout(0.6))
    vgg16.add(Dense(256, activation='relu'))
    vgg16.add(Dense(5, activation='softmax'))

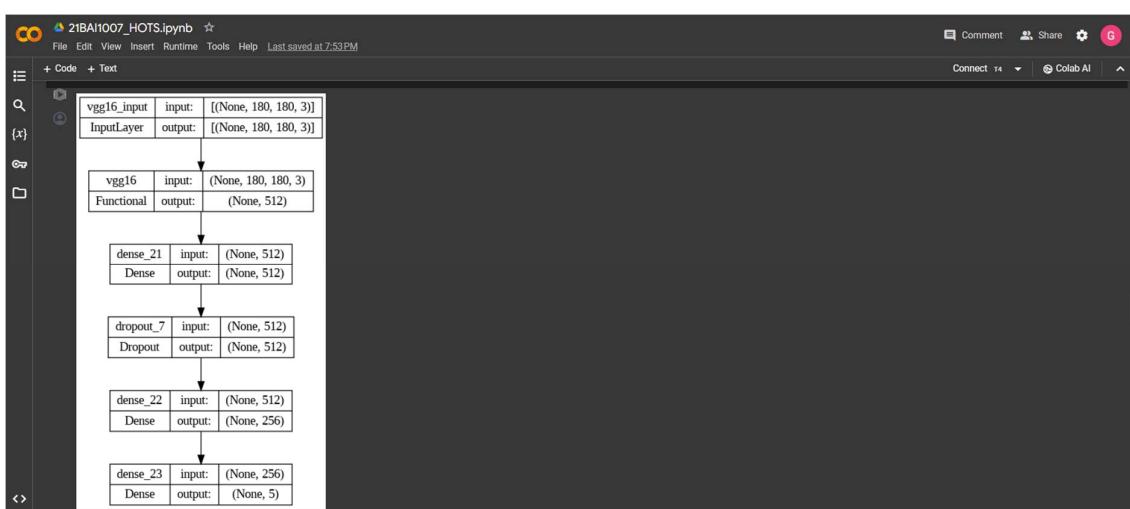
    vgg16.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001), metrics=['accuracy'])

    return vgg16

input_shape = (180, 180, 3)
num_classes = 5

vgg16 = vgg16_model(input_shape, num_classes)

plot_model(vgg16, show_shapes=True, show_layer_names=True)
```

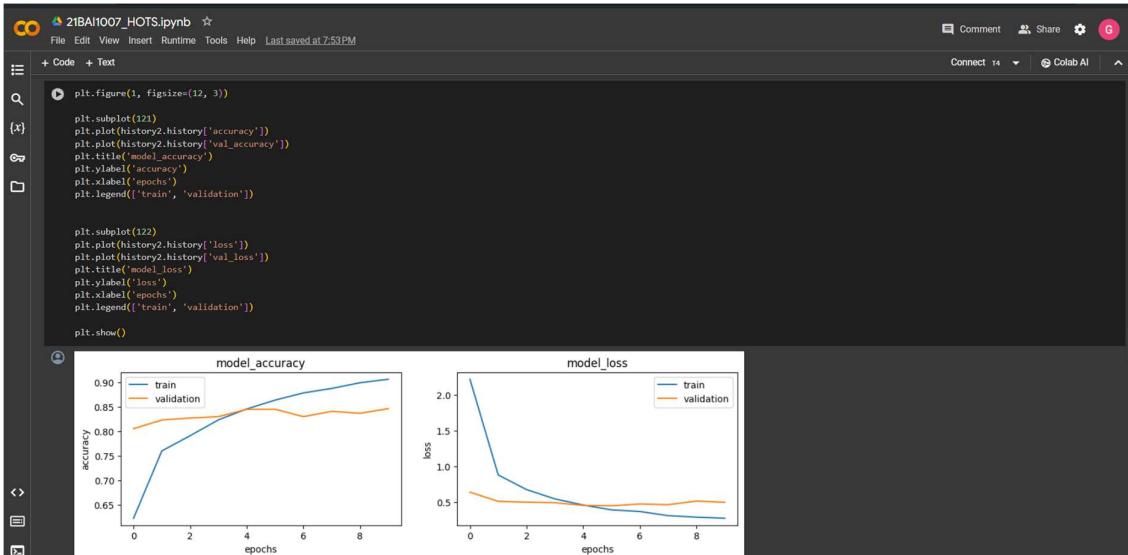


21BAI007_HOTS.ipynb

```
+ Code + Text
```

```
[ ] history2 = vgg16.fit(train_ds, validation_data=val_ds, epochs=10)
```

```
Epoch 1/10
92/92 [=====] - 14s 124ms/step - loss: 2.2218 - accuracy: 0.6226 - val_loss: 0.6371 - val_accuracy: 0.8052
Epoch 2/10
92/92 [=====] - 12s 124ms/step - loss: 0.8778 - accuracy: 0.7599 - val_loss: 0.5077 - val_accuracy: 0.8229
Epoch 3/10
92/92 [=====] - 12s 125ms/step - loss: 0.6729 - accuracy: 0.7909 - val_loss: 0.4966 - val_accuracy: 0.8276
Epoch 4/10
92/92 [=====] - 12s 125ms/step - loss: 0.5422 - accuracy: 0.8232 - val_loss: 0.4888 - val_accuracy: 0.8297
Epoch 5/10
92/92 [=====] - 12s 128ms/step - loss: 0.4566 - accuracy: 0.8454 - val_loss: 0.4512 - val_accuracy: 0.8447
Epoch 6/10
92/92 [=====] - 12s 126ms/step - loss: 0.3883 - accuracy: 0.8634 - val_loss: 0.4455 - val_accuracy: 0.8447
Epoch 7/10
92/92 [=====] - 12s 125ms/step - loss: 0.3657 - accuracy: 0.8781 - val_loss: 0.4714 - val_accuracy: 0.8297
Epoch 8/10
92/92 [=====] - 12s 128ms/step - loss: 0.3074 - accuracy: 0.8873 - val_loss: 0.4607 - val_accuracy: 0.8406
Epoch 9/10
92/92 [=====] - 12s 126ms/step - loss: 0.2860 - accuracy: 0.8988 - val_loss: 0.5120 - val_accuracy: 0.8365
Epoch 10/10
92/92 [=====] - 12s 123ms/step - loss: 0.2709 - accuracy: 0.9060 - val_loss: 0.4950 - val_accuracy: 0.8460
```



21BAI1007_HOTS.ipynb

```

def resnet152_model(in_shape, num_classes):
    """Importing the model
    resnet152 = Sequential()
    pretrained_model = keras.applications.ResNet152(include_top=False, input_shape=in_shape, pooling='avg', classes=num_classes, weights='imagenet')

    for layer in pretrained_model.layers:
        layer.trainable = False
    resnet152.add(pretrained_model)

    # Add dense layers for classification
    resnet152.add(Dense(512, activation='relu'))
    resnet152.add(Dropout(0.5))
    resnet152.add(Dense(256, activation='relu'))
    resnet152.add(Dense(5, activation='softmax'))

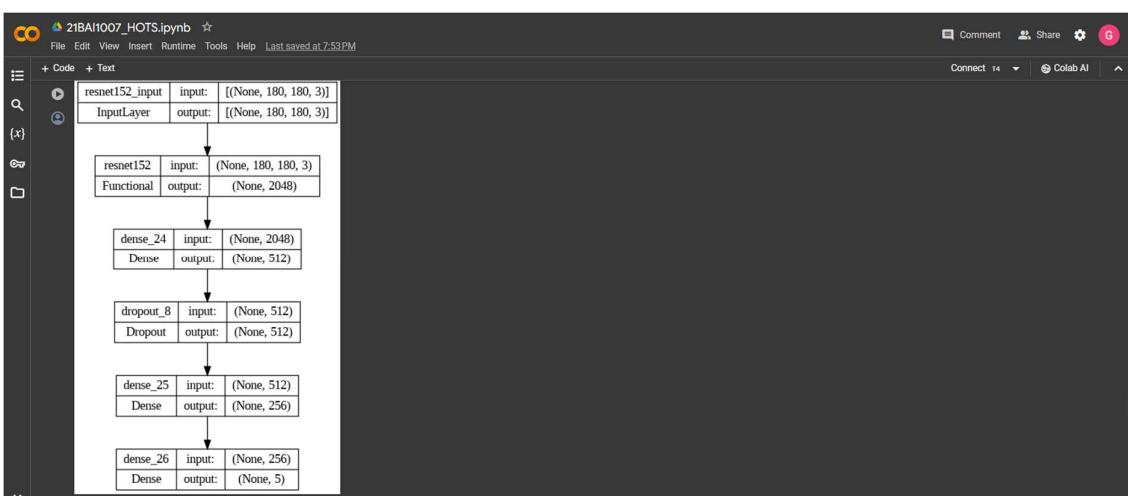
    resnet152.compile(loss='categorical_crossentropy', optimizer=Adam(learning_rate=0.001), metrics='accuracy')

    return resnet152

[ ] input_shape = (180, 180, 3)
num_classes = 5

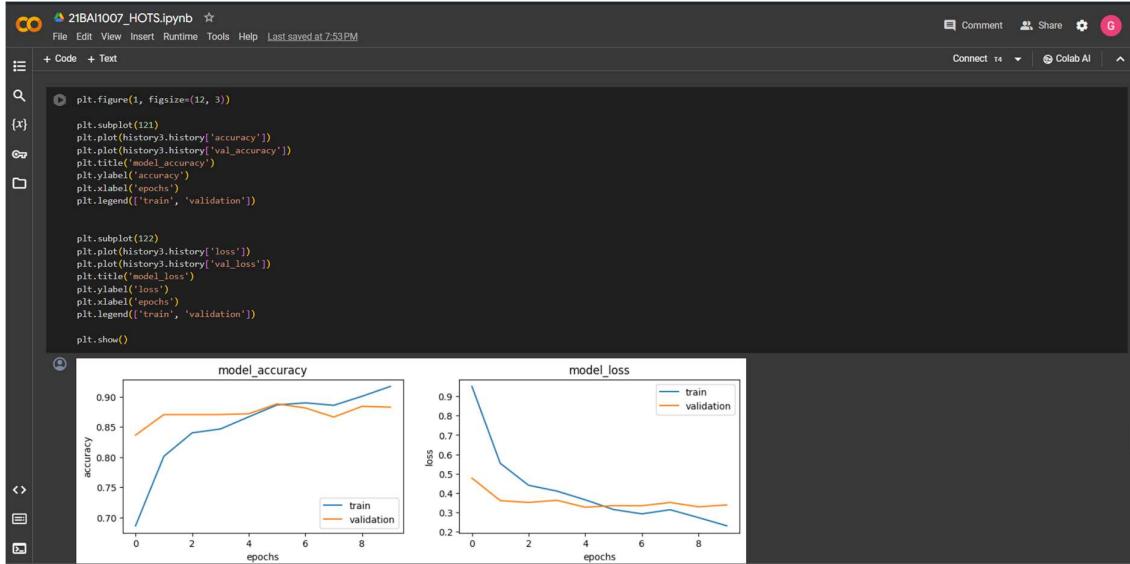
resnet152 = resnet152_model(input_shape, num_classes)
plot_model(resnet152, show_shapes=True, show_layer_names=True)

```



```
2IBAI1007_HOTS.ipynb ☆
File Edit View Insert Runtime Tools Help Last saved at 7:53PM
Comment Share G 6
+ Code + Text
history3 = resnet152.fit(train_ds, validation_data=val_ds, epochs=10)

Epoch 1/10
92/92 [=====] - 36s 292ms/step - loss: 0.9514 - accuracy: 0.6863 - val_loss: 0.4774 - val_accuracy: 0.8365
Epoch 2/10
92/92 [=====] - 21s 220ms/step - loss: 0.5539 - accuracy: 0.8914 - val_loss: 0.3609 - val_accuracy: 0.8786
Epoch 3/10
92/92 [=====] - 21s 220ms/step - loss: 0.4403 - accuracy: 0.8403 - val_loss: 0.3513 - val_accuracy: 0.8706
Epoch 4/10
92/92 [=====] - 22s 231ms/step - loss: 0.4102 - accuracy: 0.8467 - val_loss: 0.3628 - val_accuracy: 0.8706
Epoch 5/10
92/92 [=====] - 21s 220ms/step - loss: 0.3651 - accuracy: 0.8668 - val_loss: 0.3267 - val_accuracy: 0.8719
Epoch 6/10
92/92 [=====] - 21s 219ms/step - loss: 0.3149 - accuracy: 0.8866 - val_loss: 0.3349 - val_accuracy: 0.8883
Epoch 7/10
92/92 [=====] - 21s 225ms/step - loss: 0.2925 - accuracy: 0.8908 - val_loss: 0.3344 - val_accuracy: 0.8815
Epoch 8/10
92/92 [=====] - 20s 218ms/step - loss: 0.3138 - accuracy: 0.8859 - val_loss: 0.3512 - val_accuracy: 0.8665
Epoch 9/10
92/92 [=====] - 21s 225ms/step - loss: 0.2732 - accuracy: 0.9009 - val_loss: 0.3289 - val_accuracy: 0.8842
Epoch 10/10
92/92 [=====] - 21s 230ms/step - loss: 0.2307 - accuracy: 0.9172 - val_loss: 0.3386 - val_accuracy: 0.8828
```



```
2IBAI1007_HOTS.ipynb ☆
File Edit View Insert Runtime Tools Help Last saved at 7:53PM
Comment Share G
+ Code + Text
▼ Predicting an image using the best model - Resnet152

{x}
# Function to preprocess the input image
def preprocess_image(image_path, target_size=(180, 180)):
    img = keras.preprocessing.image.load_img(image_path, target_size=target_size)
    img_array = keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0) # Create a batch
    return img_array

# Function to make predictions using the model
def predict_flower(image_path, model):
    # Preprocess the input image
    img = preprocess_image(image_path)

    # Make predictions
    predictions = model.predict(img)

    # Decode predictions
    predicted_class = class_names[np.argmax(predictions)]

    return predicted_class

# Path to the random flower image
random_flower_image_path = "test_images/flower_test.jpg" # Replace with the path to your image

# Predict the flower class
predicted_class = predict_flower(random_flower_image_path, resnet152)

# Display the input image along with the predicted class
plt.figure(figsize=(6, 6))
img = keras.preprocessing.image.load_img(random_flower_image_path)
plt.imshow(img)
plt.title(f"Predicted Class: {predicted_class}")
plt.axis('off')
plt.show()
```

A screenshot of a Jupyter Notebook titled "21BAI1007_HOTS.ipynb". The code cell contains Python code to display a flower image and its predicted class:

```
# Display the input image along with the predicted class
plt.figure(figsize=(6, 6))
img = keras.preprocessing.image.load_img(random_flower_image_path)
plt.imshow(img)
plt.title("Predicted Class: " + predicted_class)
plt.axis('off')
plt.show()
```

The output cell shows the image of a pink lotus flower with the caption "Predicted Class: roses".

Comparison between ResNet50, ResNet150 and VGG16 pretrained models

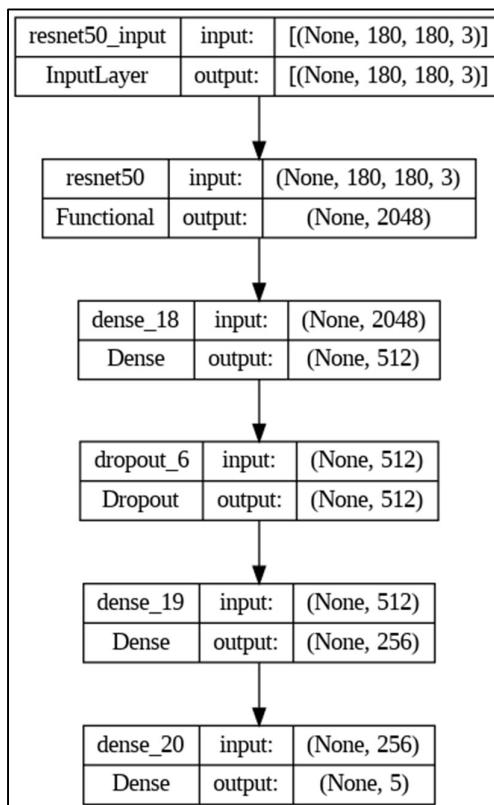


Fig: ResNet50

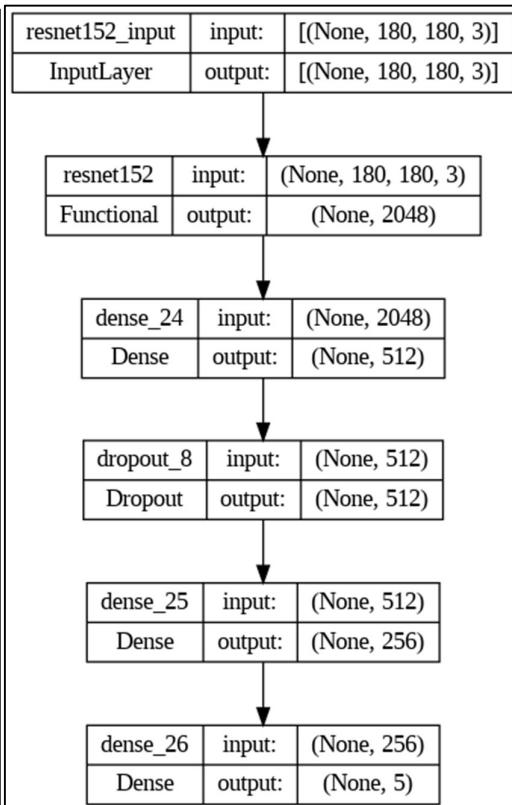


Fig: ResNet152

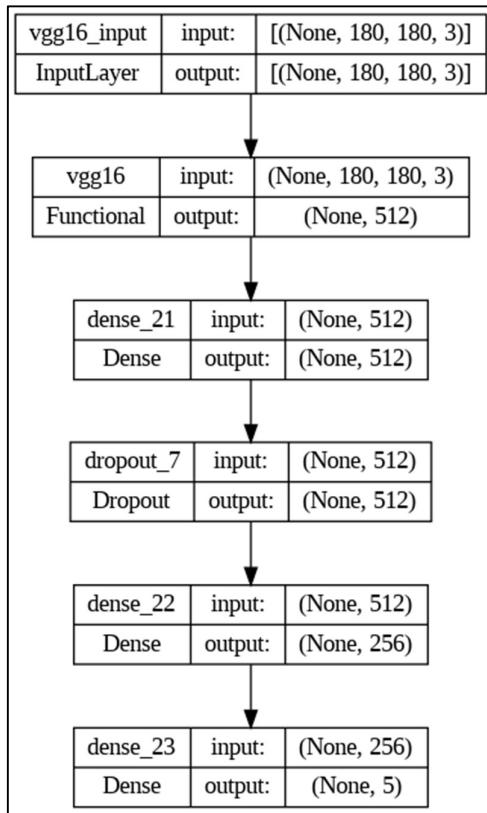


Fig: VGG16

Comparison based on training performance on Flowers Dataset

Model	VGG16	ResNet50	ResNet152
Training Loss	0.2709	0.2397	0.2307
Training Accuracy	0.9060	0.9108	0.9172
Validation Loss	0.4950	0.3458	0.3386
Validation Accuracy	0.8460	0.8774	0.8828

Results

Hence by comparing the three models based on its performance on classifying the Flowers dataset, ResNet152 has performed the best with an accuracy of 88.28% compared to 87.74% for ResNet50 and 84.60% for VGG16.