**Lab Assignment 9\_1**

**Neural Network & Deep Learning**

**Transfer Learning**

PART B

|  |  |
| --- | --- |
| Roll No: C0009 | Name: Samarth Borade |
| Class : B | Batch : EB1 |
| Date of Experiment: 27/02/24 | Date of Submission |
| Grade : |  |

**B.1 Software Code written by student:**

#Samarth Borade

#C009

#BTI SEM 10

#EXP 9: Transfer Learning

# Step 1

import tensorflow as tf

from tensorflow.keras.layers import Input, Dense, Flatten

from tensorflow.keras.models import Model

from tensorflow.keras.applications.vgg16 import VGG16

from tensorflow.keras.applications.vgg16 import preprocess\_input

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.preprocessing import image\_dataset\_from\_directory

import cv2

import numpy as np

import os

import pandas as pd

from keras.applications import VGG16

import ssl

# Workaround to avoid SSL certificate verification error

ssl.\_create\_default\_https\_context = ssl.\_create\_unverified\_context

# Load VGG16 model

vgg\_model = VGG16()

vgg\_model.summary()

# %%

# Step 2

from keras.preprocessing import image

from keras.applications.vgg16 import preprocess\_input, decode\_predictions

import numpy as np

import matplotlib.pyplot as plt

# Load and preprocess image

img\_path = './TRAIN.1/B/TRAIN.1\_BIODEG\_ORI\_0.jpg'

img = image.load\_img(img\_path, *target\_size*=(224, 224))

x = image.img\_to\_array(img)

x = np.expand\_dims(x, *axis*=0)

x = preprocess\_input(x)

# Show the image

plt.imshow(img)

plt.axis('off')

plt.show()

# Predict class of the image

preds = vgg\_model.predict(x)

predicted\_class = decode\_predictions(preds, *top*=1)[0][0]

print('Predicted class:', predicted\_class[1])

# %%

# Step 3

from keras.models import Model

# Customize VGG16 model for feature extraction

feat\_extractor = Model(*inputs*=vgg\_model.input, *outputs*=vgg\_model.get\_layer('fc2').output)

# Use model as feature extractor

features = feat\_extractor.predict(x)

# Step 4: Remove the last layer of the model and create a new model

from keras.models import Model

new\_vgg\_model = Model(*inputs*=vgg\_model.input, *outputs*=vgg\_model.layers[-2].output) # excluding the last layer

num\_layers = len(new\_vgg\_model.layers)

print("Number of layers in the model after removing the last layer:", num\_layers)

A screenshot of a computer program

Description automatically generated

# VGG model requires 224\*224 input so we are going to re-size all images

IMAGE\_SIZE = [224, 224]

# %%

#file\_path="../TEST/"

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# create generator

datagen = ImageDataGenerator(*rescale* = 1./255,

*shear\_range* = 0.2,

*zoom\_range* = 0.2,

*horizontal\_flip* = True,

*vertical\_flip*=True,

*rotation\_range*=30)

# prepare an iterators for each dataset

train\_it = datagen.flow\_from\_directory('./TRAIN.1',

*class\_mode*='categorical',

*target\_size*=(224, 224),

*batch\_size*=2,

*seed*=7)

test\_it = datagen.flow\_from\_directory('./TEST',

*class\_mode*='categorical',

*target\_size*=(224, 224),

*batch\_size*=2,

*seed*=7)

# %% A black background with white text

Description automatically generated

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras import Sequential

# ## Using VGG16 model

# %%

# VGG16 model

# %%

vgg = VGG16(*input\_shape*=IMAGE\_SIZE + [3], *weights*='imagenet', *include\_top*=False)

# %%

for layer in vgg.layers:

layer.trainable = False

# %%

output\_classes = 2

# %%

##Adding flatten and Dense Layer

x = Flatten()(vgg.output)

x = Dense(1000, *activation*='relu')(x)

prediction = Dense(output\_classes, *activation*='softmax')(x)

# %%

# create a model object

model = Model(*inputs*=vgg.input, *outputs*=prediction)

# %%

# view the structure of the model

model.summary()

# tell the model what cost and optimization method to use

from tensorflow.keras.losses import CategoricalCrossentropy

model.compile(

*loss*=CategoricalCrossentropy(),

*optimizer*='adam',

*metrics*=['accuracy']

)

history\_vgg= model.fit(

train\_it,

*validation\_data*=test\_it,

*epochs*=1 # keep epochs=5 if you want to just check as it take more computational time

)



from keras.preprocessing.image import load\_img

image = load\_img('./tomato.jpeg', *target\_size*=(224, 224))

img = np.array(image)

img = img / 255.0

img = img.reshape(1,224,224,3)

label = model.predict(img)

l={"Biodegradable":label[0][0],"NON-Biodegradable":label[0][1]}

def get\_key(*val*):

for key, value in l.items():

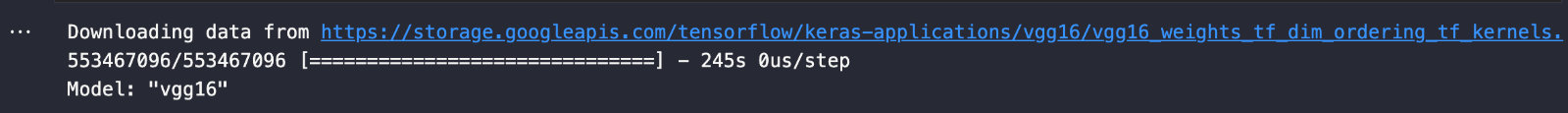
if *val* == value:

return key

return "key doesn't exist"

#label[0][1]

Output:



For tomato:

A close up of a tomato

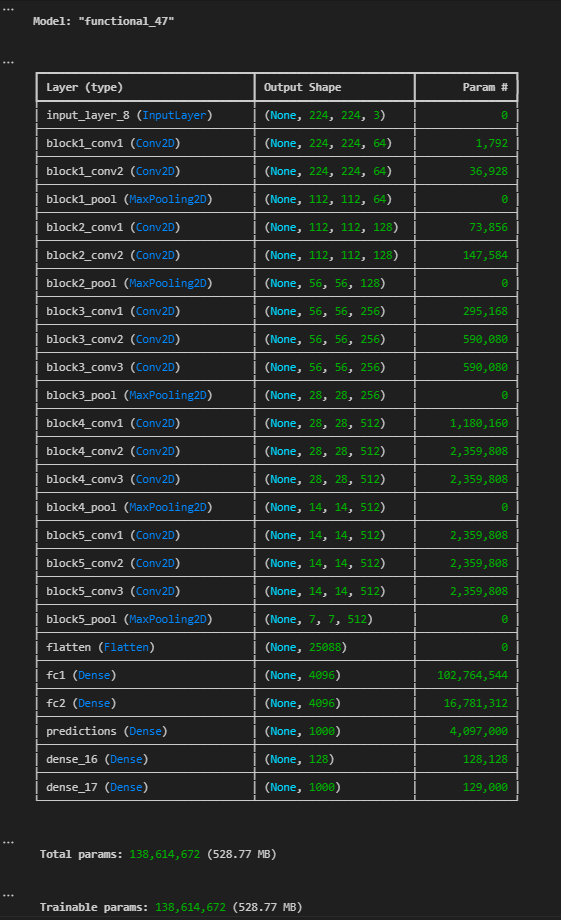
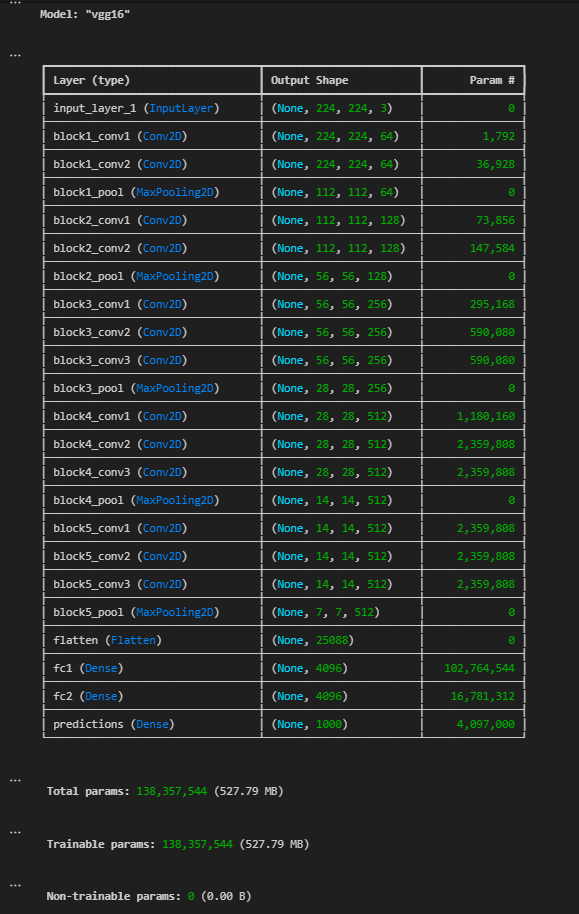
Description automatically generated

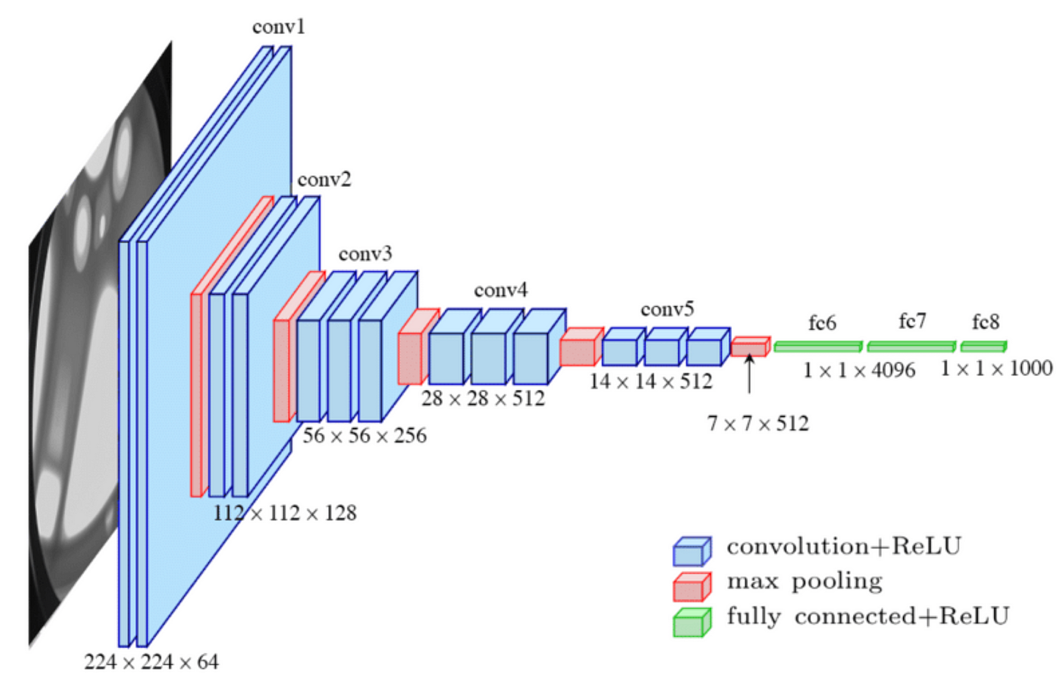
A screenshot of a computer

Description automatically generated

**B.3 Observations and learning:**

VGG16 model vs New updated model:

****



In this code, I implemented transfer learning using the VGG16 pre-trained model for image classification. Initially, I loaded the VGG16 model and observed its structure. Then, I preprocessed an image and predicted its class using the VGG16 model. After that, I extracted features from the VGG16 model and created a new model by removing the last layer. The new model was then customized by adding Dense layers for classification. Data generators were prepared for training and testing datasets. Subsequently, I compiled and trained the model, observing its performance over one epoch.

**B.4 Conclusion:**

Overall, this experiment provided valuable insights into transfer learning and model customization using pre-trained architectures like VGG16. While the initial training epoch gave a glimpse of the model's performance, further training and evaluation are necessary for a comprehensive assessment of its capabilities in image classification tasks.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*