**Lab Assignment 8**

**Neural Network & Deep Learning**

**Transfer Learning**

## PART B

(PART B: TO BE COMPLETED BY STUDENTS)

**(Students must submit the soft copy as per following segments within two hours of the practical. The soft copy must be uploaded on the Blackboard or emailed to the concerned lab in charge faculties at the end of the practical in case the there is no Black board access available)**

|  |  |
| --- | --- |
| Roll No. C026 | Name: Anirbaan Ghatak |
| Program : BTI | Division: B |
| Batch: EB1 | Date of Experiment: 23/02/24 |
| Date of Submission: 23/02/24 | Grade : |

**B.1 Tasks given in PART A to be completed here**

*(****Students must write the answers of the task(s) given in the PART A  )***

#Anirbaan Ghatak C026 EB1

# %%

import keras

# %%

import keras

from keras.applications.vgg16 import VGG16, decode\_predictions, preprocess\_input

# Load the pre-trained VGG16 model

model = VGG16(weights='imagenet', include\_top=False)  # Load without the top classification layers

# Get summary of the model's architecture

model.summary()

# Get information about the layers

for layer in model.layers:

    print(f"Layer Name: {layer.name}, Input Shape: {layer.input\_shape}, Output Shape: {layer.output\_shape}")

# %%

from keras.preprocessing.image import load\_img, img\_to\_array

import numpy as np

# Load an image and pre-process it

img\_path = "MCL750s.jpg"  # Replace with your image path

img = load\_img(img\_path, target\_size=(224, 224))  # Resize to match VGG16 input size

x = img\_to\_array(img)

x = np.expand\_dims(x, axis=0)  # Add batch dimension

x = preprocess\_input(x)

# Predict the class using VGG16 (requires loading top layers)

model = VGG16(weights='imagenet')  # Load with top layers

predictions = model.predict(x)

predicted\_class = decode\_predictions(predictions, top=4)[0][0]

print(f"Predicted class: {predicted\_class}")

# %%

# Freeze convolutional base layers to prevent them from being updated during training

for layer in model.layers[:15]:

    layer.trainable = False

# Use the model as a feature extractor

feature\_output = model.predict(x)  # Extract features from the convolutional base

# Use the extracted features for downstream tasks (e.g., classification, clustering)

# ...

# %%

from keras.layers import Dense, Flatten

# Add new layers on top of the frozen base

num\_classes = 1000  # Adjust based on your classification task

new\_model = keras.Sequential([

    model,

    Flatten(),

    Dense(256, activation='relu'),

    Dense(num\_classes, activation='softmax')

])

# Print the total number of parameters

print(f"Total number of parameters: {new\_model.count\_params()}")

# %%

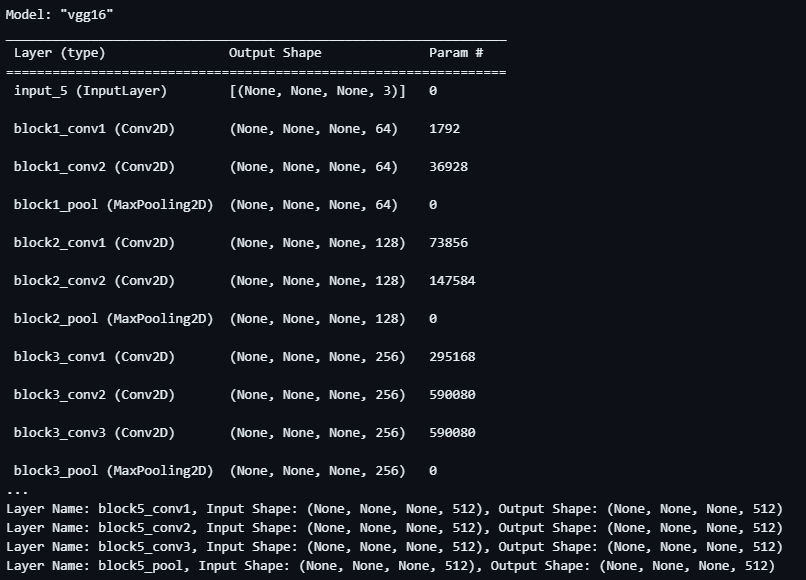
# Freeze only the convolutional base layers, allowing the rest to be trained

for layer in new\_model.layers[:15]:

    layer.trainable = False

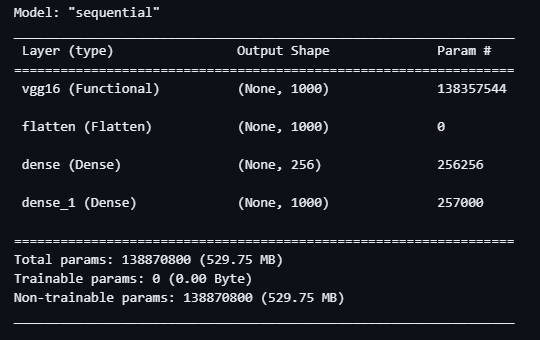
# %%

new\_model.summary()

******

******

******

******

**B.2 Observations and Learning:**

*(****Students must write the observations and learning based on their understanding built about the subject matter and inferences drawn***)

This code explores the power of pre-trained models like VGG16 for image classification. It leverages transfer learning, adding a custom classifier on top of the pre-trained base for specific tasks. Feature extraction and fine-tuning demonstrate the model's versatility beyond simple predictions. By freezing the base, the code prevents overfitting and utilizes parameters efficiently, making it ideal for tasks with limited data. Overall, this showcases how pre-trained models can be powerful tools in deep learning, achieving better results with less effort.

**B.3 Conclusion:**

*(****Students must write the conclusive statements as per the attainment of individual outcomes listed above and learning/observation noted in section B.2)***

This code demonstrates the effectiveness of pre-trained models like VGG16 for image classification. It employs transfer learning by adding a custom classifier on top of the pre-trained base, adapting the model for specific tasks. Feature extraction and fine-tuning capabilities highlight the model's flexibility beyond basic predictions. Freezing the pre-trained layers prevents overfitting and improves parameter efficiency, making it suitable for tasks with limited data. Overall, this code showcases the potential of pre-trained models as powerful tools in deep learning, enabling improved performance with reduced effort.