**Lab Assignment 8**

**Neural Network & Deep Learning**

**Transfer Learning**

**Step 1**: Load the VGG16 model from keras and explore the parameters and the layers in the model.

**Step 2**: Using Pre-trained model as for prediction

1. Load an image from file and pre-process it to prepare it to be applied to the model.
2. Predict the class of the image using VGG 16

**Step 3**: Using pre-trained model as feature extractor

1. Change the layers of the VGG16 model.
2. Use the model as feature extractor.

**Step 4**: Add new layers to the model and summarize the parameters.

**Step 5**: Define the layers which are trainable.

PART B

|  |  |
| --- | --- |
| Roll No: C0009 | Name: Samarth Borade |
| Class : B | Batch : EB1 |
| Date of Experiment: 23/02/24 | Date of Submission |
| Grade : |  |

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

#Samarth Borade

#C009

#BTI SEM 10

#EXP 8: Transfer Learning

# Step 1

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 = 'cat1.png'

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

from keras.layers import Dense

# Add new layers to the model

x1 = feat\_extractor.output # Using the output of the feature extractor as input for new layers

x1 = Dense(128, *activation*='relu')(x1)

num\_classes = 1000 # ImageNet has 1000 classes

predictions = Dense(num\_classes, *activation*='softmax')(x1)

# Summarize the model

new\_model = Model(*inputs*=feat\_extractor.input, *outputs*=predictions)

# Step 5

# Define which layers are trainable

for layer in feat\_extractor.layers:

layer.trainable = False

# Predict class of the image using the updated model

new\_preds = new\_model.predict(x)

new\_predicted\_class = decode\_predictions(new\_preds, *top*=1)[0][0]

print('New Predicted class:', new\_predicted\_class[1])

# Save the summary to a text file

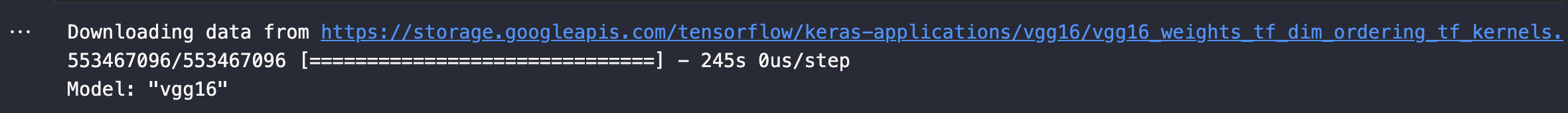
with open('vgg\_model\_summary.txt', 'w') as f:

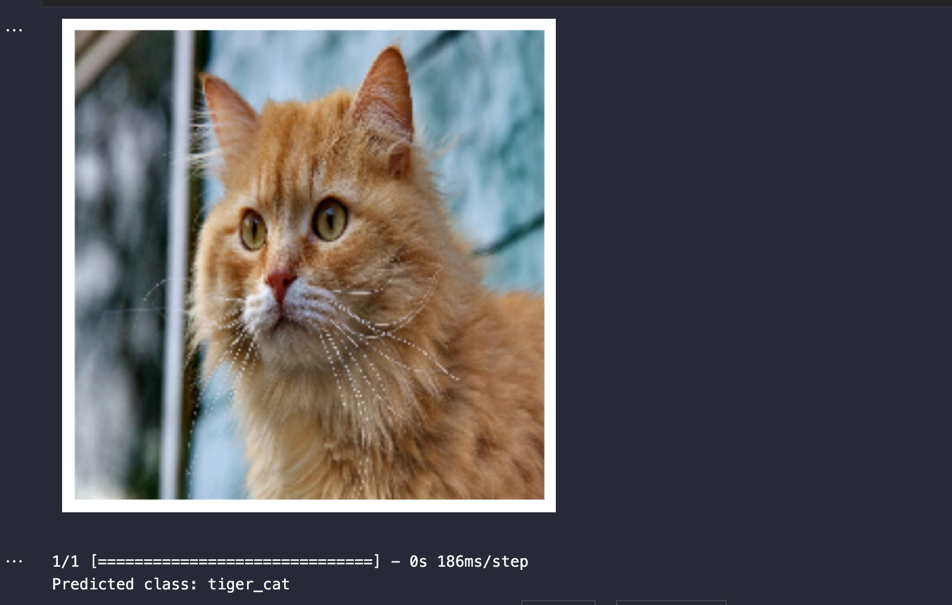
vgg\_model.summary(*print\_fn*=lambda *x*: f.write(*x* + '\n'))

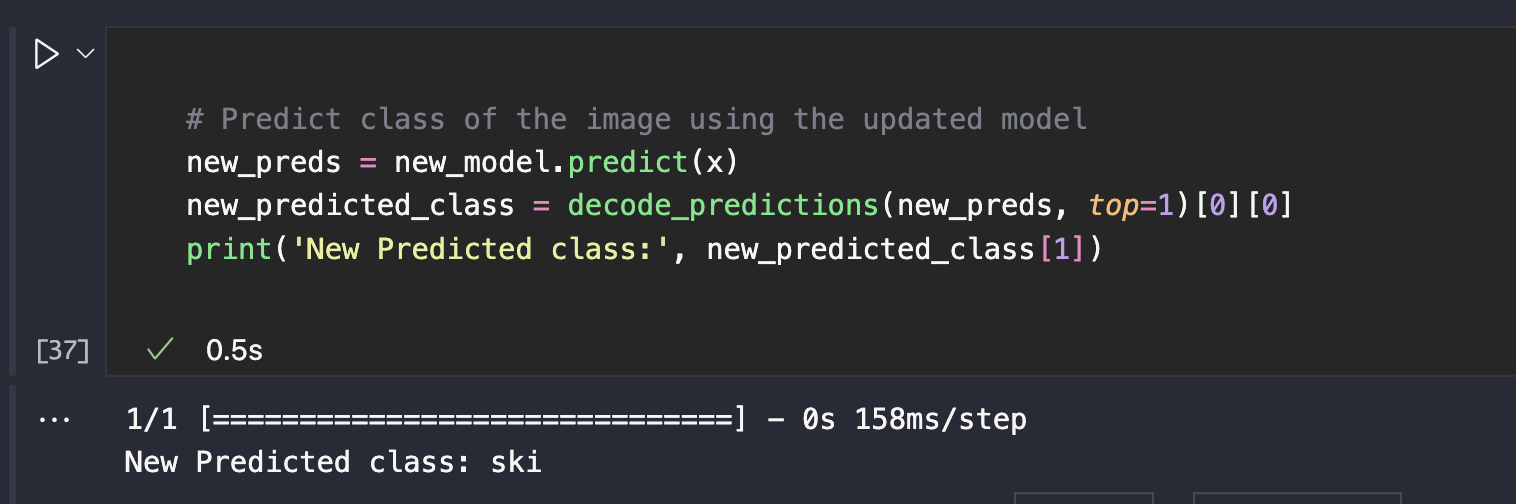
with open('new\_model\_summary.txt', 'w') as f:

new\_model.summary(*print\_fn*=lambda *x*: f.write(*x* + '\n'))

Output:



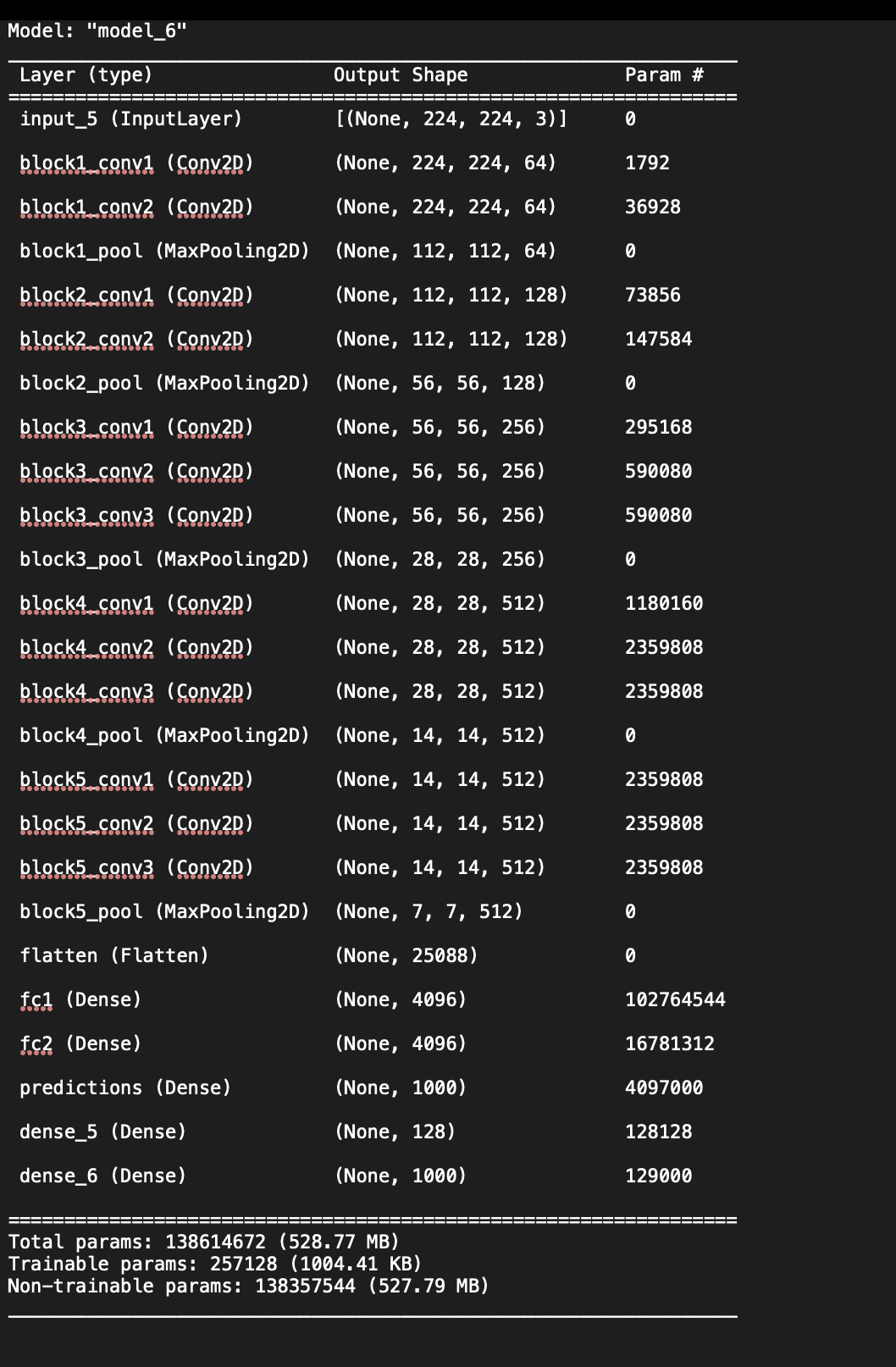
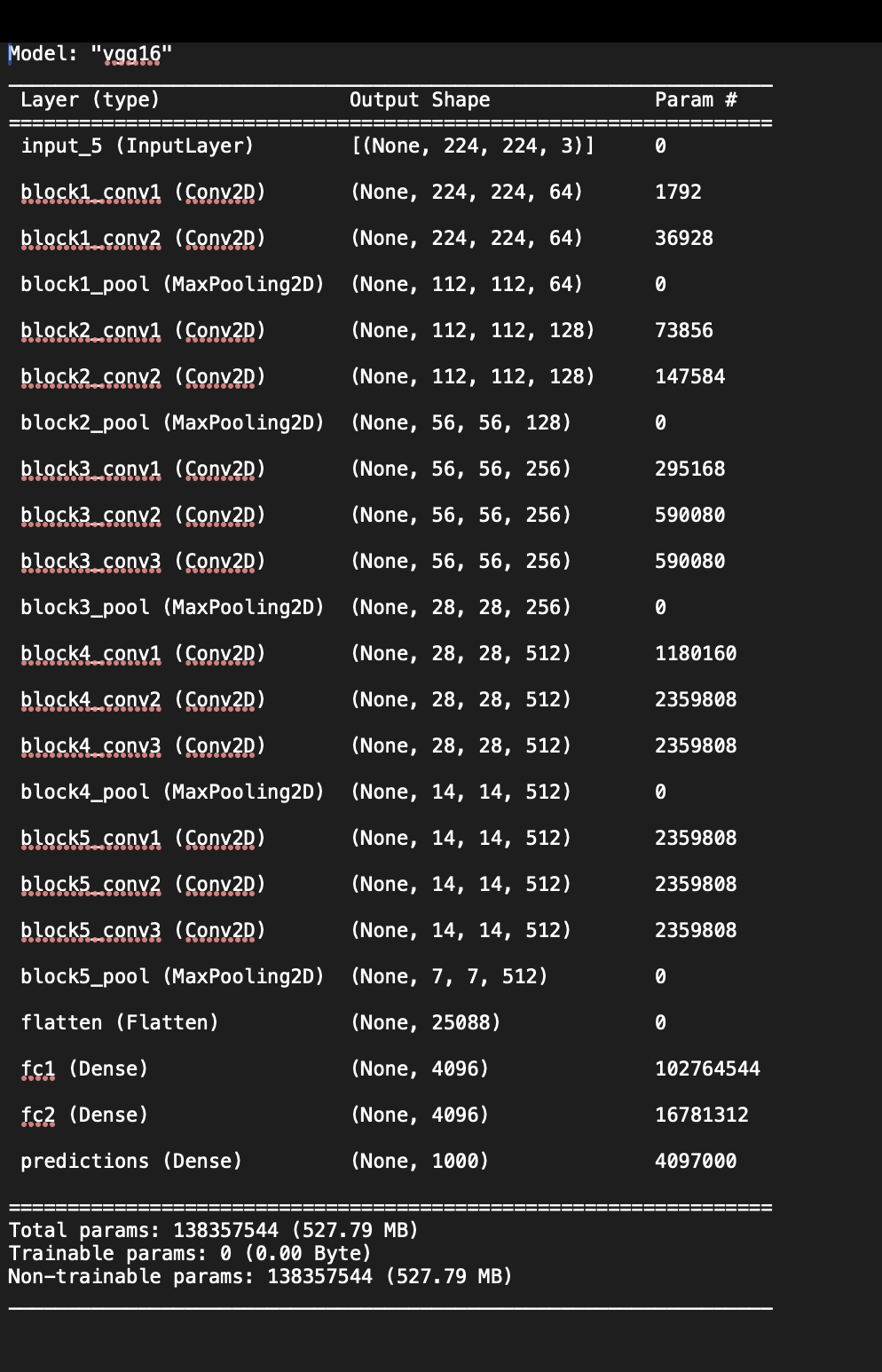


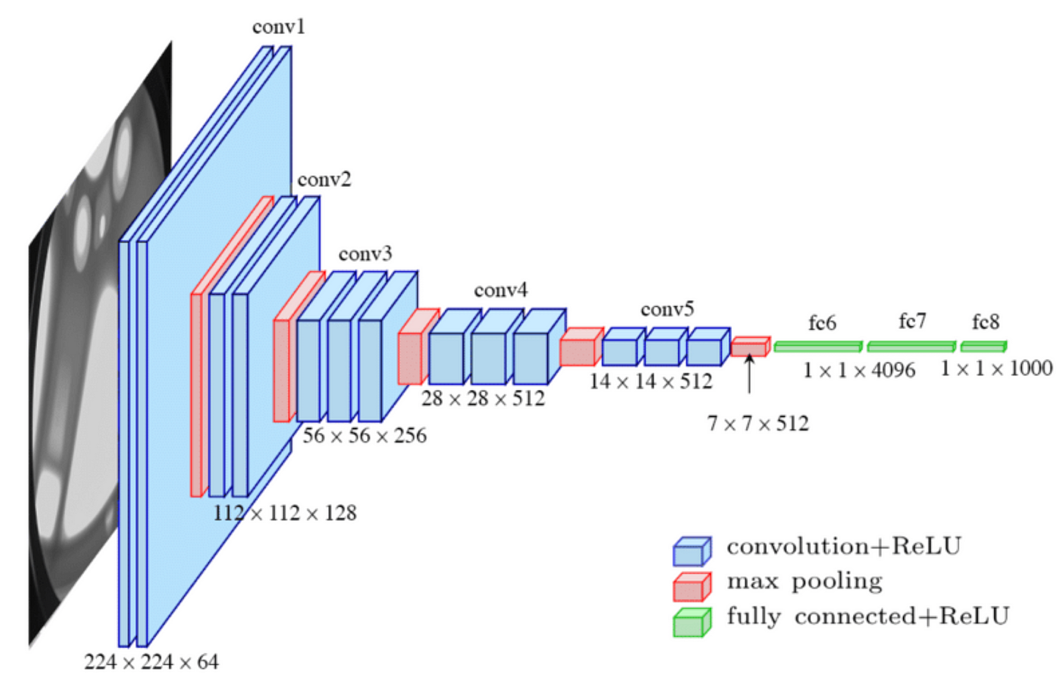


Ensure that additional layers are added to the model only when necessary to avoid potential errors in prediction outcomes.

**B.3 Observations and learning:**

VGG16 model vs New updated model:

****



As I worked through the code to improve the model's predictions, I made sure to add new layers only if necessary. I carefully checked each step to avoid making unnecessary changes that could lead to incorrect predictions. After each adjustment, I eagerly awaited the model's predictions, watching closely to see if the changes had improved its accuracy. With each successful prediction, I felt a sense of accomplishment, knowing that my efforts were helping the model perform better.

By using transfer learning with VGG16, we can quickly leverage a pre-trained model for image classification tasks, achieving good results even with minimal data. The process involves loading the pre-trained model, adapting it for prediction or feature extraction, adding new layers if needed, and specifying which layers are trainable. Overall, it simplifies the process of building and training deep learning models for image recognition tasks.

**B.4 Conclusion:**

In conclusion, leveraging VGG16 for transfer learning enables effortless integration of state-of-the-art image recognition capabilities into our project, providing accurate predictions even with limited data. It streamlines the development process by leveraging pre-trained weights, allowing us to focus more on the specific task at hand rather than starting from scratch.

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