ASSIGNMENT

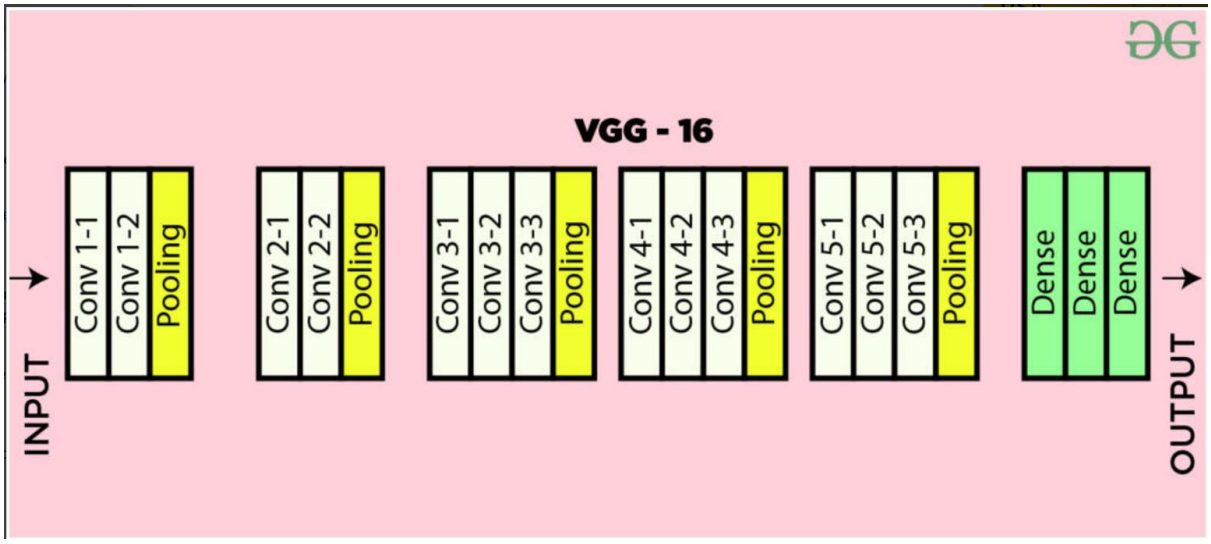
PROBLEM STATEMENT: Train and deploy a VGG16 Model Using CIFAR-10 Dataset

Link: https://github.com/NiharikaAmritkar/VGG16

SOLUTION:

PART 1: MODEL DEVELOPMENT IN PYTHON

Implemented the VGG16 model in TensorFlow framework.



Methodology:

- Implemented necessary libraries:

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import tensorflow as tf
```

[2]:
```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D
from keras.layers import Dropout, Flatten, BatchNormalization
from keras.regularizers import l2
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping
from keras.models import load_model
```

+ Code + Markdown

[3]:
```
#importing the dataset
from tensorflow.keras.datasets import cifar10
```

- Train -test and validation split of the dataset:

[4]:
```
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 3s 0us/step
```

[5]:
```
x_train.shape, x_test.shape, y_train.shape, y_test.shape
```

[5]:
```
((50000, 32, 32, 3), (10000, 32, 32, 3), (50000, 1), (10000, 1))
```

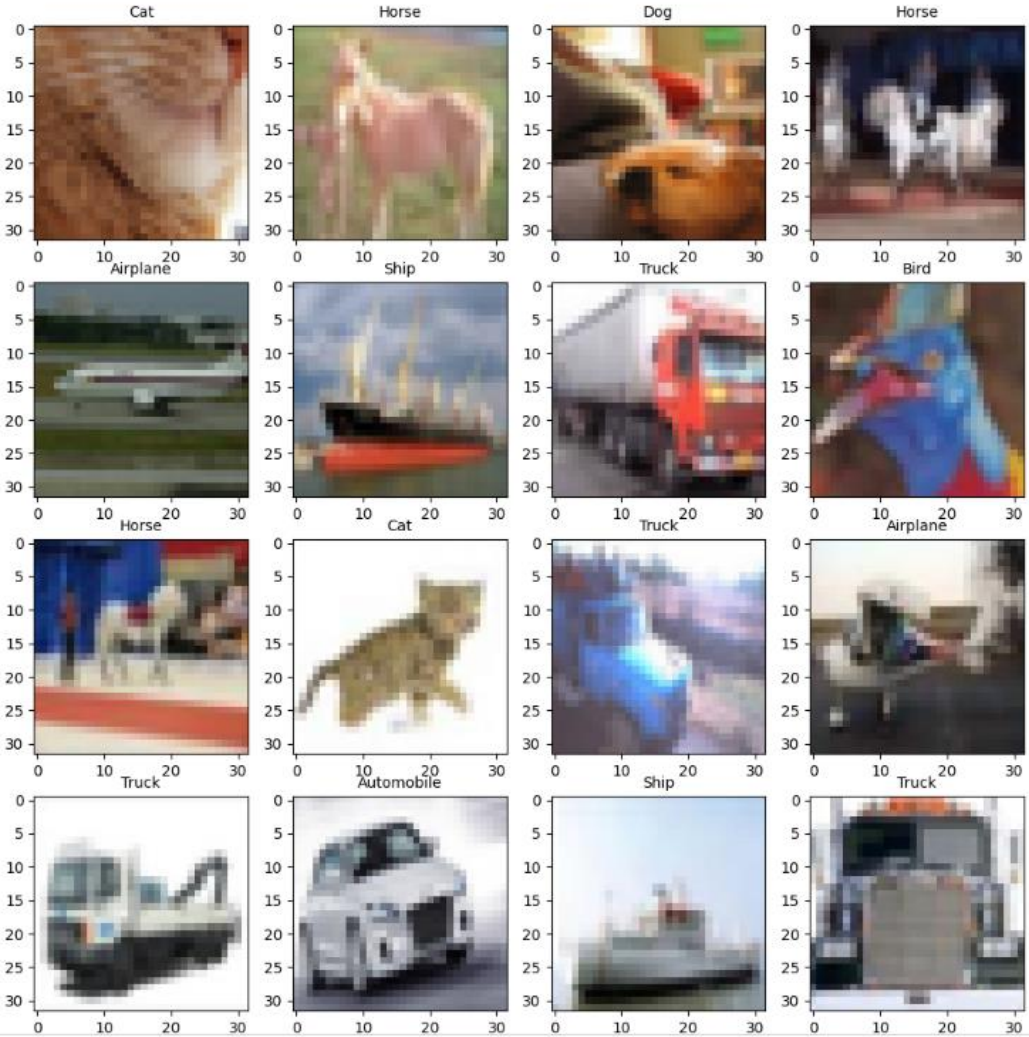
[6]:
```
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val =  train_test_split(x_train, y_train, test_size=0.2, random_state=0)
```

[7]:
```
x_train.shape, x_val.shape, y_train.shape, y_val.shape
```

[7]:
```
((40000, 32, 32, 3), (10000, 32, 32, 3), (40000, 1), (10000, 1))
```

- Displaying the dataset:

[8]:
```
classes=["Airplane", "Automobile", "Bird", "Cat", "Deer", "Dog", "Frog", "Horse", "Ship", "Truck"]

plt.figure(figsize=(12, 12))
for i in range(16):
    plt.subplot(4, 4, i + 1)
    plt.imshow(x_train[i])
    plt.title(classes[y_train[i][0]], fontsize= 10)
plt.show()
```



- Conversion of labels to integers:

```
y_train = tf.keras.utils.to_categorical(y_train, num_classes=10)
y_val = tf.keras.utils.to_categorical(y_val, num_classes=10)
y_test= tf.keras.utils.to_categorical(y_test, num_classes=10)
```

- Data preprocessing:

11]:
```
print(mean, std)
```

```
120.70063406575521 64.15108741792801
```

12]:
```
x_train = (x_train-mean)/(std+1e-7)
x_test  = (x_test-mean) /(std+1e-7)
x_val = (x_val-mean)/(std+1e-7)
```

Data augmentation:

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
data_generator = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.15,
    height_shift_range=0.15,
    horizontal_flip=True,
    vertical_flip= True,
    zoom_range=0.1,

)
```

- Model building:

```
model= Sequential()
# CV1 1 layer
model.add(Conv2D(16, (3,3), padding='same', activation='relu', input_shape=(32,32,3)))
model.add(BatchNormalization())
#2 layer
model.add(Conv2D(16, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#3 layer
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))
#CV2 4 layer
model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#5 layer
model.add(Conv2D(32, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#6 layer
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.3))
#CV3 7 layer
model.add(Conv2D(64, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#8 layer
model.add(Conv2D(64, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#9 layer
model.add(Conv2D(64, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#10 layer
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.4))
#CV4 11 layer
model.add(Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#12 layer
model.add(Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#13 layer
model.add(Conv2D(128, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#14 layer
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.5))
#CV5 15 layer
model.add(Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#16 layer
model.add(Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#17 layer
model.add(Conv2D(256, (3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
#18 layer
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.6))
#F 19 layer
model.add(Flatten())
#D1 20 layer
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.7))
#D2 21 layer
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.8))
#D3 22 layer
model.add(Dense(10, activation='softmax'))
```

```
model1.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 16, 32, 32)	448
batch_normalization (BatchNormalization)	(None, 16, 32, 32)	128
conv2d_1 (Conv2D)	(None, 16, 32, 32)	2,320
batch_normalization_1 (BatchNormalization)	(None, 16, 32, 32)	128
max_pooling2d (MaxPooling2D)	(None, 16, 16, 16)	0
dropout (Dropout)	(None, 16, 16, 16)	0
conv2d_2 (Conv2D)	(None, 32, 16, 16)	4,640
batch_normalization_2 (BatchNormalization)	(None, 32, 16, 16)	64
conv2d_3 (Conv2D)	(None, 32, 16, 16)	9,248
batch_normalization_3 (BatchNormalization)	(None, 32, 16, 16)	64
max_pooling2d_1 (MaxPooling2D)	(None, 32, 8, 8)	0
dropout_1 (Dropout)	(None, 32, 8, 8)	0
conv2d_4 (Conv2D)	(None, 64, 8, 8)	18,496
batch_normalization_4 (BatchNormalization)	(None, 64, 8, 8)	32
conv2d_5 (Conv2D)	(None, 64, 8, 8)	36,928
batch_normalization_5 (BatchNormalization)	(None, 64, 8, 8)	32
conv2d_6 (Conv2D)	(None, 64, 8, 8)	36,928
batch_normalization_6 (BatchNormalization)	(None, 64, 8, 8)	32
max_pooling2d_2 (MaxPooling2D)	(None, 64, 4, 4)	0
dropout_2 (Dropout)	(None, 64, 4, 4)	0
conv2d_7 (Conv2D)	(None, 128, 4, 4)	73,856
batch_normalization_7 (BatchNormalization)	(None, 128, 4, 4)	16
conv2d_8 (Conv2D)	(None, 128, 4, 4)	147,584
batch_normalization_8 (BatchNormalization)	(None, 128, 4, 4)	16
conv2d_9 (Conv2D)	(None, 128, 4, 4)	147,584
batch_normalization_9 (BatchNormalization)	(None, 128, 4, 4)	16
max_pooling2d_3 (MaxPooling2D)	(None, 128, 2, 2)	0
dropout_3 (Dropout)	(None, 128, 2, 2)	0
conv2d_10 (Conv2D)	(None, 256, 2, 2)	295,168
batch_normalization_10 (BatchNormalization)	(None, 256, 2, 2)	8
conv2d_11 (Conv2D)	(None, 256, 2, 2)	590,080
batch_normalization_11 (BatchNormalization)	(None, 256, 2, 2)	8
conv2d_12 (Conv2D)	(None, 256, 2, 2)	590,080
batch_normalization_12 (BatchNormalization)	(None, 256, 2, 2)	8
max_pooling2d_4 (MaxPooling2D)	(None, 256, 1, 1)	0
dropout_4 (Dropout)	(None, 256, 1, 1)	0
flatten (Flatten)	(None, 256)	0
dense (Dense)	(None, 128)	32,896
batch_normalization_13 (BatchNormalization)	(None, 128)	512
dropout_5 (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 128)	16,512
batch_normalization_14 (BatchNormalization)	(None, 128)	512
dropout_6 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1,290

Total params: 2,085,634 (7.65 MB)

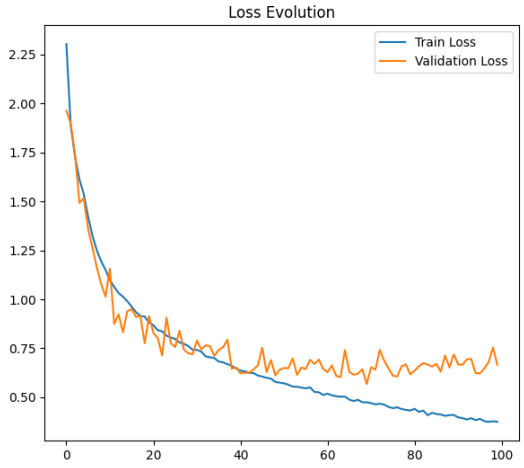
Trainable params: 2,084,846 (7.65 MB)

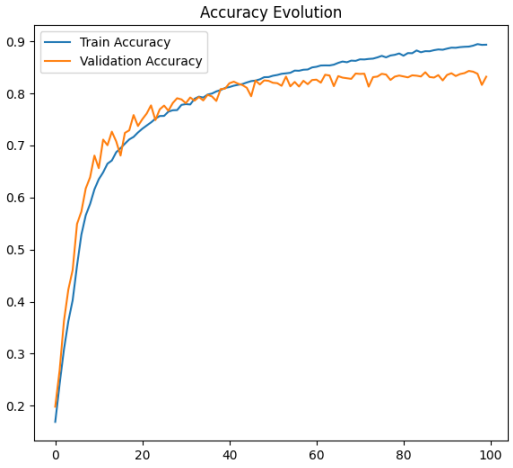
[19]:
```
optimizer= Adam(learning_rate=0.01, epsilon=1e-07)
model1.compile(optimizer= optimizer, loss='categorical_crossentropy', metrics=['accuracy'])
```

[20]:
```
early_stopping= EarlyStopping(monitor='val_loss', patience=30, verbose=1)
model1.fit(x_train, y_train, batch_size=128, epochs=200, validation_data=(x_val, y_val), callbacks=[early_stopping])
```

```
Epoch 1/200
```

- Result evaluation:





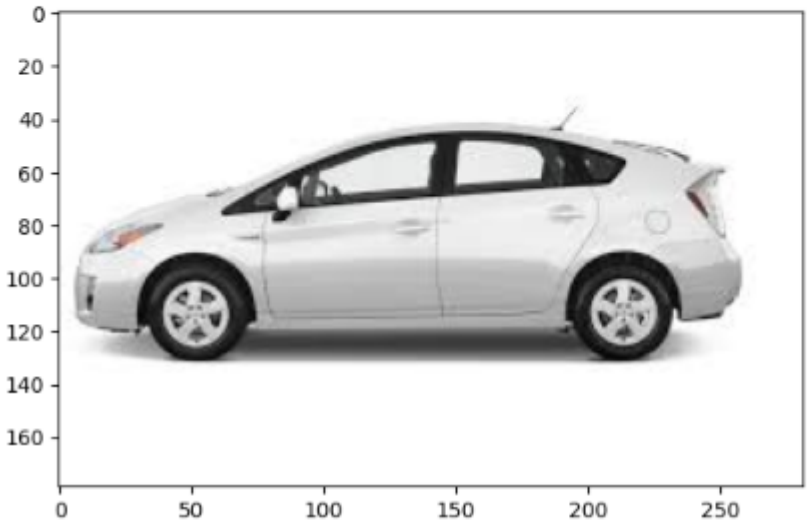

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=1)

print('\nTest Accuracy:', test_acc)
print('Test Loss:    ', test_loss)
```

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.8300 - loss: 0.6896

Test Accuracy: 0.8306999802589417
Test Loss:     0.6906129121780396
```

- Predicting results:

[29]:
```
image3_path= '/kaggle/input/testing-images/d1.jpg'
image3= cv2.imread(image3_path)
image3 = cv2.cvtColor(image3, cv2.COLOR_BGR2RGB)
plt.imshow(image3)
```

[29]:
```
<matplotlib.image.AxesImage at 0x7b3d7b2ea170>
```



[30]:
```
image3 = cv2.resize(image3, (32,32))
image3 = (image3-mean)/(std+1e-7)
image3 = image3.reshape((1, 32, 32, 3))
```

[31]:
```
prediction3 = model.predict(image3)
```

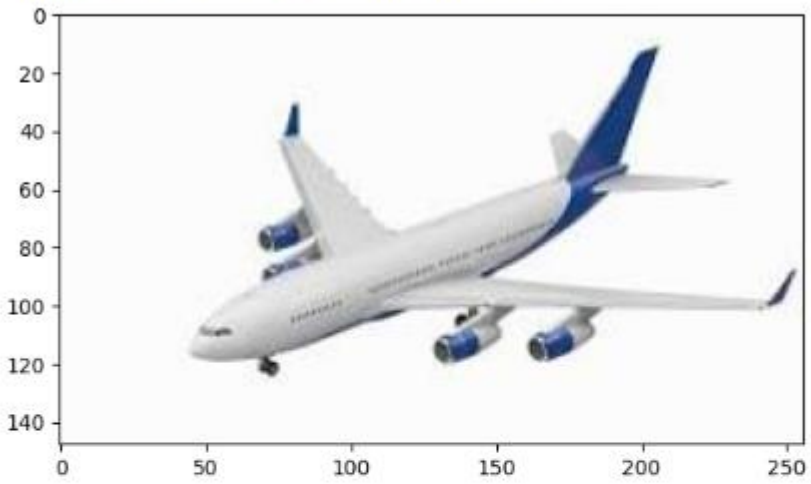
```
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 782ms/step
```

[33]:
```
predicted_class3 = prediction3.argmax()
print('Predicted class: ', classes[predicted_class3])
```

```
Predicted class:  Automobile
```

[38]:
```
image2_path= '/kaggle/input/testing-images/a1.jpg'
image2= cv2.imread(image2_path)
image2 = cv2.cvtColor(image2, cv2.COLOR_BGR2RGB)
plt.imshow(image2)
```

[38]:
```
<matplotlib.image.AxesImage at 0x7b3d587c1a20>
```



+ Code + Markdown

[39]:
```
image2 = cv2.resize(image2, (32,32))
image2 = (image2-mean)/(std+1e-7)
image2 = image2.reshape((1, 32, 32, 3))
```

[40]:
```
prediction2 = model.predict(image2)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 17ms/step
```

[41]:
```
predicted_class2 = prediction2.argmax()

print('Predicted class: ', classes[predicted_class2])
```

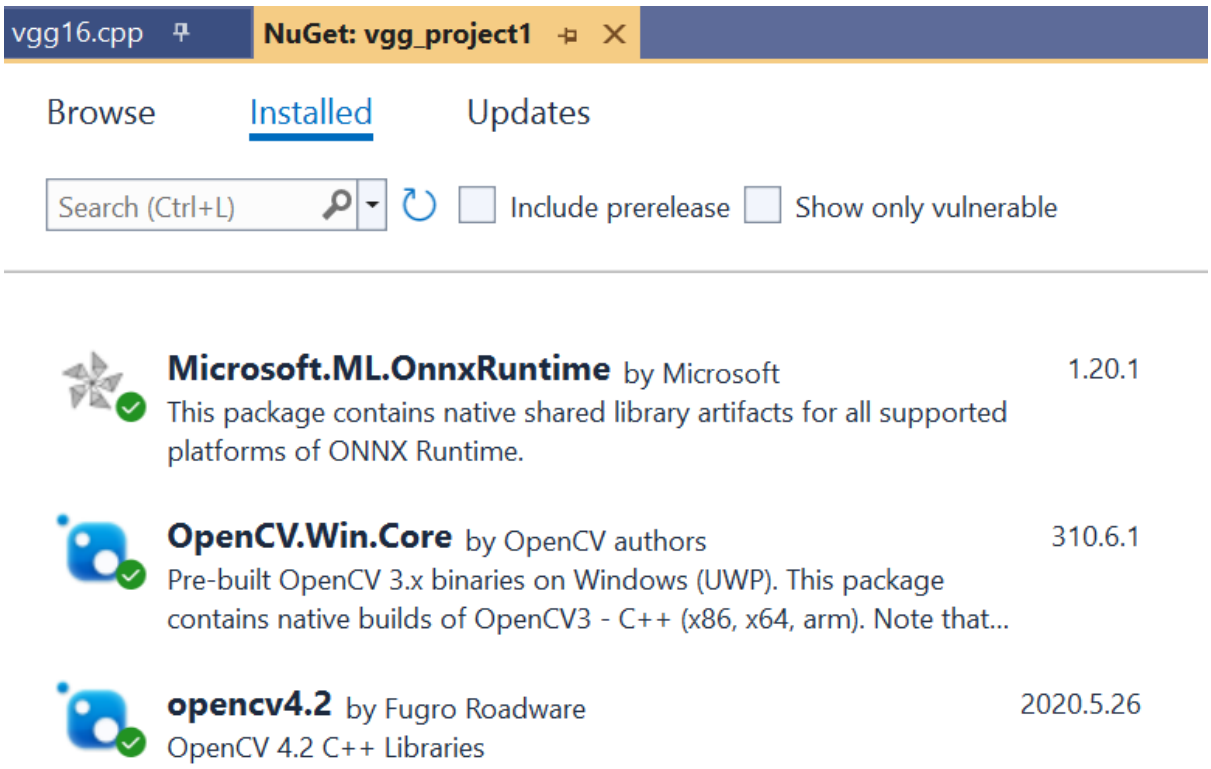
```
Predicted class:  Airplane
```

- Model to ONNX format:

```
[49]: model.save("vgg16.h5")

+ Code  + Markdown

[64]: model = tf.keras.models.load_model("vgg16.h5", compile=False)

[65]: input_signature= [tf.TensorSpec([None, 32, 32, 3], tf.float32)]

[69]: model.output_names = ['output']

onnx_model,_ = tf2onnx.convert.from_keras(model, input_signature=input_signature, opset=13)
onnx.save(onnx_model, 'vgg16.onnx')

WARNING: AutoGraph could not transform <function trace_model_call.<locals>._wrapped_model at 0x7b42dbae2c20> and will run it as-is.
Please report this to the TensorFlow team. When filing the bug, set the verbosity to 10 (on Linux, `export AUTOGRAPH_VERBOSITY=10`) and attach the full output.
Cause: closure mismatch, requested ('input_signature', 'model'), but source function had ()
To silence this warning, decorate the function with @tf.autograph.experimental.do_not_convert

+ Code  + Markdown
```

TASK 2: MODEL DEPLOYMENT IN C++

- Downloading necessary packages:



- Code implementation:

vgg_project1 (Global Scope)

```
#include <fstream>
#include <sstream>
#include <iostream>


#include <opencv2/dnn.hpp>
#include <opencv2/imgproc.hpp>
#include <opencv2/highgui.hpp>
#include <opencv2/dnn/dnn.hpp>
#include <opencv2/imgcodecs.hpp>


using namespace std;
using namespace cv;
using namespace dnn;


int main() {
    try{

    vector<string> classes;
    ifstream ifs("classes.txt"); // Path to the class names file
    if (!ifs.is_open()) {
        cerr << "Error opening classes.txt" << endl;
        return -1;
    }
    string line;
    while (getline(ifs, line)) {
        classes.push_back(line);
    }
```

```
    string path = "a1.jpg"; //reading and loading the image
    Mat img = imread(path);
    Mat rgbimg;
    cvtColor(img, rgbimg, COLOR_BGR2RGB); // converting image to RGB channel

    Mat resized_img;
    resize(rgbimg, resized_img, Size(32, 32), INTER_LINEAR); //resizing the image into [32,32]

    //image preprocessing:
    float mean = 120.70063406575521;
    float std = 64.15108741792801;
    Mat img_float;
    resized_img.convertTo(img_float, CV_32F, 1.0/255); // Convert to float
    Mat normalized_img = (img_float - mean) / (std + 1e-7f); //normalizing

    int IMG_HEIGHT = 32;
    int IMG_WIDTH = 32;
    int sz[4] = { 1, IMG_HEIGHT, IMG_WIDTH, 3 };
    Mat blob = Mat(4, sz, CV_32F, normalized_img.data); //convert to BHWC format
    //cout << "blob:" << blob.size << endl;

    string modelpath = "vgg16.onnx";
    Net net = readNetFromONNX(modelpath);
    net.setPreferableBackend(DNN_BACKEND_OPENCV);
    net.setPreferableTarget(DNN_TARGET_CPU);
    if (net.empty()) {
        cerr << "Failed to load the ONNX model!" << endl;
        return -1;
    }

    net.setInput(blob); //passing the inputs to model
    Mat outputs = net.forward();

    Point classIdPoint;
    double confidence;
    minMaxLoc(outputs, nullptr, &confidence, nullptr, &classIdPoint);
    int predictedClass = classIdPoint.x;

    string className = (predictedClass < classes.size()) ? classes[predictedClass] : "Unknown";
    cout << "Predicted Class: " << className << endl;
    cout << "Confidence: " << confidence << endl;
    cout << "Output probabilities: " << outputs << endl;

    for (int i = 0; i < classes.size(); ++i) {
        cout << "Class " << i << ": " << classes[i] << endl;
    }


    }

    catch (const std::exception& e) {
        std::cerr << "Error: " << e.what() << endl;
        return -1;
    }
    return 0;
```

RESULTS:

Input1:

Output1:

```
Microsoft Visual Studio Debu
[ INFO:0] global C:\build\master_winpack-build-win64-vc14\opencv\modules\core\src\ocl.cpp (891) cv::ocl::haveOpenCL Init
ialize OpenCL runtime...
Predicted Class: Deer
Confidence: 0.468693
Output probabilities: [0.014634821, 0.00053116266, 0.069748096, 0.30807915, 0.46869293, 0.040760804, 0.061702657, 0.0187
66707, 0.014515476, 0.002568247]
Class 0: Airplane
Class 1: Automobile
Class 2: Bird
Class 3: Cat
Class 4: Deer
Class 5: Dog
Class 6: Frog
Class 7: Horse
Class 8: Ship
Class 9: Truck

C:\Users\Niharika\source\repos\vgg_project1\x64\Debug\vgg_project1.exe (process 22436) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Input2:

Output2:

```
Microsoft Visual Studio Debu
[ INFO:0] global C:\build\master_winpack-build-win64-vc14\opencv\modules\core\src\ocl.cpp (891) cv::ocl::haveOpenCL Init
ialize OpenCL runtime...
Predicted Class: Deer
Confidence: 0.458067
Output probabilities: [0.014541978, 0.00054808543, 0.069807261, 0.31754342, 0.45806658, 0.04119676, 0.062347621, 0.01886
3214, 0.014523101, 0.0025620109]
Class 0: Airplane
Class 1: Automobile
Class 2: Bird
Class 3: Cat
Class 4: Deer
Class 5: Dog
Class 6: Frog
Class 7: Horse
Class 8: Ship
Class 9: Truck

C:\Users\Niharika\source\repos\vgg_project1\x64\Debug\vgg_project1.exe (process 28968) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Input3:

Output3:

```
Microsoft Visual Studio Debu
[ INFO:0] global C:\build\master_winpack-build-win64-vc14\opencv\modules\core\src\ocl.cpp (891) cv::ocl::haveOpenCL Init
ialize OpenCL runtime...
Predicted Class: Deer
Confidence: 0.464313
Output probabilities: [0.014707376, 0.00054616993, 0.069917351, 0.31110942, 0.46431309, 0.040802363, 0.062432058, 0.0187
92156, 0.014770325, 0.0026096099]
Class 0: Airplane
Class 1: Automobile
Class 2: Bird
Class 3: Cat
Class 4: Deer
Class 5: Dog
Class 6: Frog
Class 7: Horse
Class 8: Ship
Class 9: Truck

C:\Users\Niharika\source\repos\vgg_project1\x64\Debug\vgg_project1.exe (process 25280) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

CHALLENGES AND OUTCOMES:

1. The input shape of blob was [1,3,32,32] whereas the model expected input shape of [1,32,32,3]

2. The model after deployment is not accurate in predicting the class of the input image. For any input image, it predicts the same "deer" class but with different confidence or maximum value of SoftMax function probabilities for the respective outputs.
3. The python model accuracy was descent. As from the graph there was no sign of overfitting, but the model could be improved more.

REFERENCES:

1. https://www.geeksforgeeks.org/vgg-16-cnn-model/
2. https://gist.github.com/vietanhdev/eb7ed528ac5ad6f4b36703f0ad5e7558
3. https://docs.opencv.org/4.x/pages.html
4. https://docs.opencv.org/4.x/d0/db7/tutorial_js_table_of_contents_dnn.html
5. https://forum.opencv.org/t/dnn-forward-works-in-python-but-not-c/7937
6. https://stackoverflow.com/questions/4493554/neural-network-always-produces-same-similar-outputs-for-any-input
7. https://www.tensorflow.org/datasets/catalog/cifar10
8. https://keras.io/api/applications/vgg/
9. https://www.youtube.com/watch?v=N5t78V0s6Go&t=274s
10. https://www.youtube.com/watch?v=j5YwP292YRg&t=714s