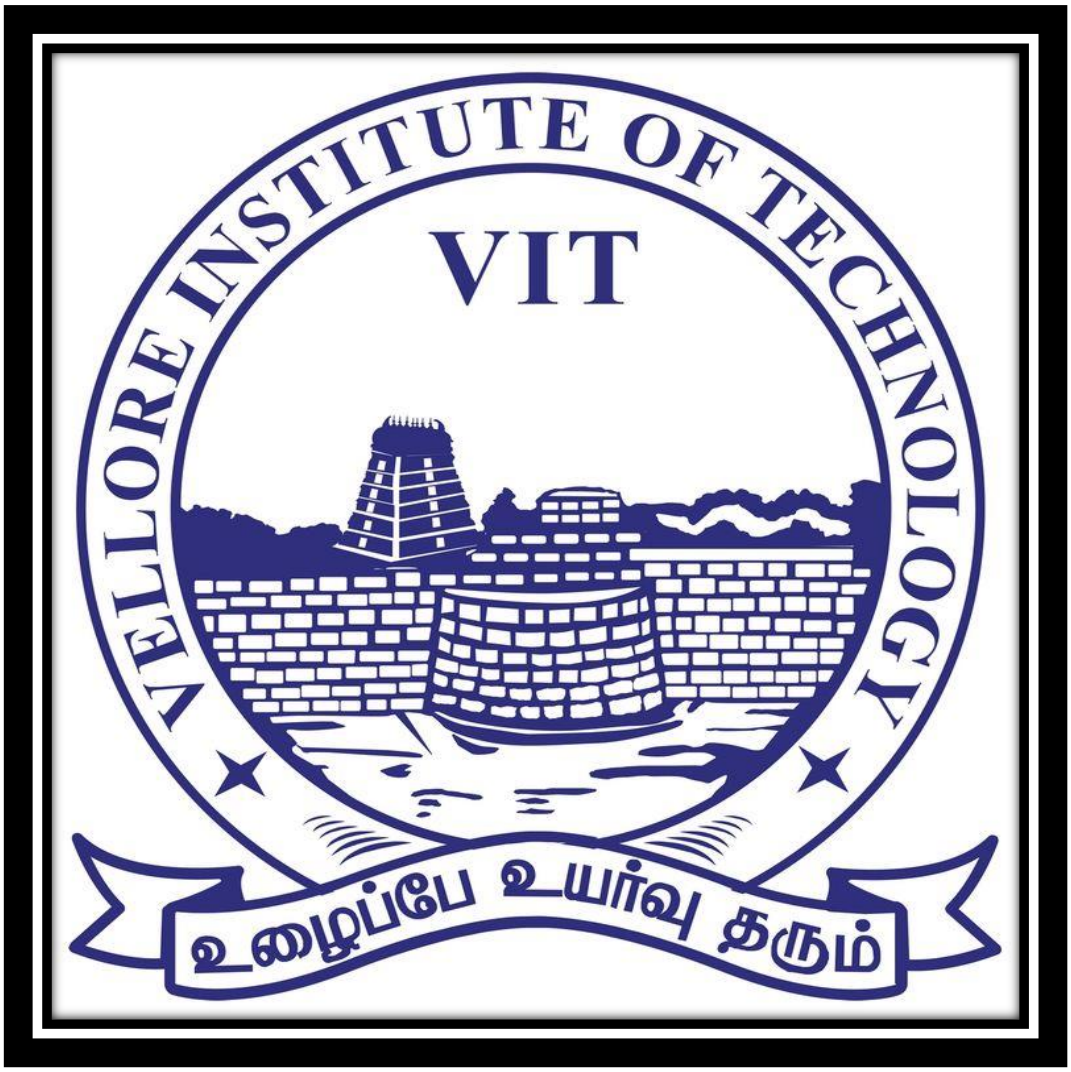
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



VELLORE INSTITUTE OF TECHNOLOGY

VIT – CHENNAI CAMPUS

DEEP LEARNING LAB

LAB ASSIGNMENT – 5

FACULTY : DR. HARINI. S

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

BCSE332P – DL Lab

Lab 5 – DL Lab Assignment

INTRODUCTION TO PYTORCH & GPU

1. Executing the code.

CODE:

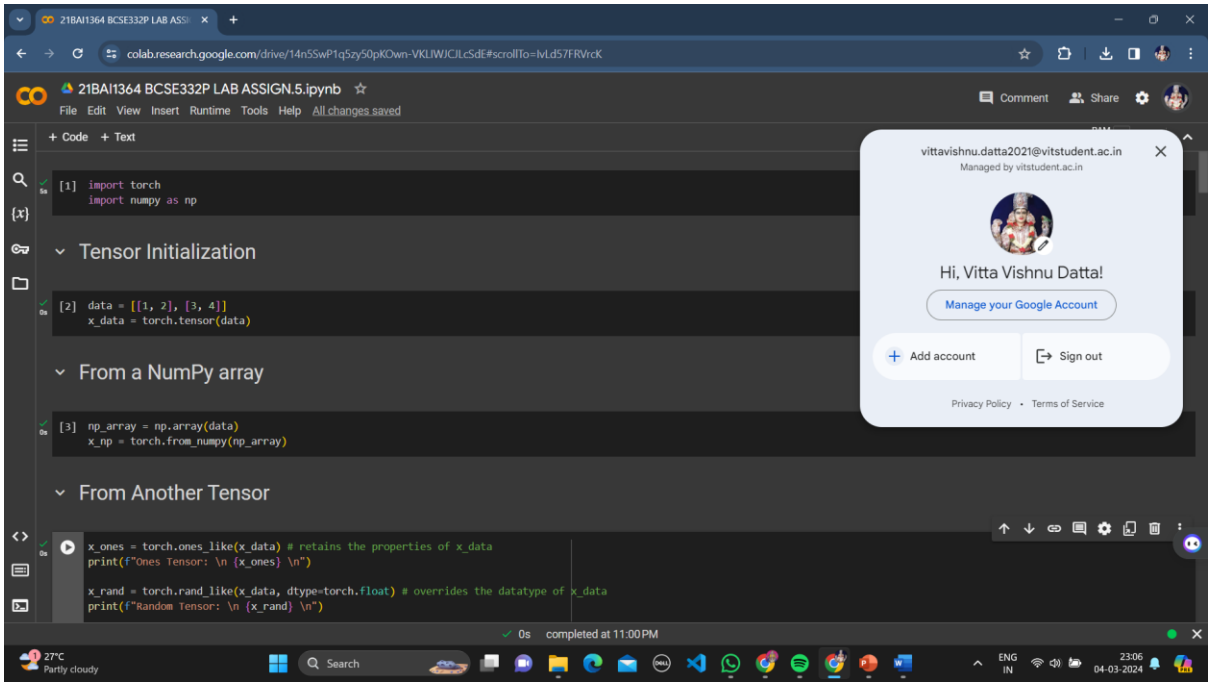
```
import torch
import numpy as np
data = [[1, 2], [3, 4]]
x_data = torch.tensor(data)
np_array = np.array(data)
x_np = torch.from_numpy(np_array)
x_ones = torch.ones_like(x_data) # retains the properties of x_data
print(f"Ones Tensor: \n {x_ones} \n")

x_rand = torch.rand_like(x_data, dtype=torch.float) # overrides the
datatype of x_data
print(f"Random Tensor: \n {x_rand} \n")
shape = (2, 3,)
rand_tensor = torch.rand(shape)
ones_tensor = torch.ones(shape)
zeros_tensor = torch.zeros(shape)

print(f"Random Tensor: \n {rand_tensor} \n")
print(f"Ones Tensor: \n {ones_tensor} \n")
print(f"Zeros Tensor: \n {zeros_tensor}")
tensor = torch.rand(3, 4)

print(f"Shape of tensor: {tensor.shape}")
print(f"Datatype of tensor: {tensor.dtype}")
print(f"Device tensor is stored on: {tensor.device}")
if torch.cuda.is_available():
  tensor = tensor.to('cuda')
```

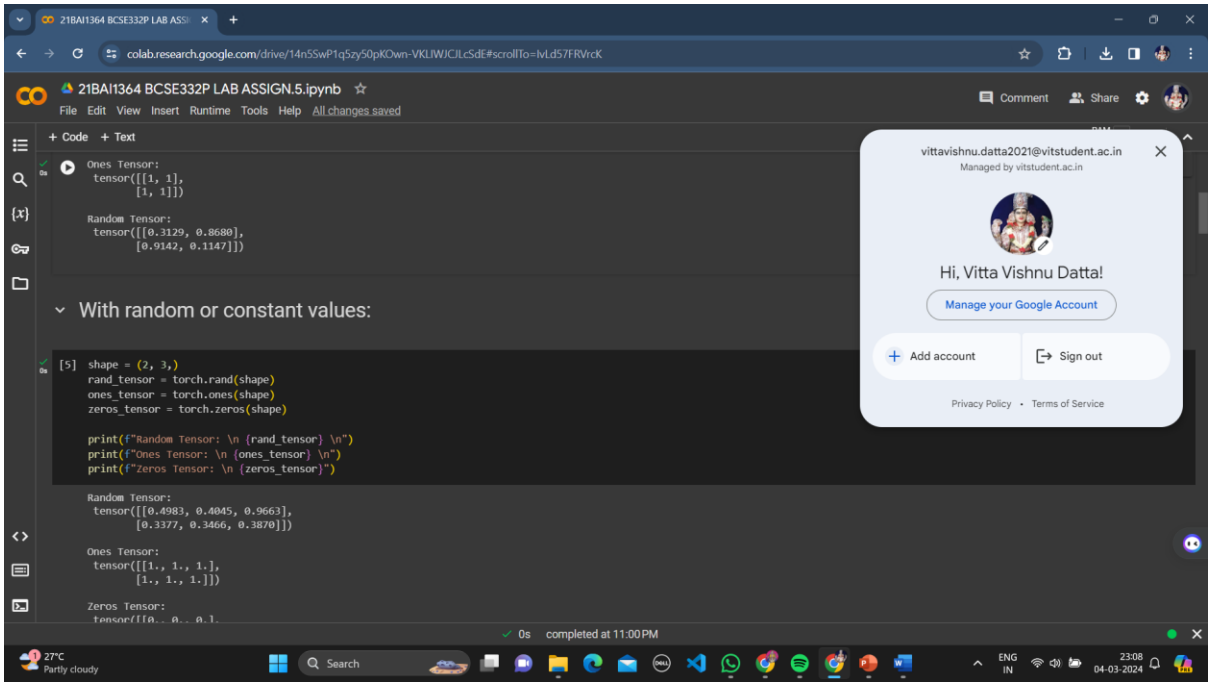
```
  print(f"Device tensor is stored on: {tensor.device}")
tensor = torch.ones(4, 4)
tensor[:,1] = 0
print(tensor)
t1 = torch.cat([tensor, tensor, tensor], dim=1)
print(t1)
# This computes the element-wise product
print(f"tensor.mul(tensor) \n {tensor.mul(tensor)} \n")
# Alternative syntax:
print(f"tensor * tensor \n {tensor * tensor}")
params = list(net.parameters())
print(len(params))
print(params[0].size())  # conv1's .weight
input = torch.randn(1, 1, 32, 32)
out = net(input)
print(out)
import torch
import torchvision
import torchvision.transforms as transforms
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True,
transform=transform)
trainloader = torch.utils.data.DataLoader(trainset,
batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True,
transform=transform)
testloader = torch.utils.data.DataLoader(testset,
batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
import matplotlib.pyplot as plt
```

```
import numpy as np

# functions to show an image


def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
# Alternative syntax:
print(f"tensor @ tensor.T \n {tensor @ tensor.T}")
print(tensor, "\n")
tensor.add_(5)
print(tensor)
t = torch.ones(5)
print(f"t: {t}")
n = t.numpy()
print(f"n: {n}")
t.add_(1)
print(f"t: {t}")
print(f"n: {n}")
n = np.ones(5)
t = torch.from_numpy(n)
np.add(n, 1, out=n)
print(f"t: {t}")
print(f"n: {n}")
import torch
from torchvision.models import resnet18, ResNet18_Weights
model = resnet18(weights=ResNet18_Weights.DEFAULT)
data = torch.rand(1, 3, 64, 64)
labels = torch.rand(1, 1000)
prediction = model(data) # forward pass
```

```
loss = (prediction - labels).sum()
loss.backward() # backward pass
optim = torch.optim.SGD(model.parameters(), lr=1e-2, momentum=0.9)
optim.step() #gradient descent
import torch
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):

    def __init__(self):
        super(Net, self).__init__()
        # 1 input image channel, 6 output channels, 5x5 square
convolution
        # kernel
        self.conv1 = nn.Conv2d(1, 6, 5)
        self.conv2 = nn.Conv2d(6, 16, 5)
        # an affine operation: y = Wx + b
        self.fc1 = nn.Linear(16 * 5 * 5, 120)  # 5*5 from image
dimension
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        # Max pooling over a (2, 2) window
        x = F.max_pool2d(F.relu(self.conv1(x)), (2, 2))
        # If the size is a square, you can specify with a single number
        x = F.max_pool2d(F.relu(self.conv2(x)), 2)
        x = torch.flatten(x, 1) # flatten all dimensions except the
batch dimension
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
print(net)
```

SCREENSHOTS OF THE CODE SNIPPET:



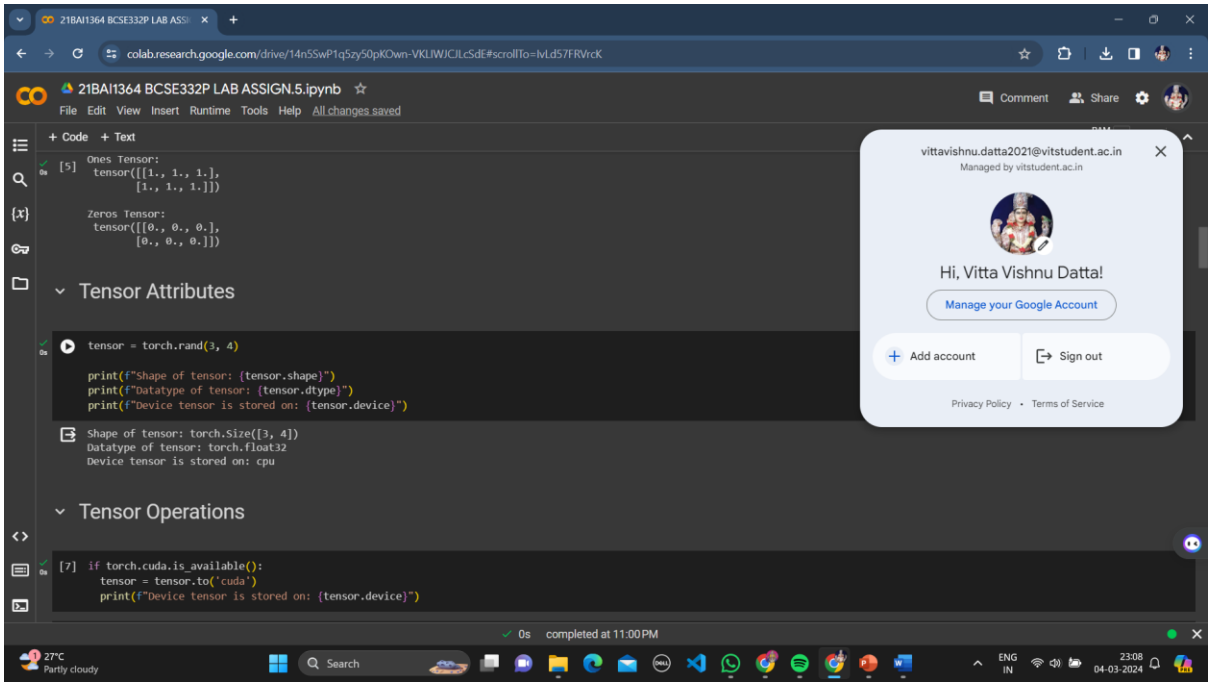


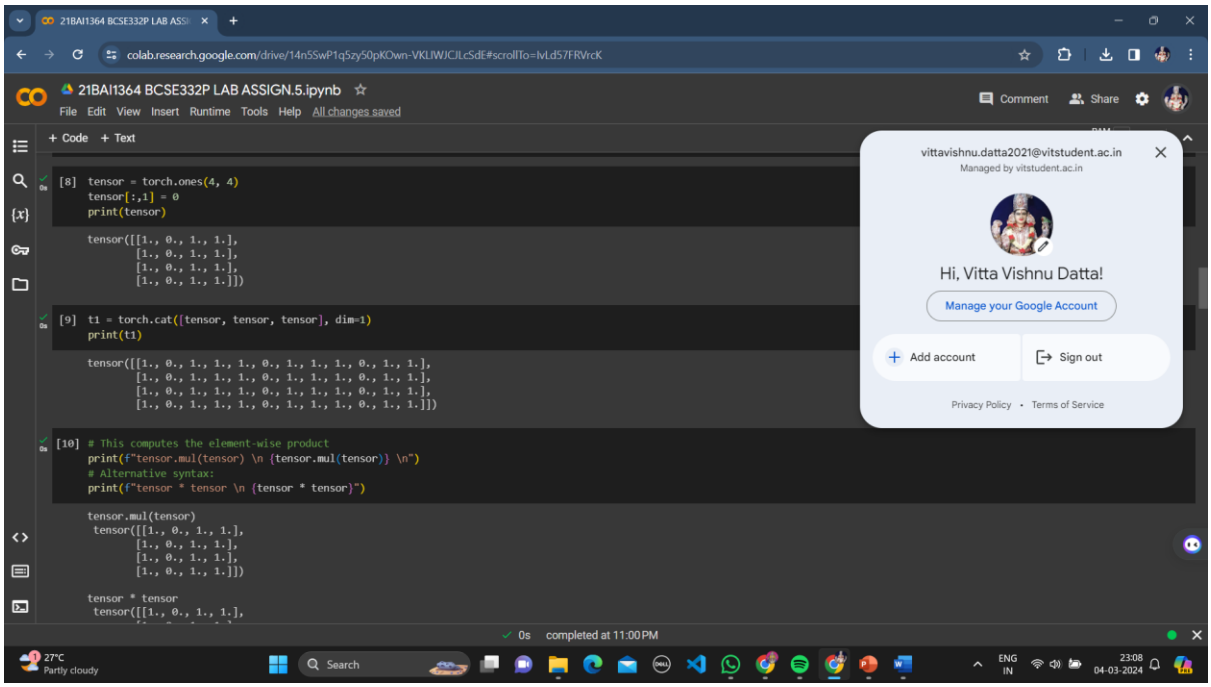
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



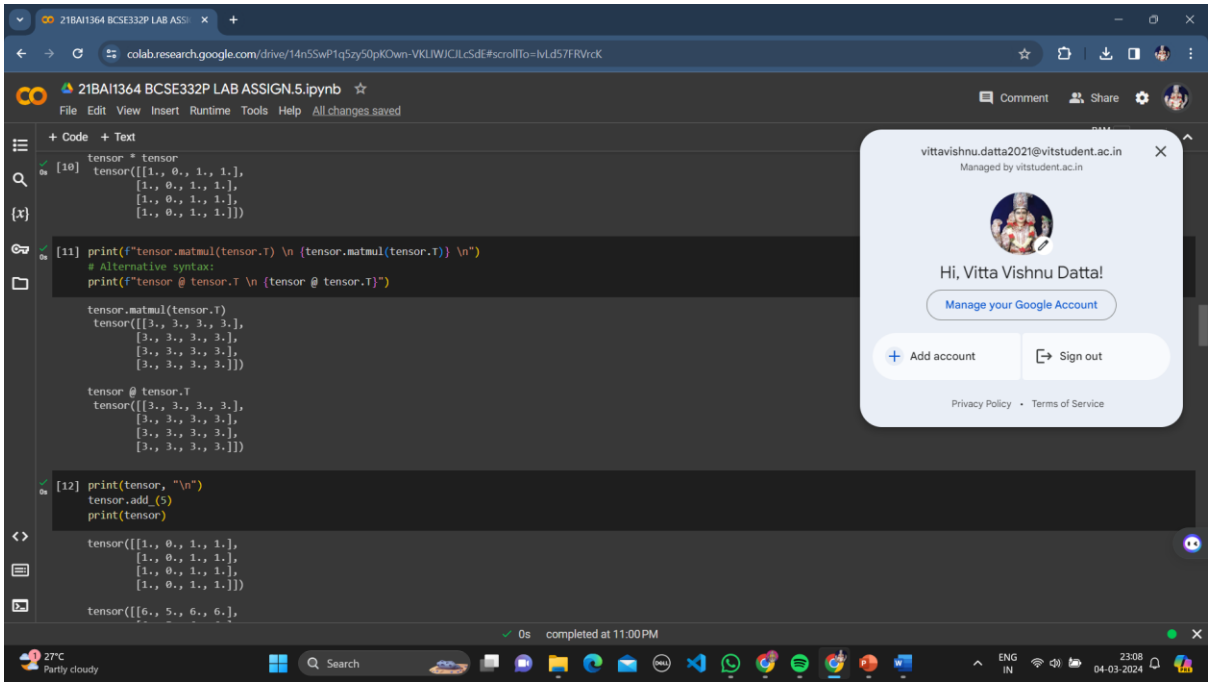


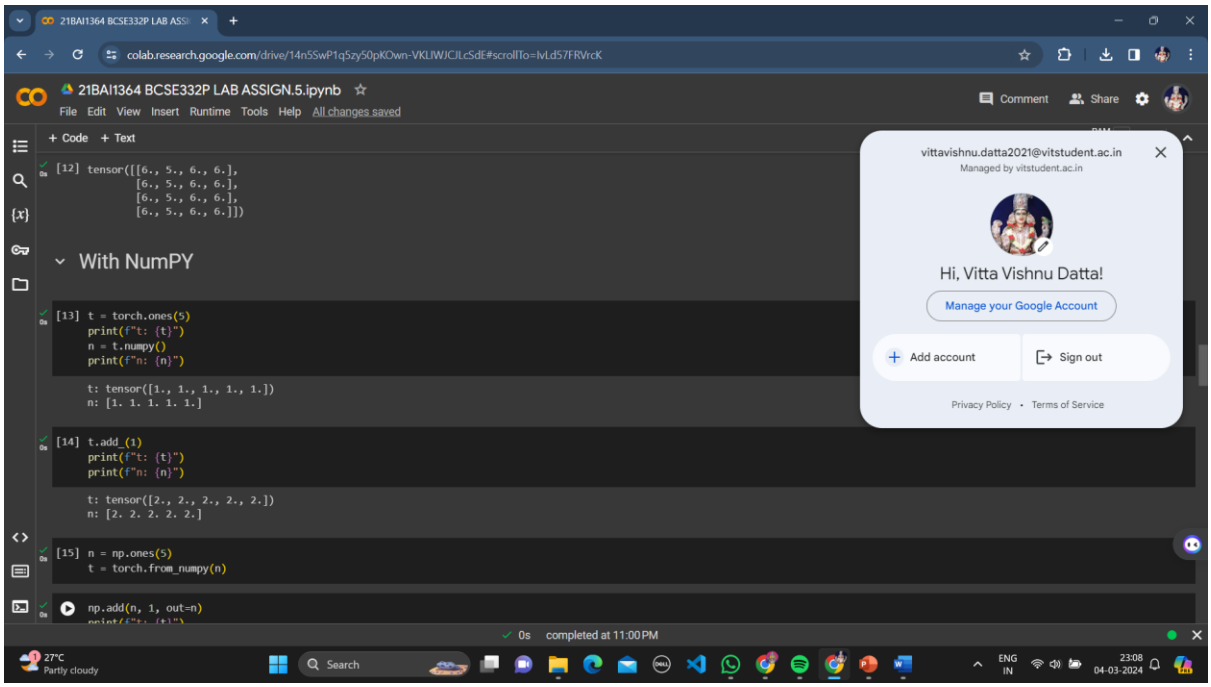
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



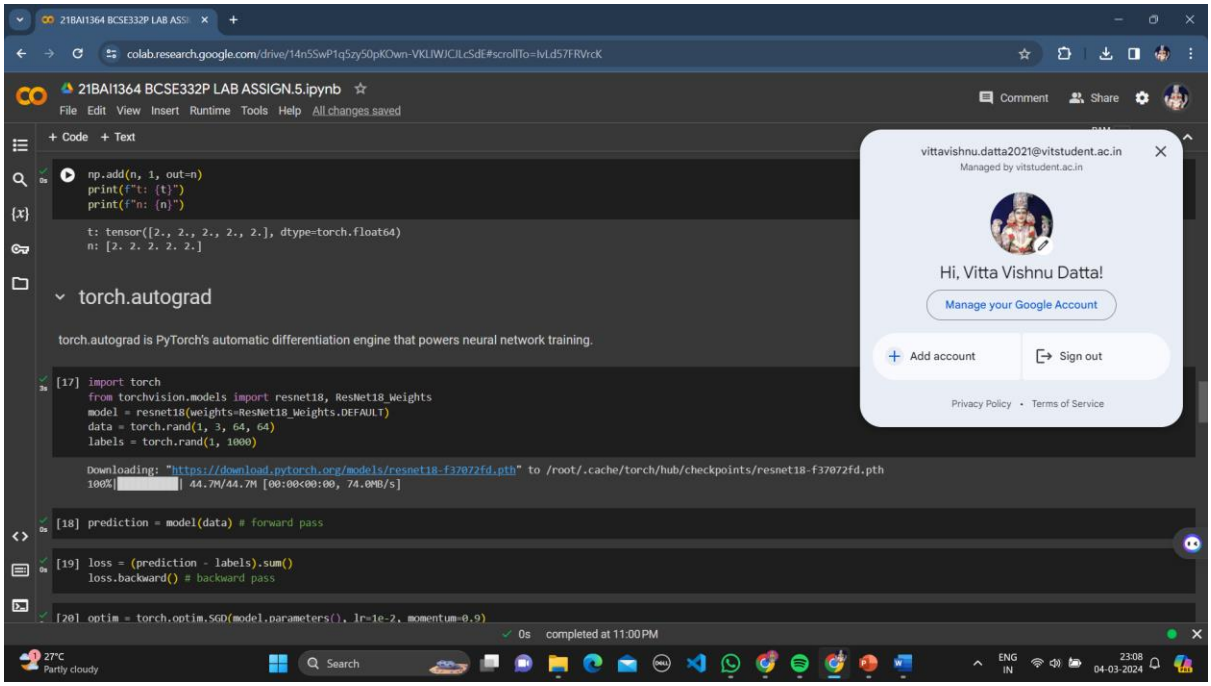


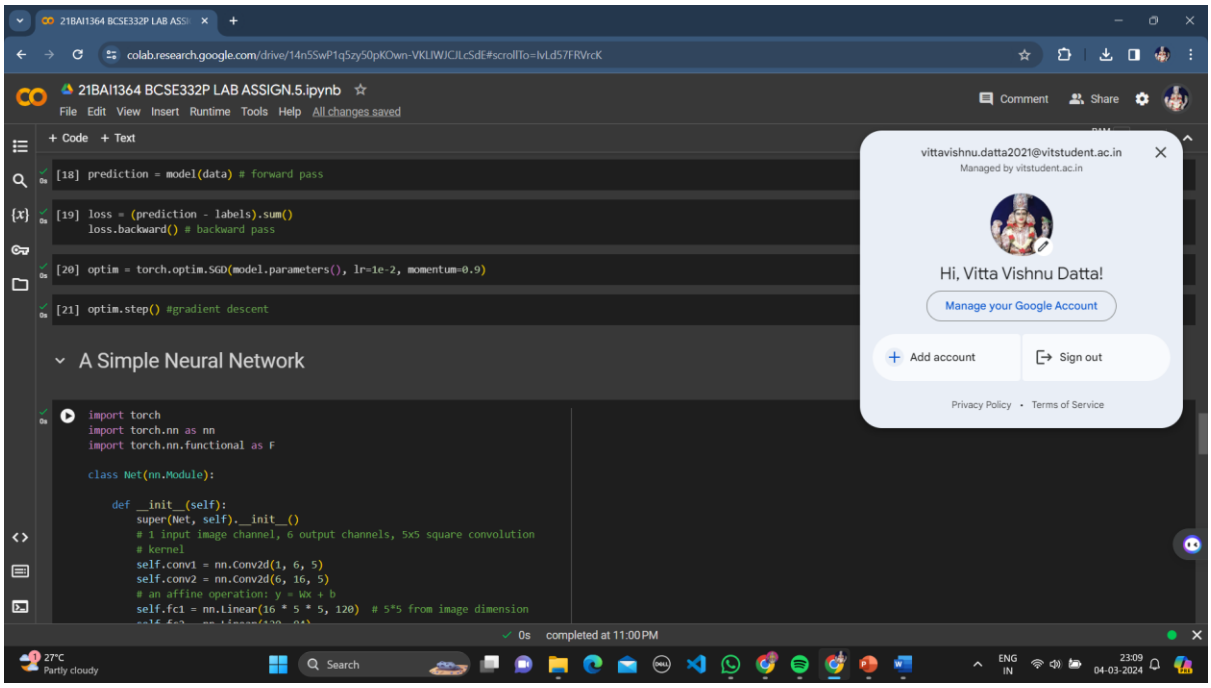
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

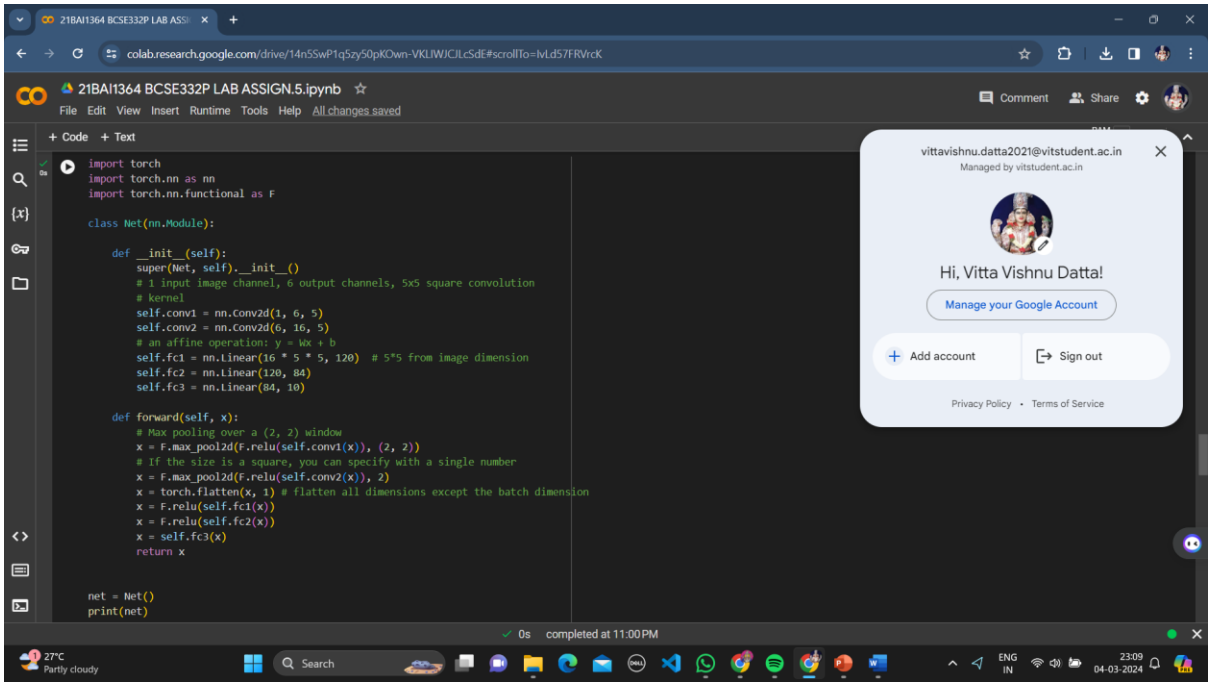


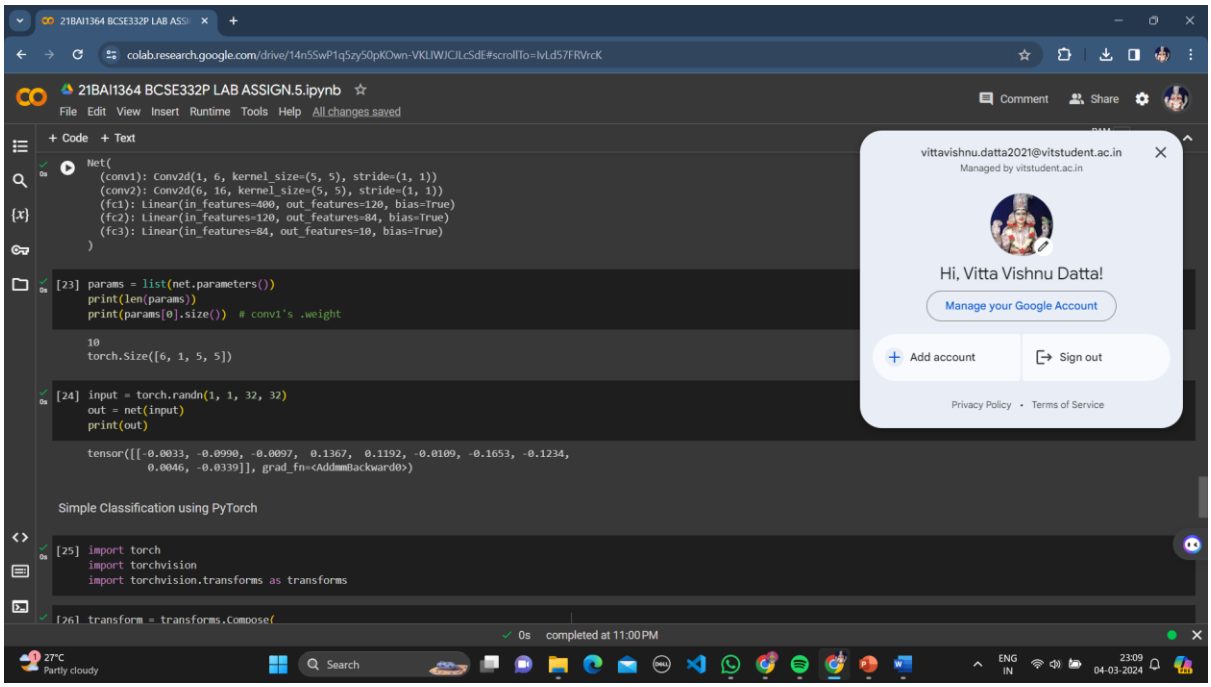


Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



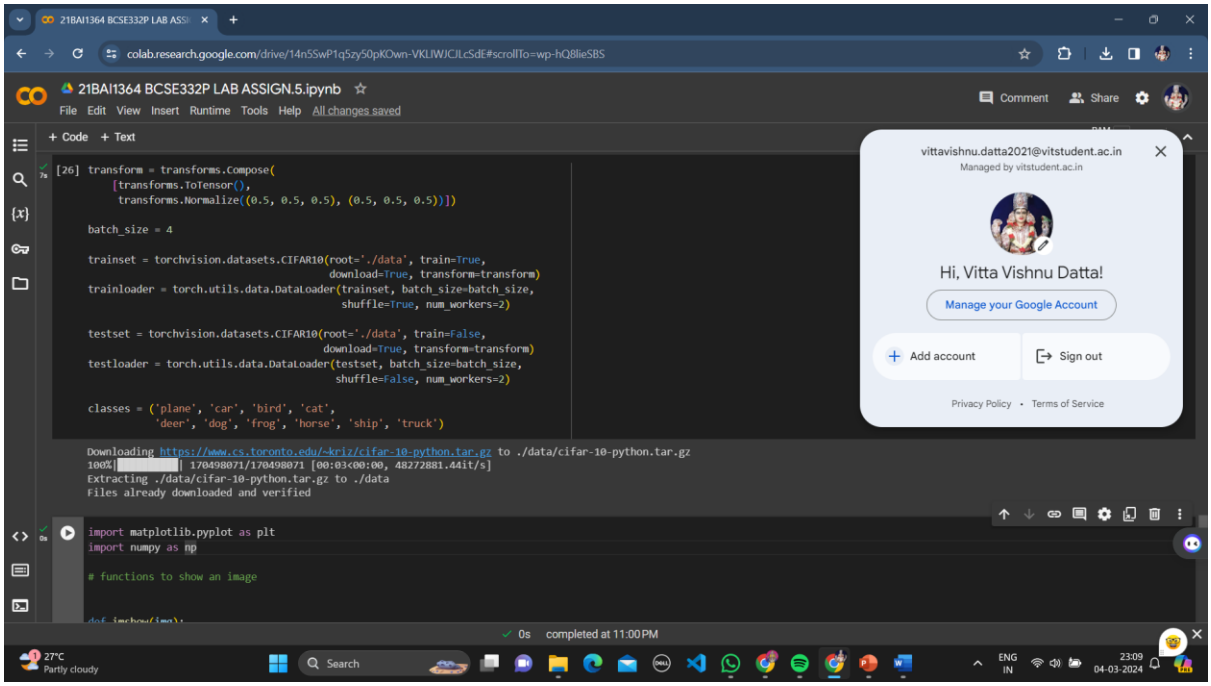


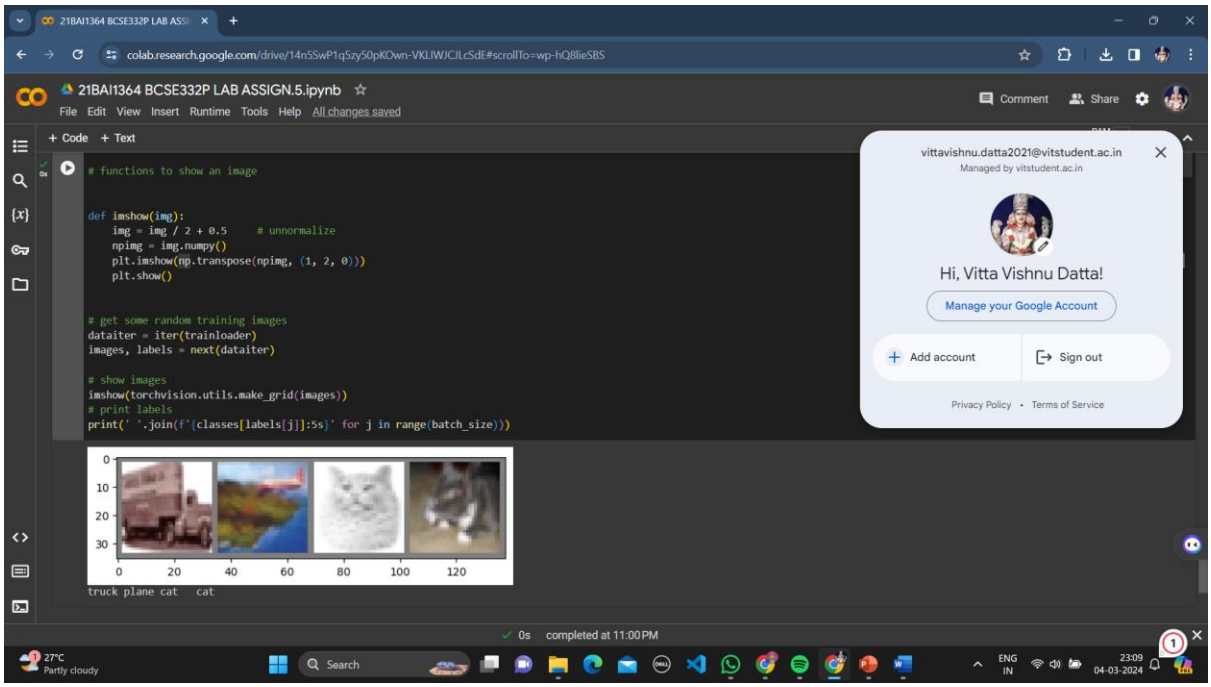
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI





2. Executing the PYTORCH & GPU

CODE:

```
import torch
import torchvision
import torchvision.transforms as transforms
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# Assuming that we are on a CUDA machine, this should print a CUDA
device:

print(device)
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 4

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True,
transform=transform)
trainloader = torch.utils.data.DataLoader(trainset,
batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True,
transform=transform)
testloader = torch.utils.data.DataLoader(testset,
batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
import matplotlib.pyplot as plt
import numpy as np

# functions to show an image
```

```
def imshow(img):
    img = img / 2 + 0.5     # unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


# get some random training images
dataiter = iter(trainloader)
images, labels = next(dataiter)

# show images
imshow(torchvision.utils.make_grid(images))
# print labels
print(' '.join(f'{classes[labels[j]]:5s}' for j in range(batch_size)))
import torch.nn as nn
import torch.nn.functional as F


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.fc1 = nn.Linear(16 * 5 * 5, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x

net = Net()
net.to(device)
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
```

```
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
for epoch in range(2):  # loop over the dataset multiple times
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # Assuming data is a list containing inputs and labels
        inputs, labels = data[0], data[1]  # Access elements using
integer indices

        # Move data to device (if applicable)
        if device is not None:
            inputs = inputs.to(device)
            labels = labels.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:  # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss /
2000:.3f}')
            running_loss = 0.0

print('Finished Training')

PATH = './cifar_net.pth'
torch.save(net.state_dict(), PATH)
dataiter = iter(testloader)
images, labels = next(dataiter)

# print images
imshow(torchvision.utils.make_grid(images))
print('GroundTruth: ', ' '.join(f'{classes[labels[j]]:5s}' for j in
range(4)))
```

LINK TO THE GOOGLE COLAB NOTEBOOK

VIT®

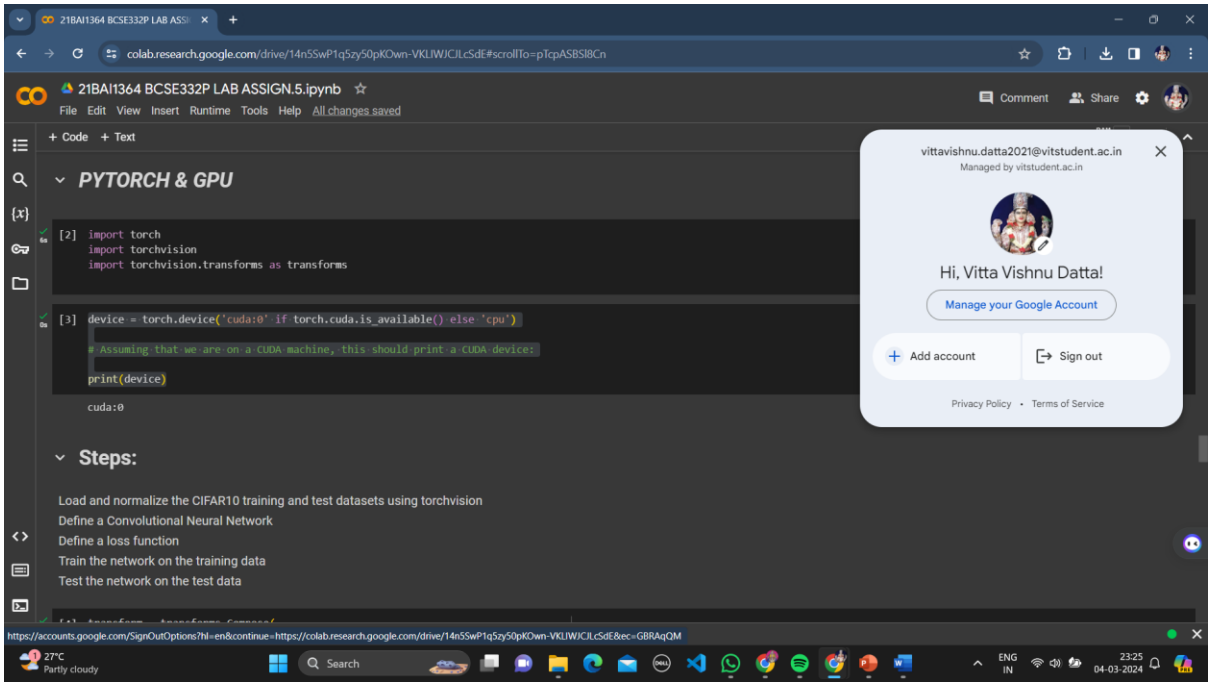
Vellore Institute of Technology

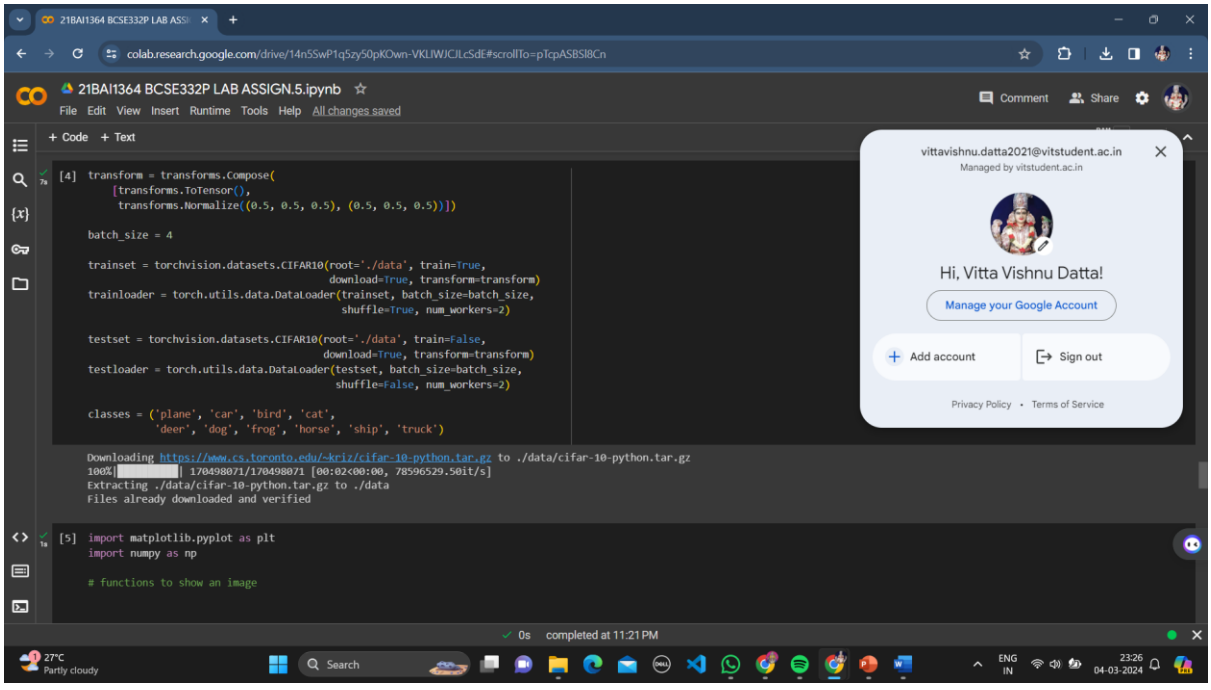
(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

https://colab.research.google.com/drive/14n5SwP1q5zy50pKOwnVKLIWJCJLcSdE?usp=chrome_ntp

SCREENSHOTS OF THE CODE SNIPPET:

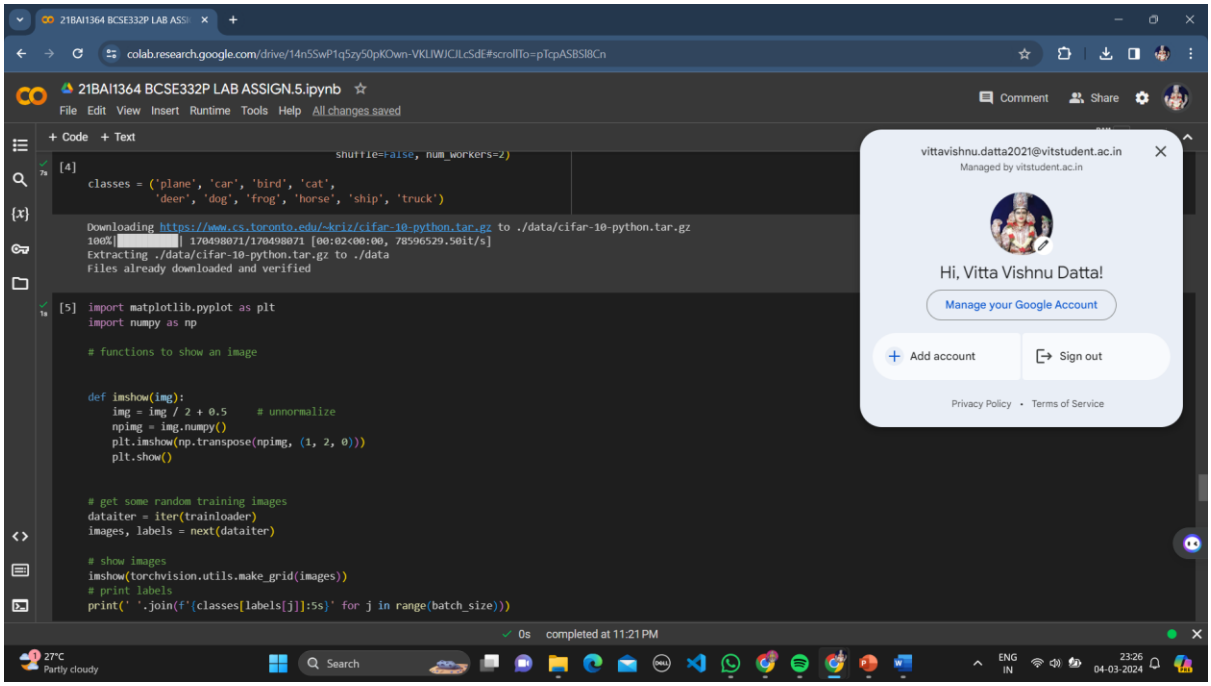


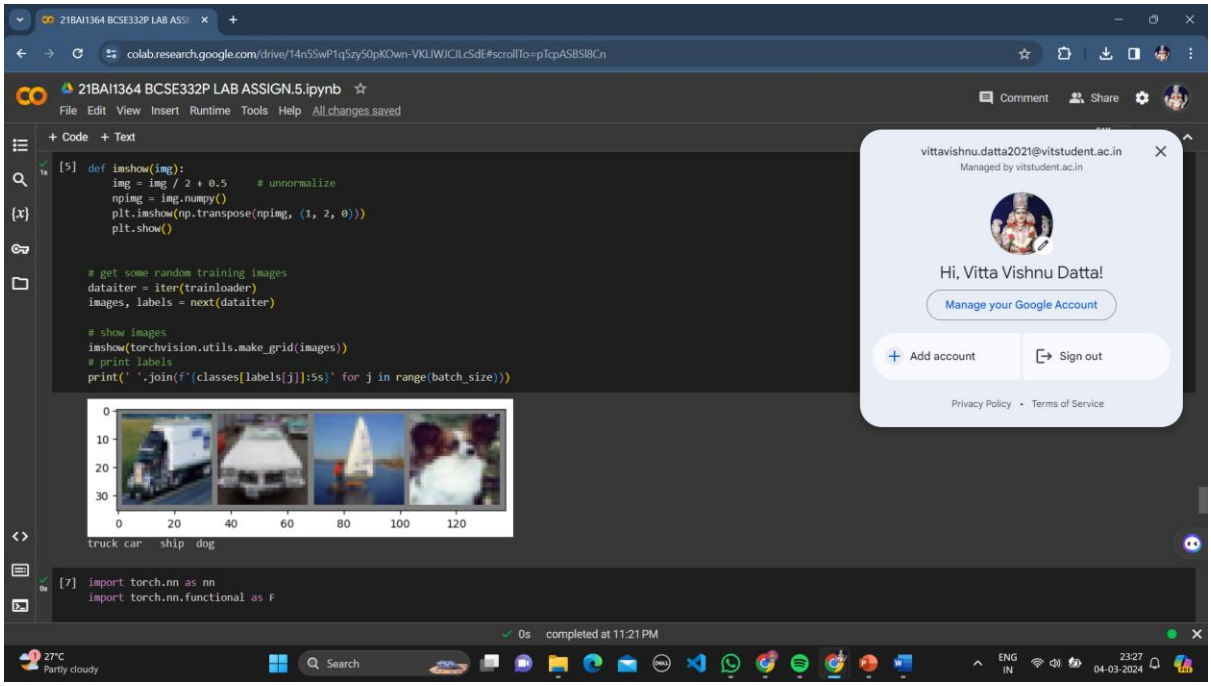


Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



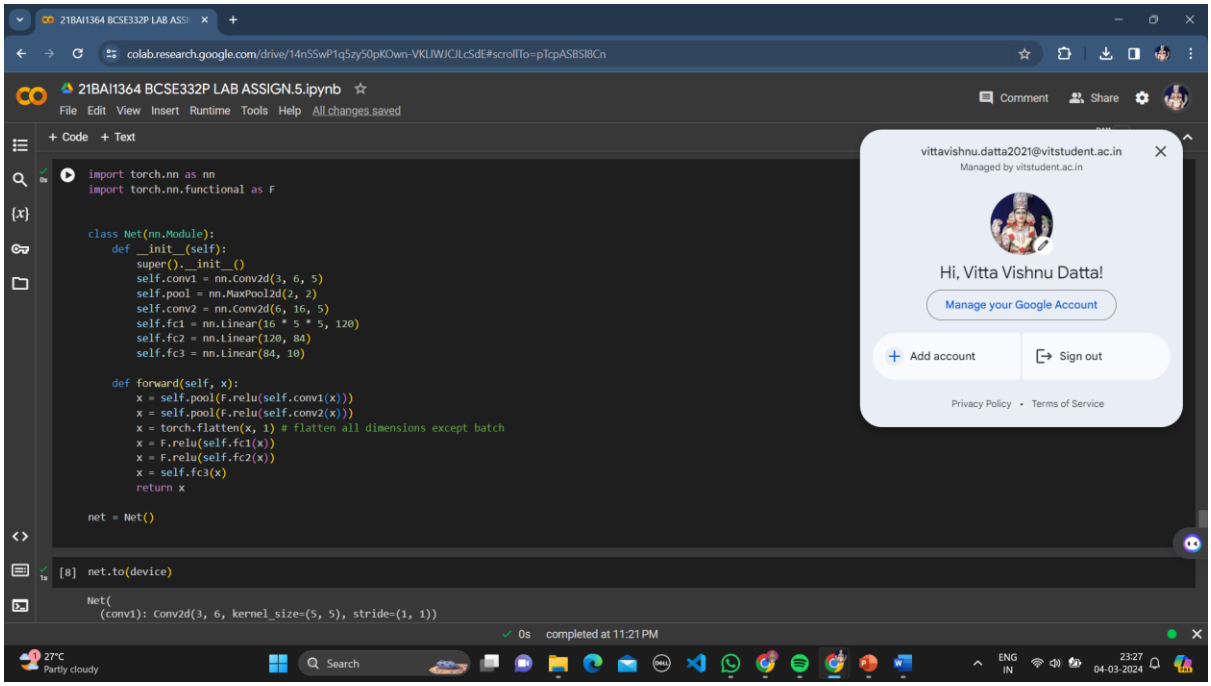


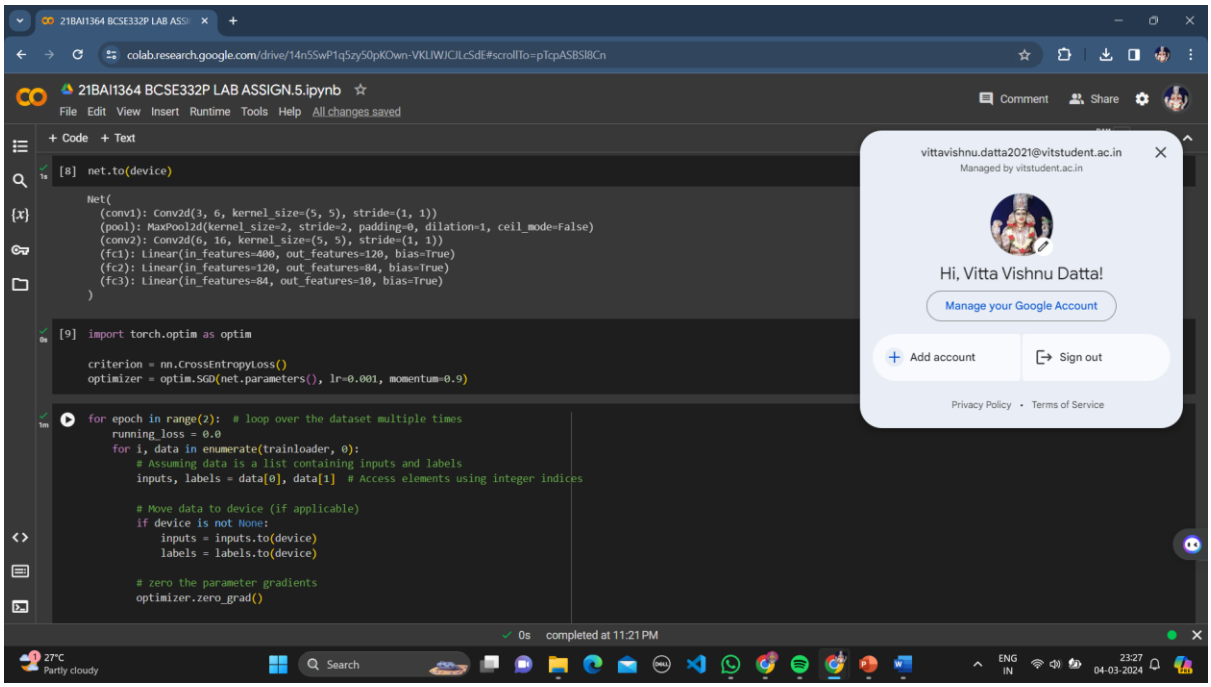
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

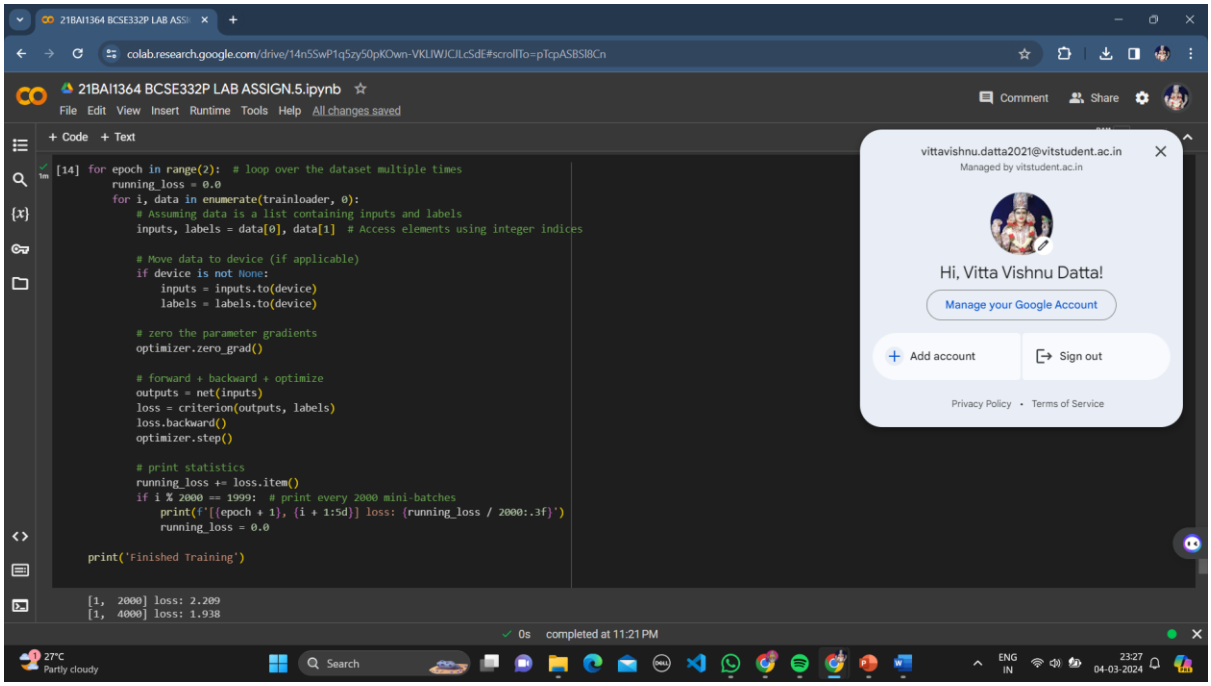


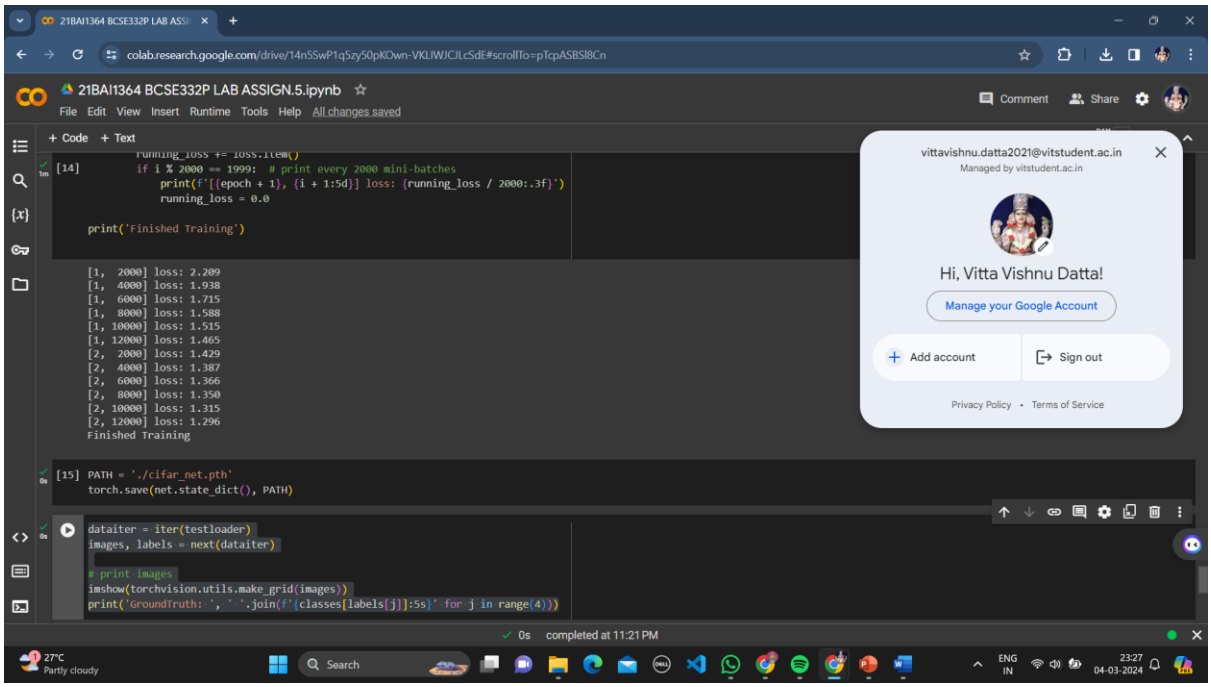


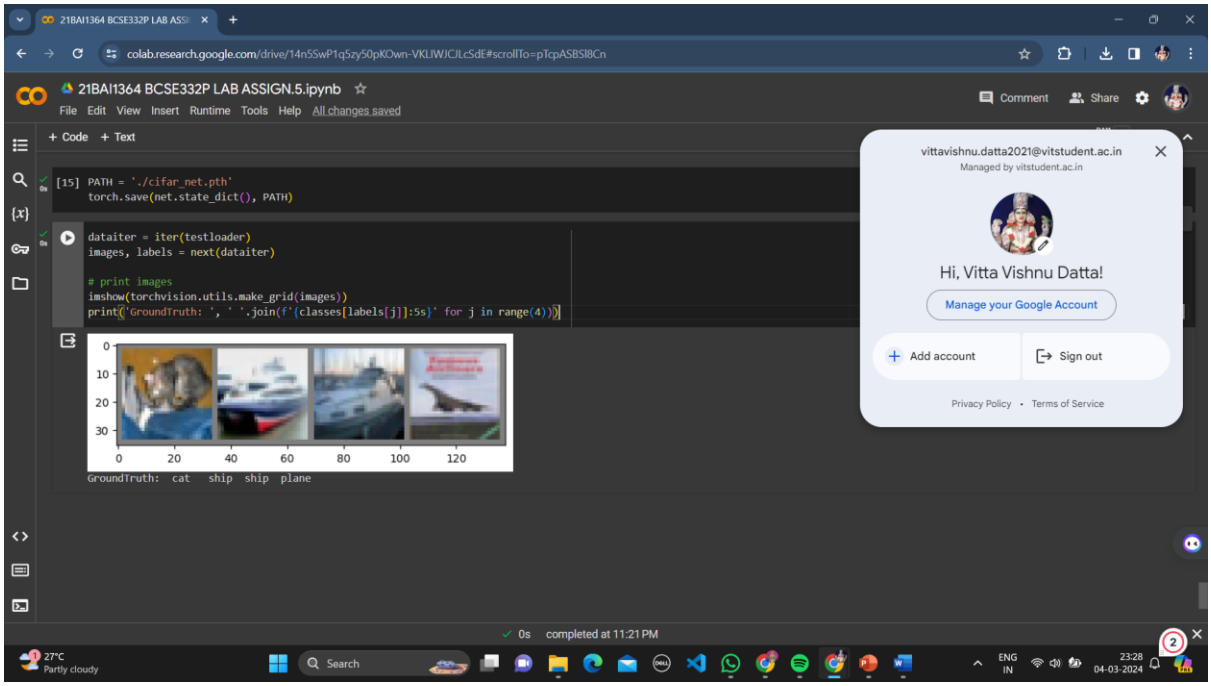
Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI







3. Visualize the filters and features for GoogleNet model using pytorch.

CODE FOR PRE-TRAINED GoogLeNet MODEL:

```
import torch
model = torch.hub.load('pytorch/vision:v0.10.0', 'googlenet',
pretrained=True)
model.eval()
# Download an example image from the pytorch website
import urllib
url, filename =
("https://github.com/pytorch/hub/raw/master/images/dog.jpg", "dog.jpg")
try: urllib.URLopener().retrieve(url, filename)
except: urllib.request.urlretrieve(url, filename)
# sample execution (requires torchvision)
from PIL import Image
from torchvision import transforms
input_image = Image.open(filename)
preprocess = transforms.Compose([
```

```
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
0.225]),
])
input_tensor = preprocess(input_image)
input_batch = input_tensor.unsqueeze(0) # create a mini-batch as
expected by the model

# move the input and model to GPU for speed if available
if torch.cuda.is_available():
    input_batch = input_batch.to('cuda')
    model.to('cuda')

with torch.no_grad():
    output = model(input_batch)
# Tensor of shape 1000, with confidence scores over ImageNet's 1000
classes
print(output[0])
# The output has unnormalized scores. To get probabilities, you can run
a softmax on it.
probabilities = torch.nn.functional.softmax(output[0], dim=0)
print(probabilities)
# Download ImageNet labels
!wget
https://raw.githubusercontent.com/pytorch/hub/master/imagenet_classes.t
xt
# Read the categories
with open("imagenet_classes.txt", "r") as f:
    categories = [s.strip() for s in f.readlines()]
# Show top categories per image
top5_prob, top5_catid = torch.topk(probabilities, 5)
for i in range(top5_prob.size(0)):
    print(categories[top5_catid[i]], top5_prob[i].item())
```

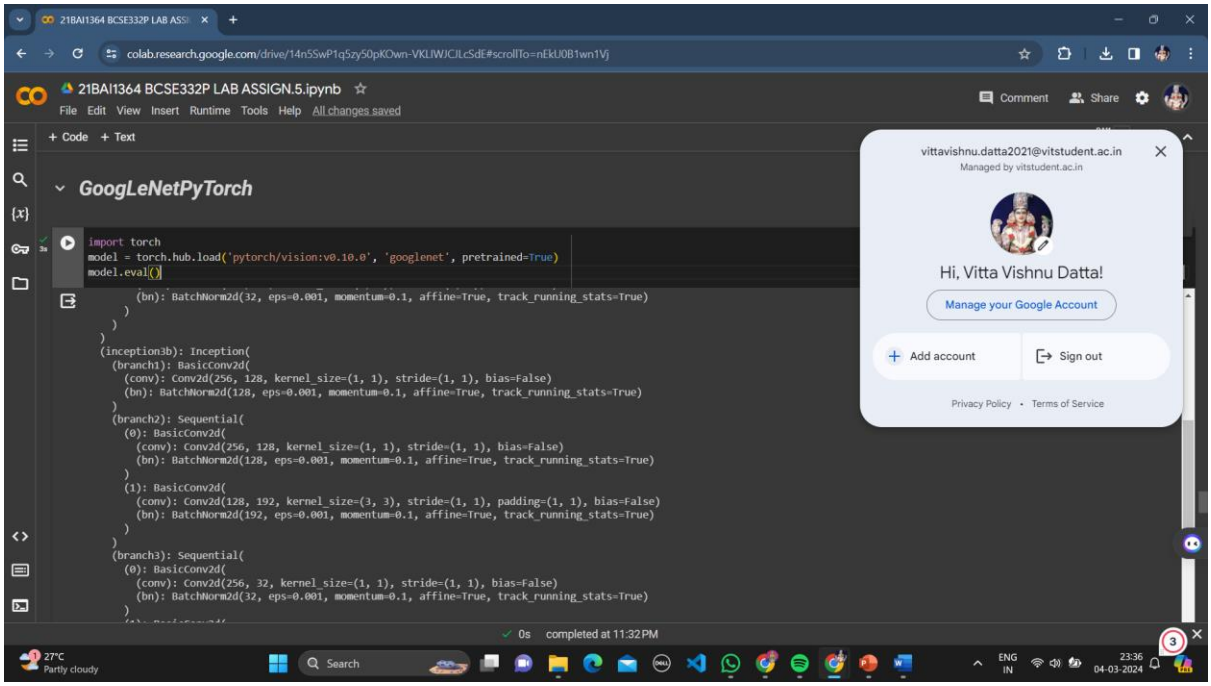
SCREENSHOTS OF THE GoogLeNet MODEL:

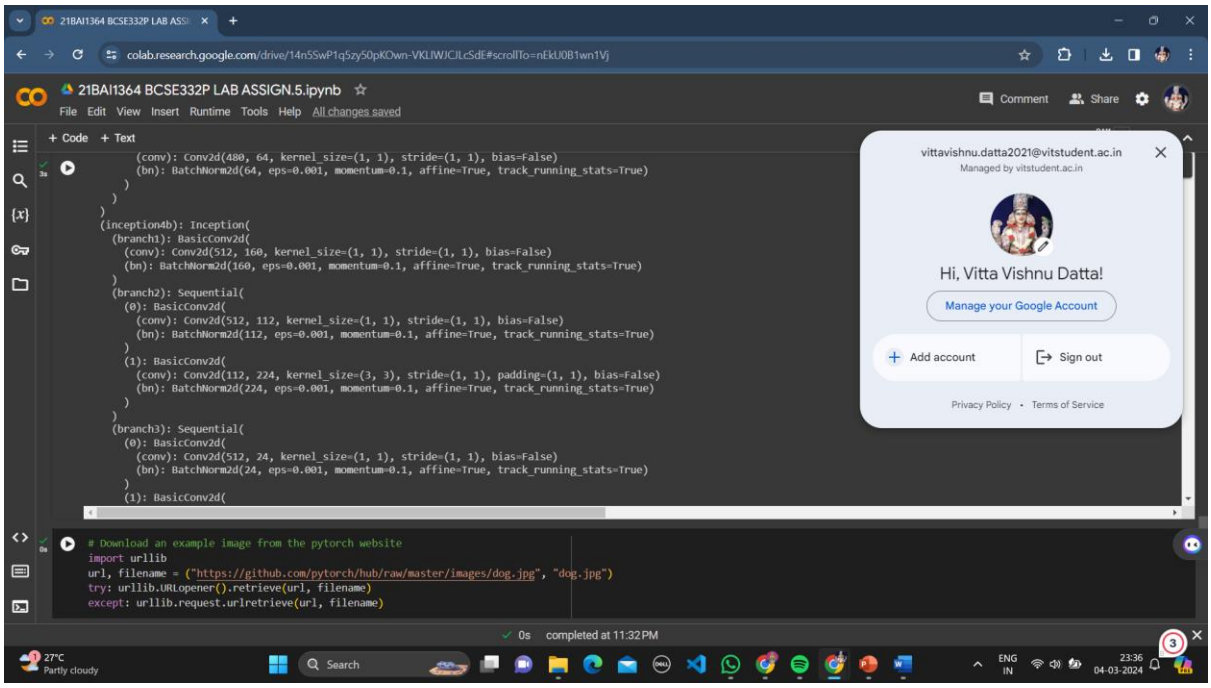
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



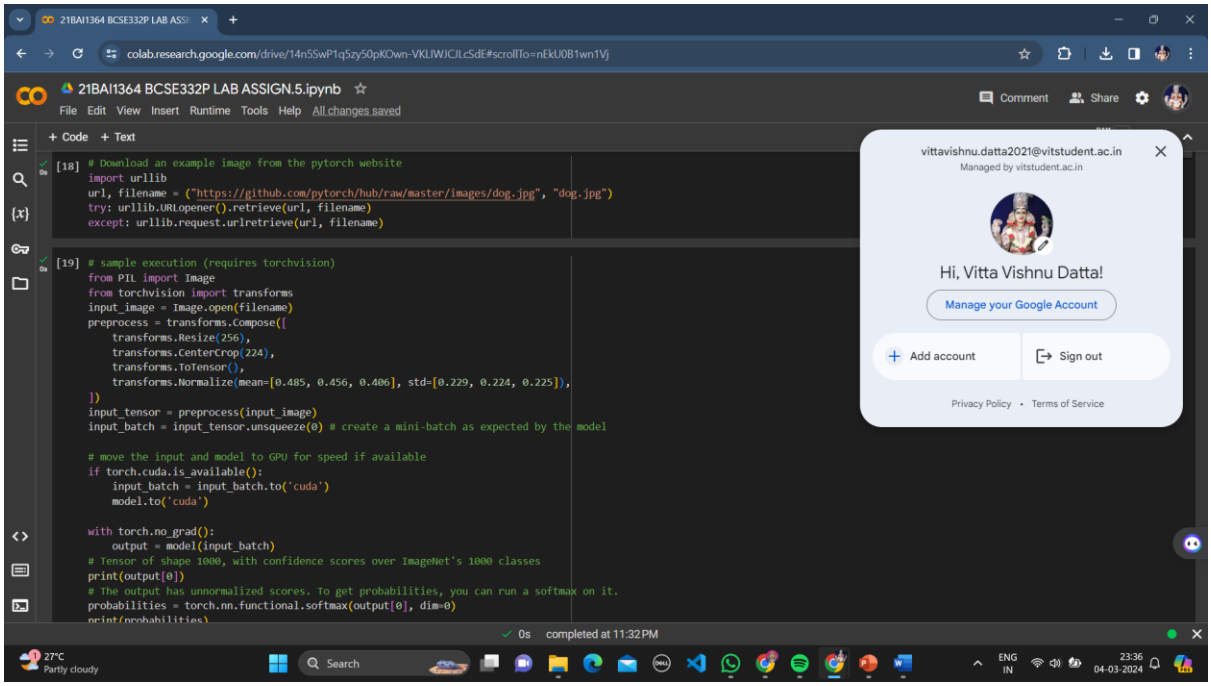


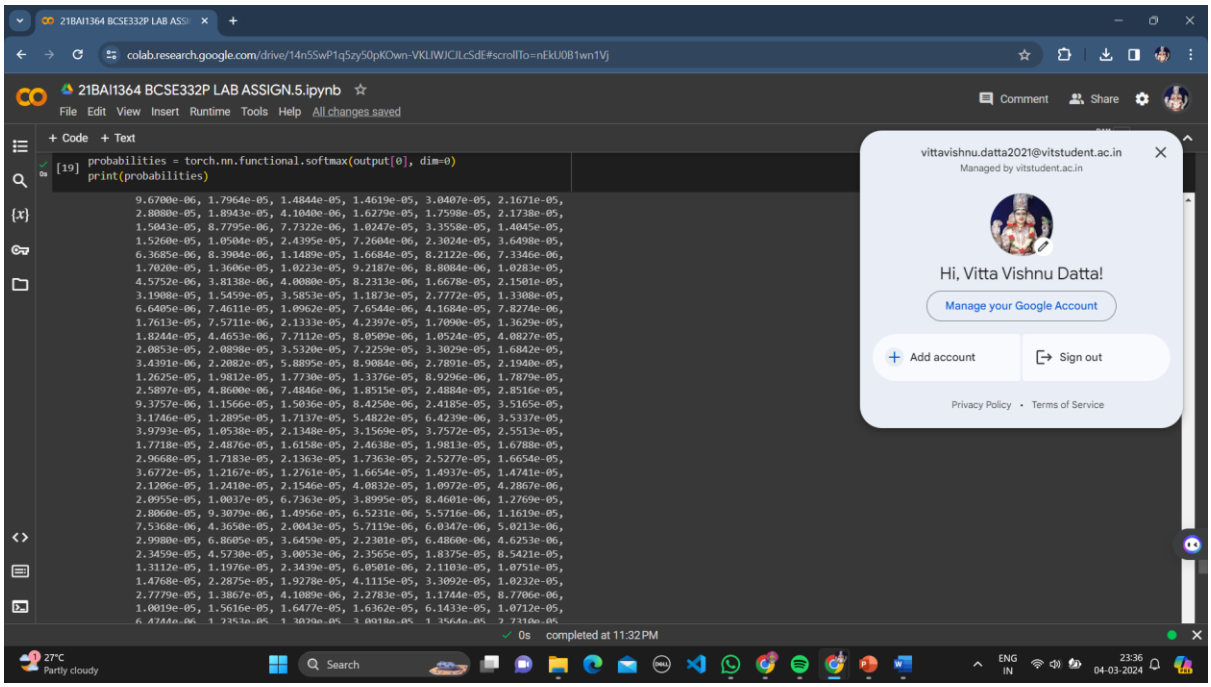
VIT®

Vellore Institute of Technology

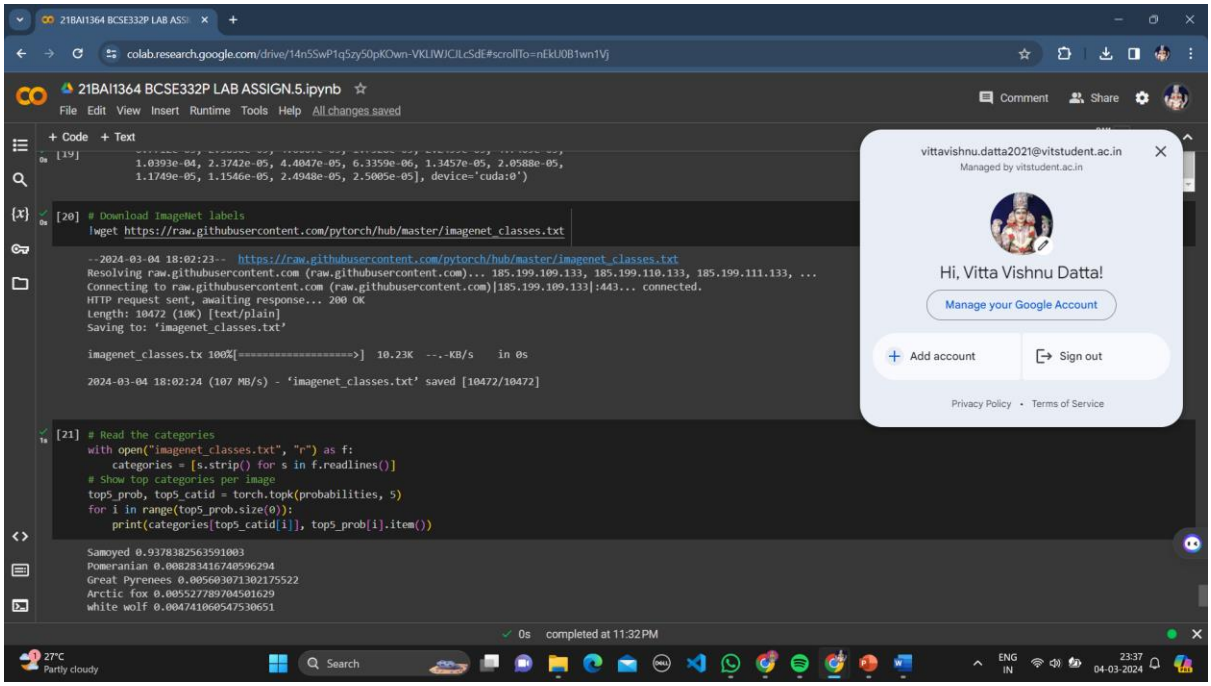
(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

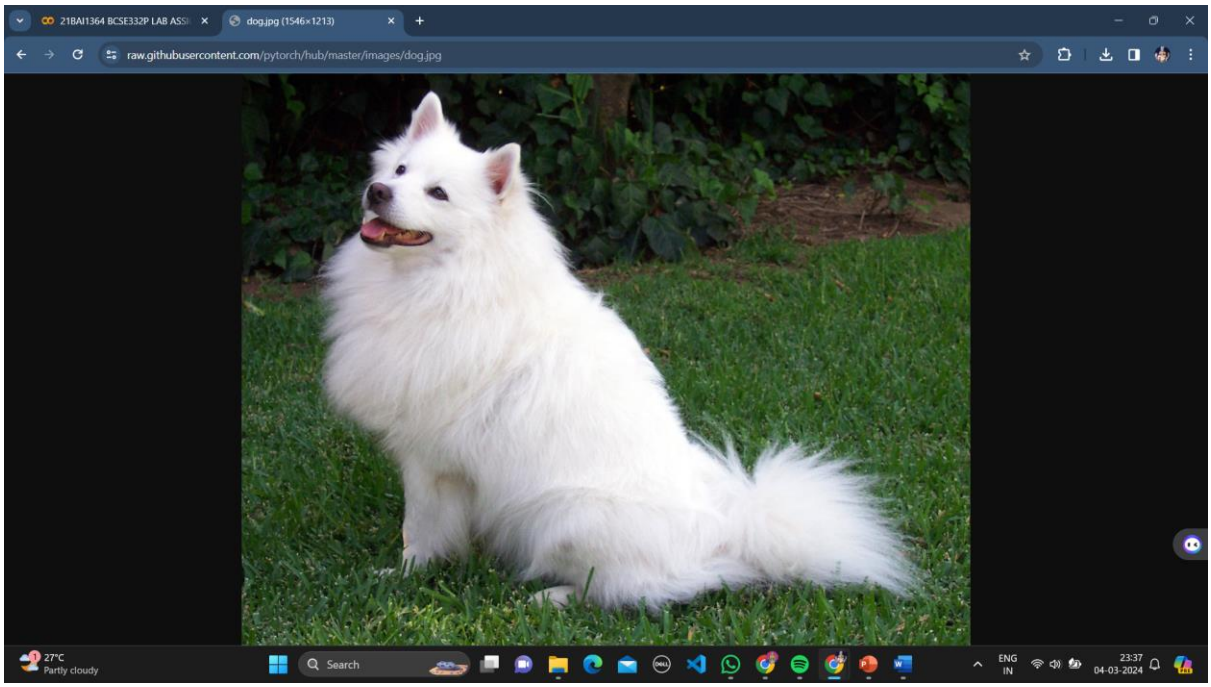




VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI



DOG IMAGE:



CODE FOR PRE-TRAINED VGG 16 MODEL:

```
import torch
import torch.nn as nn
import torchvision.models as models

# Define device (CPU or GPU)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

class BasicBlock(nn.Module):  # Basic building block for ResNet
    def __init__(self, in_channels, out_channels, stride=1):
        super(BasicBlock, self).__init__()

        # Define layers
        self.conv1 = nn.Conv2d(in_channels, out_channels,
kernel_size=3, stride=stride, padding=1, bias=False)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = nn.Conv2d(out_channels, out_channels,
kernel_size=3, padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(out_channels)

        # Shortcut connection for identity mapping (if dimensions
match)
        self.shortcut = nn.Sequential()
        if stride != 1 or in_channels != out_channels:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1,
stride=stride, bias=False),
                nn.BatchNorm2d(out_channels)
            )

    def forward(self, x):
        # Forward pass through the block
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.bn2(self.conv2(out))
        out += self.shortcut(x)  # Add shortcut connection
        out = self.relu(out)
        return out

class ResNet(nn.Module):  # Main ResNet architecture
    def __init__(self, block, num_blocks, num_classes=10):
        super(ResNet, self).__init__()
        self.in_channels = 64
```

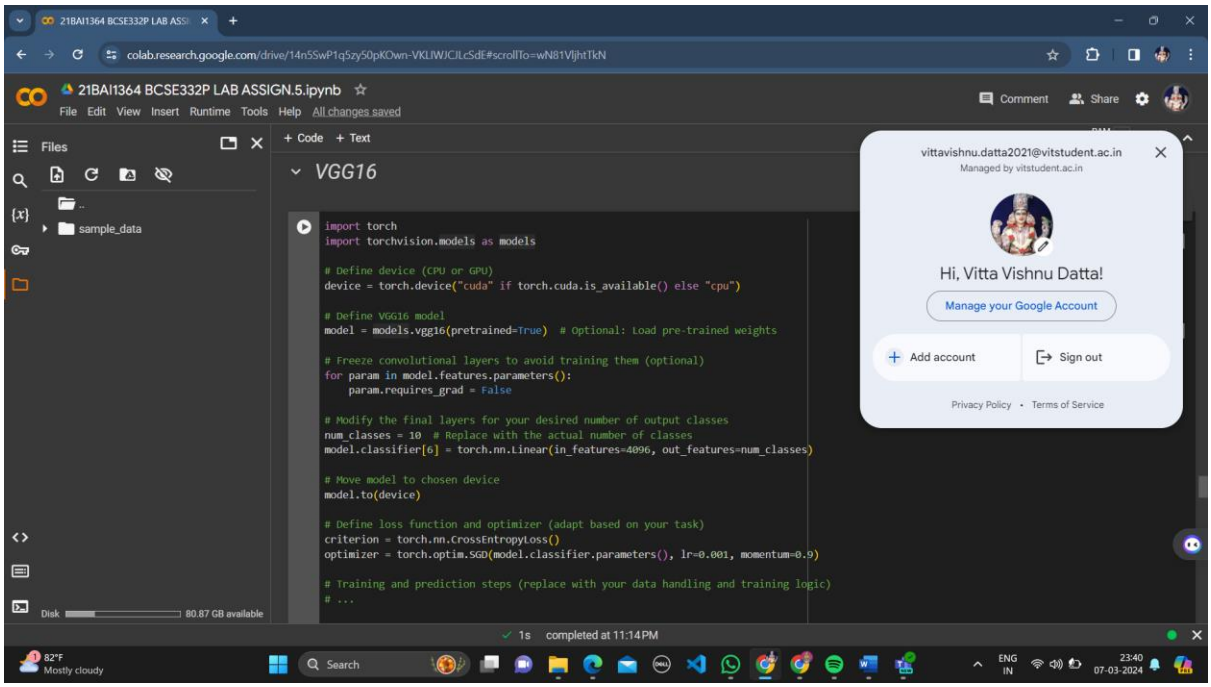
```
        # First convolutional layer
        self.conv1 = nn.Conv2d(3, 64, kernel_size=7, stride=2,
padding=3, bias=False)
        self.bn1 = nn.BatchNorm2d(64)
        self.relu = nn.ReLU(inplace=True)
        self.maxpool = nn.MaxPool2d(kernel_size=3, stride=2, padding=1)

        # Define residual layers
        self.layer1 = self._make_layer(block, 64, num_blocks[0])
        self.layer2 = self._make_layer(block, 128, num_blocks[1],
stride=2)
        self.layer3 = self._make_layer(block, 256, num_blocks[2],
stride=2)
        self.layer4 = self._make_layer(block, 512, num_blocks[3],
stride=2)

        # Final layers
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(512 * 1 * 1, num_classes)  # Output layer
for desired number of classes

    def _make_layer(self, block, out_channels, num_blocks, stride=1):
        # Helper function to create multiple residual blocks
        strides = [stride] + [1] * (num_blocks - 1)  # Adjust strides
for the first and subsequent blocks
        layers = []
        for i in range(num_blocks):
            layers.append(block(self.in_channels, out_channels,
strides[i]))
            self.in_channels = out_channels  # Update in_channels for
subsequent blocks
        return nn.Sequential(*layers)

    def forward(self, x):
        # Forward pass through the network
        out = self.relu(self.bn1(self.conv1(x)))
        out = self.maxpool(out)
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
```

```
        out = self.avgpool(out)
        out = out.view(out.size(0), -1)  # Flatten for fully connected
layer
        out = self.fc(out)
        out
import tensorflow as tf
from tensorflow.keras import layers, models

def VGG16(input_shape=(224, 224, 3), num_classes=1000):
    model = models.Sequential()

    # Block 1
    model.add(layers.Conv2D(64, (3, 3), activation='relu',
padding='same', input_shape=input_shape))
    model.add(layers.Conv2D(64, (3, 3), activation='relu',
padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 2
    model.add(layers.Conv2D(128, (3, 3), activation='relu',
padding='same'))
    model.add(layers.Conv2D(128, (3, 3), activation='relu',
padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 3
    model.add(layers.Conv2D(256, (3, 3), activation='relu',
padding='same'))
    model.add(layers.Conv2D(256, (3, 3), activation='relu',
padding='same'))
    model.add(layers.Conv2D(256, (3, 3), activation='relu',
padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    # Block 4
    model.add(layers.Conv2D(512, (3, 3), activation='relu',
padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu',
padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu',
padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))
```

```
    # Block 5
    model.add(layers.Conv2D(512, (3, 3), activation='relu',
padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu',
padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu',
padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    # Flatten and fully connected layers
    model.add(layers.Flatten())
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))

    return model

# Example usage
input_shape = (224, 224, 3)
num_classes = 1000  # Change this to match your dataset's number of
classes
vgg16_model = VGG16(input_shape, num_classes)

# Print model summary
vgg16_model.summary()

#IMPLEMENT VGG16 FOR IMAGE RECOGNITION
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.vgg16 import
preprocess_input,decode_predictions
import numpy as np
import matplotlib.pyplot as plt
# Load the VGG16 model with pre-trained weights
model = VGG16(weights='imagenet')
# Load and preprocess the input image
img_path = '/content/paint.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)
# Perform inference
preds = model.predict(x)
# Decode and print the top-3 predicted labels
print('Predicted:', decode_predictions(preds, top=3)[0])
# Step 4: Visualize the features
plt.imshow(img)
plt.title('Input Image')
plt.axis('off')
plt.show()
```

SCREENSHOTS OF THE VGG16 MODEL:

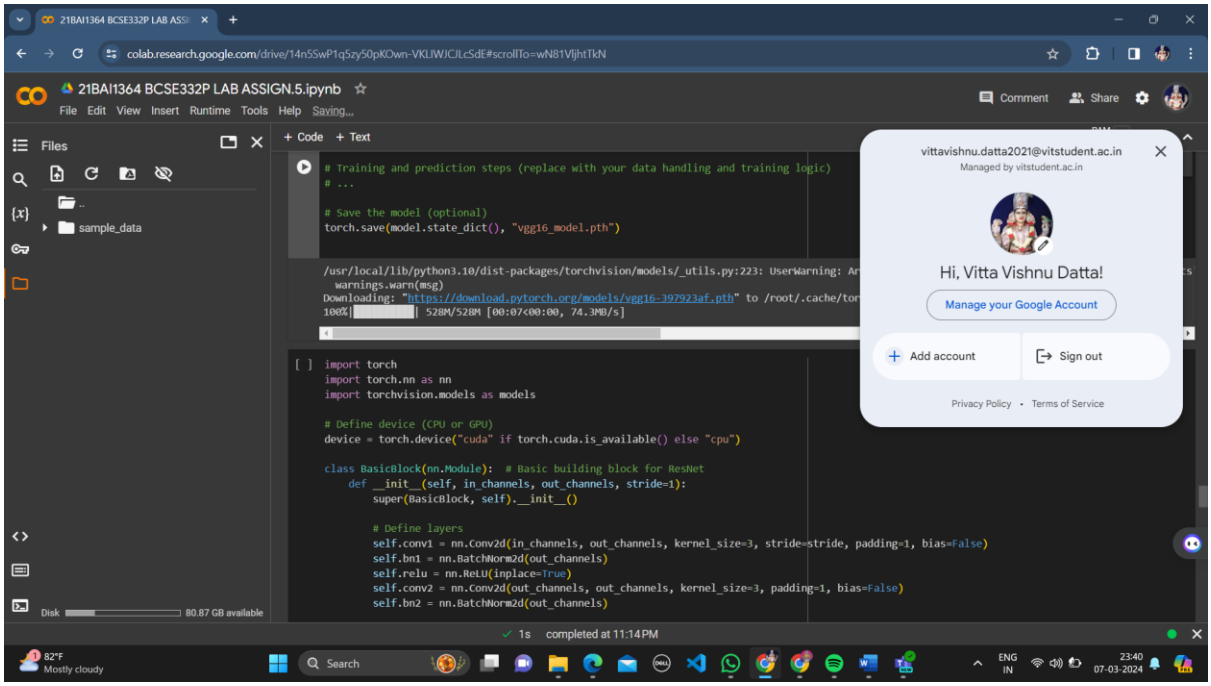


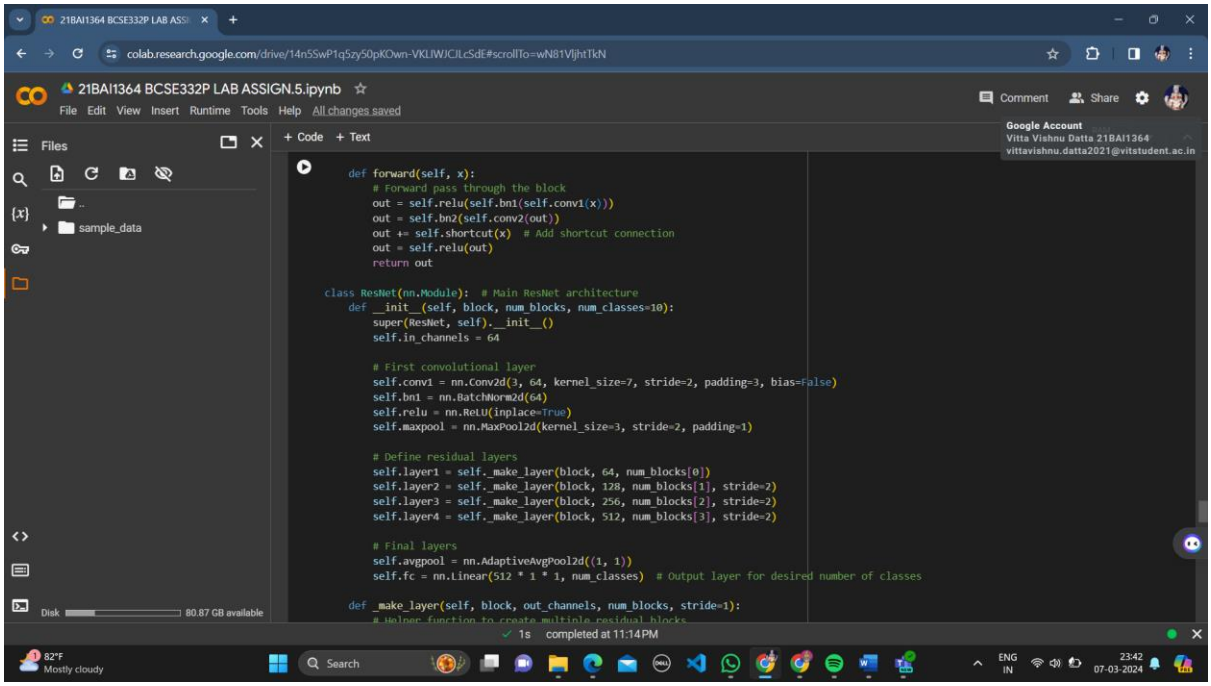
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



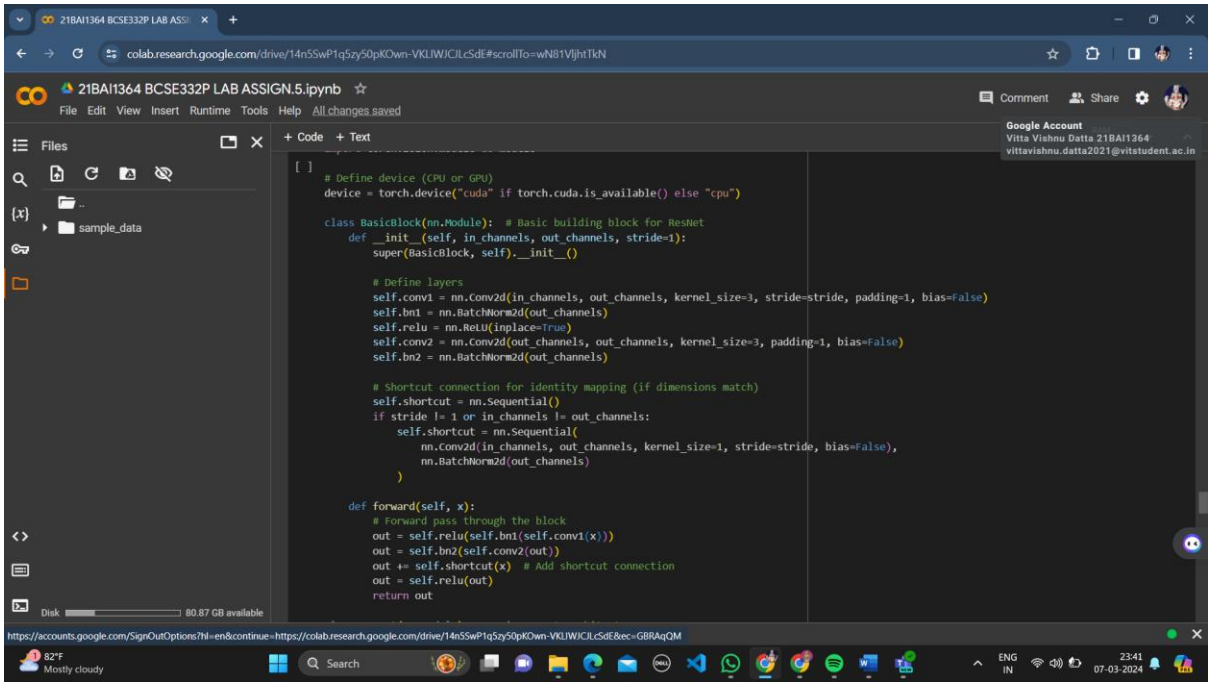


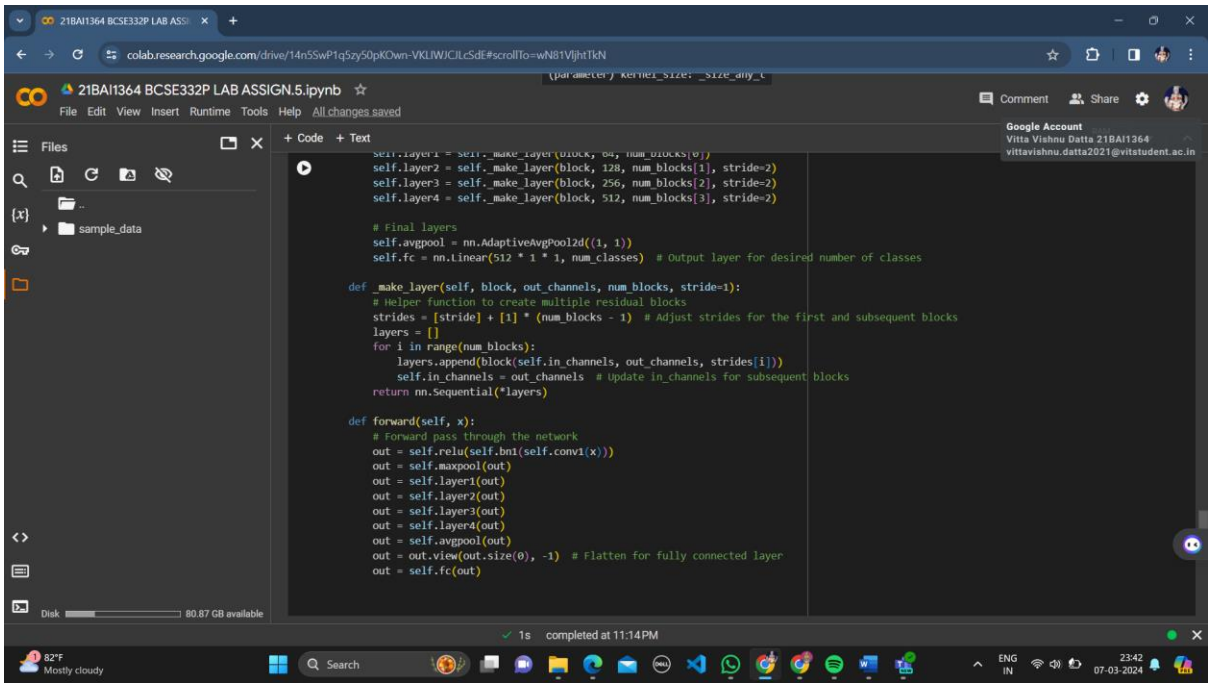
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



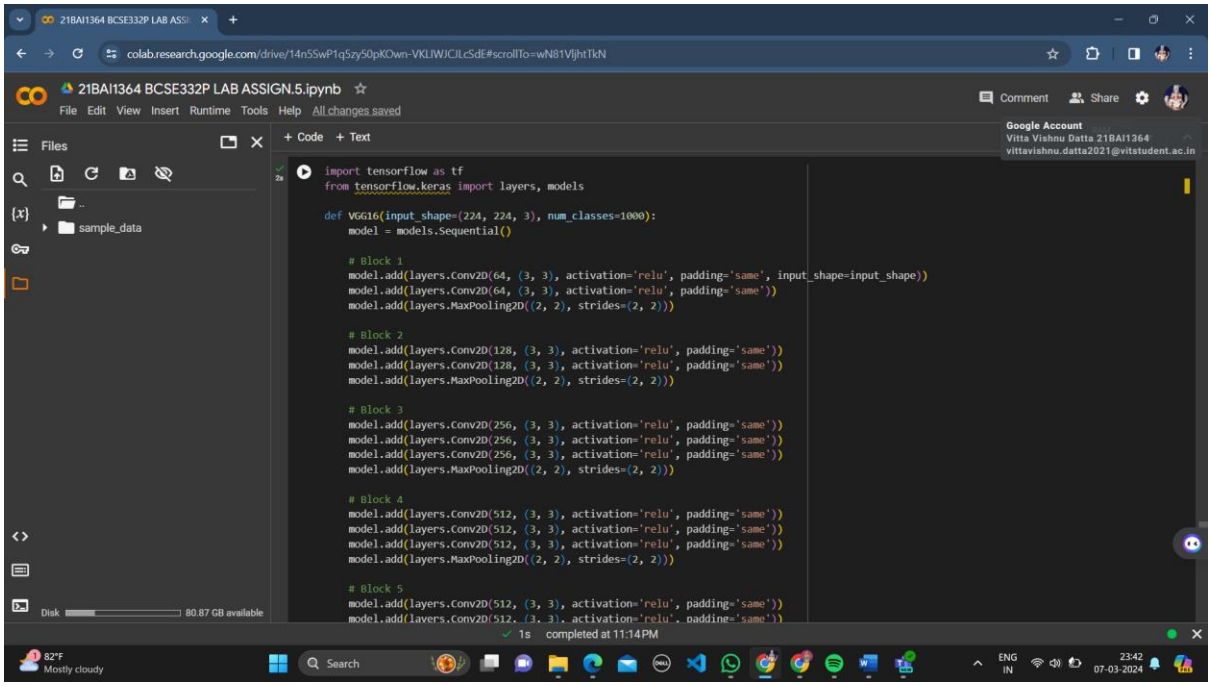


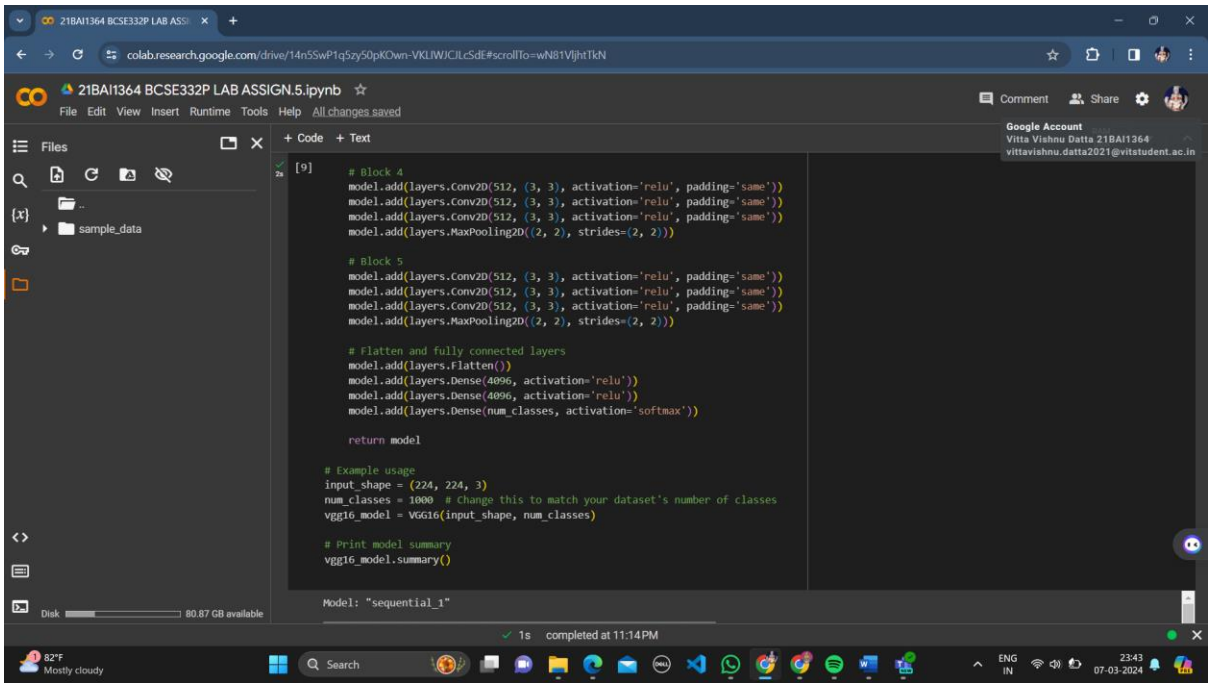
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

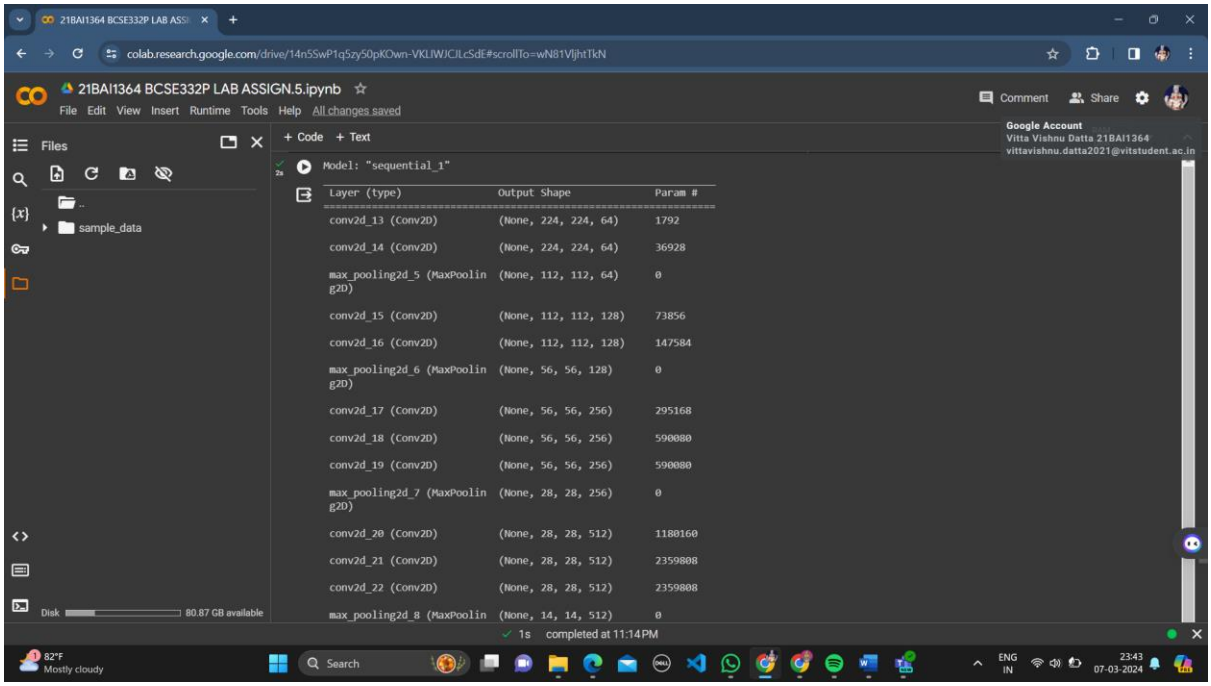


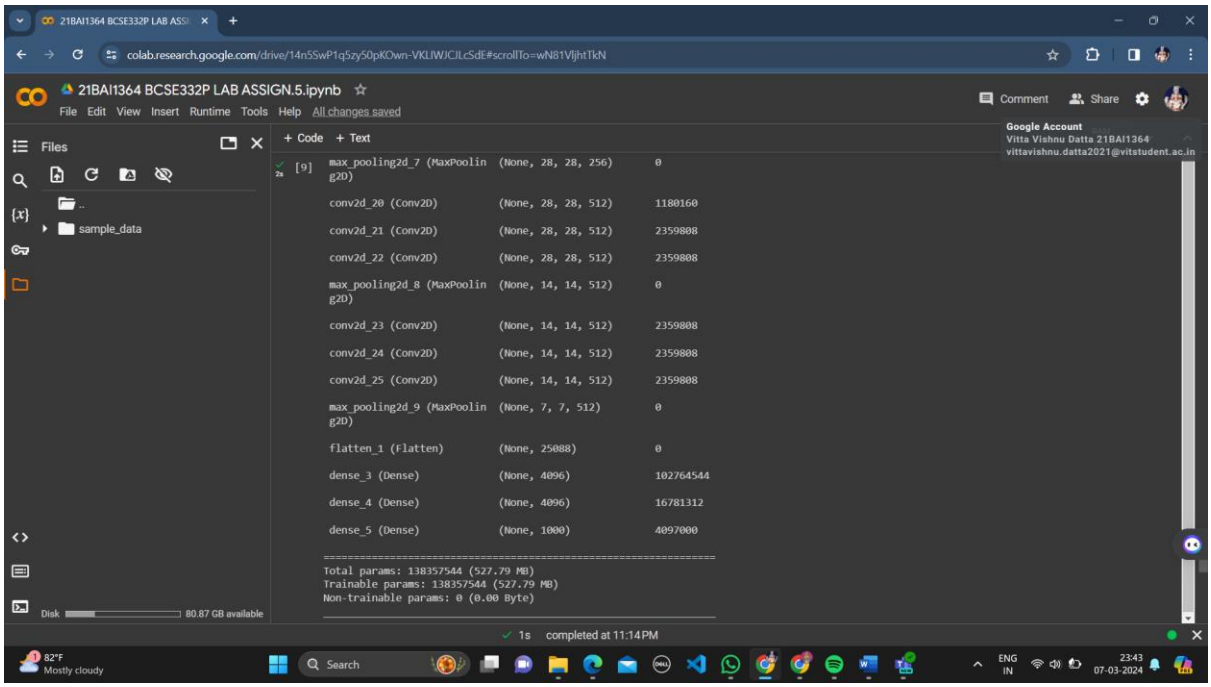


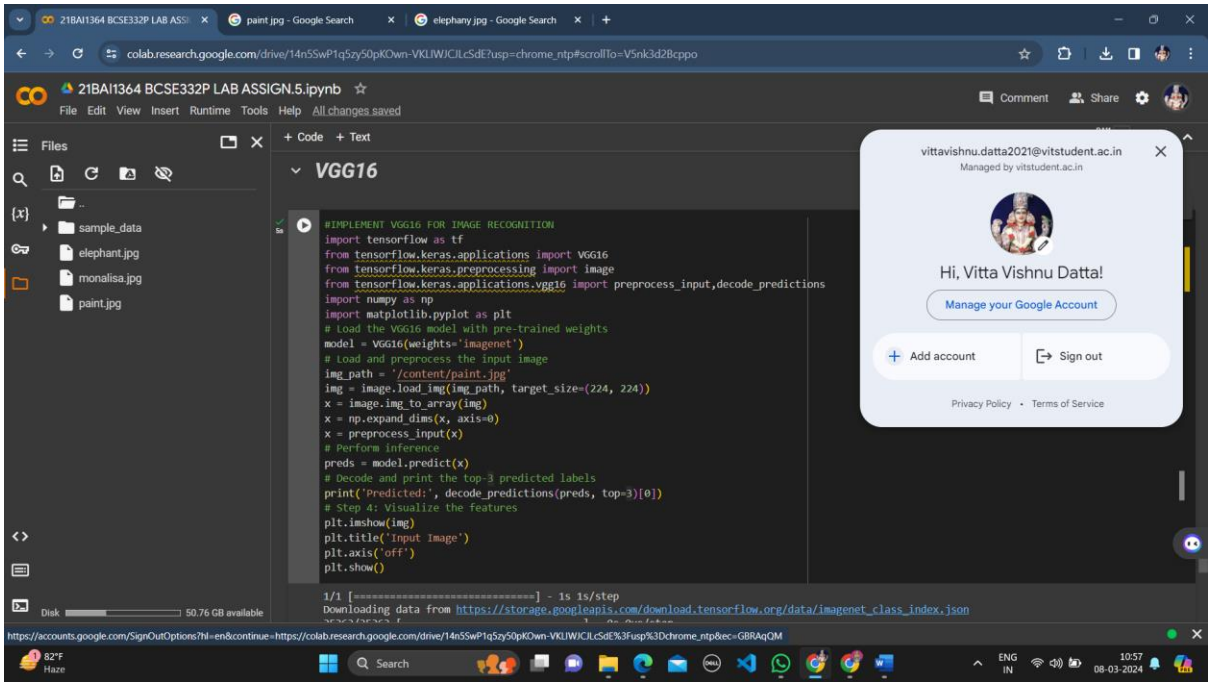
Vellore Institute of Technology

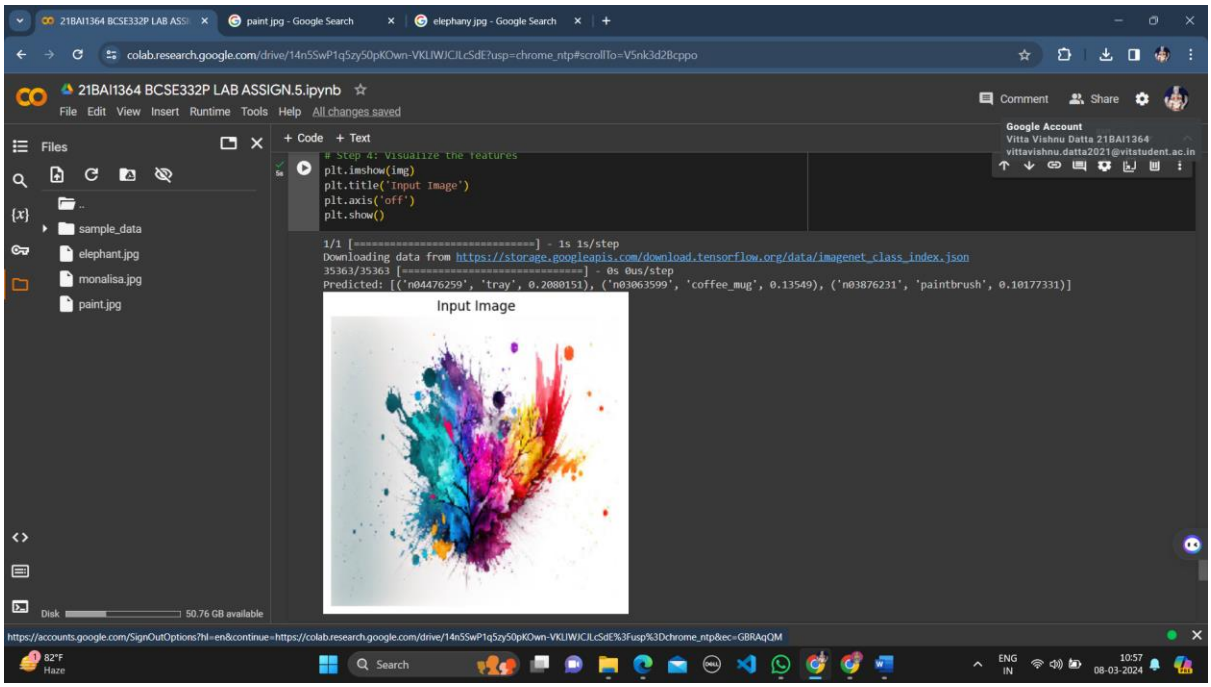
(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI









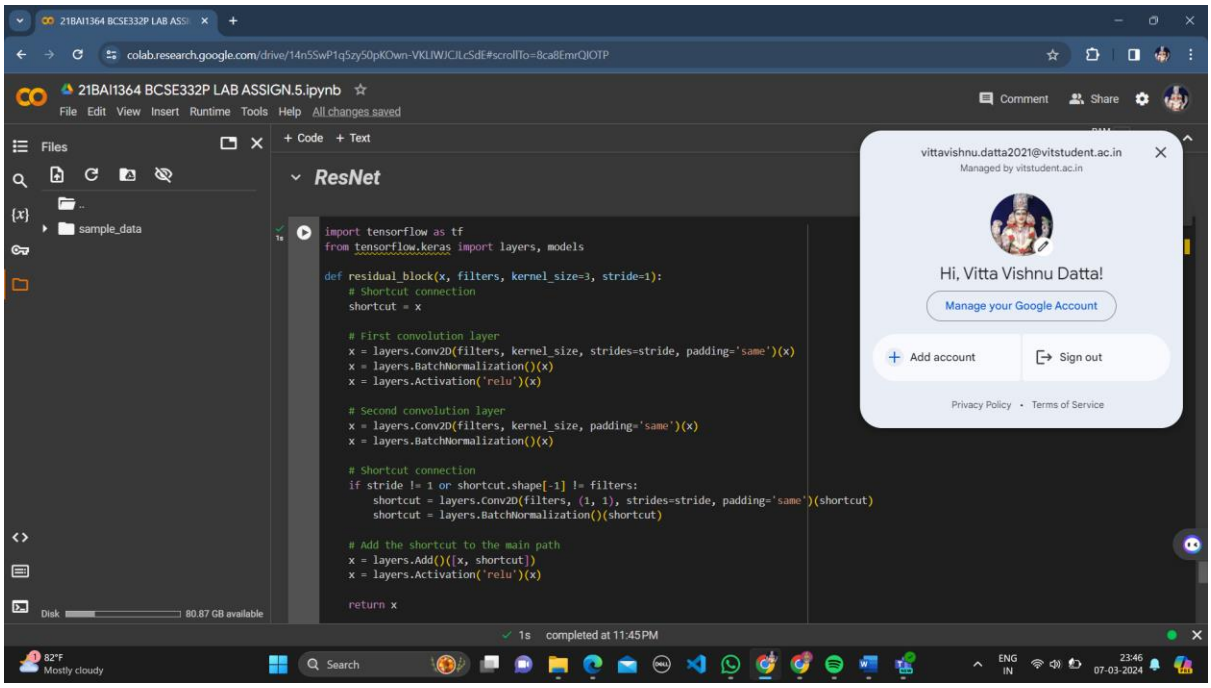
CODE FOR PRE-TRAINED ResNet MODEL:

```
import tensorflow as tf
from tensorflow.keras import layers, models

def residual_block(x, filters, kernel_size=3, stride=1):
    # Shortcut connection
    shortcut = x

    # First convolution layer
    x = layers.Conv2D(filters, kernel_size, strides=stride,
padding='same')(x)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)

    # Second convolution layer
    x = layers.Conv2D(filters, kernel_size, padding='same')(x)
    x = layers.BatchNormalization()(x)

    # Shortcut connection
    if stride != 1 or shortcut.shape[-1] != filters:
        shortcut = layers.Conv2D(filters, (1, 1), strides=stride,
padding='same')(shortcut)
        shortcut = layers.BatchNormalization()(shortcut)

    # Add the shortcut to the main path
    x = layers.Add()([x, shortcut])
    x = layers.Activation('relu')(x)

    return x

def ResNet(input_shape=(224, 224, 3), num_classes=1000):
    inputs = layers.Input(shape=input_shape)

    # Initial convolution and max pooling
    x = layers.Conv2D(64, (7, 7), strides=2, padding='same')(inputs)
    x = layers.BatchNormalization()(x)
    x = layers.Activation('relu')(x)
    x = layers.MaxPooling2D((3, 3), strides=2, padding='same')(x)

    # Residual blocks
    x = residual_block(x, filters=64)
    x = residual_block(x, filters=64)
    x = residual_block(x, filters=128, stride=2)
```

```
    x = residual_block(x, filters=128)
    x = residual_block(x, filters=256, stride=2)
    x = residual_block(x, filters=256)
    x = residual_block(x, filters=512, stride=2)
    x = residual_block(x, filters=512)

    # Global average pooling and dense layer
    x = layers.GlobalAveragePooling2D()(x)
    x = layers.Dense(num_classes, activation='softmax')(x)

    model = models.Model(inputs, x, name='resnet')

    return model

# Example usage
input_shape = (224, 224, 3)
num_classes = 1000  # Change this to match your dataset's number of
classes
resnet_model = ResNet(input_shape, num_classes)

# Print model summary
resnet_model.summary()


import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.resnet50 import preprocess_input,
decode_predictions

# Step 1: Load the ResNet50 model
model = ResNet50(weights='imagenet', include_top=True)

# Step 2: Load and preprocess the input image
img_path = '/content/monalisa.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)

# Step 3: Pass the preprocessed image through the ResNet50 model
```

```
features = model.predict(x)

# Step 4: Visualize the input image
plt.imshow(img)
plt.title('Input Image')
plt.axis('off')
plt.show()

# Step 5: Classify the image
preds = decode_predictions(features, top=3)[0]

# Step 6: Display the results
for pred in preds:
    print(f'{pred[1]}: {pred[2]}')
```

SCREENSHOTS OF THE ResNet MODEL:

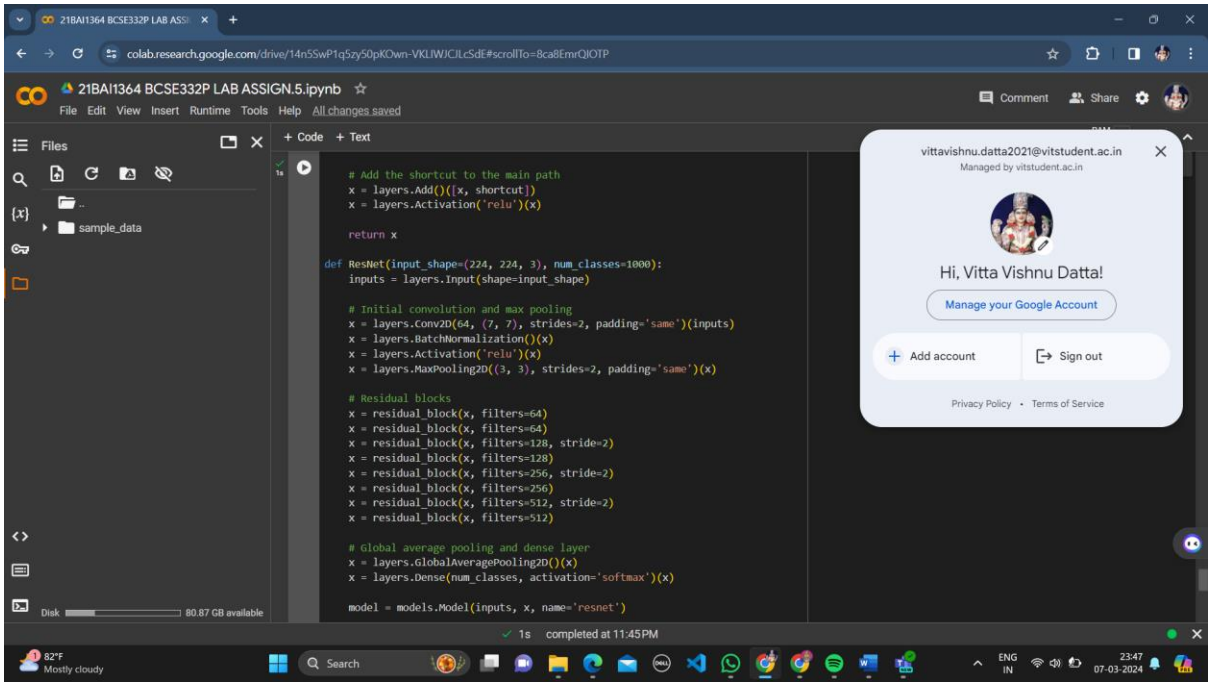


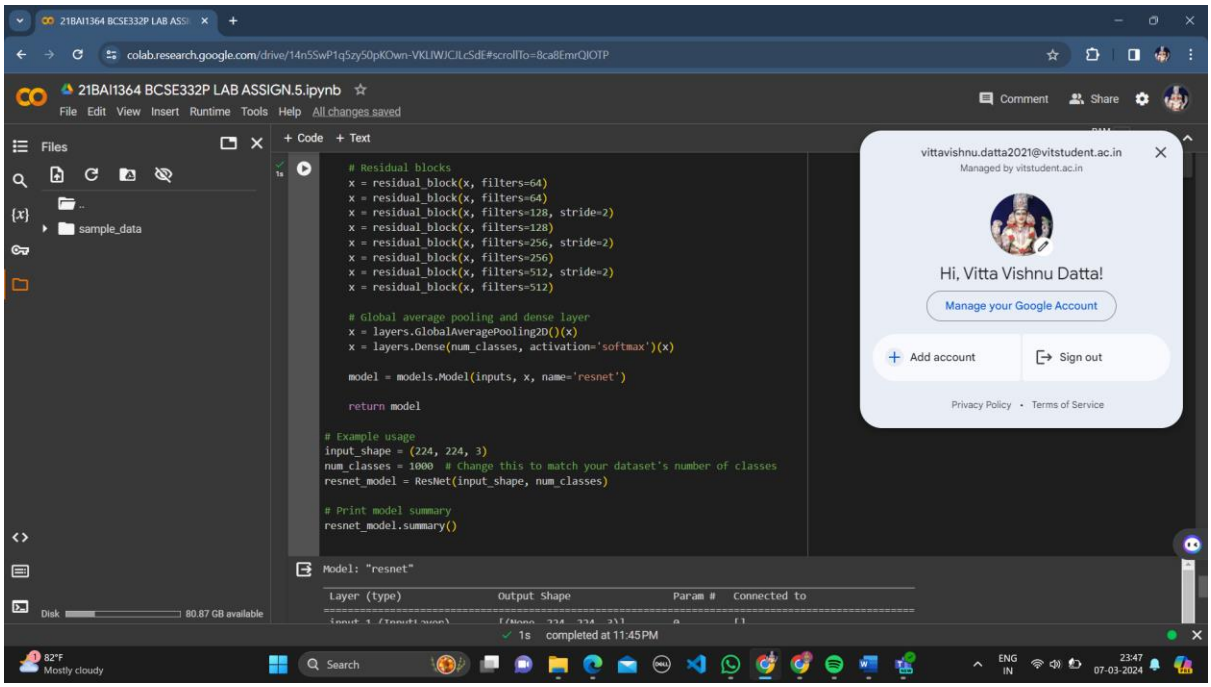
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



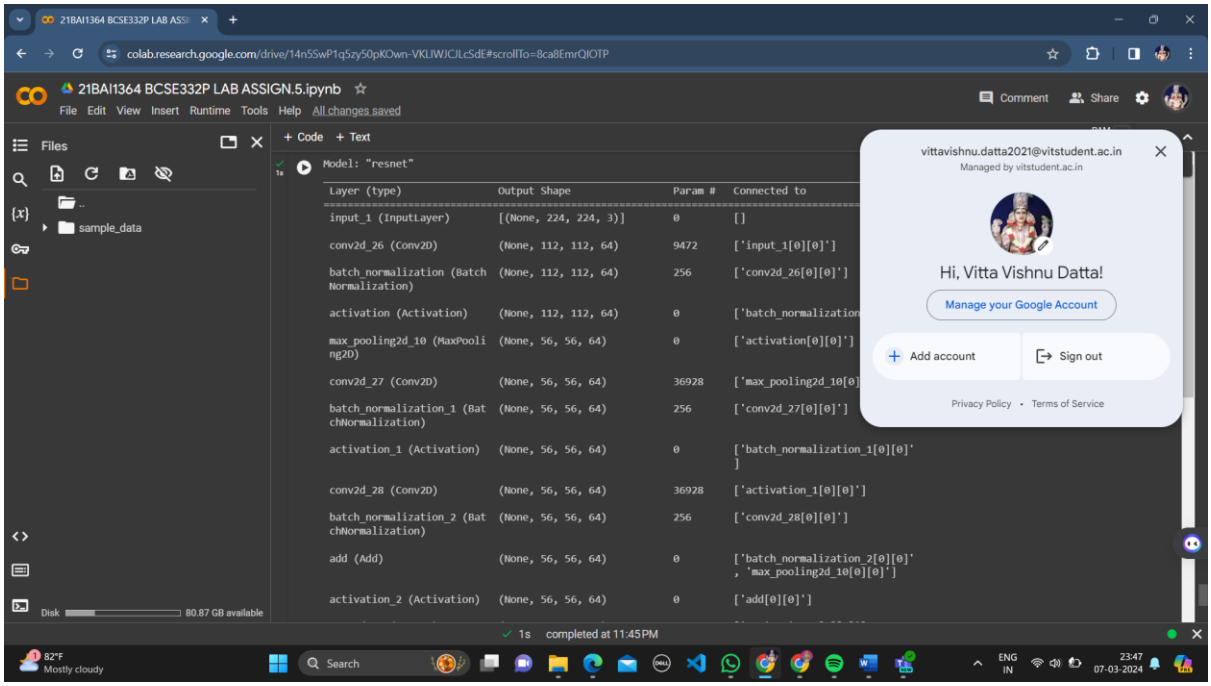


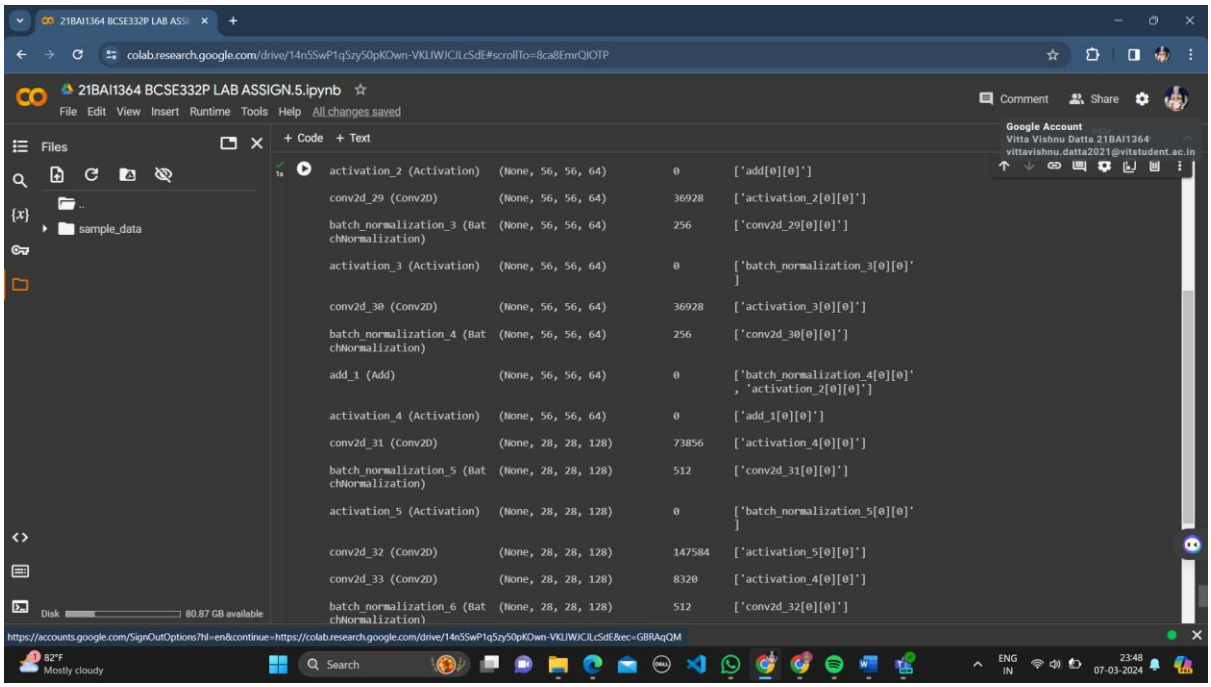
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



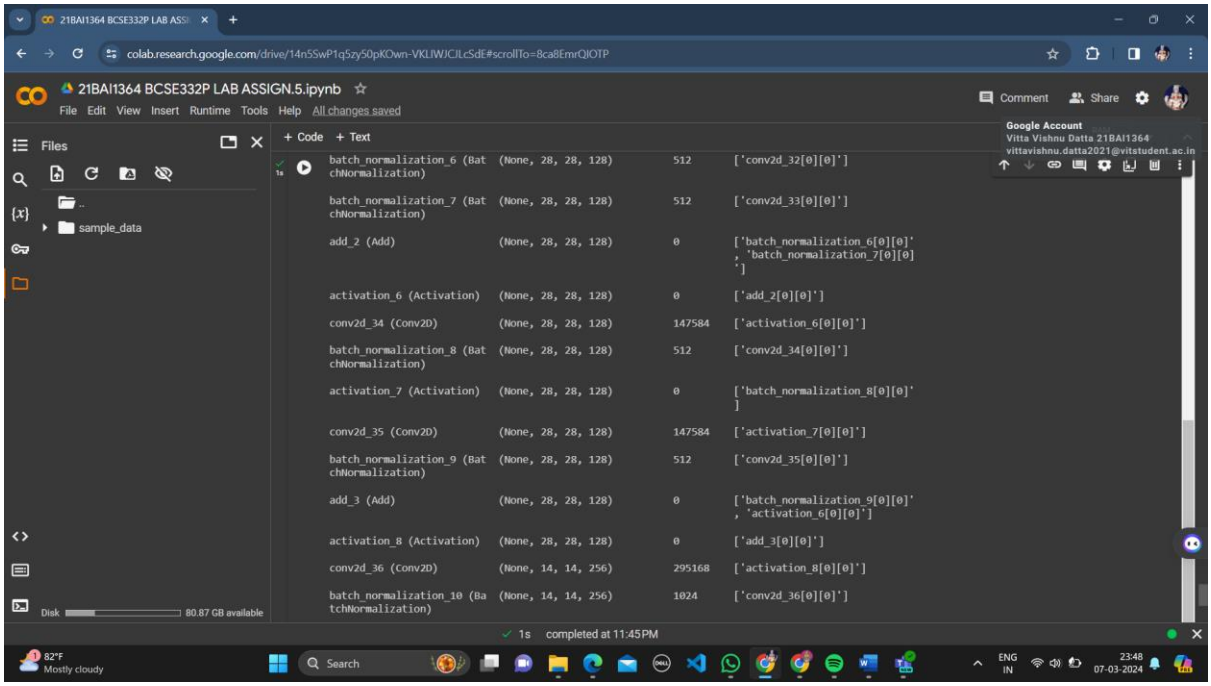


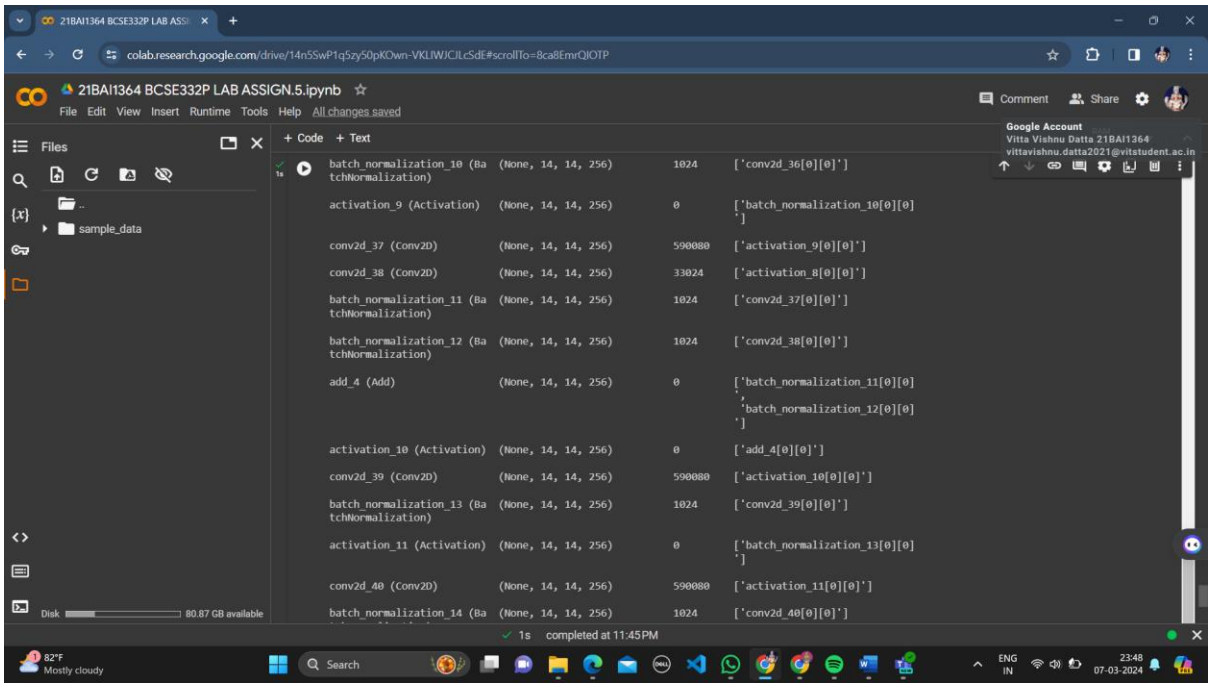
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



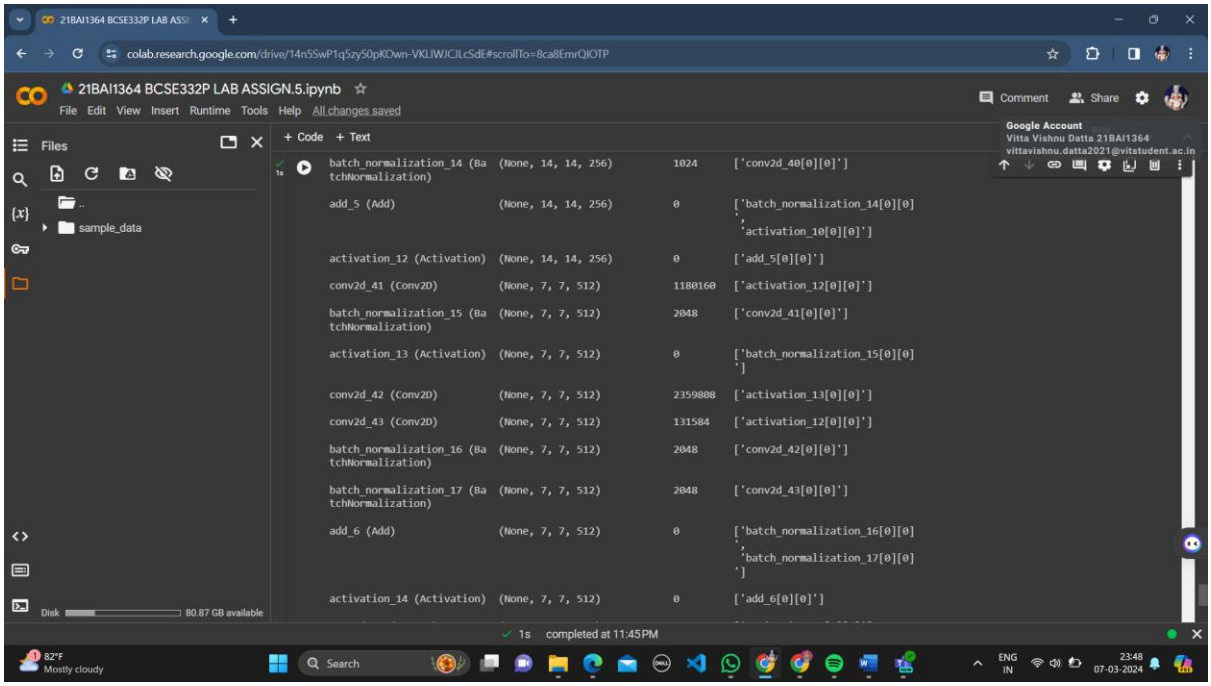


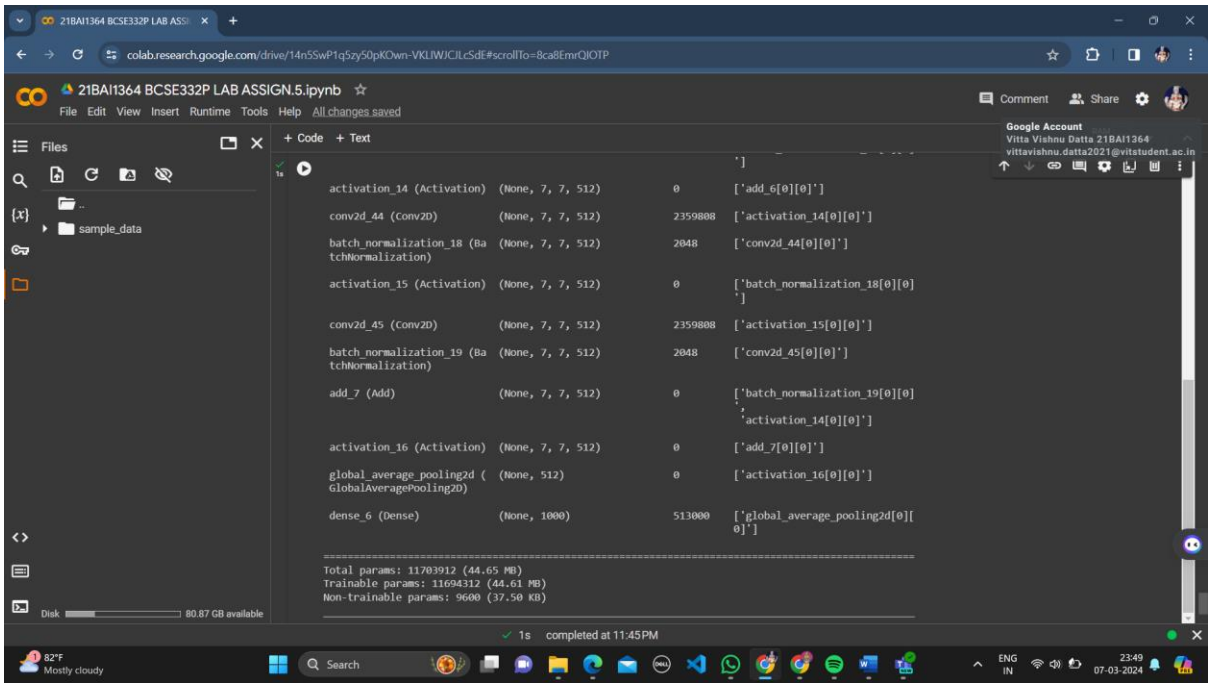
VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI



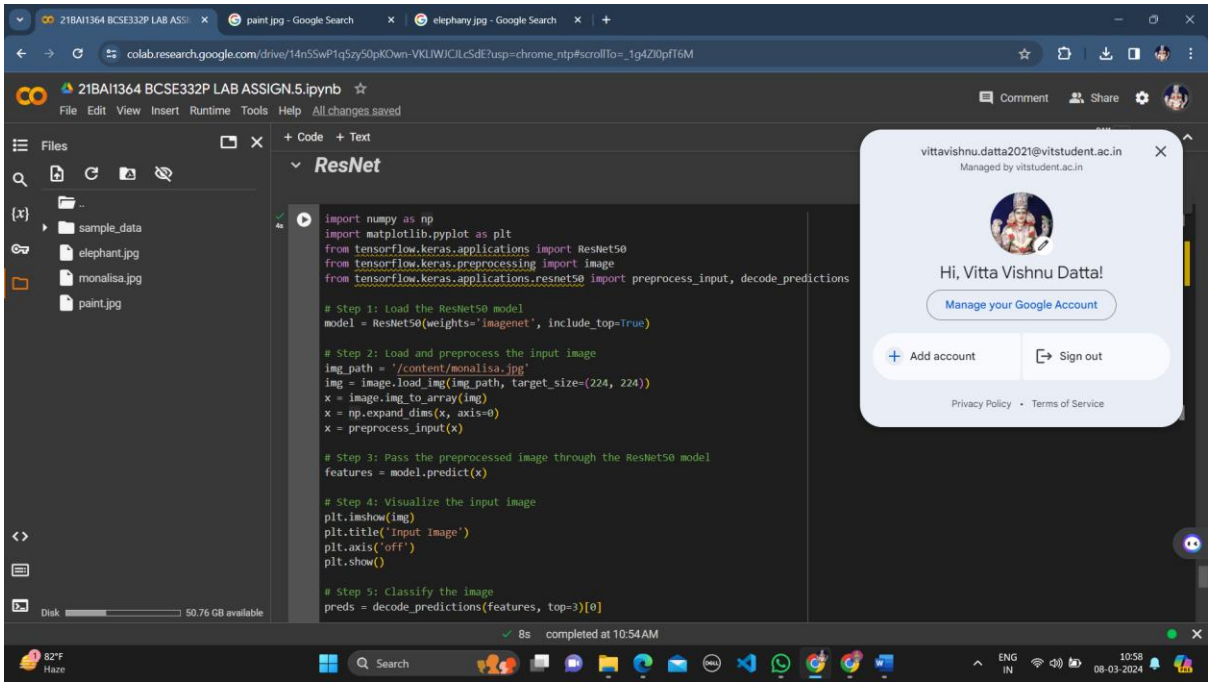


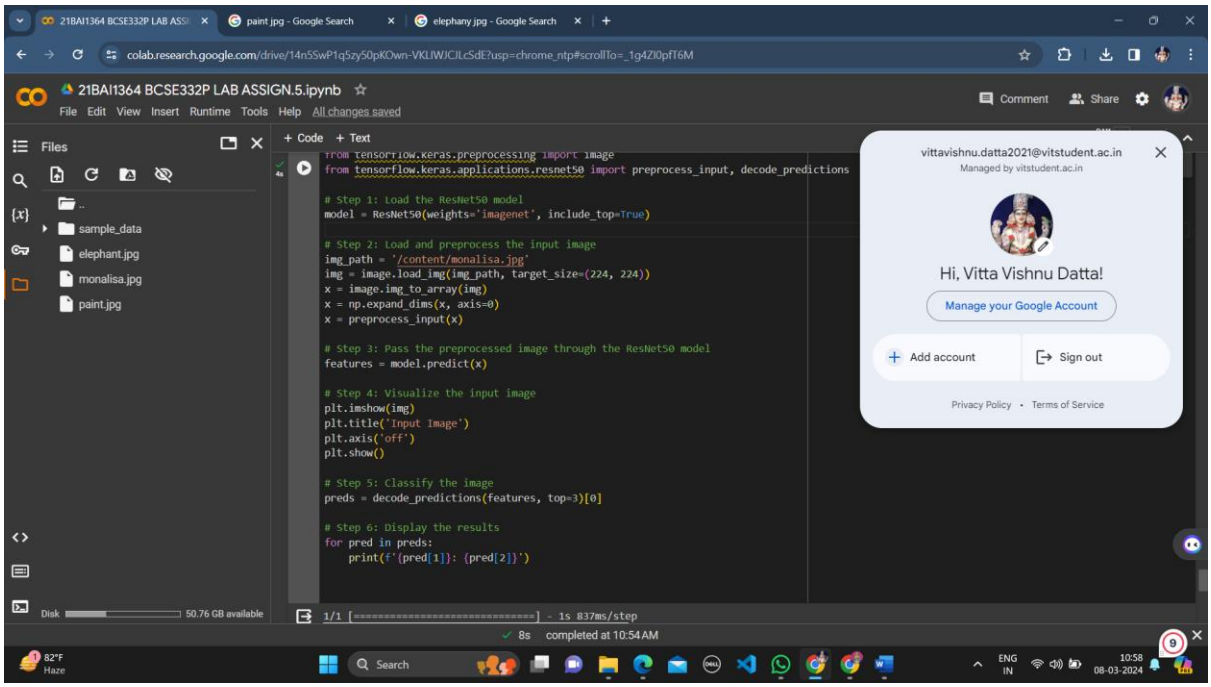
VIT®

Vellore Institute of Technology

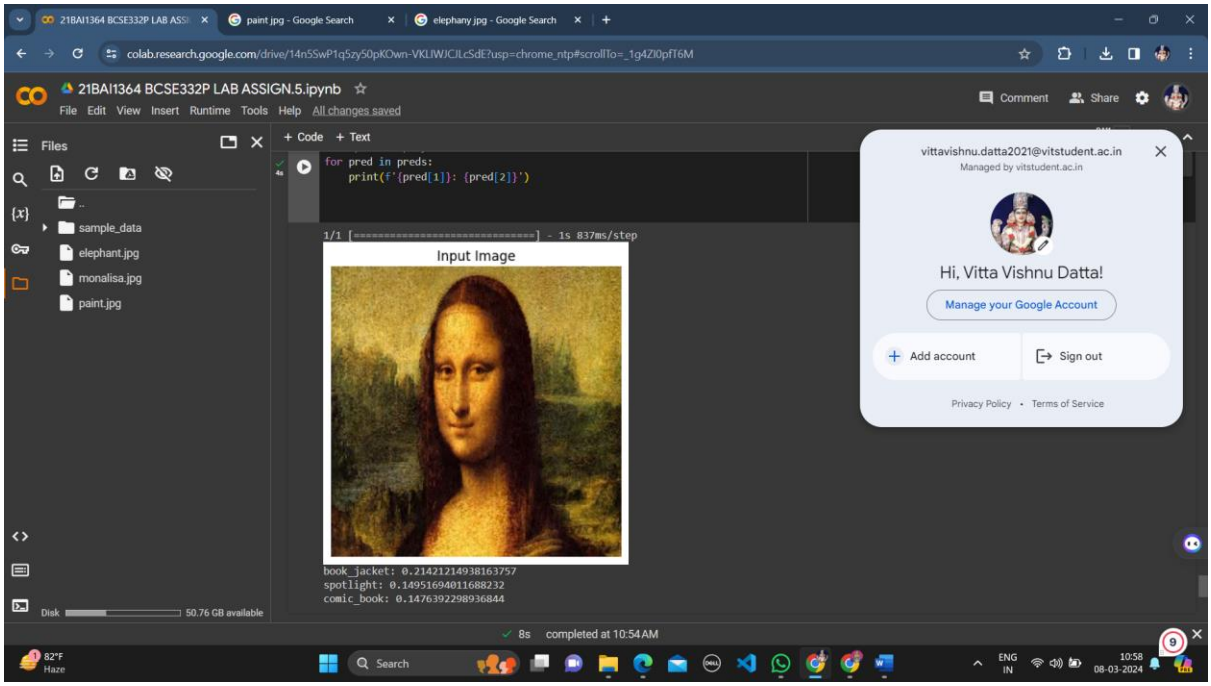
(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI





VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI



COMPARING RESULTS OF VGG16 & GOOGLE NET MODELS:

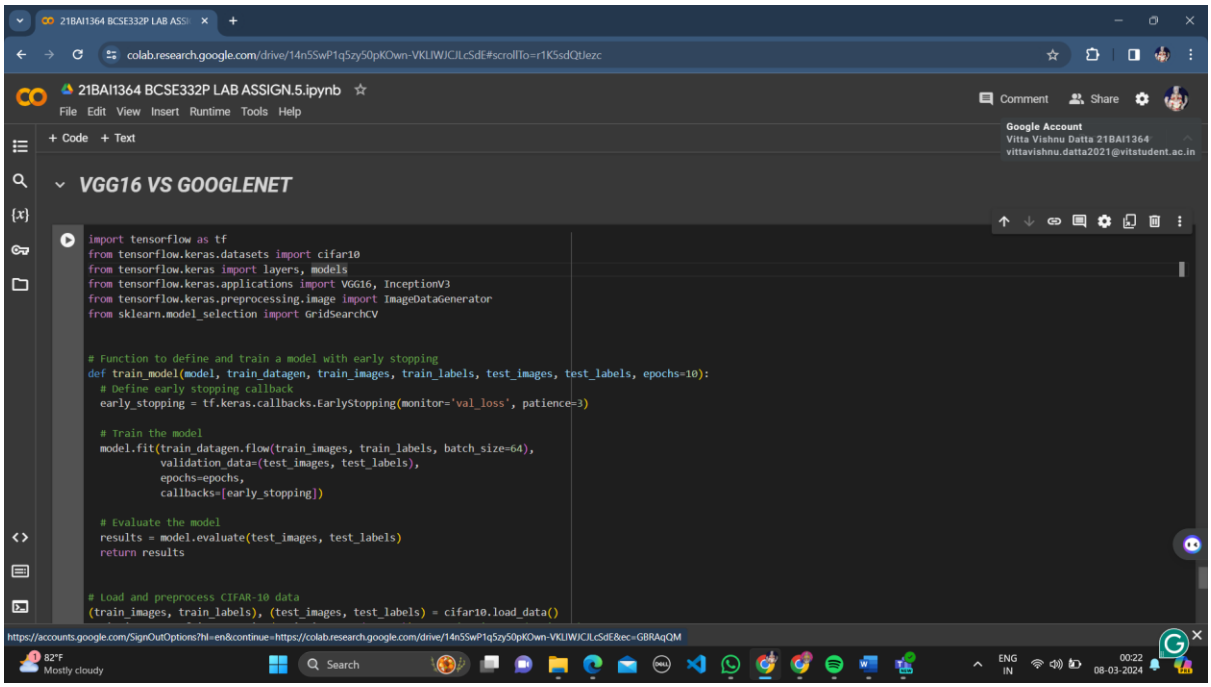
```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
from tensorflow.keras import layers, models
from tensorflow.keras.applications import VGG16, InceptionV3
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import GridSearchCV


# Function to define and train a model with early stopping
def train_model(model, train_datagen, train_images, train_labels,
test_images, test_labels, epochs=10):
  # Define early stopping callback
  early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss',
patience=3)
```

```
  # Train the model
  model.fit(train_datagen.flow(train_images, train_labels,
batch_size=64),
            validation_data=(test_images, test_labels),
            epochs=epochs,
            callbacks=[early_stopping])

  # Evaluate the model
  results = model.evaluate(test_images, test_labels)
  return results


# Load and preprocess CIFAR-10 data
(train_images, train_labels), (test_images, test_labels) =
cifar10.load_data()
train_images = tf.image.resize(train_images, (75, 75))  # Resize images
(Option 1)
test_images = tf.image.resize(test_images, (75, 75))  # Resize images
(Option 1)
train_images, test_images = train_images / 255.0, test_images / 255.0

# Define data augmentation for training
train_datagen = ImageDataGenerator(
    rotation_range=15,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=False,
    fill_mode='nearest'
)

# Define common parameters
input_shape = (75, 75, 3)  # Updated input shape after resizing (Option
1)
num_classes = 10

# Function to define a model based on a pre-trained model choice
def create_model(model_choice, input_shape=input_shape):
  if model_choice == 'VGG16':
    # VGG16 model
    base_model = VGG16(weights='imagenet', include_top=False,
input_shape=input_shape)
```

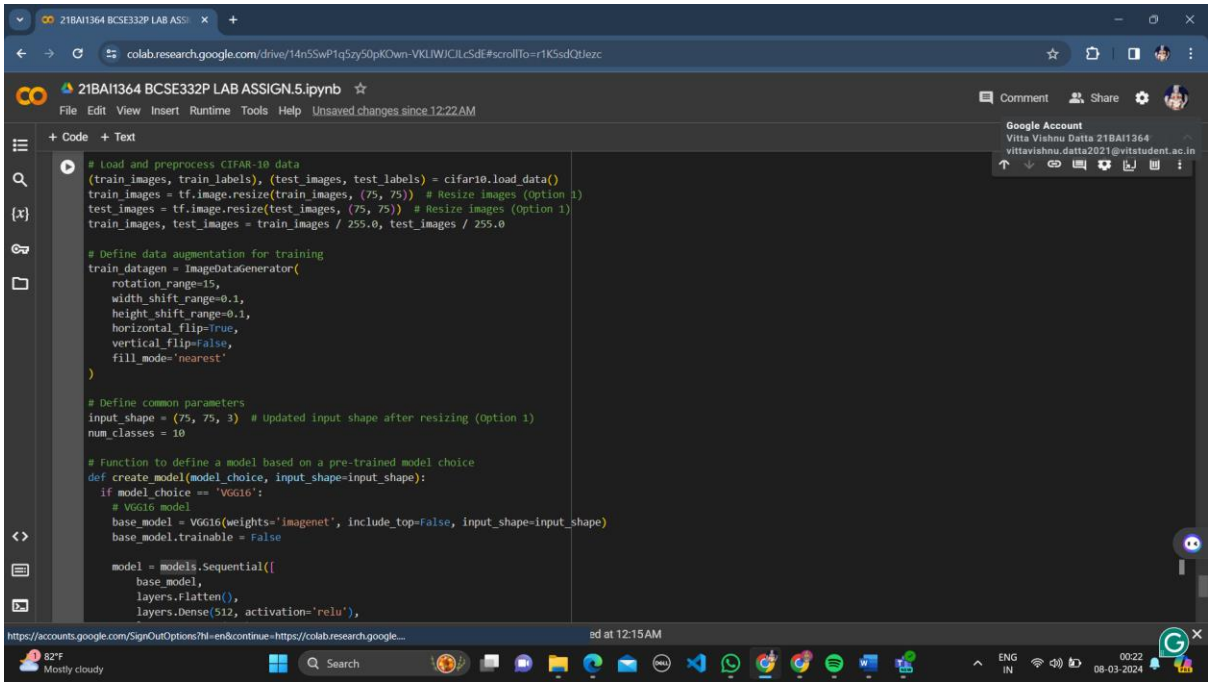
```
    base_model.trainable = False

    model = models.Sequential([
        base_model,
        layers.Flatten(),
        layers.Dense(512, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])
  elif model_choice == 'InceptionV3':

    # Inception (GoogleNet) model
    base_model = InceptionV3(weights='imagenet', include_top=False,
input_shape=input_shape)
    base_model.trainable = False

    model = models.Sequential([
        base_model,
        layers.GlobalAveragePooling2D(),
        layers.Dense(512, activation='relu'),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation='softmax')
    ])
  else:
    raise ValueError(f"Invalid model choice: {model_choice}")
  model.compile(optimizer='adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
  return model


# Define models for comparison (modify list to include others)
model_choices = ['VGG16', 'InceptionV3']

# Hyperparameter tuning with GridSearchCV (optional)
param_grid = {
    'epochs': [10, 20, 30],
    'learning_rate': [0.001, 0.0001]
}

for model_choice in model_choices:
  print(f"\nEvaluating Model: {model_choice}")

  # Create the model
```

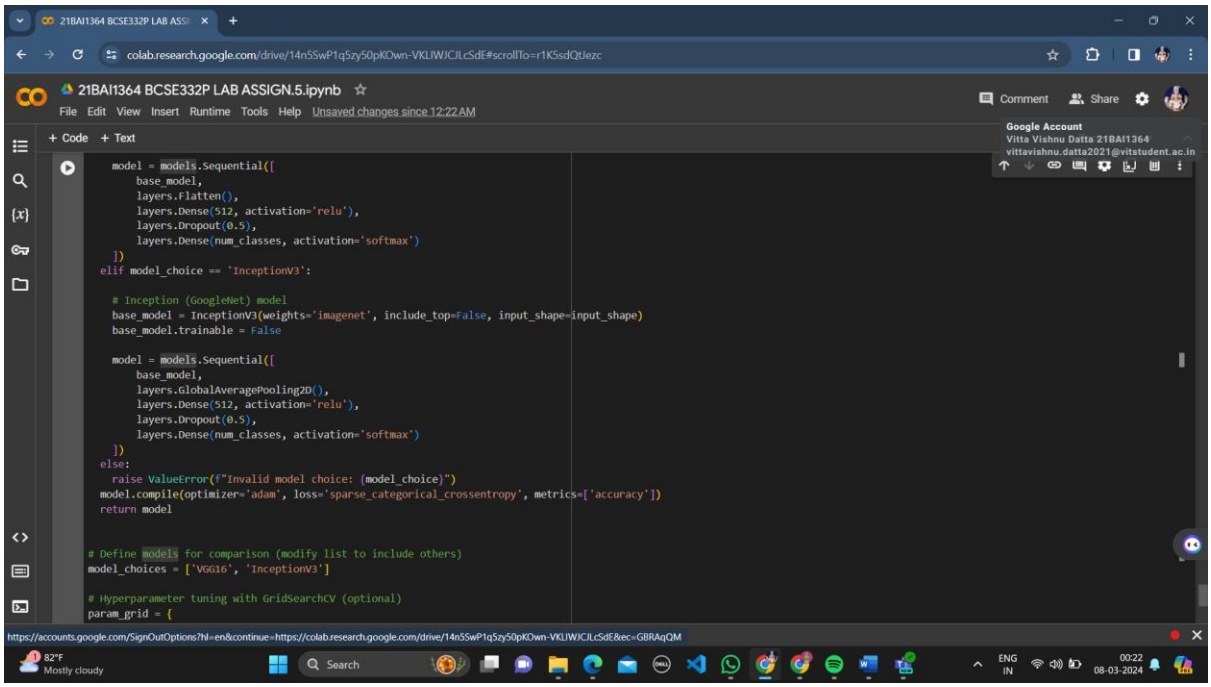
```
  model = create_model(model_choice)

  # Hyperparameter tuning with GridSearchCV (uncomment to activate)
  # grid_search = GridSearchCV(model, param_grid, cv=3)
  # grid_search.fit(train_datagen.flow(train_images, train_labels,
batch_size=64), epochs=epochs)
  # means = grid_search.cv_results_['mean_test_score']
  # std = grid_search.cv_results_['std
```

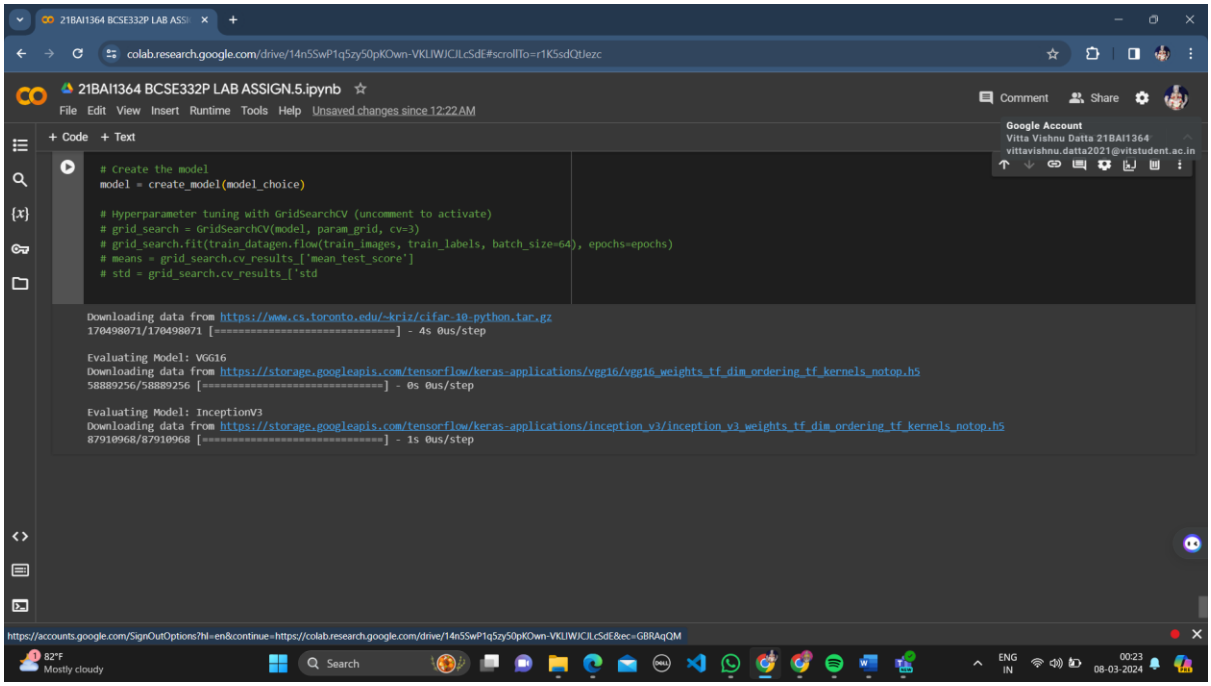
SCREENSHOTS OF THE CODE SNIPPET:



VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)
CHENNAI

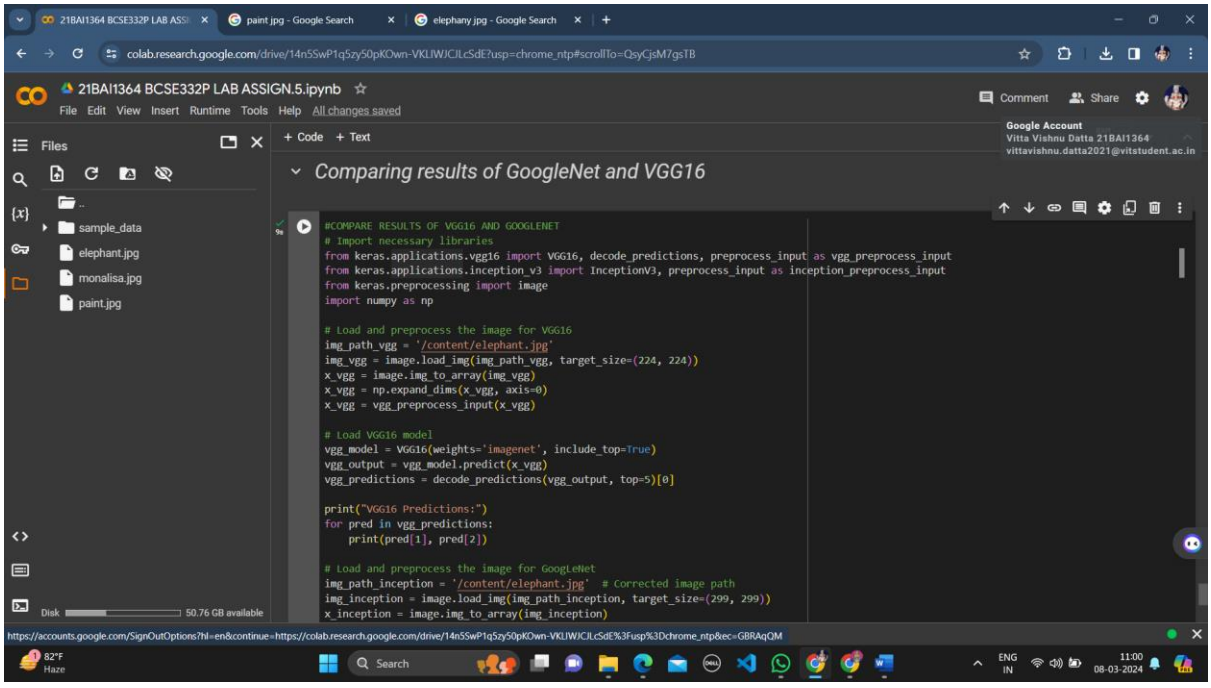







```python
#COMPARE RESULTS OF VGG16 AND GOOGLENET
# Import necessary libraries
from keras.applications.vgg16 import VGG16, decode_predictions,
preprocess_input as vgg_preprocess_input
from keras.applications.inception_v3 import InceptionV3,
preprocess_input as inception_preprocess_input
from keras.preprocessing import image
import numpy as np

# Load and preprocess the image for VGG16
img_path_vgg = '/content/elephant.jpg'
img_vgg = image.load_img(img_path_vgg, target_size=(224, 224))
x_vgg = image.img_to_array(img_vgg)
x_vgg = np.expand_dims(x_vgg, axis=0)
x_vgg = vgg_preprocess_input(x_vgg)

# Load VGG16 model
vgg_model = VGG16(weights='imagenet', include_top=True)
vgg_output = vgg_model.predict(x_vgg)
vgg_predictions = decode_predictions(vgg_output, top=5)[0]

print("VGG16 Predictions:")
```

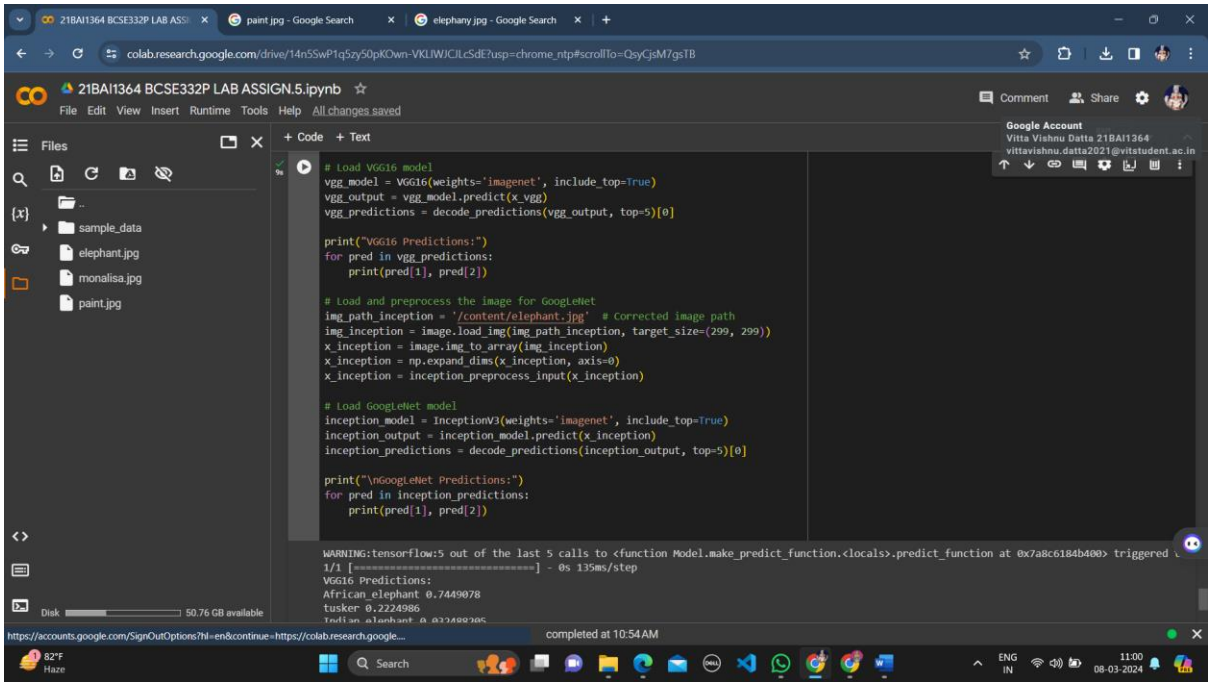
```
for pred in vgg_predictions:
    print(pred[1], pred[2])

# Load and preprocess the image for GoogLeNet
img_path_inception = '/content/elephant.jpg'  # Corrected image path
img_inception = image.load_img(img_path_inception, target_size=(299,
299))
x_inception = image.img_to_array(img_inception)
x_inception = np.expand_dims(x_inception, axis=0)
x_inception = inception_preprocess_input(x_inception)

# Load GoogLeNet model
inception_model = InceptionV3(weights='imagenet', include_top=True)
inception_output = inception_model.predict(x_inception)
inception_predictions = decode_predictions(inception_output, top=5)[0]

print("\nGoogLeNet Predictions:")
for pred in inception_predictions:
    print(pred[1], pred[2])
```

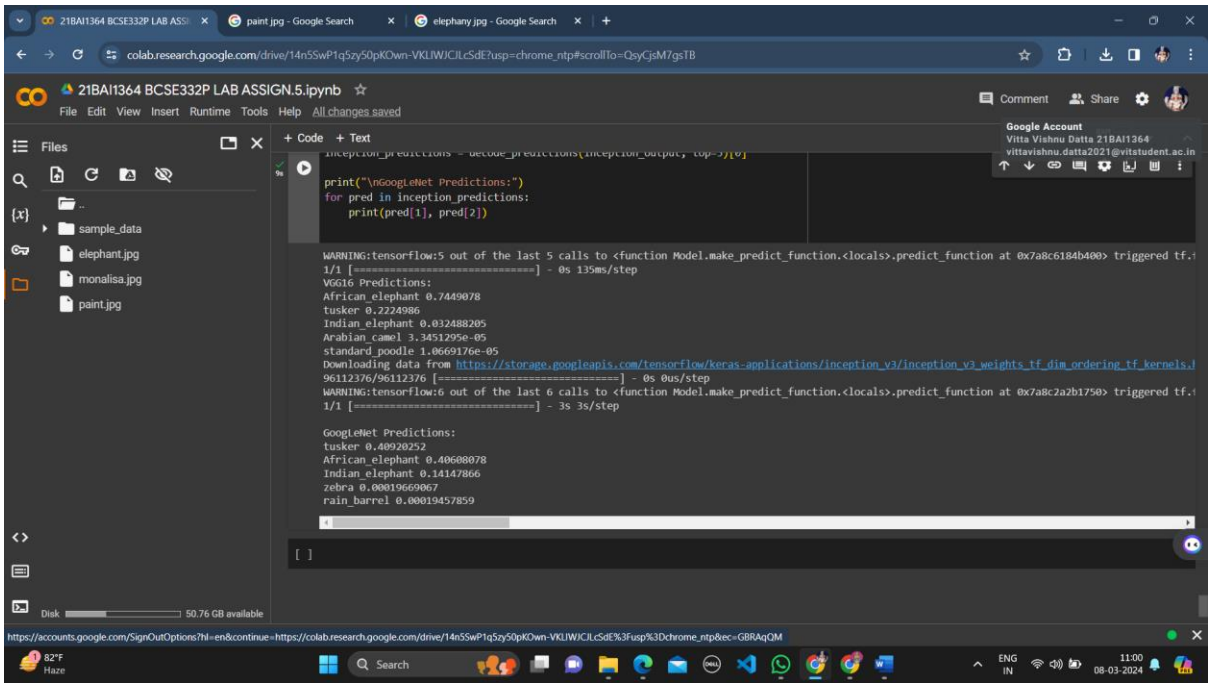


Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI





VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

PREPARED AND SUBMITTED BY:

VITTA VISHNU DATTA.

21BAI1364 – SCOPE.

VIT – CHENNAI CAMPUS.