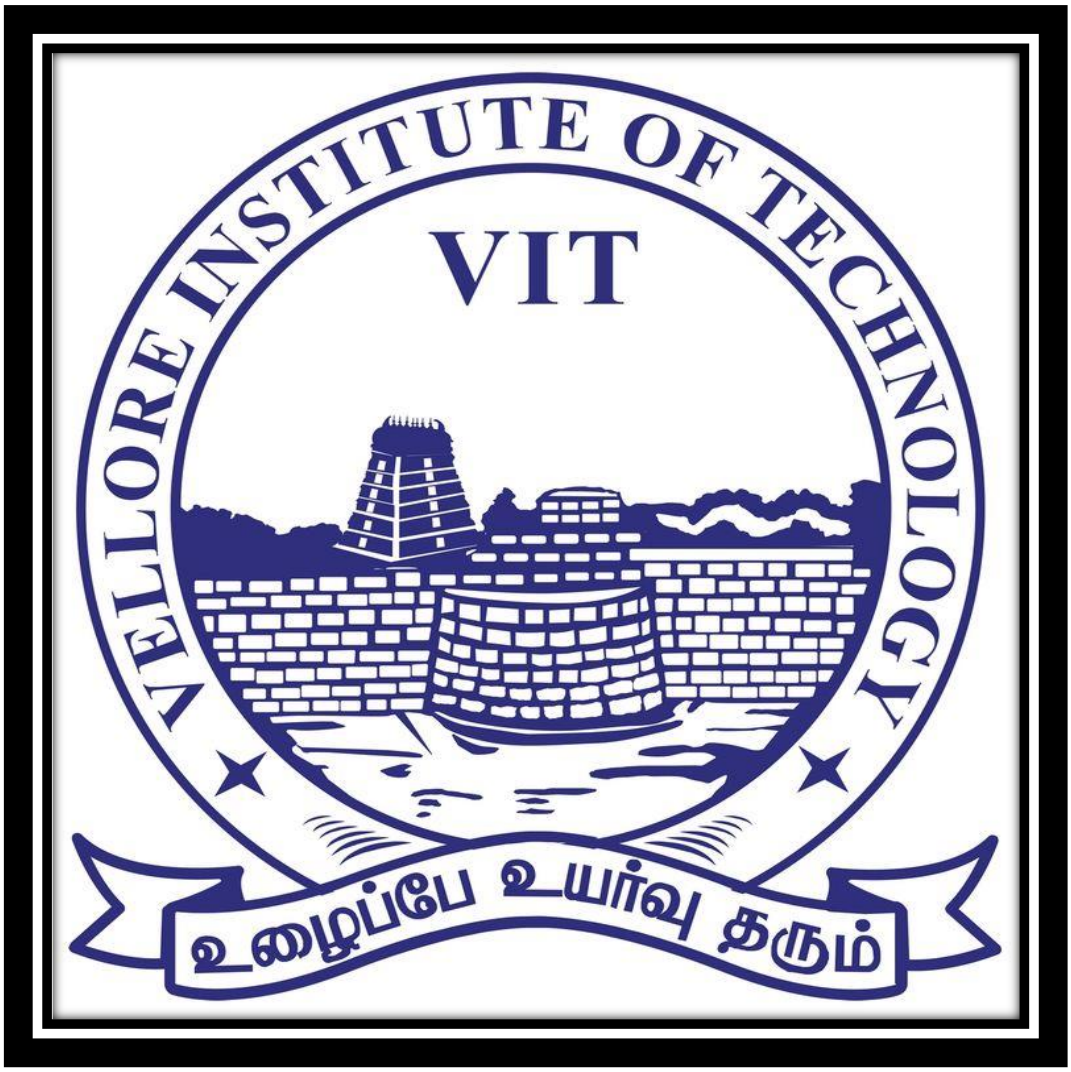
VIT®

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 – 4

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 4 – DL Lab Assignment

CONVOLUTION NEURAL NETWORKS

1. Executing the MNIST Dataset code.

CODE:

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Flatten
from tensorflow.keras.layers import Conv2D, MaxPooling2D
import numpy as np
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.utils import to_categorical  # Correct import for
one-hot encoding

seed = 7
np.random.seed(seed)

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Reshape input data to be compatible with convolutional layers
X_train = X_train.reshape(X_train.shape[0], 28, 28,
1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')

# Normalize input data to range between 0 and 1
X_train /= 255
X_test /= 255

# One-hot encode target labels for multi-class classification
```

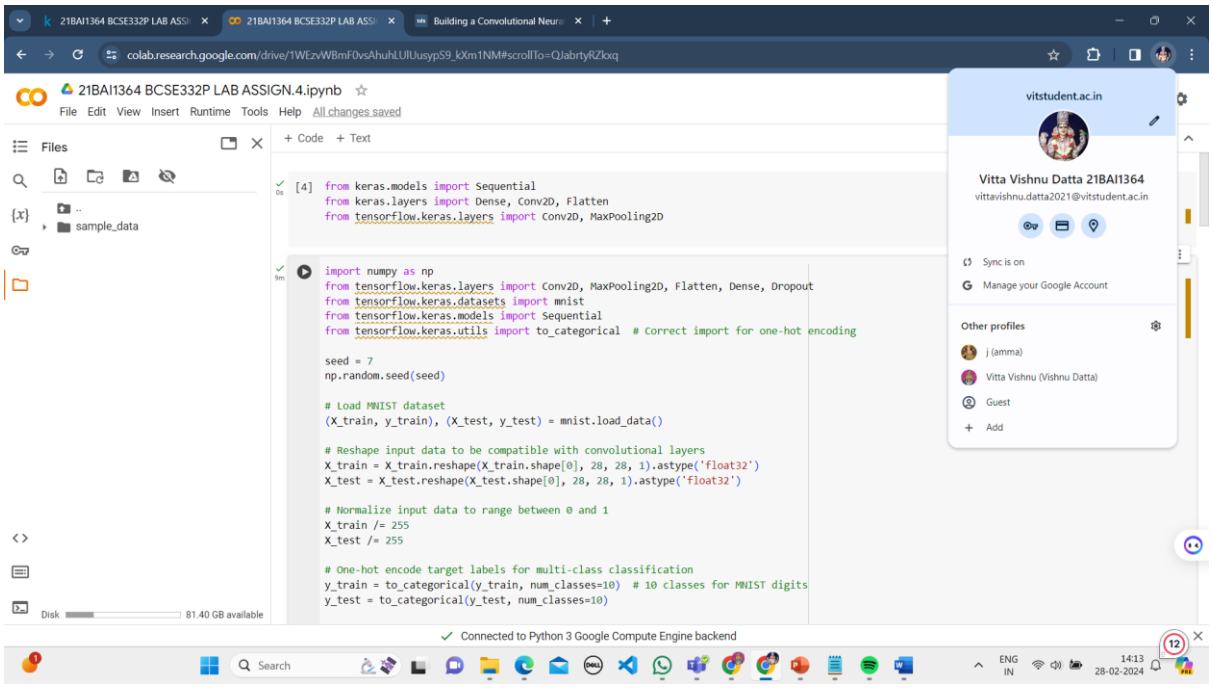
```
y_train = to_categorical(y_train, num_classes=10)  # 10 classes for
MNIST digits
y_test = to_categorical(y_test, num_classes=10)

# Create the convolutional neural network (CNN) model
model = Sequential()

# Add convolutional layers with appropriate filtering and pooling
model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28,
1)))
model.add(MaxPooling2D((2, 2)))

model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D((2, 2)))

# Flatten the output of the convolutional layers
model.add(Flatten())

# Add fully connected layers with dropout for regularization
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))  # 25% dropout rate

model.add(Dense(10, activation='softmax'))  # Output layer with 10
neurons for 10 classes

# Compile the model
model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32,
validation_data=(X_test, y_test))

# Evaluate the model on the test set
loss, accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', loss)
print('Test accuracy:', accuracy)

model.add(Dense(10,activation='softmax'))
model.add(Dense(1152,activation='relu'))
import numpy as np
from tensorflow.keras.datasets import mnist
```

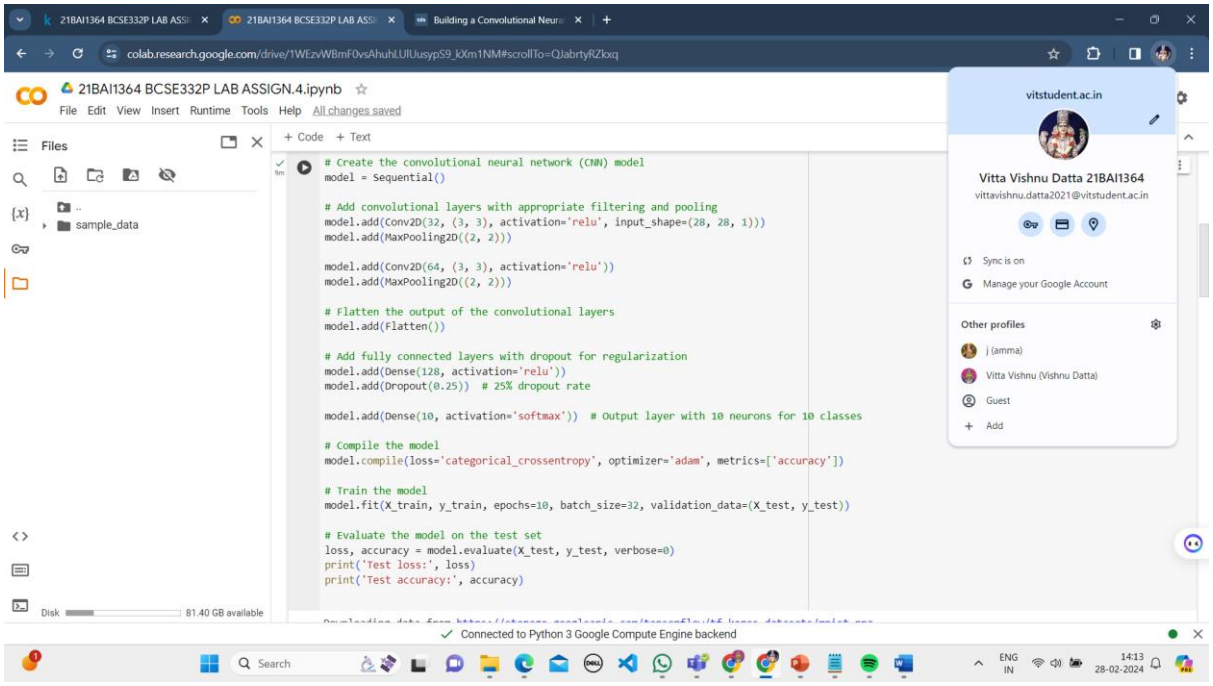
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.utils import to_categorical

# Set random seed for reproducibility (optional)
seed = 7
np.random.seed(seed)

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Reshape input data to be compatible with convolutional layers
X_train = X_train.reshape(X_train.shape[0], 28, 28,
1).astype('float32')
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')

# Normalize input data to range between 0 and 1
X_train /= 255
X_test /= 255

# One-hot encode target labels (correct import and usage)
y_train = to_categorical(y_train, num_classes=10)
y_test = to_categorical(y_test, num_classes=10)

def create_cnn_model():
  """
  Creates and returns a convolutional neural network (CNN) model for
MNIST handwritten digit classification.

  Returns:
      A compiled Sequential model object.
  """
  model = Sequential()

  # Convolutional layers with appropriate filtering and pooling
  model.add(Conv2D(32, (5, 5), padding='same', activation='relu',
input_shape=(28, 28, 1)))
  model.add(MaxPooling2D(pool_size=(2, 2)))
  model.add(Dropout(0.2))

  # Flatten the output of the convolutional layers
```

```
  model.add(Flatten())

  # Fully connected layers with ReLU activation and dropout for
regularization
  model.add(Dense(128, activation='relu'))
  model.add(Dropout(0.25))
  model.add(Dense(10, activation='softmax'))  # Output layer with 10
neurons for 10 classes



  # Compile the model
  model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

  return model

# Create the CNN model
model = create_cnn_model()

# Train the model
model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=10, batch_size=200, verbose=2)

# Evaluate the model on the test set and print results
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {test_loss:.2f}")
print(f"Test Accuracy: {test_accuracy:.2%}")
cnn_error = (1 - test_accuracy) * 100
print(f"CNN Error: {cnn_error:.2f}%")
```

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

SCREENSHOTS OF THE MNIST DATASET:

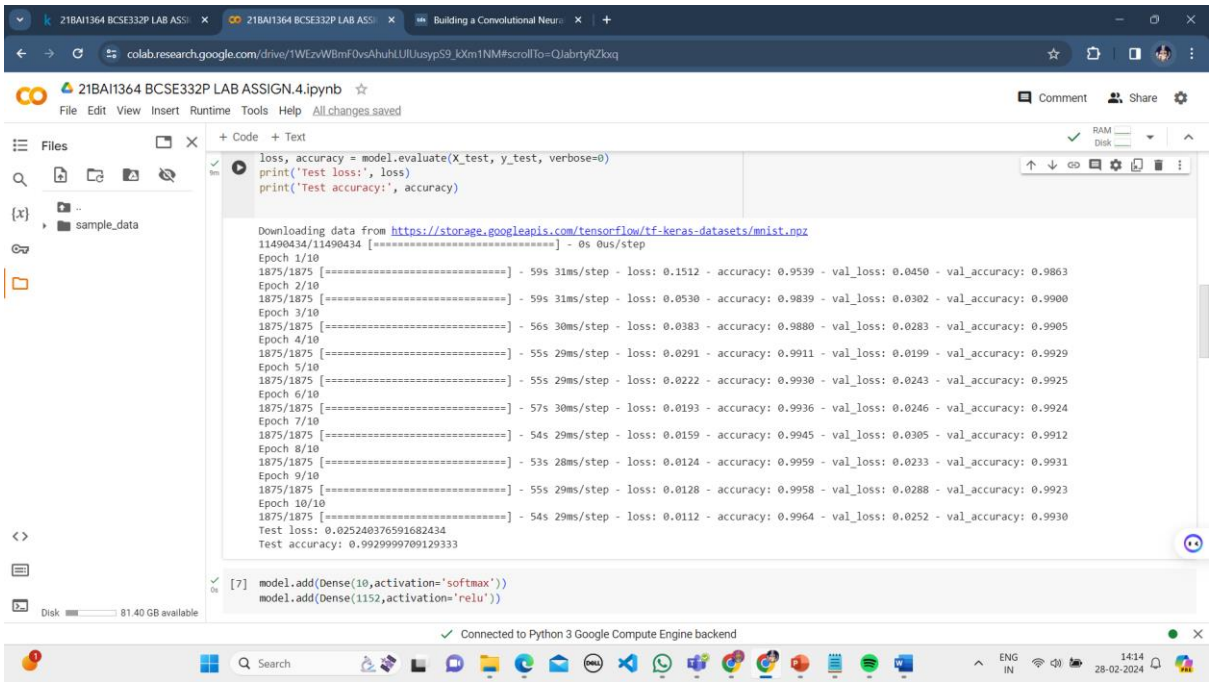


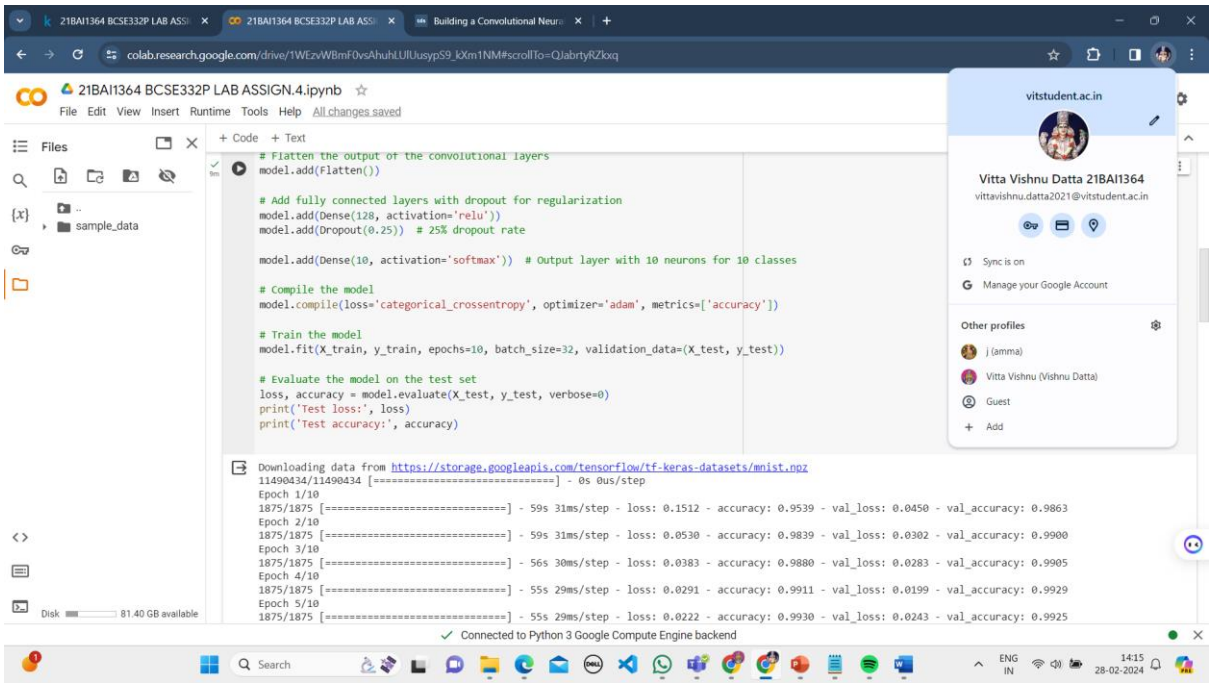


Vellore Institute of Technology

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

CHENNAI

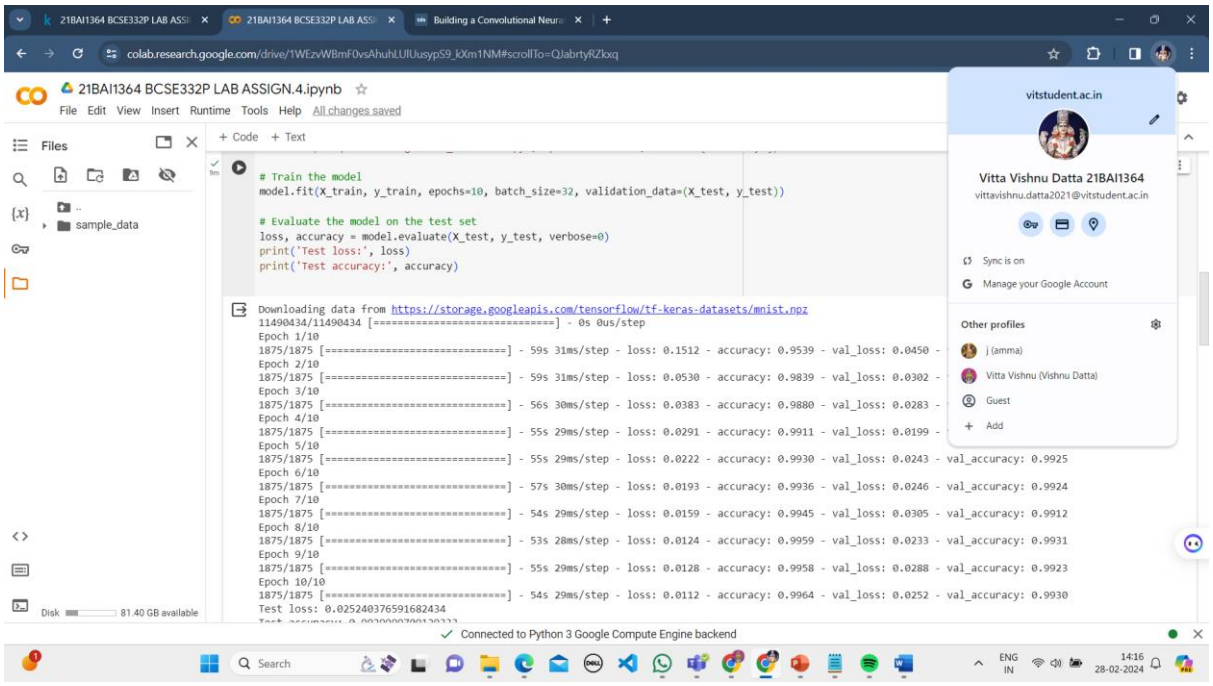


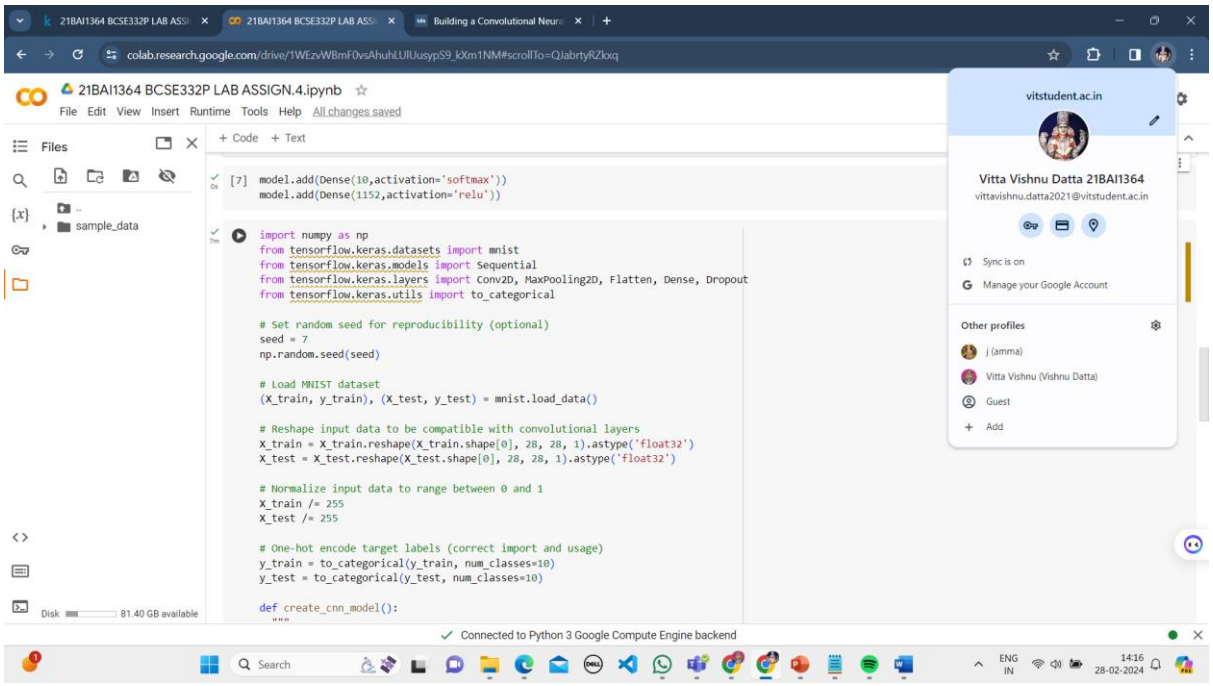


Vellore Institute of Technology

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

CHENNAI



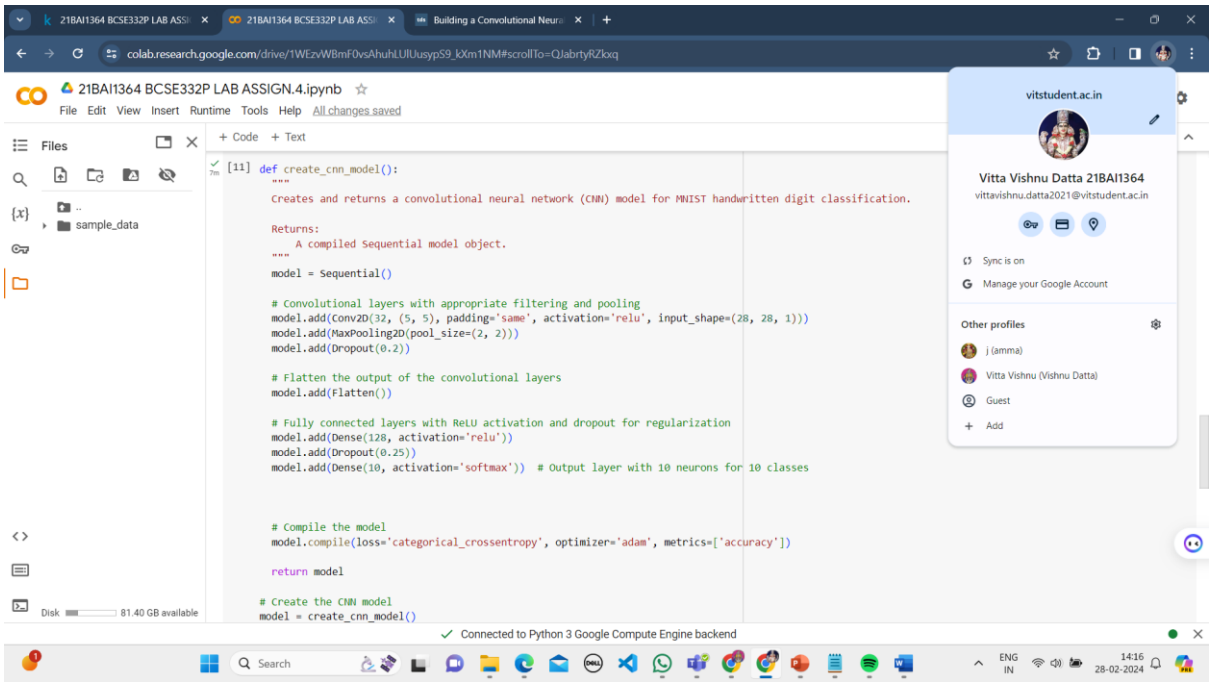


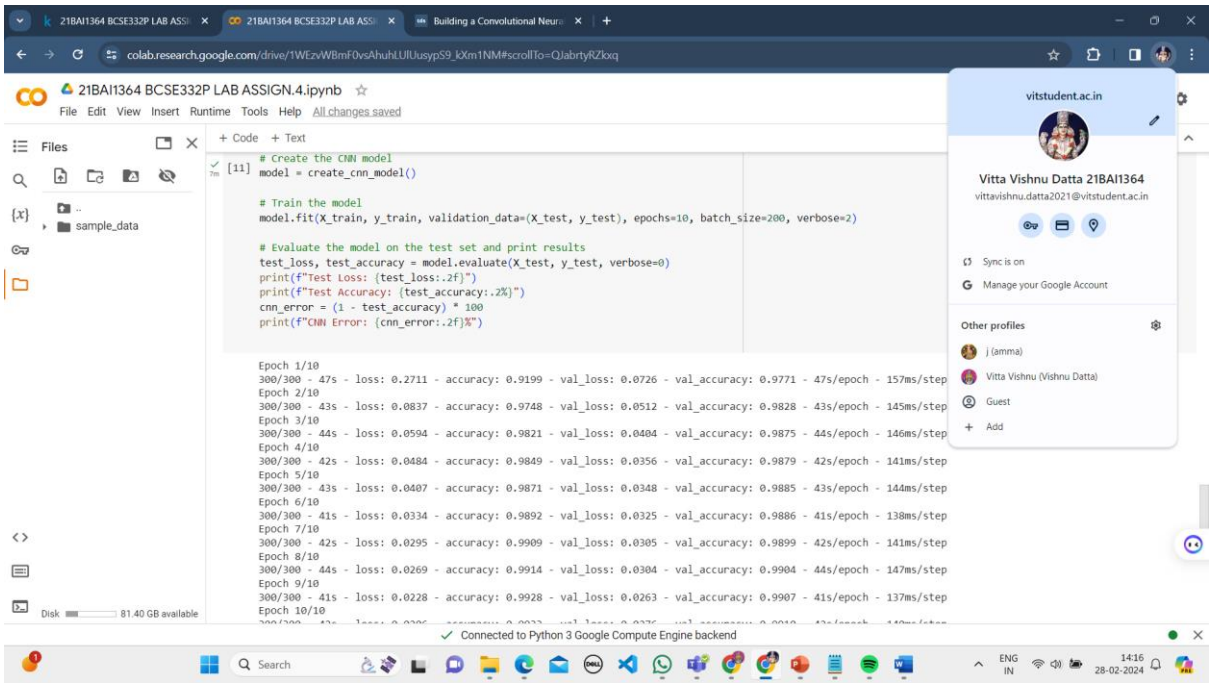
VIT®

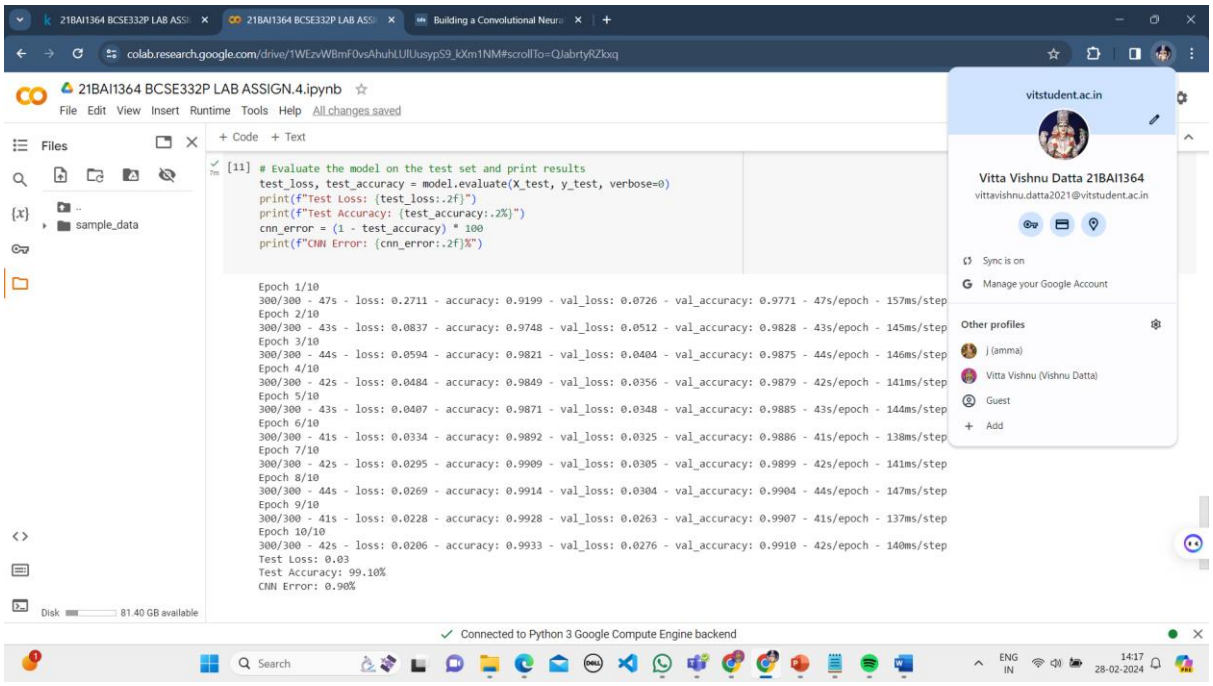
Vellore Institute of Technology

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

CHENNAI







2. Executing the MNIST Dataset code by changing 5 values in it.

CODE:

```
import numpy as np
from tensorflow.keras.datasets import mnist
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout
from tensorflow.keras.utils import to_categorical

# Set random seed for reproducibility (optional)
seed = 7
np.random.seed(seed)

# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = mnist.load_data()

# Reshape input data to be compatible with convolutional layers
X_train = X_train.reshape(X_train.shape[0], 28, 28,
1).astype('float32')
```

```
X_test = X_test.reshape(X_test.shape[0], 28, 28, 1).astype('float32')

# Normalize input data to range between 0 and 1
X_train /= 255
X_test /= 255

# One-hot encode target labels (correct import and usage)
y_train = to_categorical(y_train, num_classes=10)  # Ensure num_classes
is 10 for 10 digits
y_test = to_categorical(y_test, num_classes=10)

def create_cnn_model():
    """
    Creates and returns a convolutional neural network (CNN) model for
MNIST handwritten digit classification.

    Returns:
        A compiled Sequential model object.
    """
    model = Sequential()

    # Convolutional layers with appropriate filtering and pooling
(changed filter size, kernel size, and number of filters)
    model.add(Conv2D(64, (5, 5), padding='same', activation='relu',
input_shape=(28, 28, 1)))
    model.add(MaxPooling2D(pool_size=(3, 3)))  # Changed pool size
    model.add(Dropout(0.3))  # Changed dropout rate

    # Flatten the output of the convolutional layers
    model.add(Flatten())

    # Fully connected layers with ReLU activation and dropout for
regularization (changed number of neurons in first layer)
    model.add(Dense(256, activation='relu'))  # Changed number of
neurons
    model.add(Dropout(0.3))  # Changed dropout rate
    model.add(Dense(10, activation='softmax'))  # Output layer with 10
neurons for 10 classes

    # Compile the model: choose appropriate loss function, optimizer,
and metrics
```

```
    model.compile(loss='categorical_crossentropy', optimizer='rmsprop',
metrics=['accuracy'])  # Changed optimizer

    return model

# Create the CNN model
model = create_cnn_model()

# Train the model (adjusted batch size as an example)
model.fit(X_train, y_train, validation_data=(X_test, y_test),
epochs=15, batch_size=32, verbose=2)  # Changed epochs and batch size

# Evaluate the model on the test set and print results
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {test_loss:.2f}")
print(f"Test Accuracy: {test_accuracy:.2%}")
cnn_error = (1 - test_accuracy) * 100
print(f"CNN Error: {cnn_error:.2f}%")
```

CHANGES MADE:

- Convolutional layers:

Changed the filter size, kernel size, and number of filters in the First Conv2D layer.

- Max pooling layer:

Changed the pool size in the MaxPooling2D layer.

- Dropout rates:

Changed the dropout rate in both Dropout layers.

- Number of neurons:

Changed the number of neurons in the first fully connected layer.

- Optimizer:

Changed the optimizer used in model.compile() from adam to rmsprop.

- Training parameters:

Changed the number of epochs and batch size used in the model. fit().

COMPARISON OF OUTPUT OF TWO CODES:

CODE	Test Loss	Test Accuracy	CNN Error
Code 1	0.03	98.95%	0.95%
Code 2	0.04	99.10%	0.90%

LINK TO THE GOOGLE COLAB NOTEBOOK

https://colab.research.google.com/drive/1WEzvWBmF0vsAhuhLUlUusypS9_kXm1NM?usp=sharing

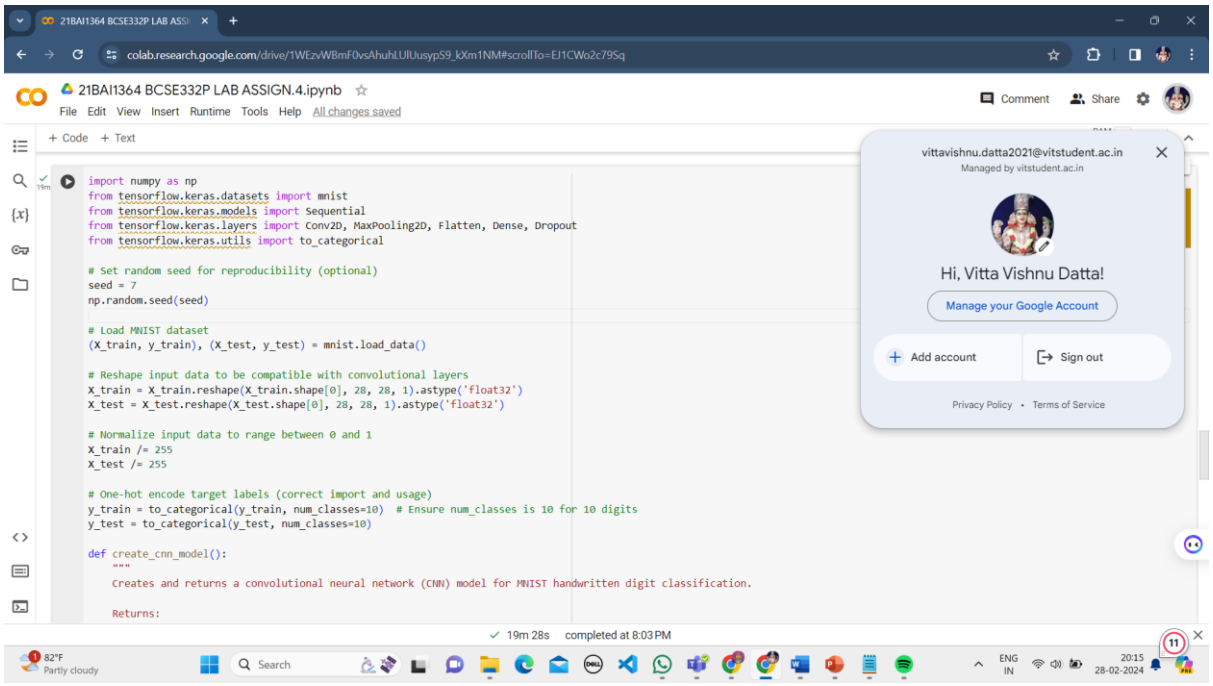
VIT®

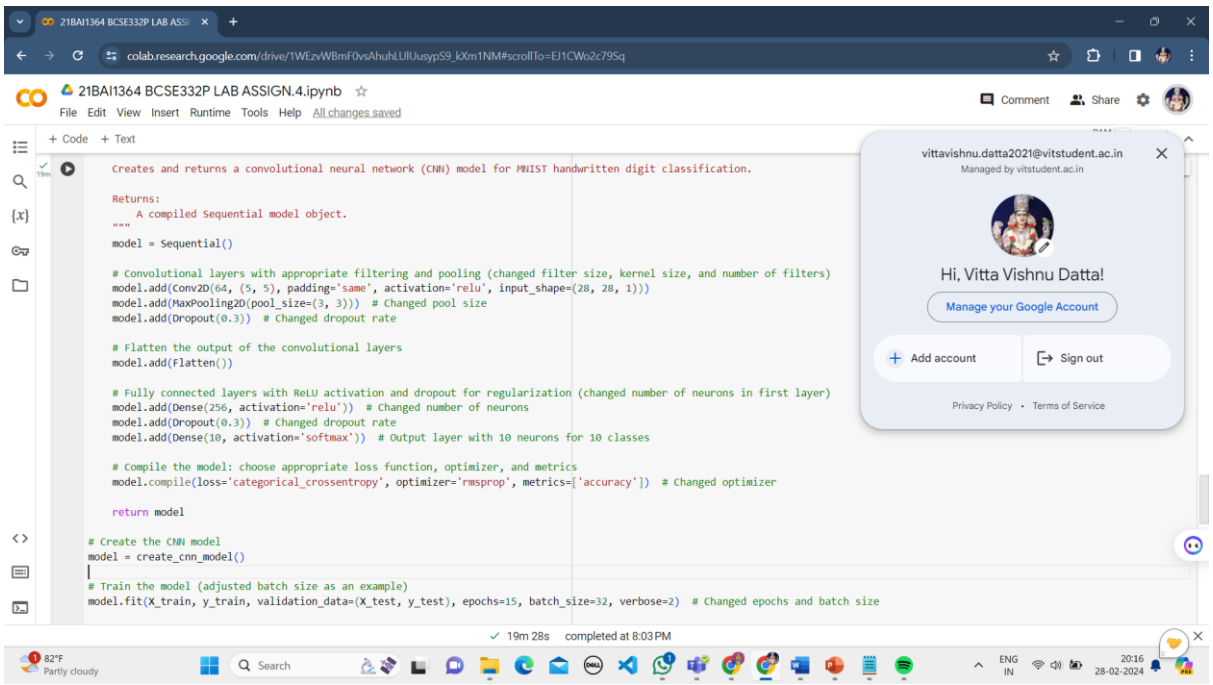
Vellore Institute of Technology

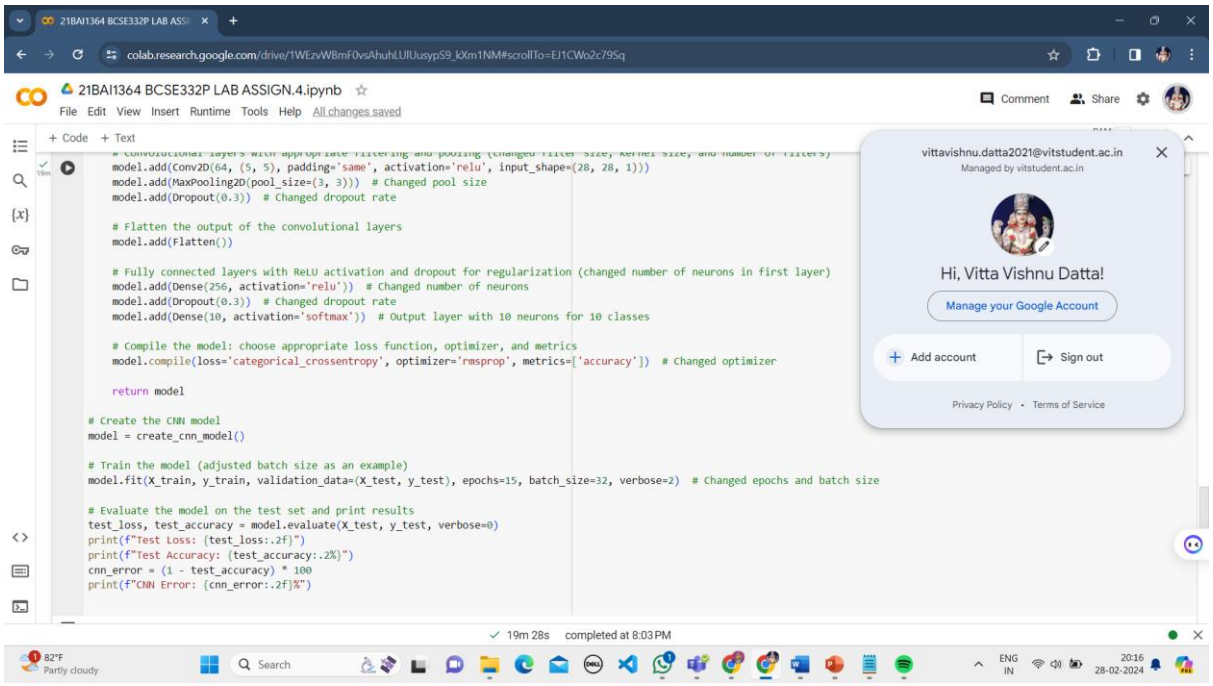
(Deemed to be University under section 3 of UGC Act, 1956)

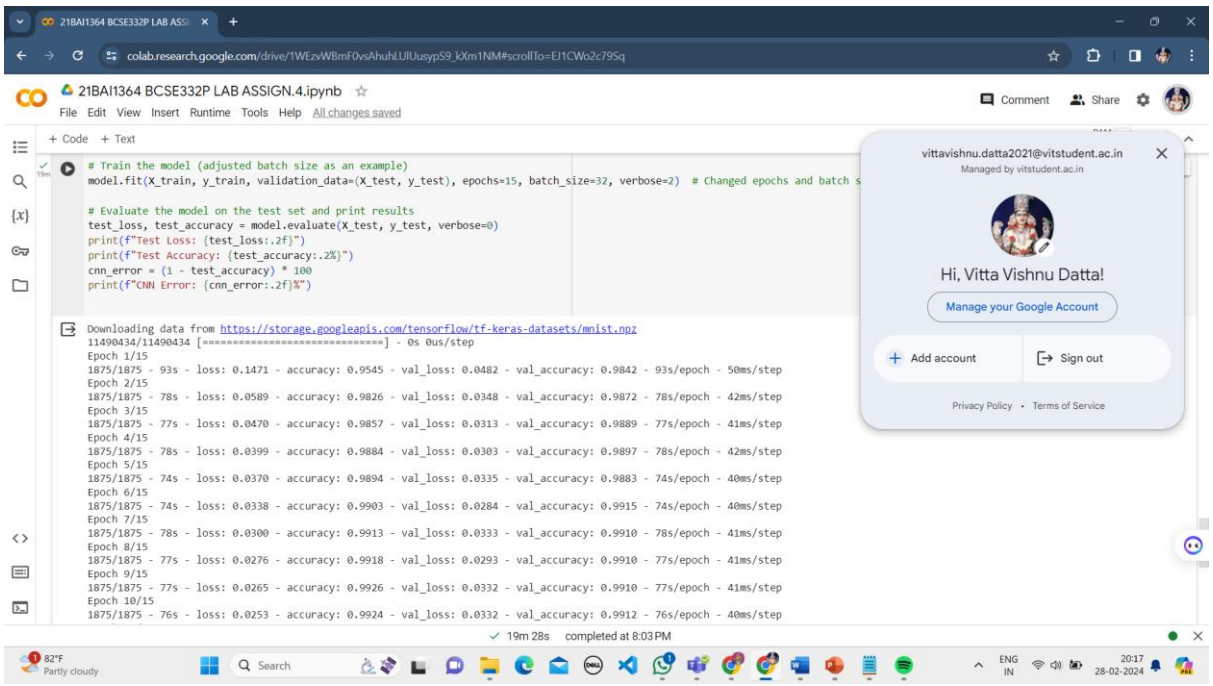
CHENNAI

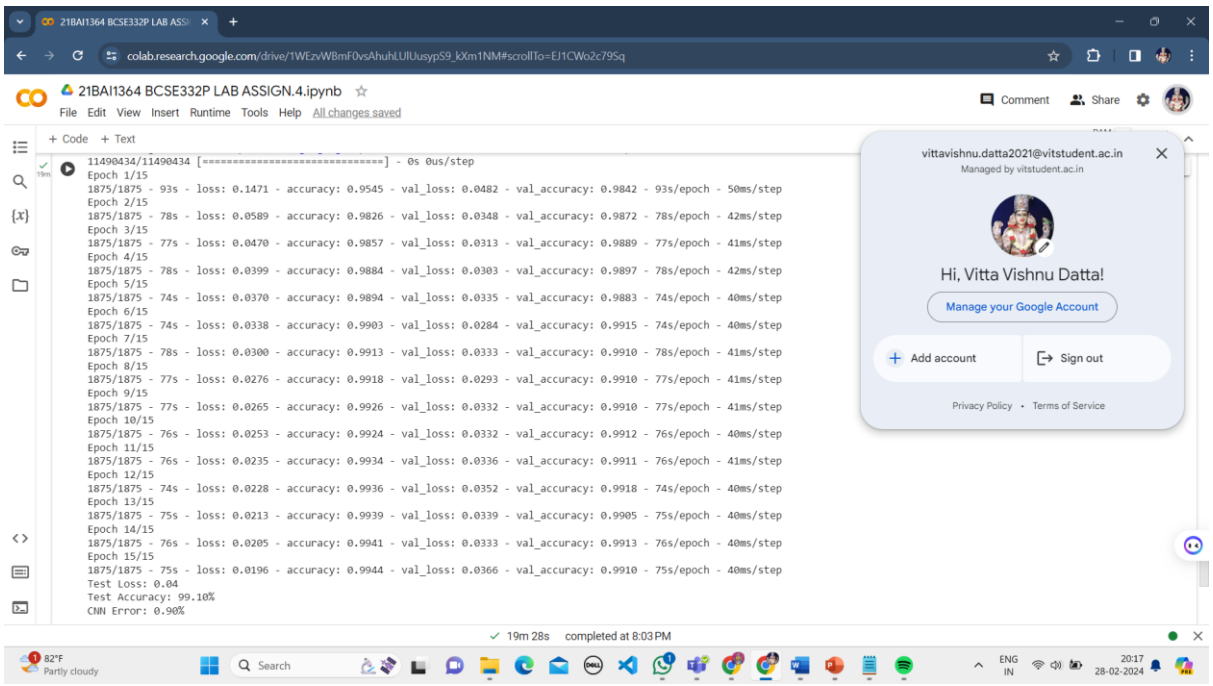
SCREENSHOTS OF THE MNIST DATASET:






21BAI1364 BCSE332P LAB ASSIGN.4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
model.add(Conv2D(64, (5, 5), padding='same', activation='relu', input_shape=(28, 28, 1)))
model.add(MaxPooling2D(pool_size=(3, 3))) # Changed pool size
model.add(Dropout(0.3)) # Changed dropout rate
# Flatten the output of the convolutional layers
model.add(Flatten())
# Fully connected layers with ReLU activation and dropout for regularization (changed number of neurons in first layer)
model.add(Dense(256, activation='relu')) # Changed number of neurons
model.add(Dropout(0.3)) # Changed dropout rate
model.add(Dense(10, activation='softmax')) # Output layer with 10 neurons for 10 classes
Compile the model: choose appropriate loss function, optimizer, and metrics
model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy']) # Changed optimizer
return model
# Create the CNN model
model = create_cnn_model()
# Train the model (adjusted batch size as an example)
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=15, batch_size=32, verbose=2) # Changed epochs and batch size
Evaluate the model on the test set and print results
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {test_loss:.2f}")
print(f"Test Accuracy: {test_accuracy:.2%}")
cnn_error = (1 - test_accuracy) * 100
print(f"CNN Error: {cnn_error:.2f}%")
vittavishnu.datta2021@vitstudent.ac.in
Managed by vitstudent.ac.in
Hi, Vitta Vishnu Datta!
Manage your Google Account
Add account
Sign out
Privacy Policy • Terms of Service
19m 28s completed at 8:03 PM


21BAI1364 BCSE332P LAB ASSIGN.4.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
# Train the model (adjusted batch size as an example)
model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=15, batch_size=32, verbose=2) # Changed epochs and batch s
Evaluate the model on the test set and print results
test_loss, test_accuracy = model.evaluate(X_test, y_test, verbose=0)
print(f"Test Loss: {test_loss:.2f}")
print(f"Test Accuracy: {test_accuracy:.2%}")
cnn_error = (1 - test_accuracy) * 100
print(f"CNN Error: {cnn_error:.2f}%")
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
Epoch 1/15
1875/1875 - 93s - loss: 0.1471 - accuracy: 0.9545 - val_loss: 0.0482 - val_accuracy: 0.9842 - 93s/epoch - 50ms/step
Epoch 2/15
1875/1875 - 78s - loss: 0.0589 - accuracy: 0.9826 - val_loss: 0.0348 - val_accuracy: 0.9872 - 78s/epoch - 42ms/step
Epoch 3/15
1875/1875 - 77s - loss: 0.0470 - accuracy: 0.9857 - val_loss: 0.0313 - val_accuracy: 0.9889 - 77s/epoch - 41ms/step
Epoch 4/15
1875/1875 - 78s - loss: 0.0399 - accuracy: 0.9884 - val_loss: 0.0303 - val_accuracy: 0.9897 - 78s/epoch - 42ms/step
Epoch 5/15
1875/1875 - 74s - loss: 0.0370 - accuracy: 0.9894 - val_loss: 0.0335 - val_accuracy: 0.9883 - 74s/epoch - 40ms/step
Epoch 6/15
1875/1875 - 74s - loss: 0.0338 - accuracy: 0.9903 - val_loss: 0.0284 - val_accuracy: 0.9915 - 74s/epoch - 40ms/step
Epoch 7/15
1875/1875 - 78s - loss: 0.0300 - accuracy: 0.9913 - val_loss: 0.0333 - val_accuracy: 0.9910 - 78s/epoch - 41ms/step
Epoch 8/15
1875/1875 - 77s - loss: 0.0276 - accuracy: 0.9918 - val_loss: 0.0293 - val_accuracy: 0.9910 - 77s/epoch - 41ms/step
Epoch 9/15
1875/1875 - 77s - loss: 0.0265 - accuracy: 0.9926 - val_loss: 0.0332 - val_accuracy: 0.9910 - 77s/epoch - 41ms/step
Epoch 10/15
1875/1875 - 76s - loss: 0.0253 - accuracy: 0.9924 - val_loss: 0.0332 - val_accuracy: 0.9912 - 76s/epoch - 40ms/step
vittavishnu.datta2021@vitstudent.ac.in
Managed by vitstudent.ac.in
Hi, Vitta Vishnu Datta!
Manage your Google Account
Add account
Sign out
Privacy Policy • Terms of Service
19m 28s completed at 8:03 PM



3. Visualize the filters and features for AlexNet, GoogleNet, and ResNet

CODE FOR PRE-TRAINED VGG 16 MODEL:

```
# load vgg model
from keras.applications.vgg16 import VGG16
# load the model
model = VGG16()
# summarize the model
model.summary()
# summarize filters in each convolutional layer
from keras.applications.vgg16 import VGG16
from matplotlib import pyplot
# load the model
model = VGG16()
# summarize filter shapes
```

```
for layer in model.layers:
 # check for convolutional layer
 if 'conv' not in layer.name:continue
 # get filter weights
 filters, biases = layer.get_weights()
 print(layer.name, filters.shape)
# retrieve weights from the second hidden layer
filters, biases = model.layers[1].get_weights()
# normalize filter values to 0-1 so we can visualize them
f_min, f_max = filters.min(), filters.max()
filters = (filters - f_min) / (f_max - f_min)
# plot first few filters
n_filters, ix = 6, 1
for i in range(n_filters):
  # get the filter
  f = filters[:, :, :, i]
  # plot each channel separately
  for j in range(3):
    # specify subplot and turn of axis
    ax = pyplot.subplot(n_filters, 3, ix)
    ax.set_xticks([])
    ax.set_yticks([])
    # plot filter channel in grayscale
    pyplot.imshow(f[:, :, j], cmap='gray')
    ix += 1
# show the figure
pyplot.show()
# cannot easily visualize filters lower down
from keras.applications.vgg16 import VGG16
from matplotlib import pyplot
# load the model
model = VGG16()
# retrieve weights from the second hidden layer
filters, biases = model.layers[1].get_weights()
# normalize filter values to 0-1 so we can visualize them
f_min, f_max = filters.min(), filters.max()
filters = (filters - f_min) / (f_max - f_min)
# plot first few filters
n_filters, ix = 6, 1
for i in range(n_filters):
  # get the filter
  f = filters[:, :, :, i]
```

```
  # plot each channel separately
  for j in range(3):
    # specify subplot and turn of axis
    ax = pyplot.subplot(n_filters, 3, ix)
    ax.set_xticks([])
    ax.set_yticks([])
    # plot filter channel in grayscale
    pyplot.imshow(f[:, :, j], cmap='gray')
    ix += 1
# show the figure
pyplot.show()
# summarize feature map size for each conv layer
from keras.applications.vgg16 import VGG16
from matplotlib import pyplot
# load the model
model = VGG16()
# summarize feature map shapes
for i in range(len(model.layers)):
  layer = model.layers[i]
  # check for convolutional layer
  if 'conv' not in layer.name:
    continue
  # summarize output shape
  print(i, layer.name, layer.output.shape)
# redefine model to output right after the first hidden layer
from keras.models import Model  # Assuming Keras

model = Model(inputs=model.inputs, outputs=model.layers[1].output)
# load the image with the required shape
from keras.preprocessing.image import load_img
img = load_img('/content/bird.jpg', target_size=(224, 224))
# convert the image to an array
from keras.preprocessing.image import img_to_array
img = img_to_array(img)
# expand dimensions so that it represents a single 'sample'
from tensorflow.keras.backend import expand_dims
img = expand_dims(img, axis=0)
# prepare the image (e.g. scale pixel values for the vgg)
from keras.applications.vgg16 import preprocess_input
img = preprocess_input(img)
# get feature map for first hidden layer
feature_maps = model.predict(img)
```

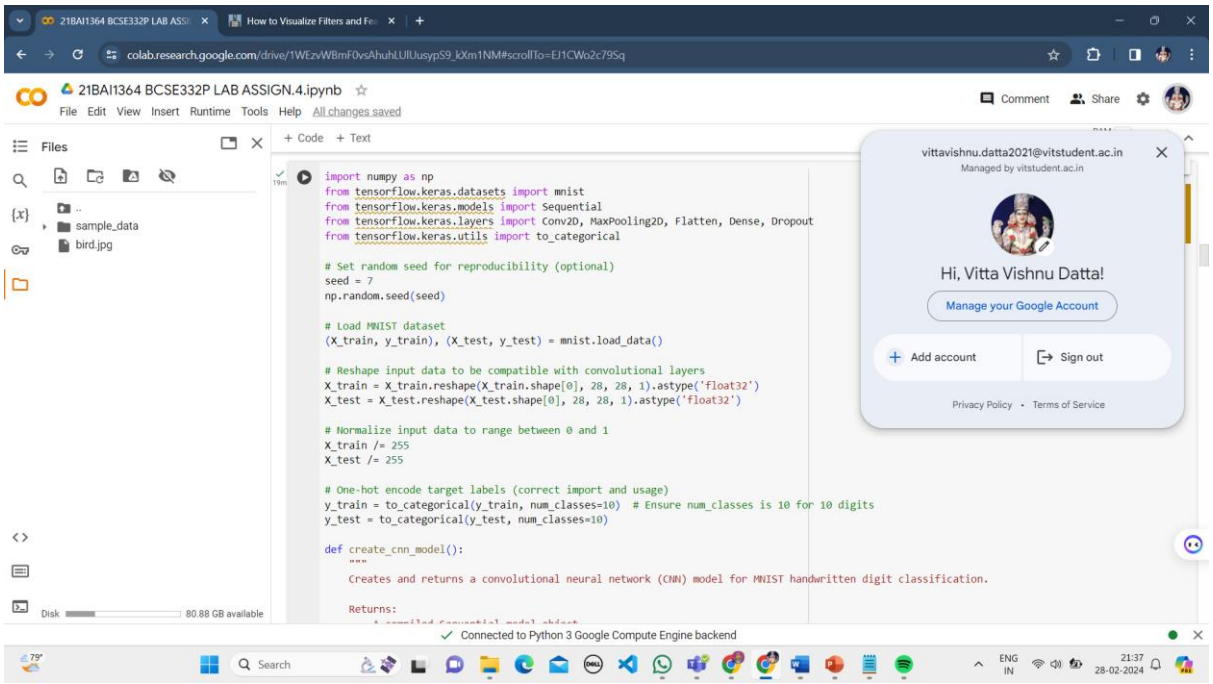
```
# plot all 64 maps in an 8x8 squares
square = 8
ix = 1
for _ in range(square):
  for _ in range(square):
    # specify subplot and turn of axis
    ax = pyplot.subplot(square, square, ix)
    ax.set_xticks([])
    ax.set_yticks([])
    # plot filter channel in grayscale
    pyplot.imshow(feature_maps[0, :, :, ix-1], cmap='gray')
    ix += 1
# show the figure
pyplot.show()
# plot feature map of first conv layer for given image
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import Model
from matplotlib import pyplot
from numpy import expand_dims
# load the model
model = VGG16()
# redefine model to output right after the first hidden layer
model = Model(inputs=model.inputs, outputs=model.layers[1].output)
model.summary()
# load the image with the required shape
img = load_img('bird.jpg', target_size=(224, 224))
# convert the image to an array
img = img_to_array(img)
# expand dimensions so that it represents a single 'sample'
img = expand_dims(img, axis=0)
# prepare the image (e.g. scale pixel values for the vgg)
img = preprocess_input(img)
# get feature map for first hidden layer
feature_maps = model.predict(img)
# plot all 64 maps in an 8x8 squares
square = 8
ix = 1
for _ in range(square):
  for _ in range(square):
```

```
    # specify subplot and turn of axis
    ax = pyplot.subplot(square, square, ix)
    ax.set_xticks([])
    ax.set_yticks([])
    # plot filter channel in grayscale
    pyplot.imshow(feature_maps[0, :, :, ix-1], cmap='gray')
    ix += 1
# show the figure
pyplot.show()
# redefine model to output right after the first hidden layer
from keras.models import Model
from keras.applications.vgg16 import VGG16

# Load the VGG16 model
model = VGG16()

# Get the maximum valid index
max_index = len(model.layers) - 2

# Generate even-indexed layer indices (assuming convolutional layers)
ixs = [i for i in range(1, max_index + 1) if i % 2 != 0]

# Define outputs based on valid indices
outputs = [model.layers[i].output for i in ixs]

# Create the new model
new_model = Model(inputs=model.inputs, outputs=outputs)

# Print model summary
new_model.summary()

import matplotlib.pyplot as plt

# Assuming `feature_maps` is a list containing feature maps from each
block
square = 8
for fmap in feature_maps:
    # Create a new figure for each block's feature maps
    fig, axes = plt.subplots(square, square, figsize=(12, 12))  #
Adjust figsize as needed

    ix = 1
```

```
    for i in range(square):
        for j in range(square):
            # Prevent index out of bounds or unnecessary indexing
            if ix >= len(fmap):
                # Handle the case where fewer than 64 maps are
available
                break
            ax = axes[i, j]
            ax.set_xticks([])
            ax.set_yticks([])

            # Use correct indexing to access individual maps
            ax.imshow(fmap[ix - 1], cmap='gray')  # Access entire
channel
            ix += 1

    # Tight layout to prevent overlapping labels
    fig.suptitle(f"Feature Maps from Block")  # Optional title
    plt.tight_layout()
    plt.show()
# visualize feature maps output from each block in the vgg model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import Model
from matplotlib import pyplot
from numpy import expand_dims
# load the model
model = VGG16()
# redefine model to output right after the first hidden layer
ixs = [2, 5, 9, 13, 17]
outputs = [model.layers[i].output for i in ixs]
model = Model(inputs=model.inputs, outputs=outputs)
# load the image with the required shape
img = load_img('bird.jpg', target_size=(224, 224))
# convert the image to an array
img = img_to_array(img)
# expand dimensions so that it represents a single 'sample'
img = expand_dims(img, axis=0)
# prepare the image (e.g. scale pixel values for the vgg)
img = preprocess_input(img)
```

```
# get feature map for first hidden layer
feature_maps = model.predict(img)
# plot the output from each block
square = 8
for fmap in feature_maps:
  # plot all 64 maps in an 8x8 squares
  ix = 1
  for _ in range(square):
    for _ in range(square):
      # specify subplot and turn of axis
      ax = pyplot.subplot(square, square, ix)
      ax.set_xticks([])
      ax.set_yticks([])
      # plot filter channel in grayscale
      pyplot.imshow(fmap[0, :, :, ix-1], cmap='gray')
      ix += 1
  # show the figure
  pyplot.show()
```

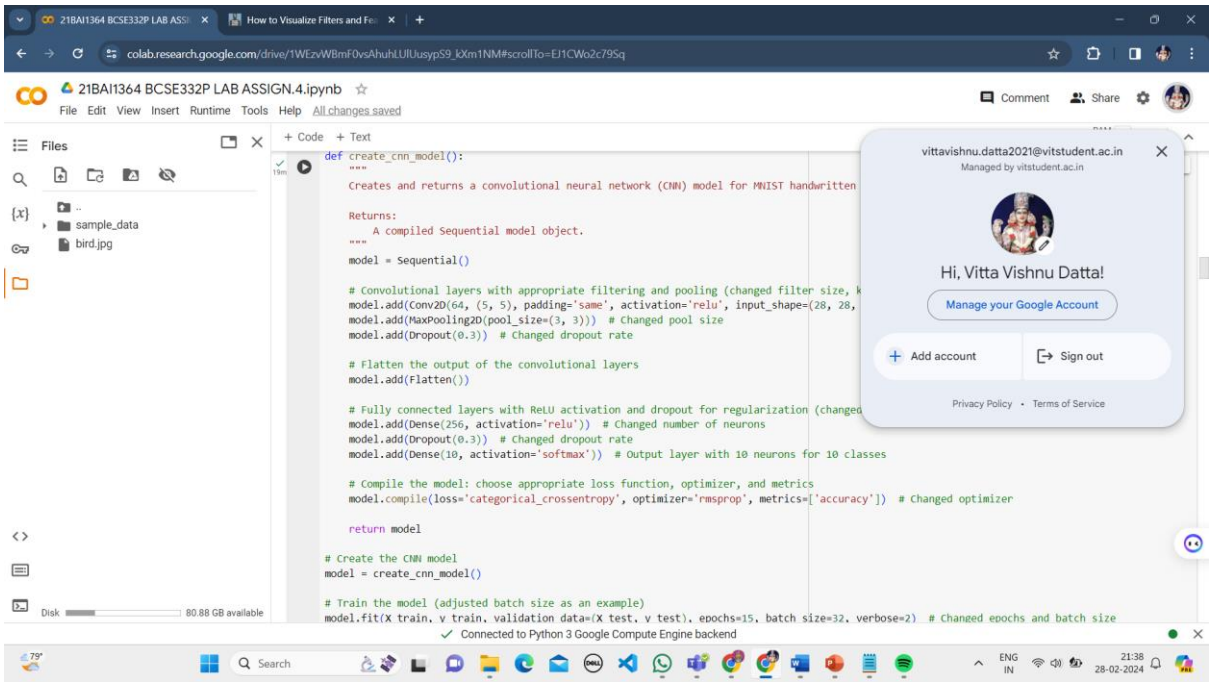
SCREENSHOTS OF THE VGG16 MODEL:

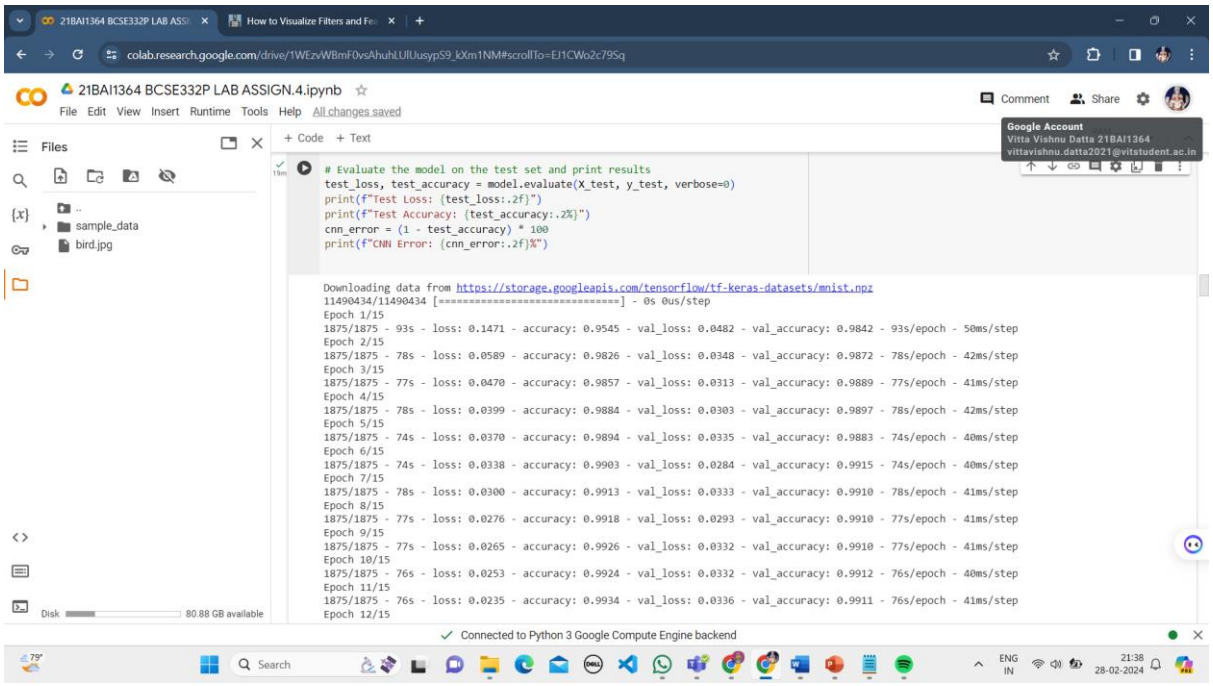


Vellore Institute of Technology

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

CHENNAI



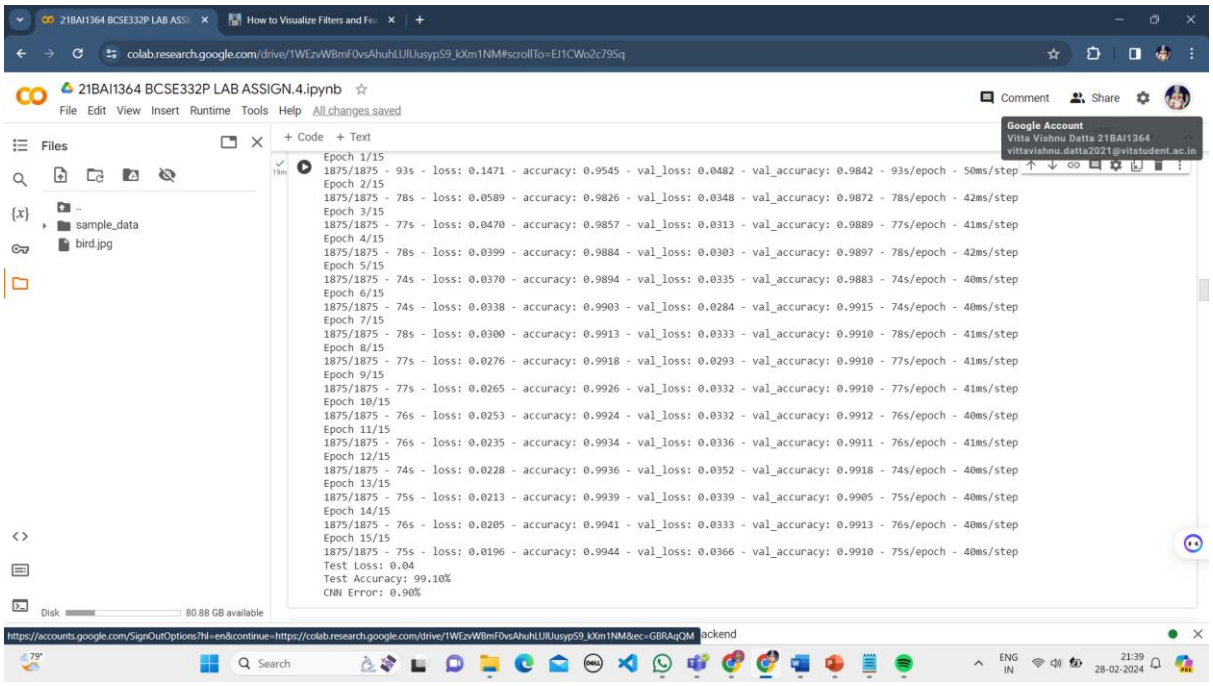


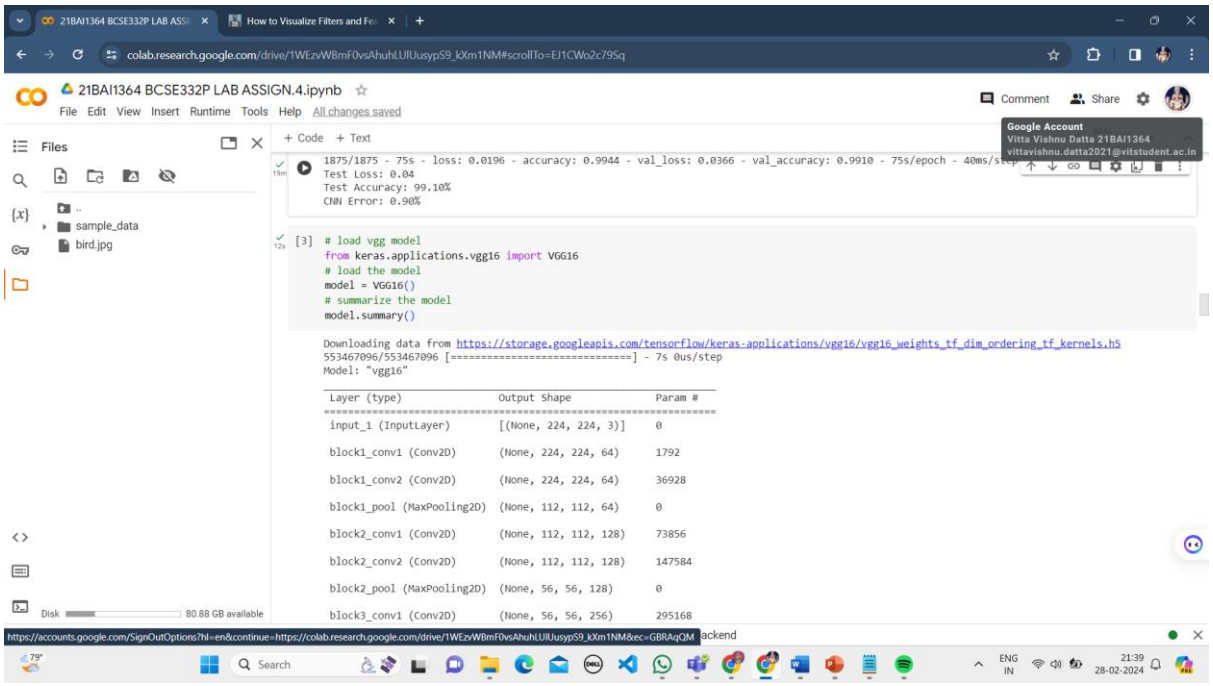
VIT®

Vellore Institute of Technology

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

CHENNAI



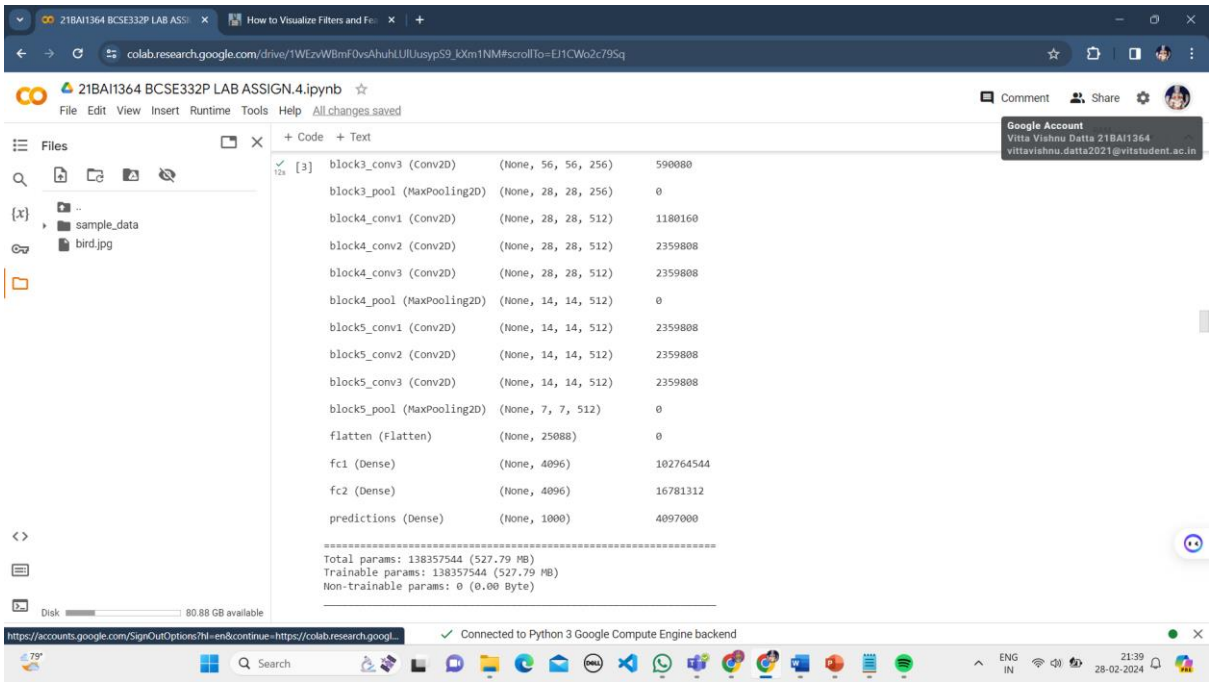


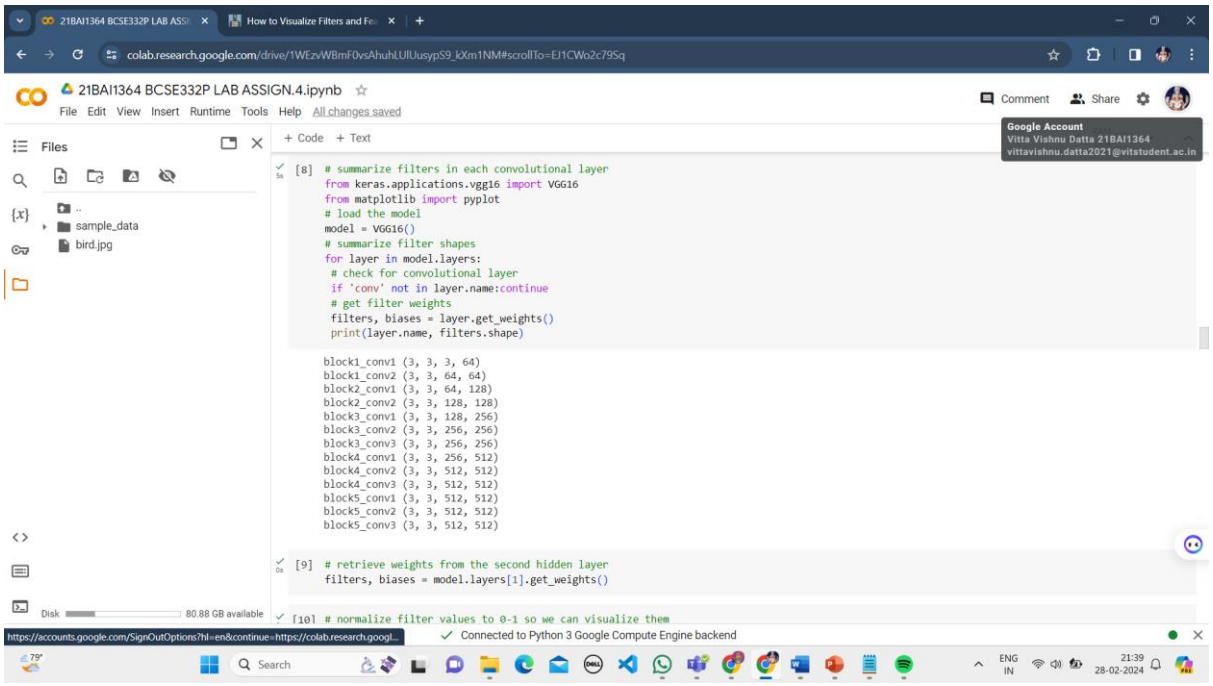
VIT®

Vellore Institute of Technology

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

CHENNAI



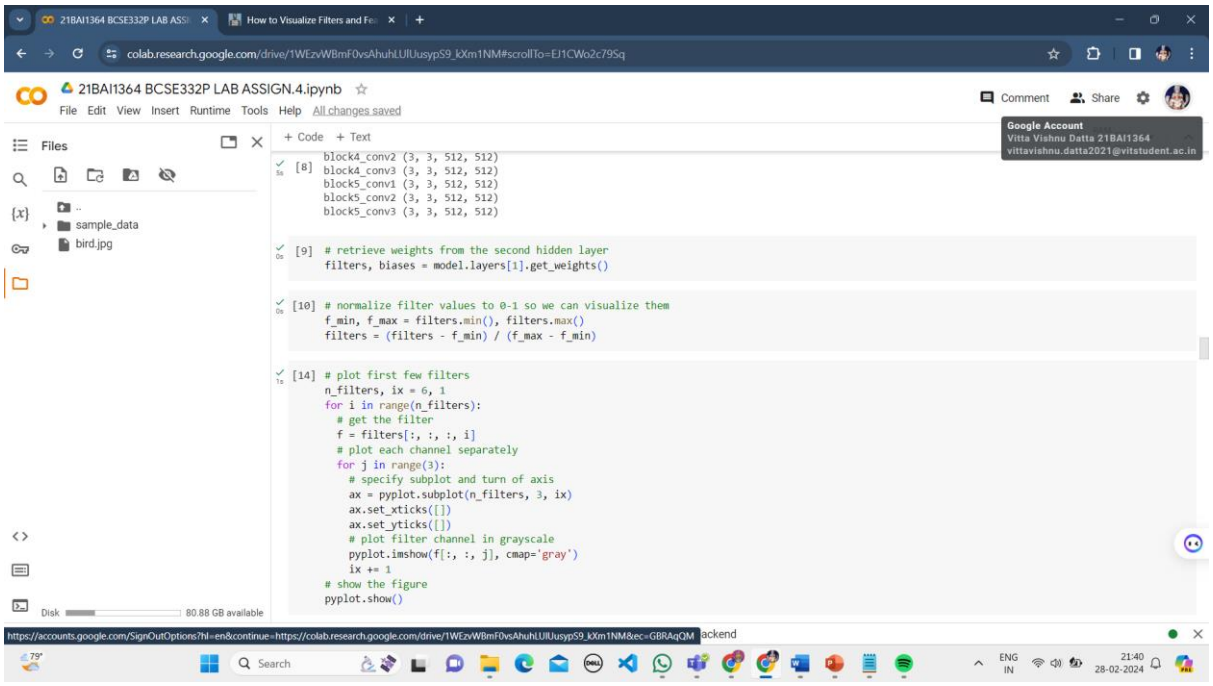


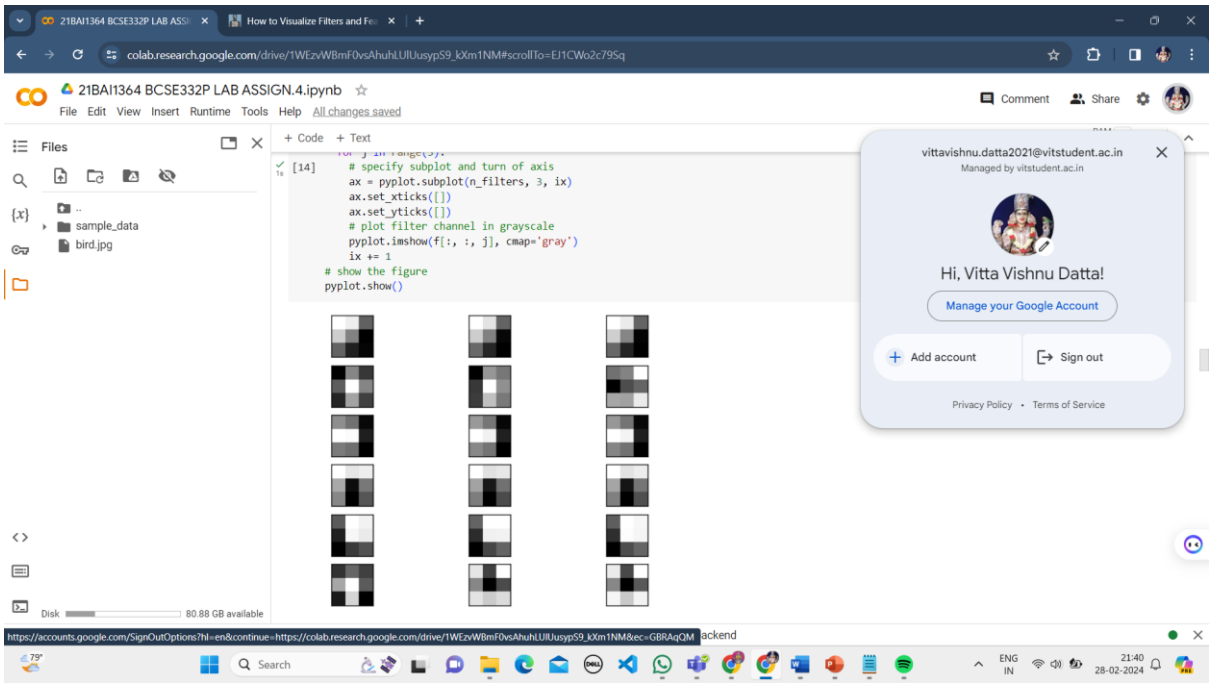
VIT®

Vellore Institute of Technology

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

CHENNAI



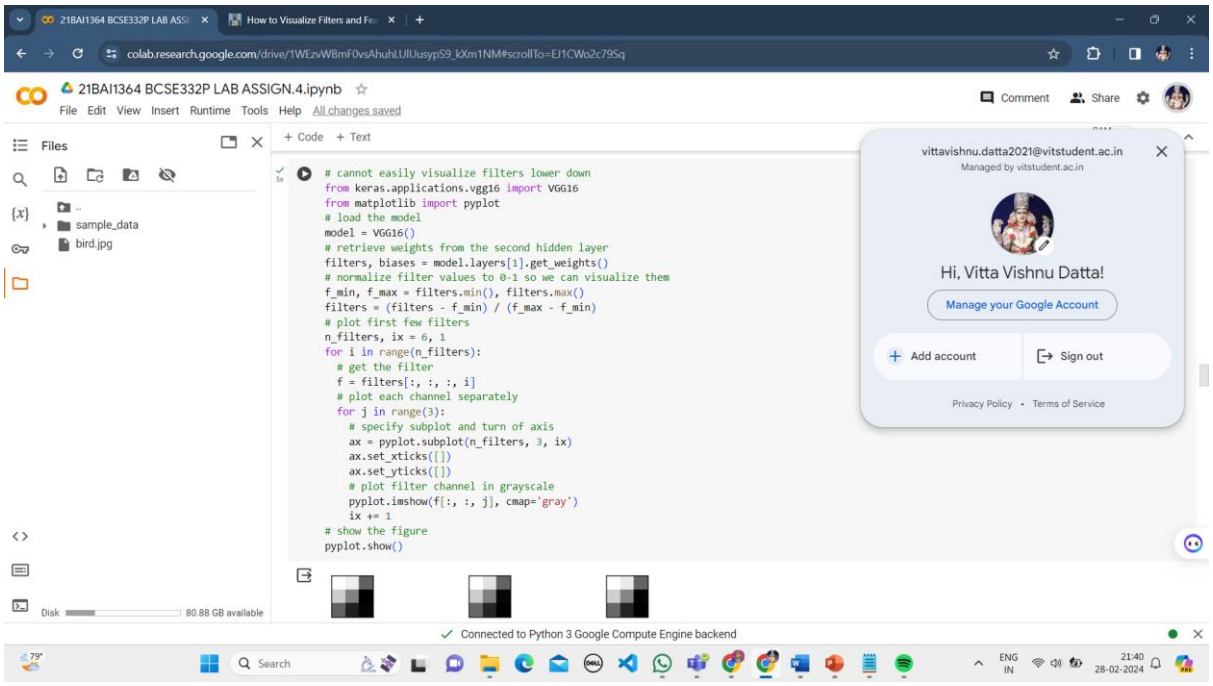


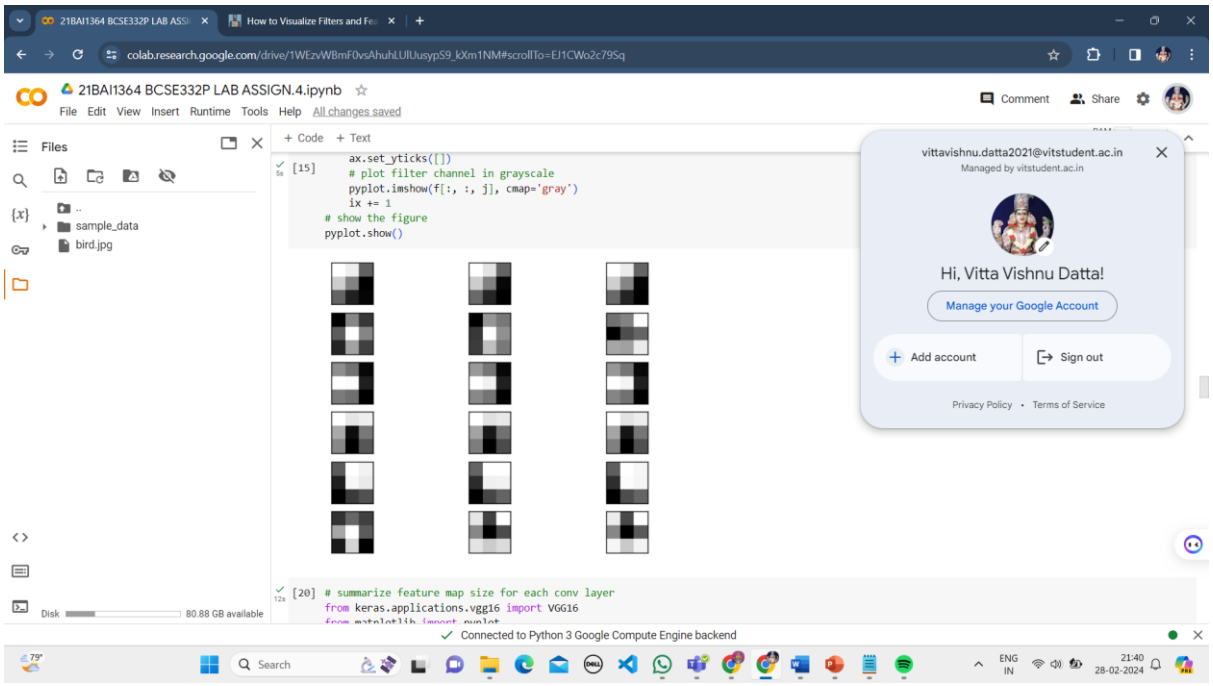
VIT®

Vellore Institute of Technology

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

CHENNAI



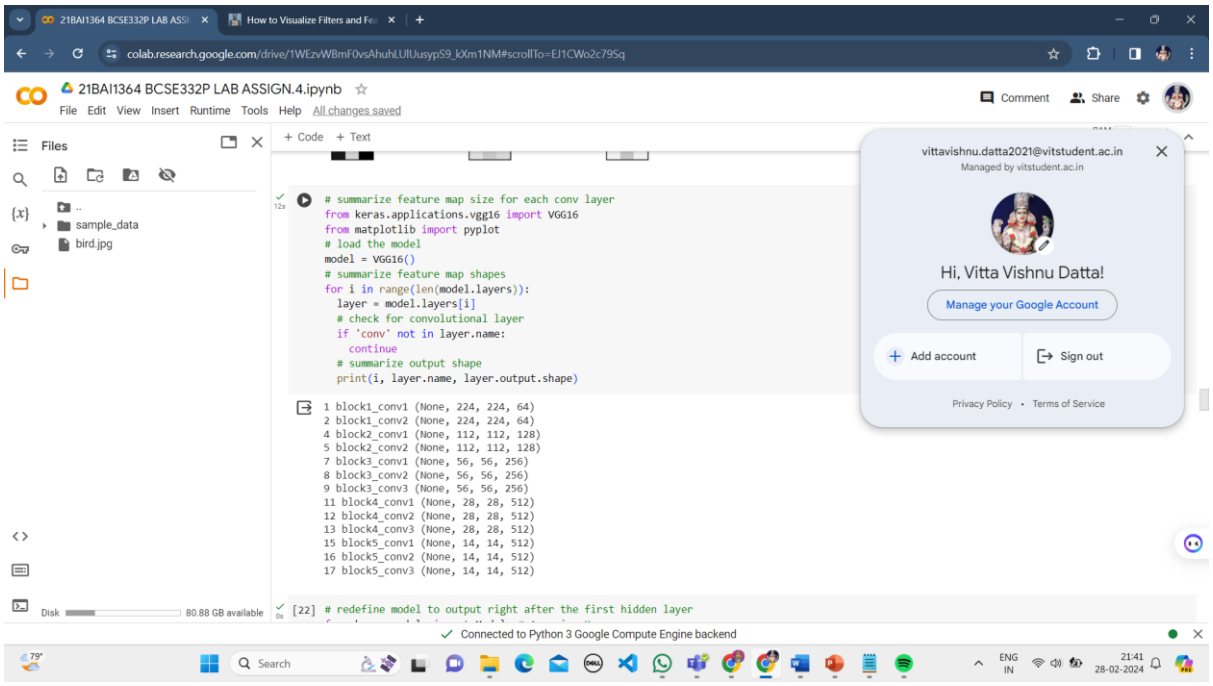


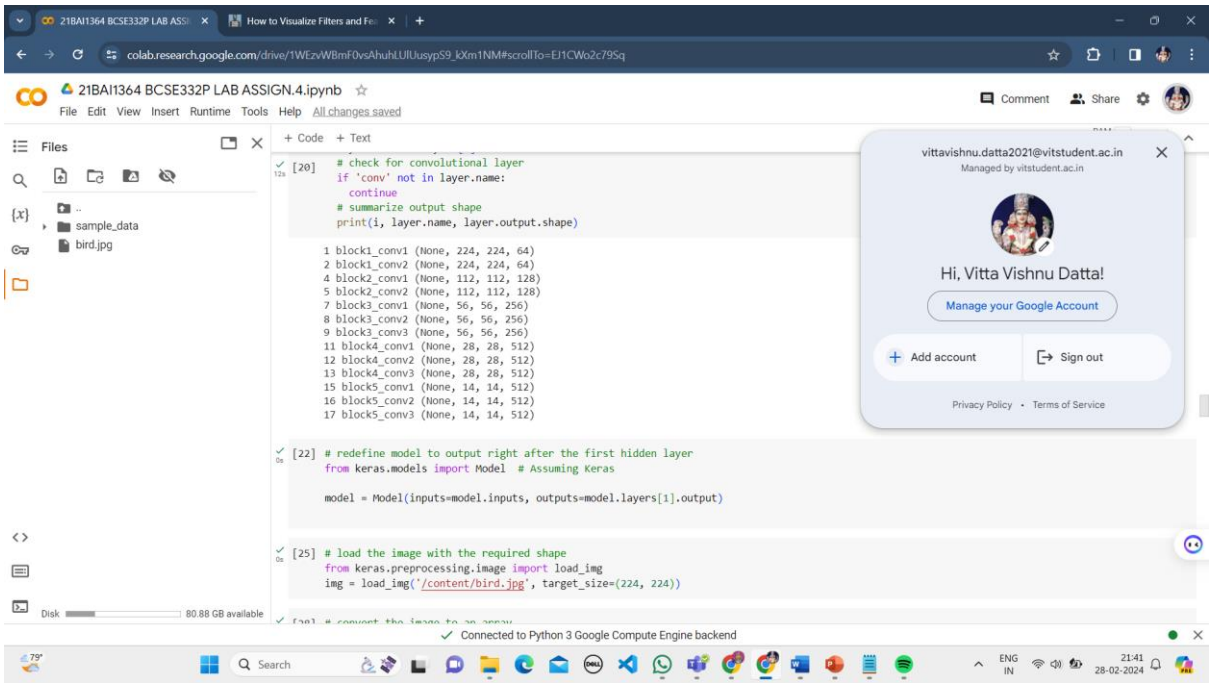
VIT®

Vellore Institute of Technology

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

CHENNAI



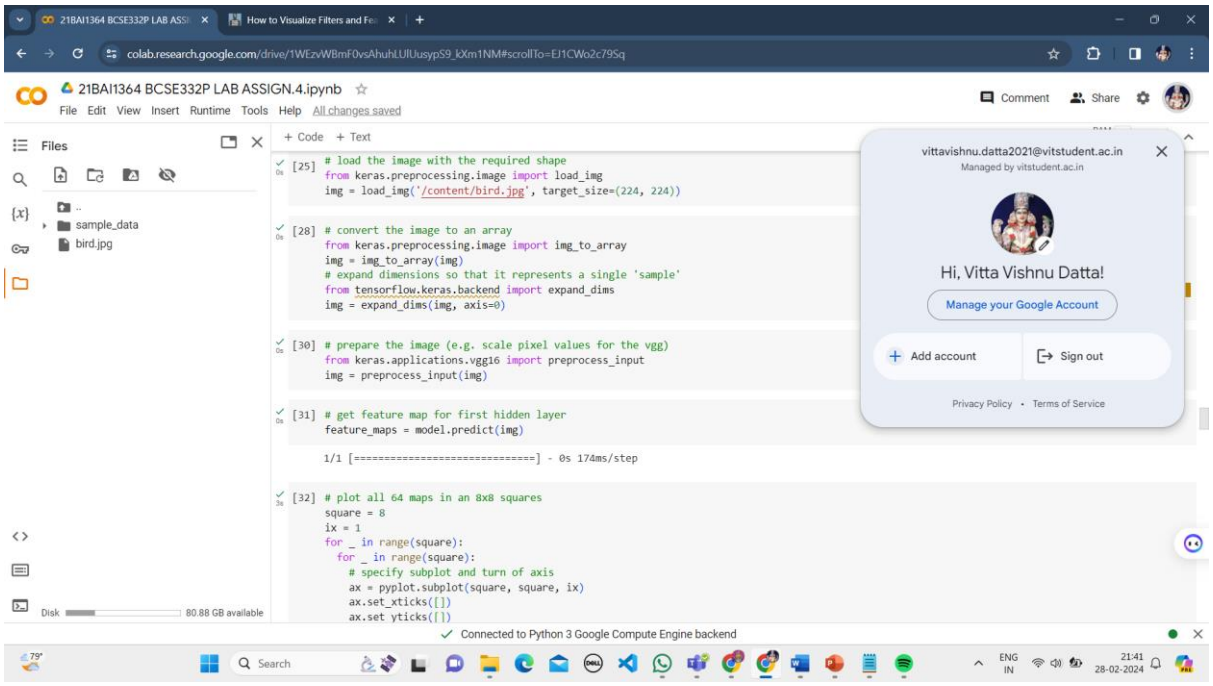


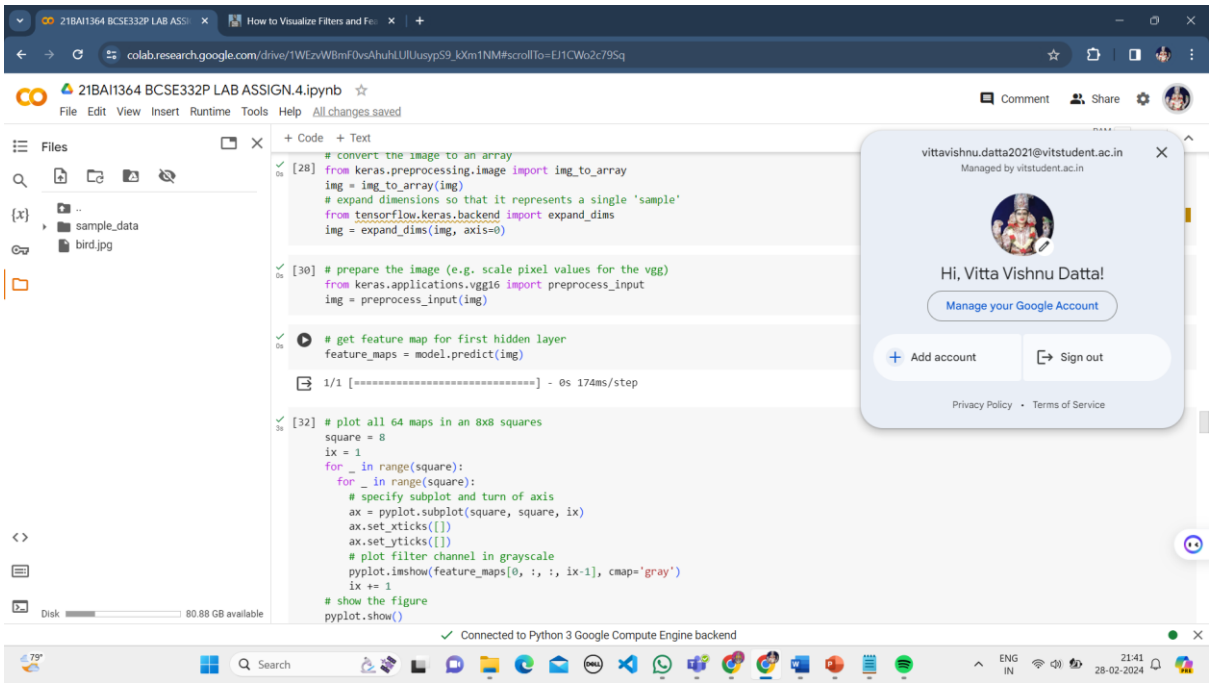
VIT®

Vellore Institute of Technology

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

CHENNAI



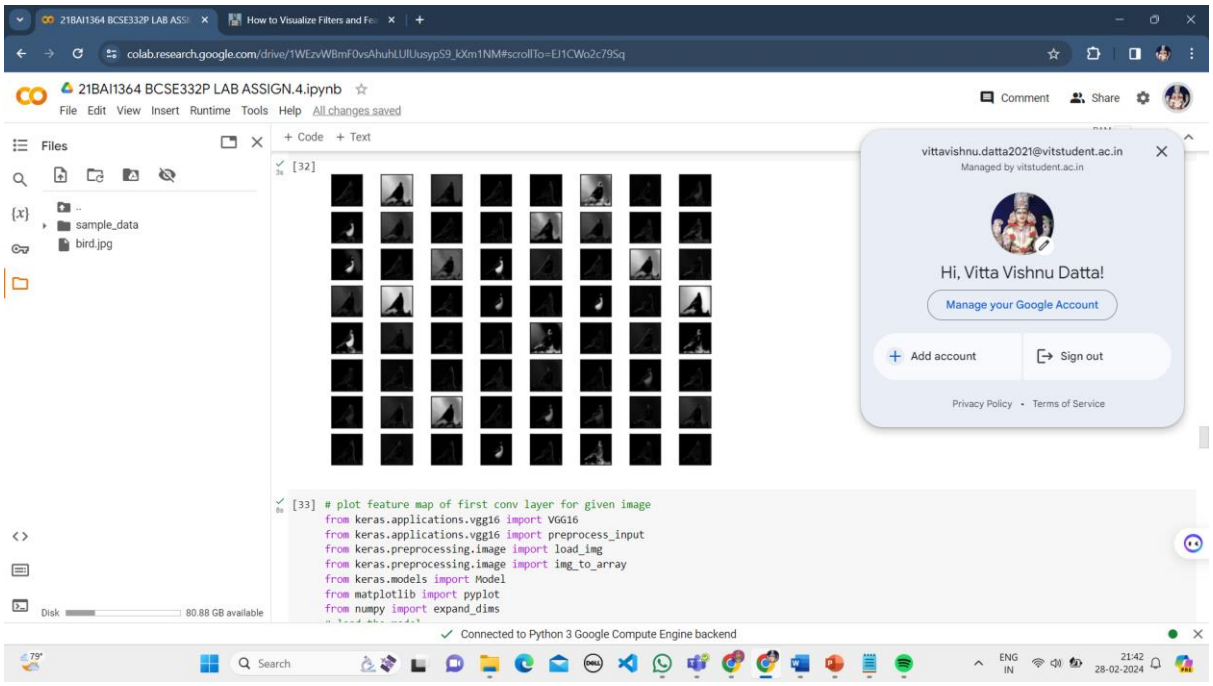


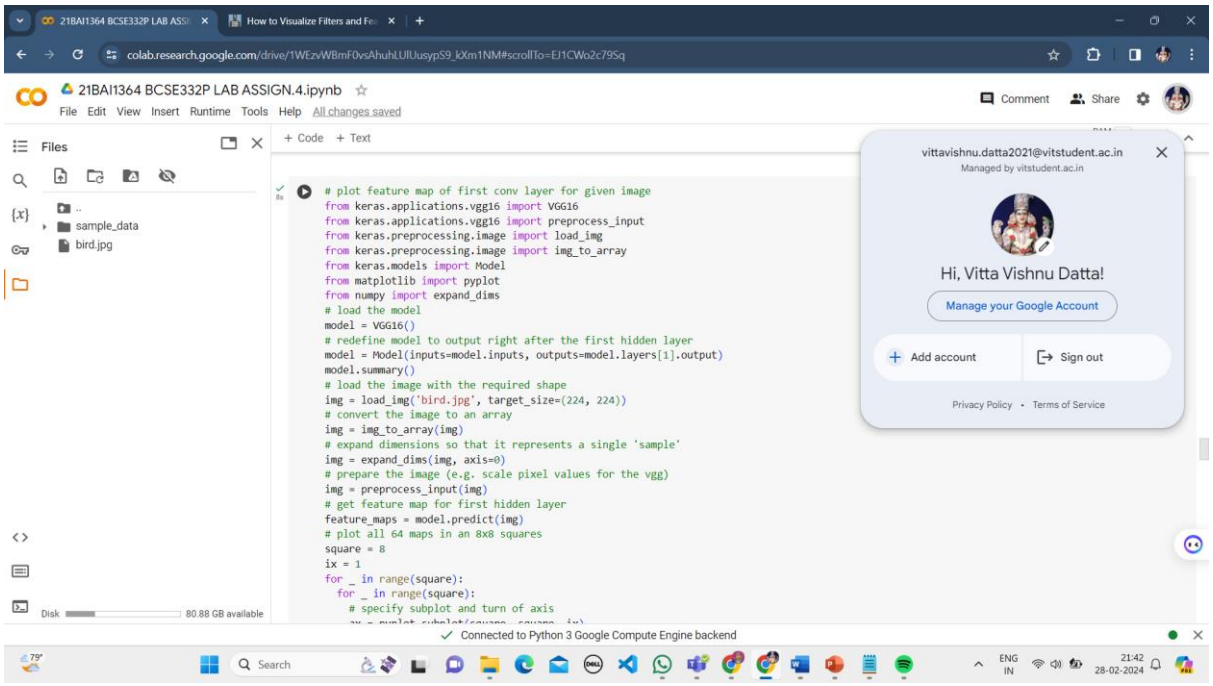
VIT®

Vellore Institute of Technology

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

CHENNAI



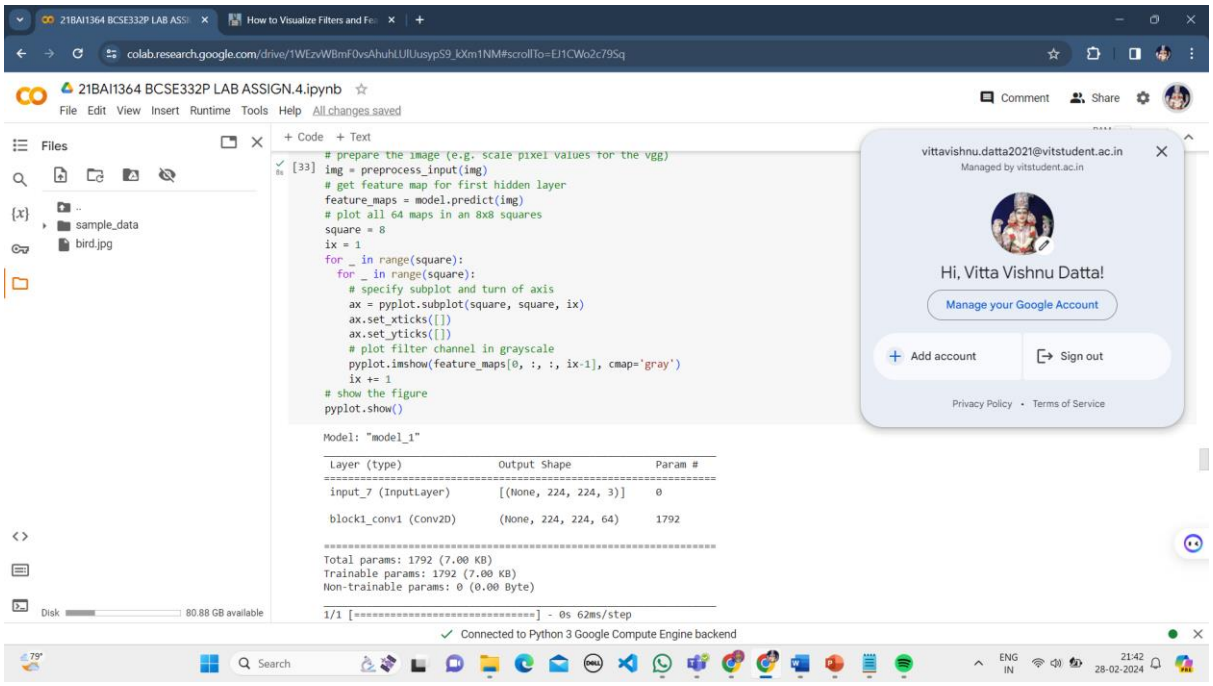


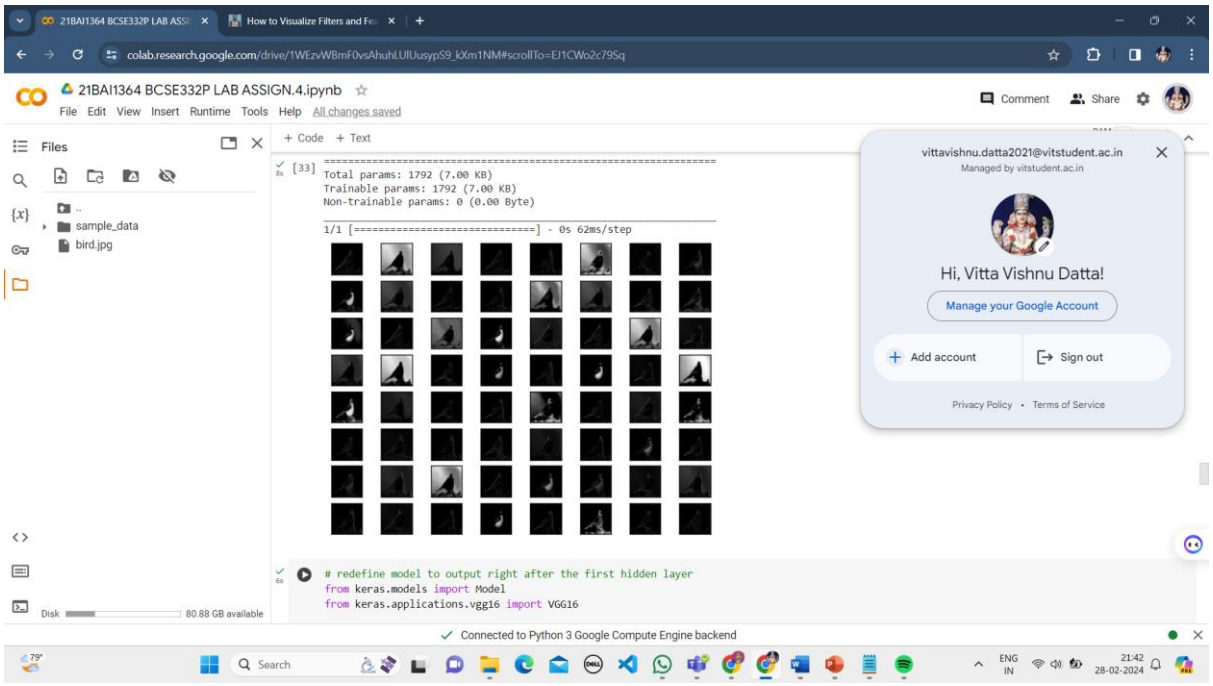
VIT®

Vellore Institute of Technology

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

CHENNAI



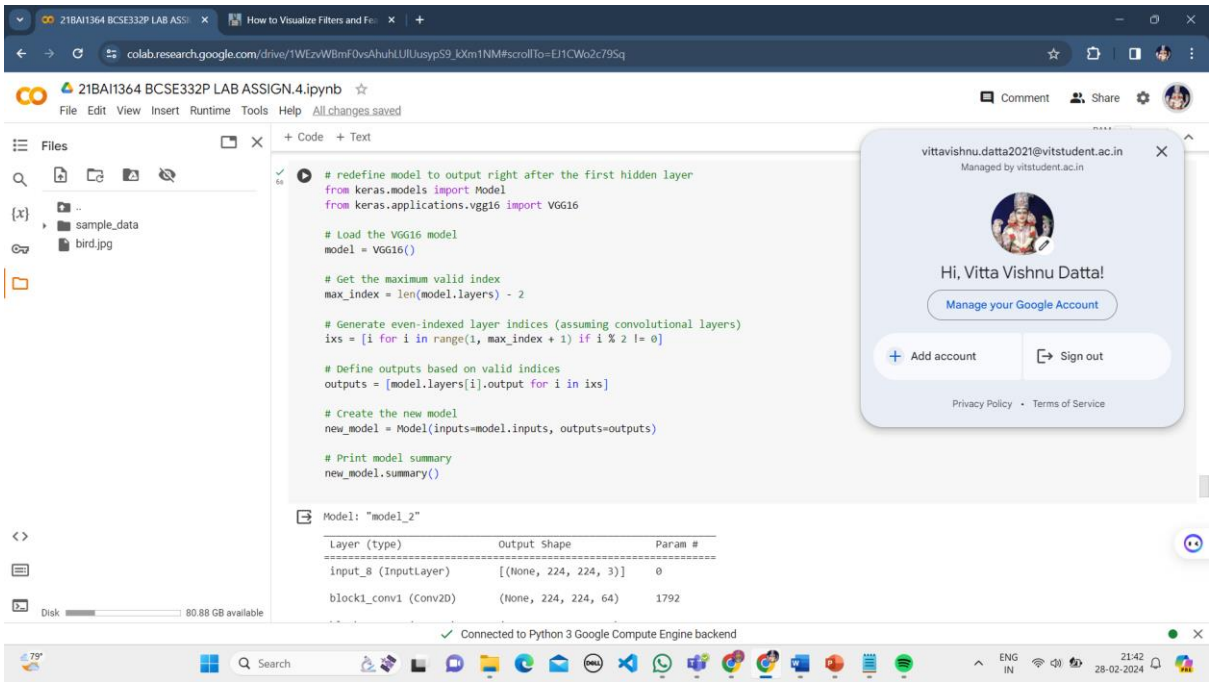


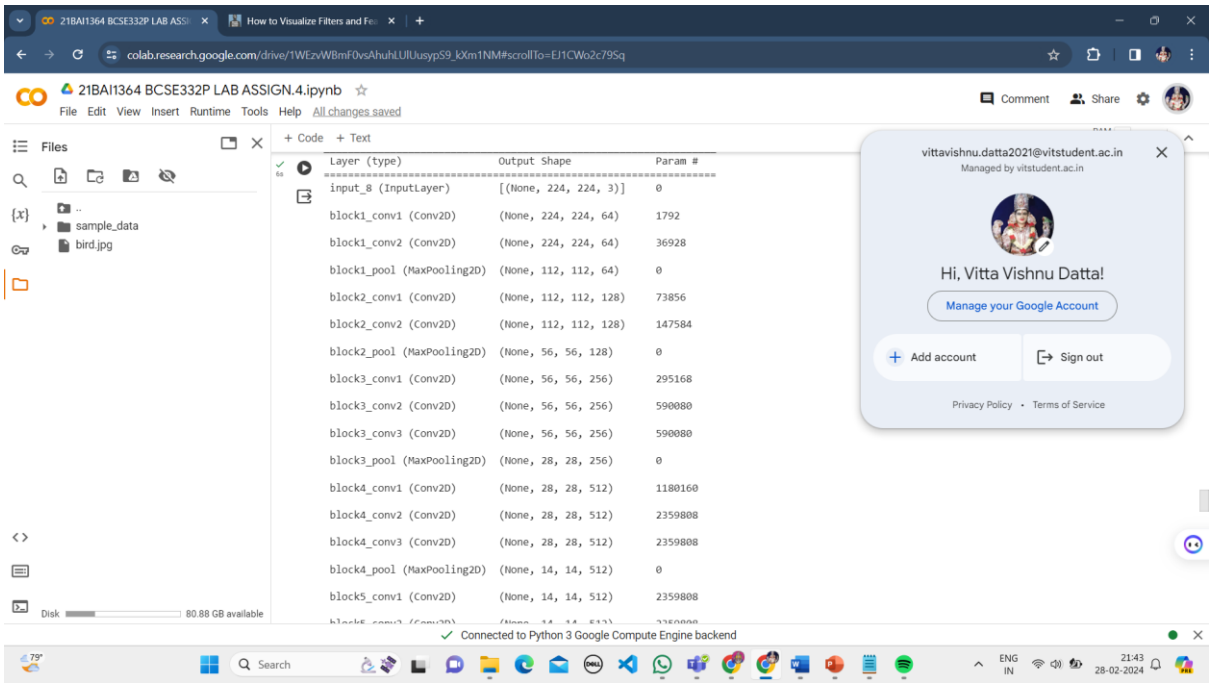
VIT®

Vellore Institute of Technology

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

CHENNAI



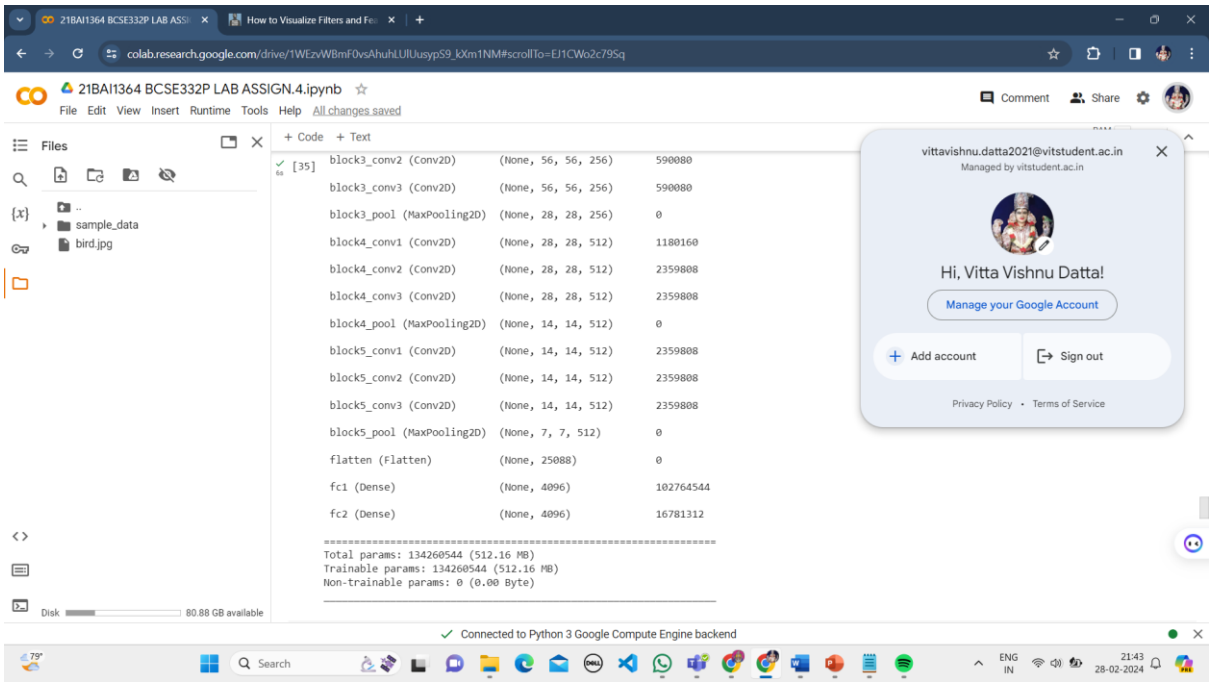


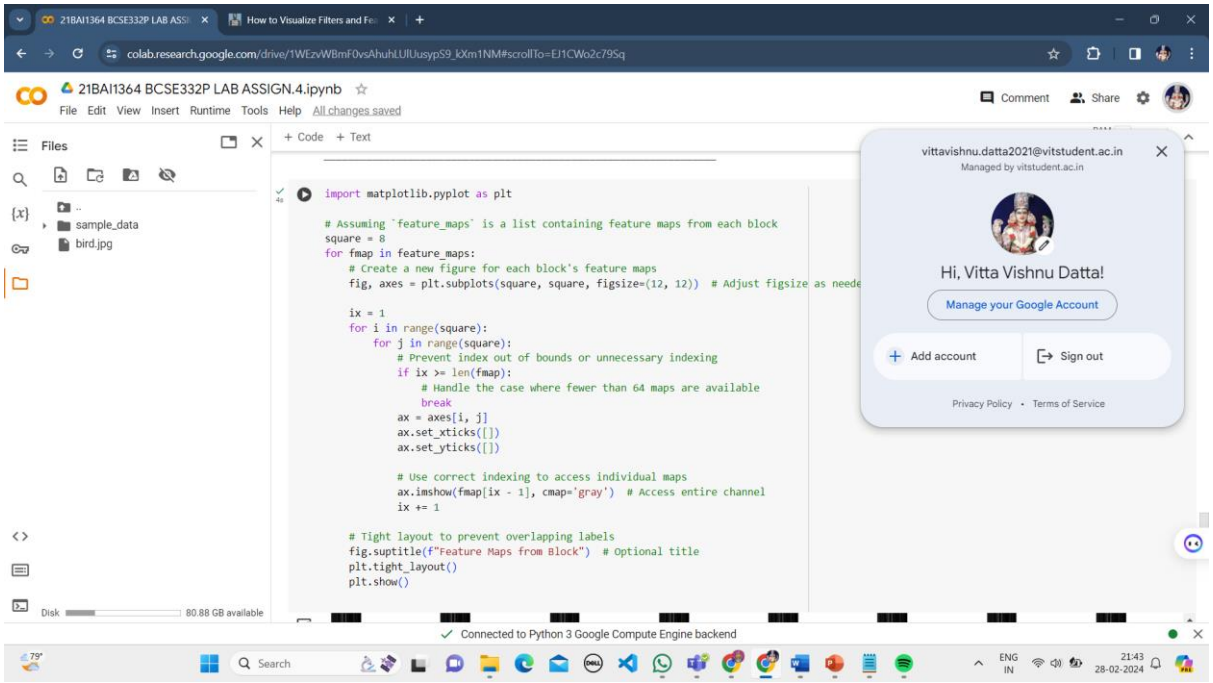
VIT®

Vellore Institute of Technology

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

CHENNAI



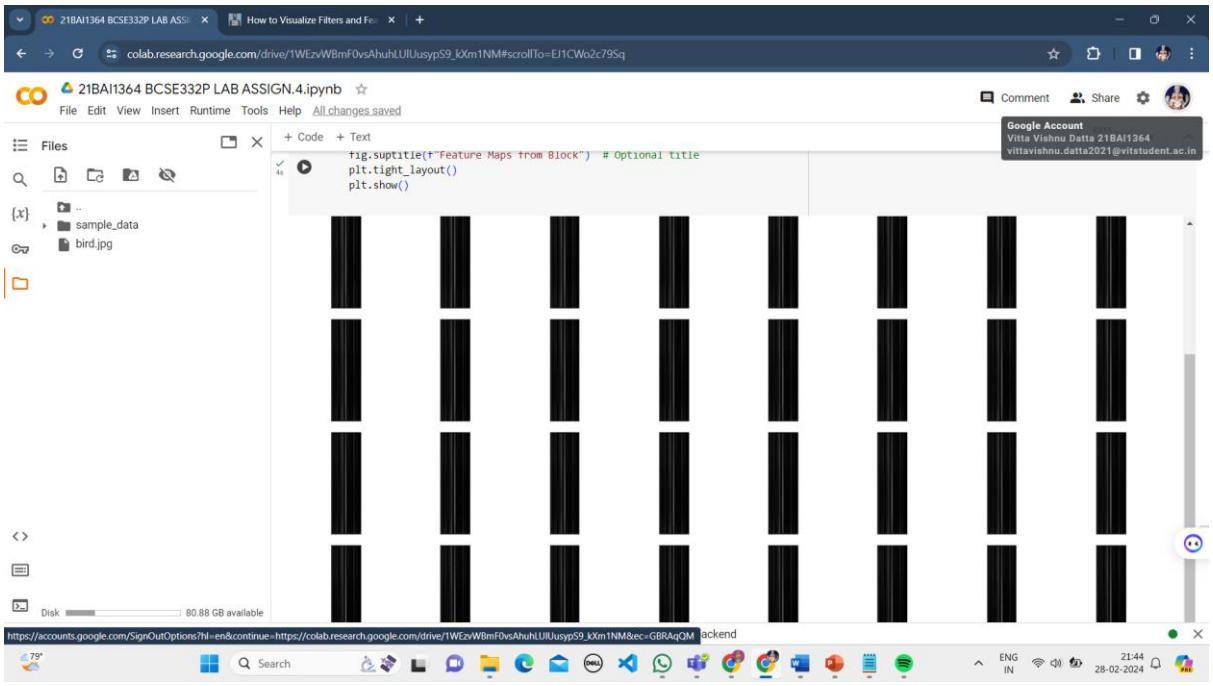


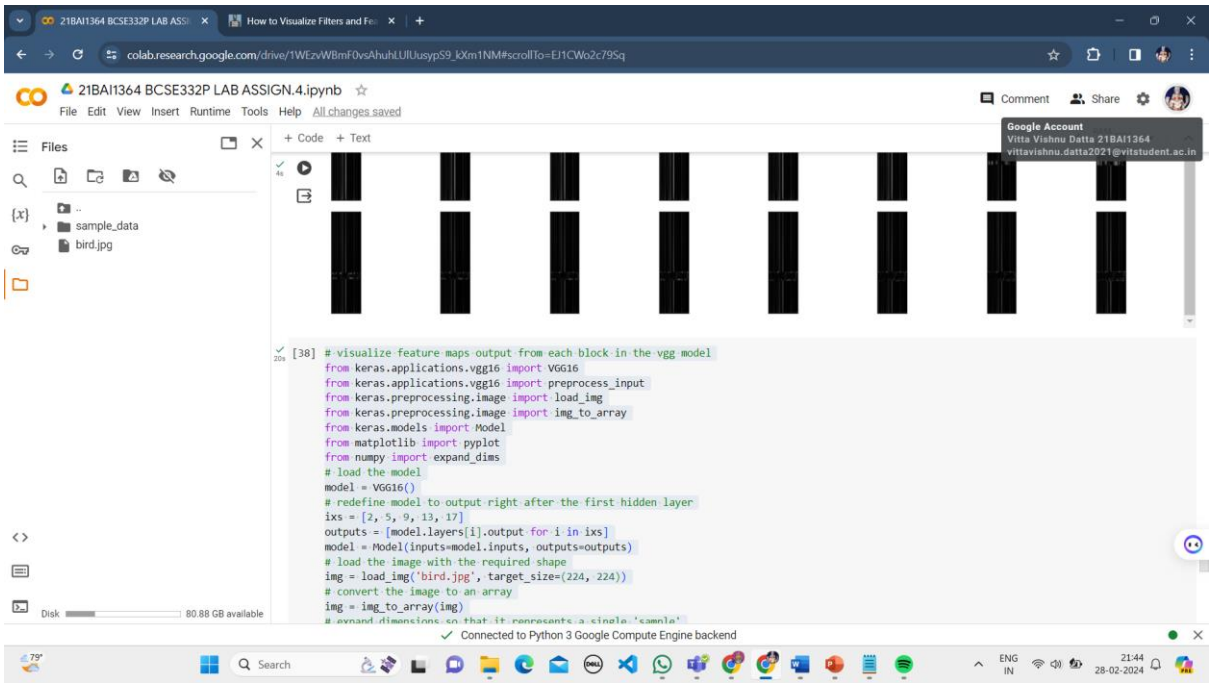
VIT®

Vellore Institute of Technology

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

CHENNAI



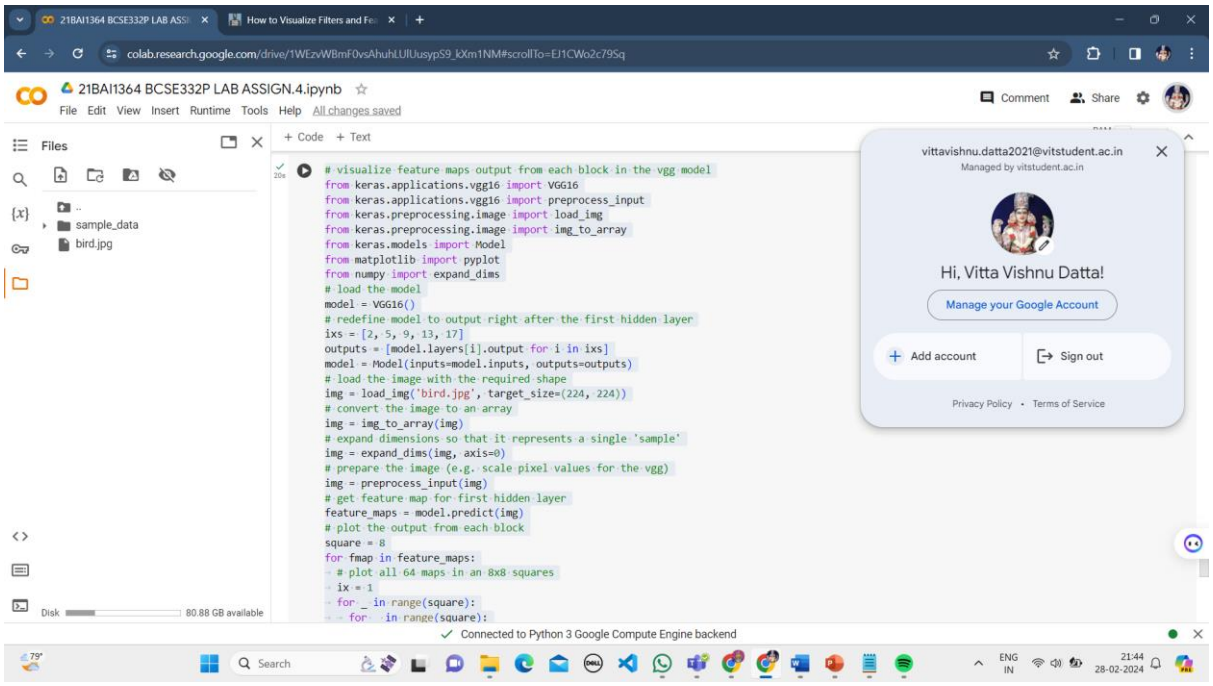


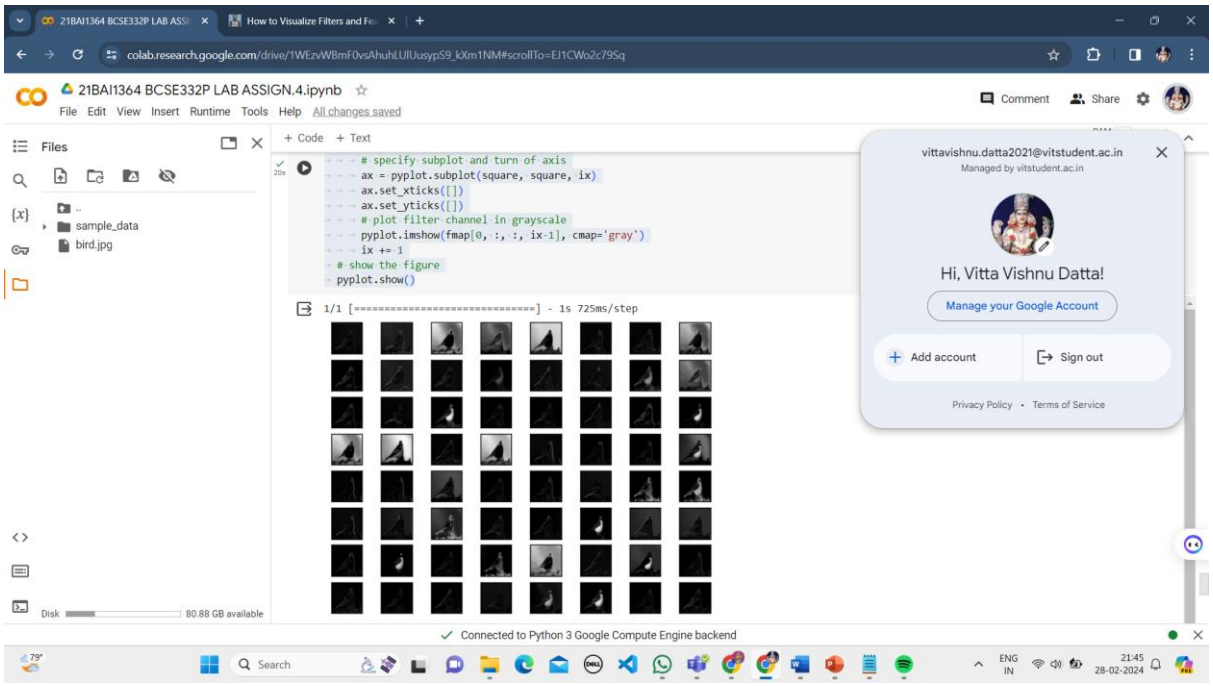
VIT®

Vellore Institute of Technology

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

CHENNAI



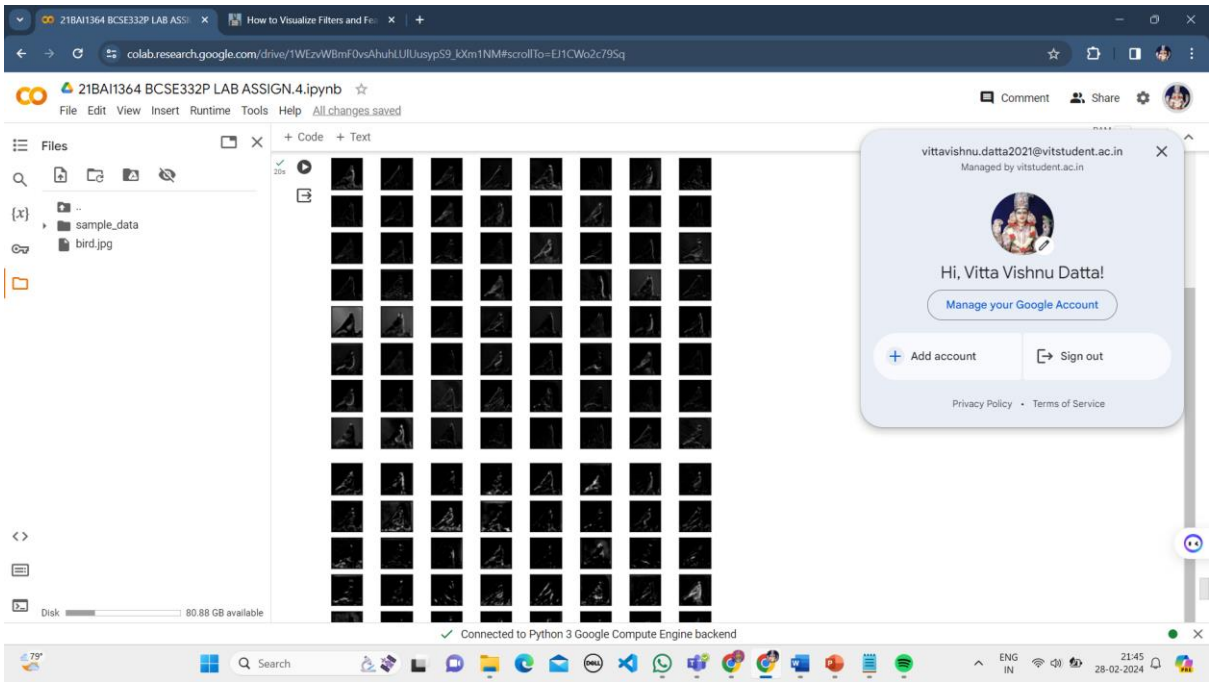


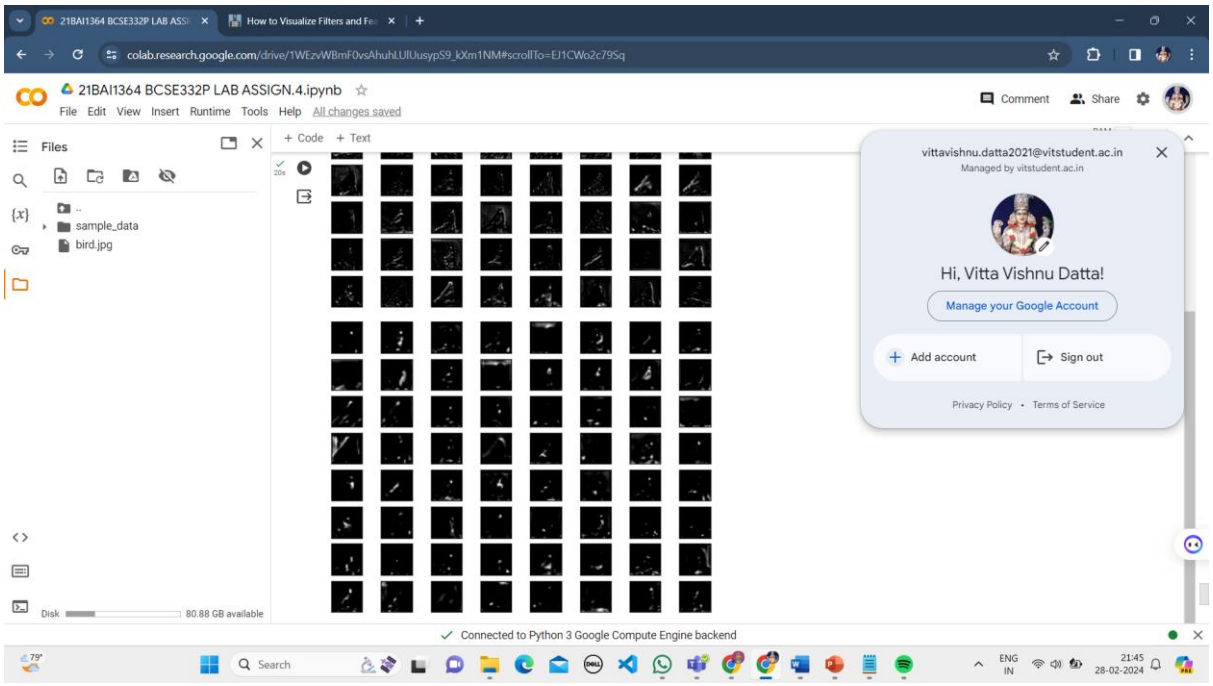
VIT®

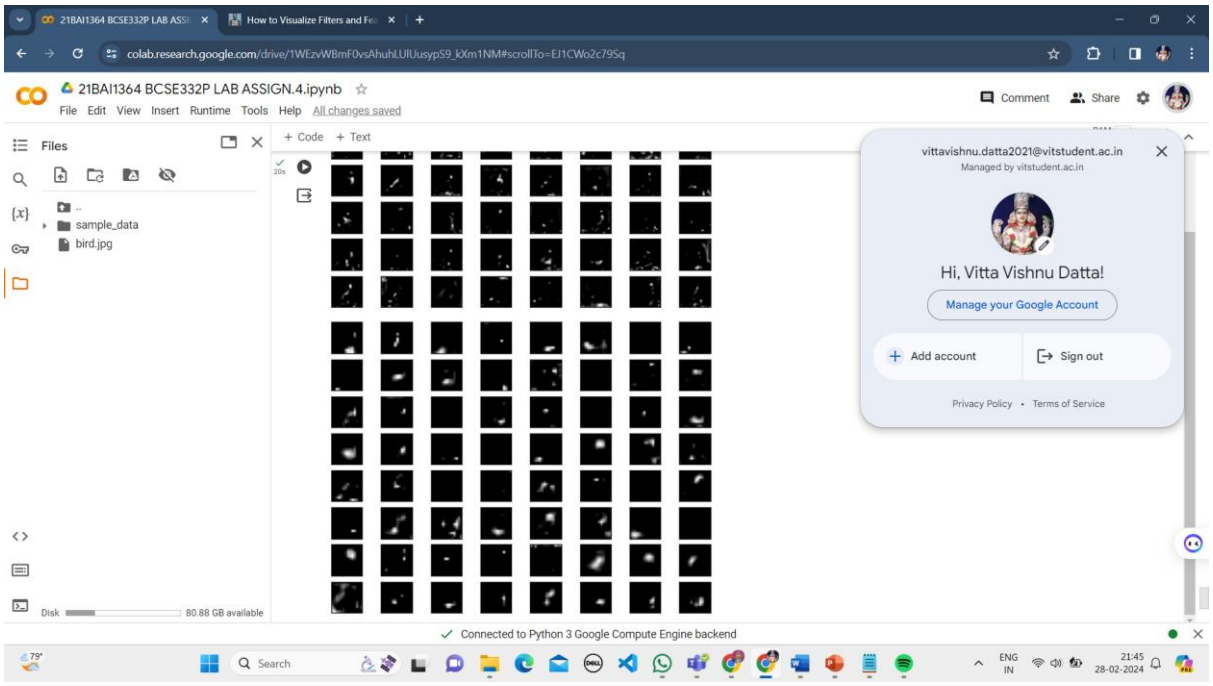
Vellore Institute of Technology

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

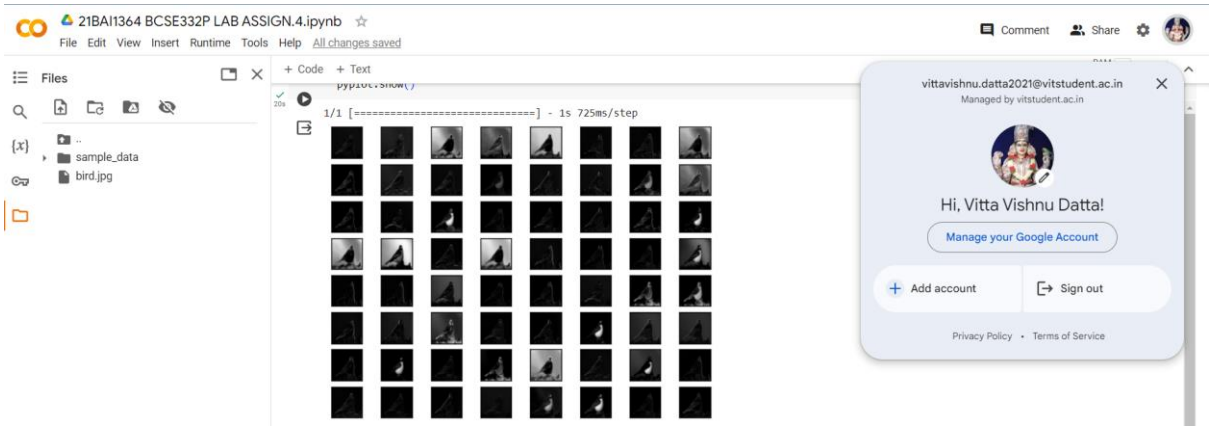
CHENNAI



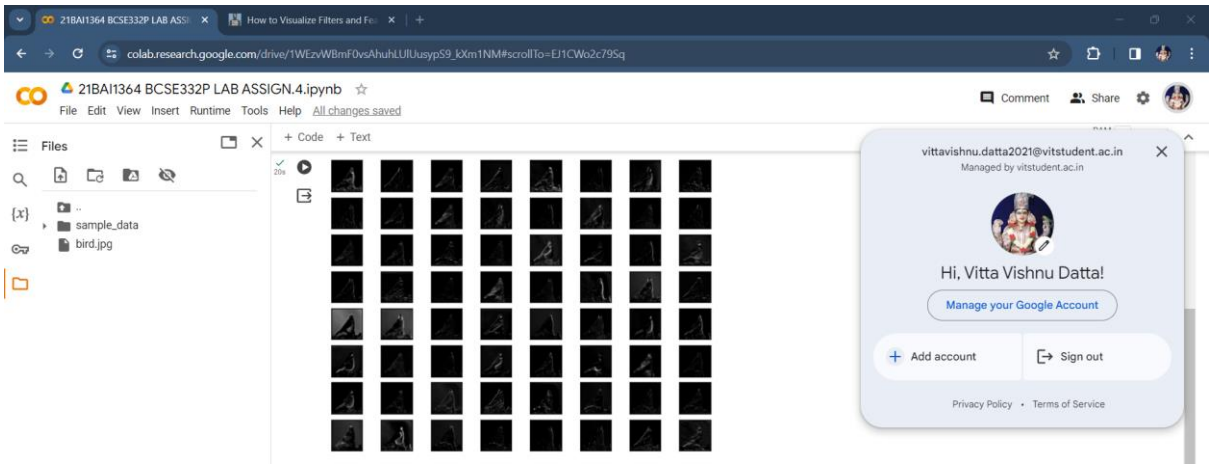




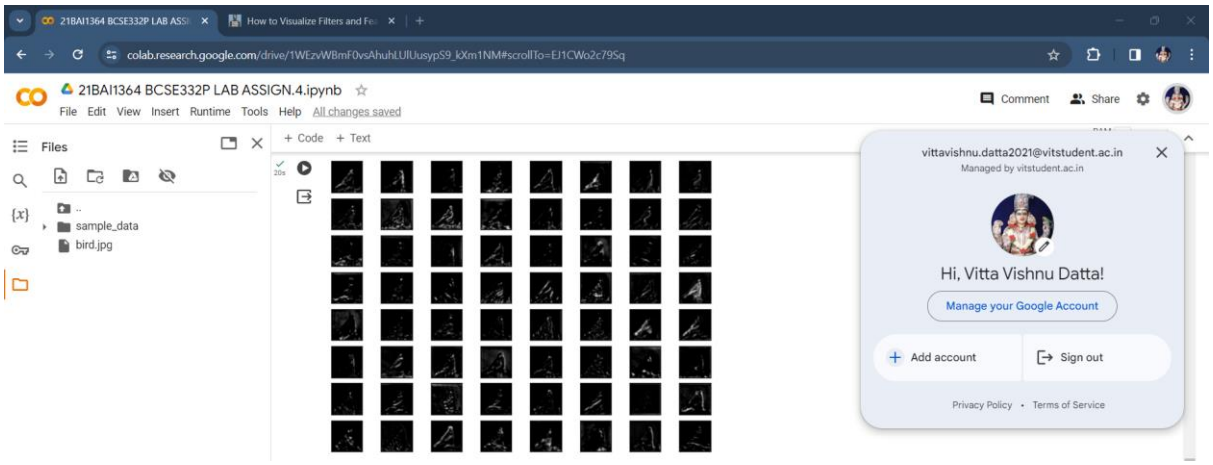
Visualization of the Feature Maps Extracted From Block 1 in the VGG16 Model



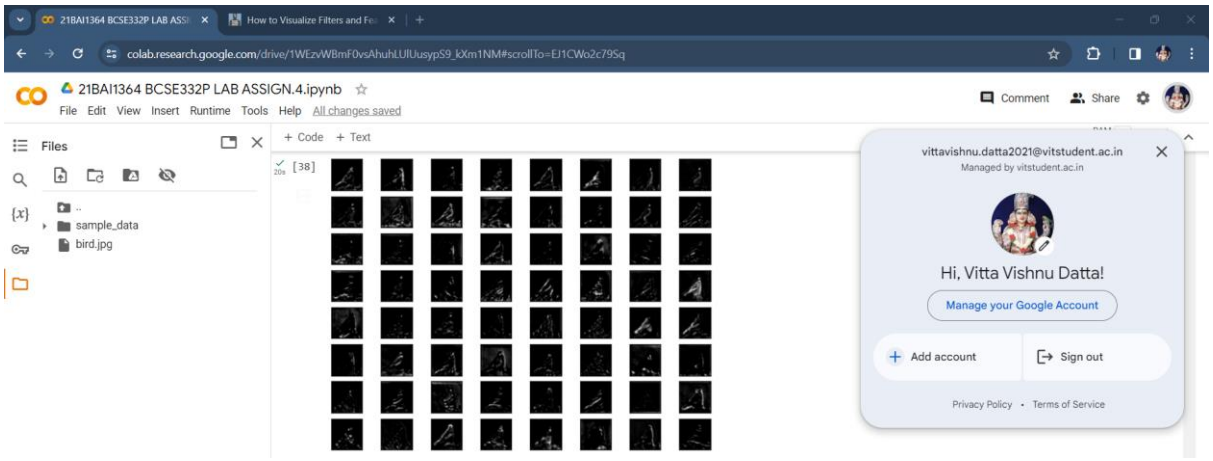
Visualization of the Feature Maps Extracted From Block 2 in the VGG16 Model



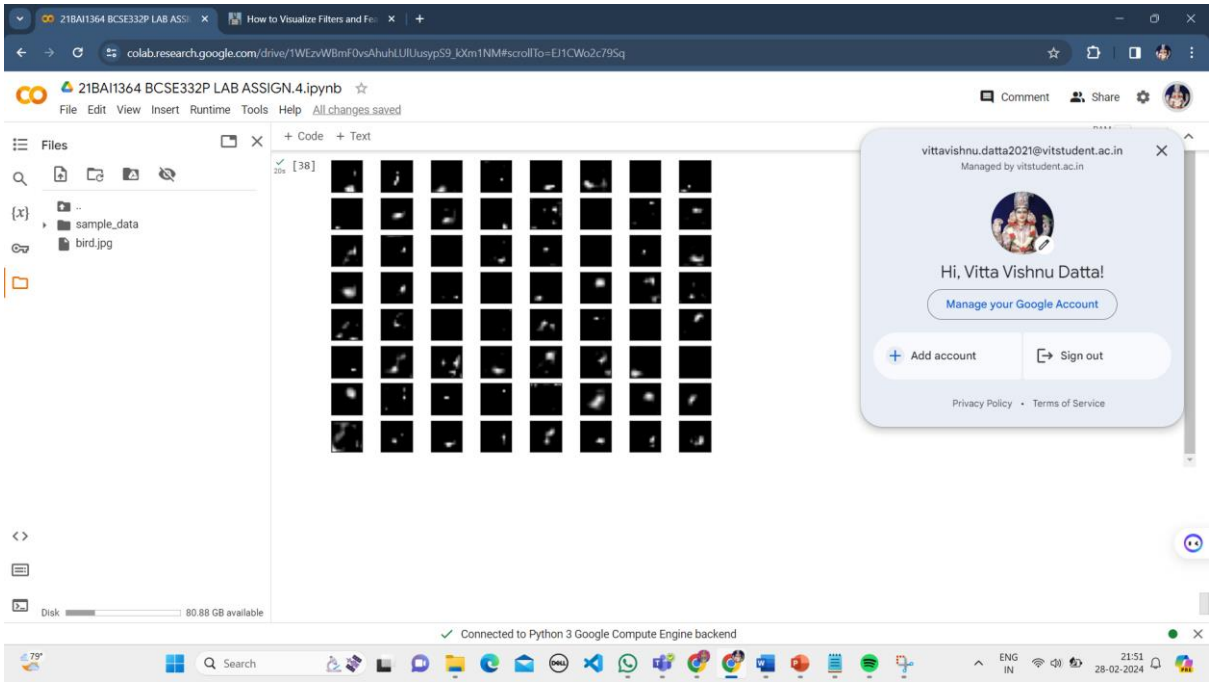
Visualization of the Feature Maps Extracted From Block 3 in the VGG16 Model



Visualization of the Feature Maps Extracted From Block 4 in the VGG16 Model



Visualization of the Feature Maps Extracted From Block 5 in the VGG16 Model



CODE FOR PRE-TRAINED VGG 16 MODEL:

```
import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras import layers, models
import matplotlib.pyplot as plt

# Load the CIFAR-10 dataset
(train_data, val_data), info = tfds.load(
    'cifar10',
    split=['train[:80%]', 'train[80%:]'],
    with_info=True,
    as_supervised=True
)

# Resize images to match AlexNet's input size (227x227 pixels)
def preprocess_image(image, label):
    image = tf.image.resize(image, (227, 227))
    return image, label

train_data = train_data.map(preprocess_image)
val_data = val_data.map(preprocess_image)
# Define AlexNet-like architecture
def alexnet_like(input_shape=(227, 227, 3), num_classes=10):
    model = models.Sequential()

    # Adjust the layers for smaller input size
    model.add(layers.Conv2D(96, (11, 11), strides=(4, 4),
activation='relu', input_shape=input_shape))
    model.add(layers.MaxPooling2D((3, 3), strides=(2, 2)))

    model.add(layers.Conv2D(256, (5, 5), activation='relu'))
    model.add(layers.MaxPooling2D((3, 3), strides=(2, 2)))

    model.add(layers.Conv2D(384, (3, 3), activation='relu'))
    model.add(layers.Conv2D(384, (3, 3), activation='relu'))
    model.add(layers.Conv2D(256, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((3, 3), strides=(2, 2)))

    model.add(layers.Flatten())
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dropout(0.5))
```

```
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dropout(0.5))

    model.add(layers.Dense(num_classes, activation='softmax'))

    return model

import tensorflow as tf

# Assuming `alexnet_like` function exists and defines the model
architecture

# Define placeholder variables for data and labels (customize based on
your data)
image_data = tf.keras.Input(shape=(224, 224, 3))  # Example image input
shape (adjust as needed)
label_data = tf.keras.Input(shape=(1,))  # Example label shape (adjust
as needed)

# Define your training data loading and preprocessing logic here
(replace with your specific code)
# You can use libraries like pandas, NumPy, or image processing
libraries to load and preprocess your training data.
# For demonstration purposes, we'll use placeholder data:
train_images = tf.random.normal(shape=(10000, 224, 224, 3))  # Replace
with your actual training images
train_labels = tf.random.uniform(shape=(10000,), minval=0, maxval=9,
dtype=tf.int32)  # Replace with your actual labels

# Create the model using the `alexnet_like` function
model = alexnet_like(inputs=image_data)

# Compile the model with appropriate loss and optimizer
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Define batch size and number of epochs
batch_size = 128
epochs = 10
```

```
# Create TensorFlow datasets from training data, assuming proper
preprocessing is done
train_dataset = tf.data.Dataset.from_tensor_slices((train_images,
train_labels))
train_dataset =
train_dataset.shuffle(buffer_size=50000).batch(batch_size)

# Shuffle and batch validation data (assuming you have validation data)
# Replace with your validation data loading and preprocessing logic
# val_images = ... (your validation images)
# val_labels = ... (your validation labels)
# val_dataset = tf.data.Dataset.from_tensor_slices((val_images,
val_labels))
# val_dataset = val_dataset.batch(batch_size)

# Train the model
history = model.fit(train_dataset, epochs=epochs,
validation_data=None)  # Replace with `val_dataset` if you have
validation data

# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
# Uncomment the following line if you have validation data
# plt.plot(history.history['val_accuracy'], label='Validation
Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
# Uncomment the following line if you have validation data
# plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()

from keras.models import Sequential
```

```
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Define the AlexNet model
model = Sequential()

# Convolutional layers
model.add(Conv2D(96, (11, 11), strides=(4, 4), activation='relu',
input_shape=(224, 224, 3)))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

model.add(Conv2D(256, (5, 5), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

model.add(Conv2D(384, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(384, (3, 3), activation='relu', padding='same'))
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))

# Fully connected layers
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000, activation='softmax'))  # Change 1000 to the
number of classes in your problem

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Print the model summary
model.summary()
```

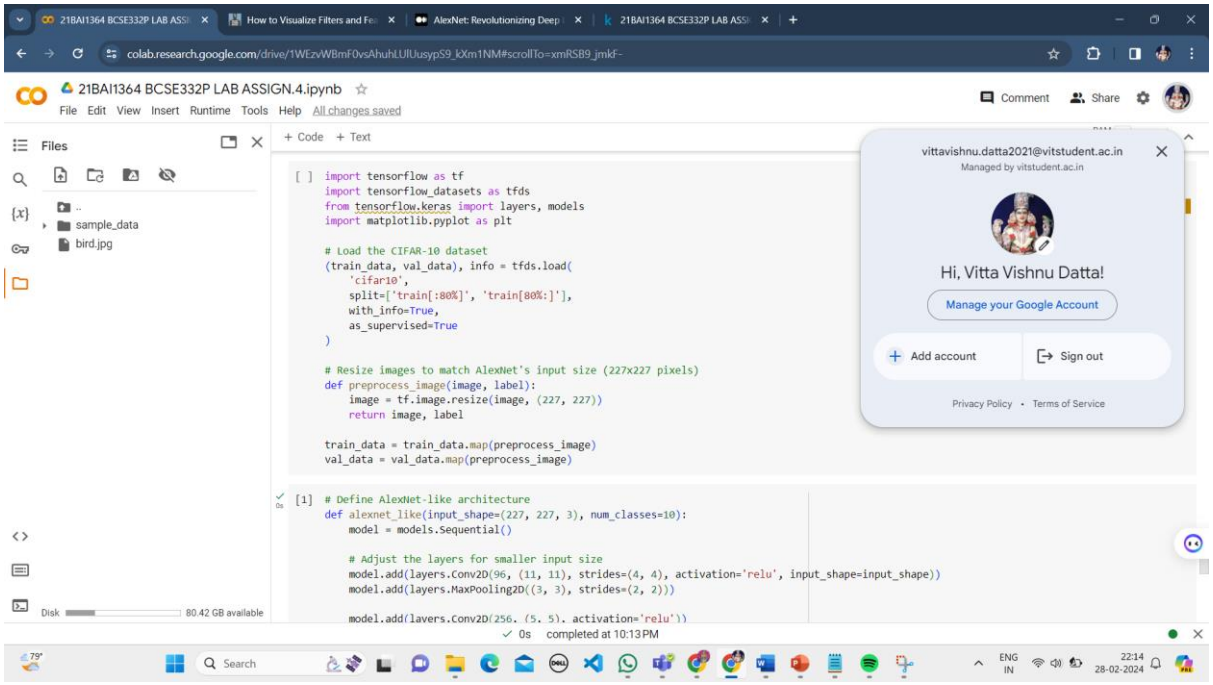
VIT®

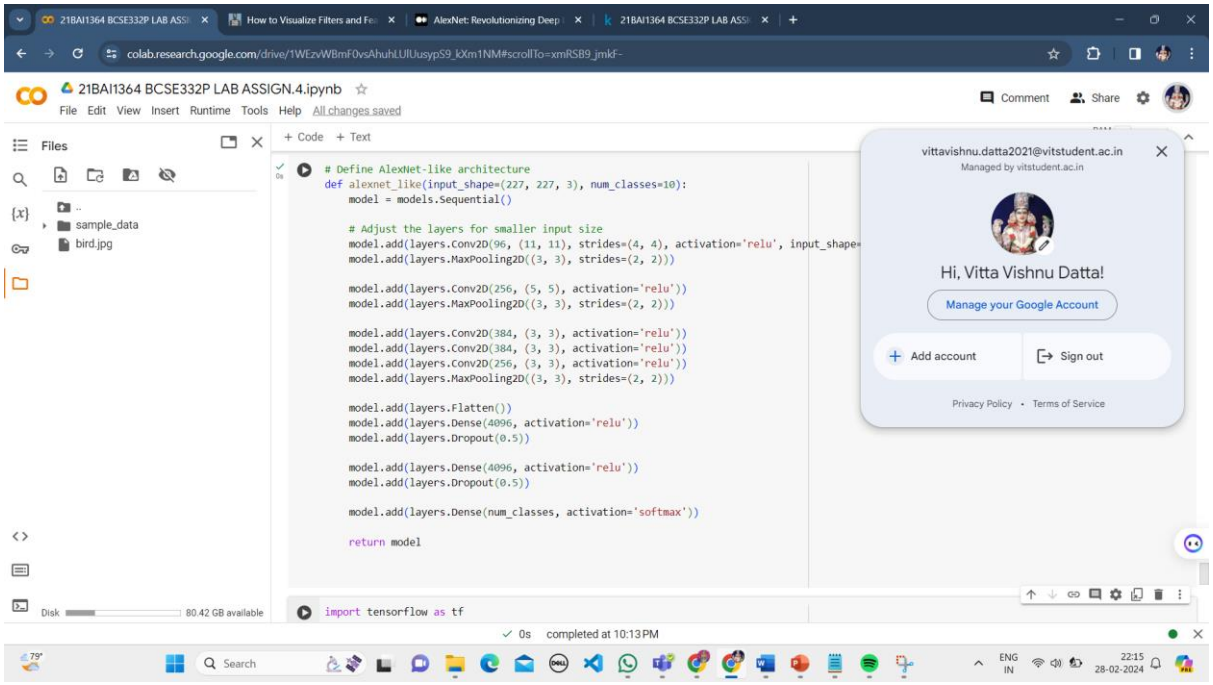
Vellore Institute of Technology

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

CHENNAI

SCREENSHOTS OF THE ALEX-NET MODEL:

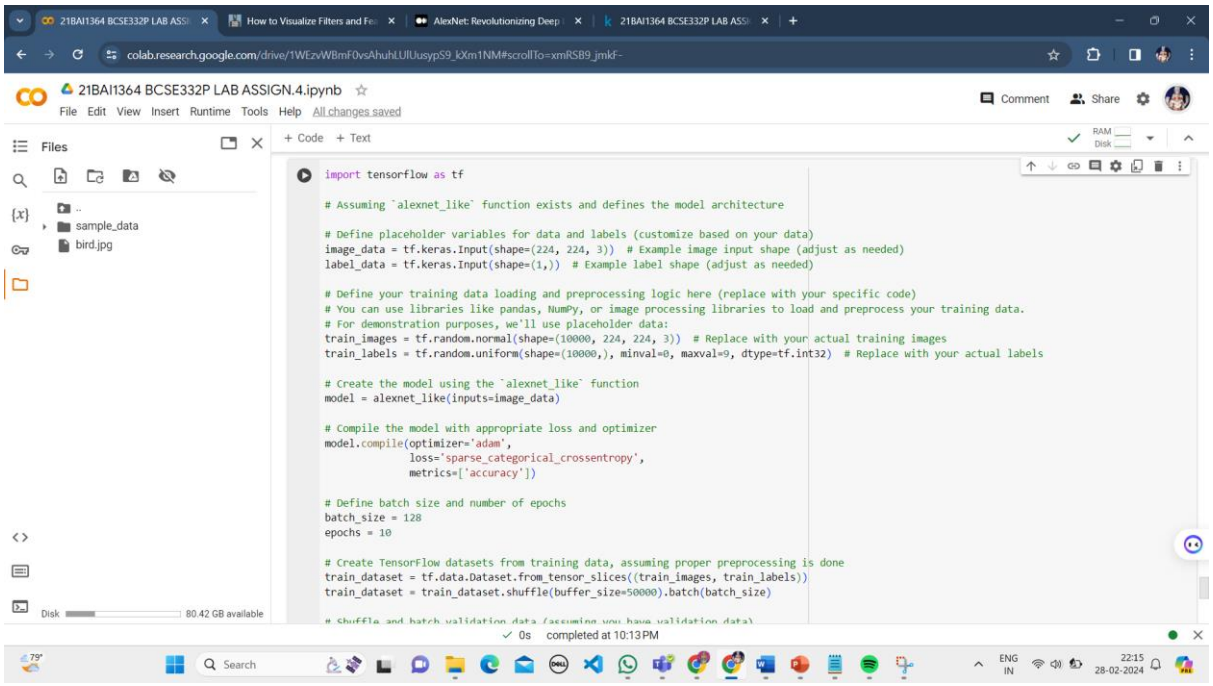


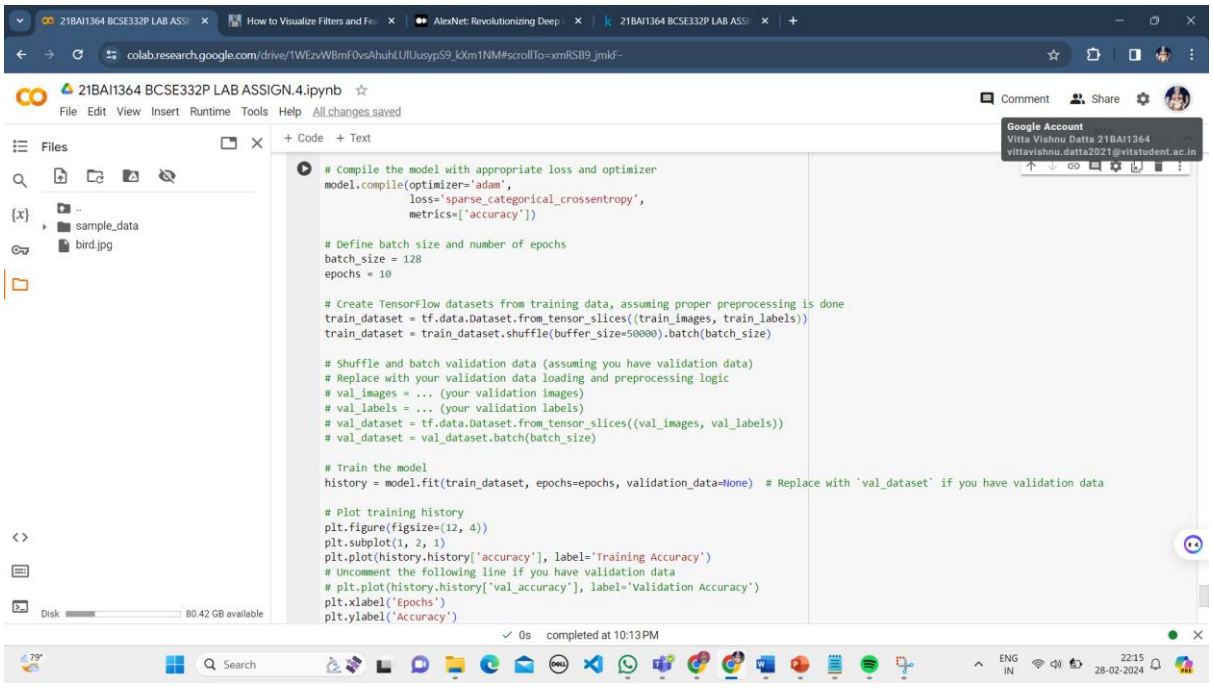


Vellore Institute of Technology

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

CHENNAI



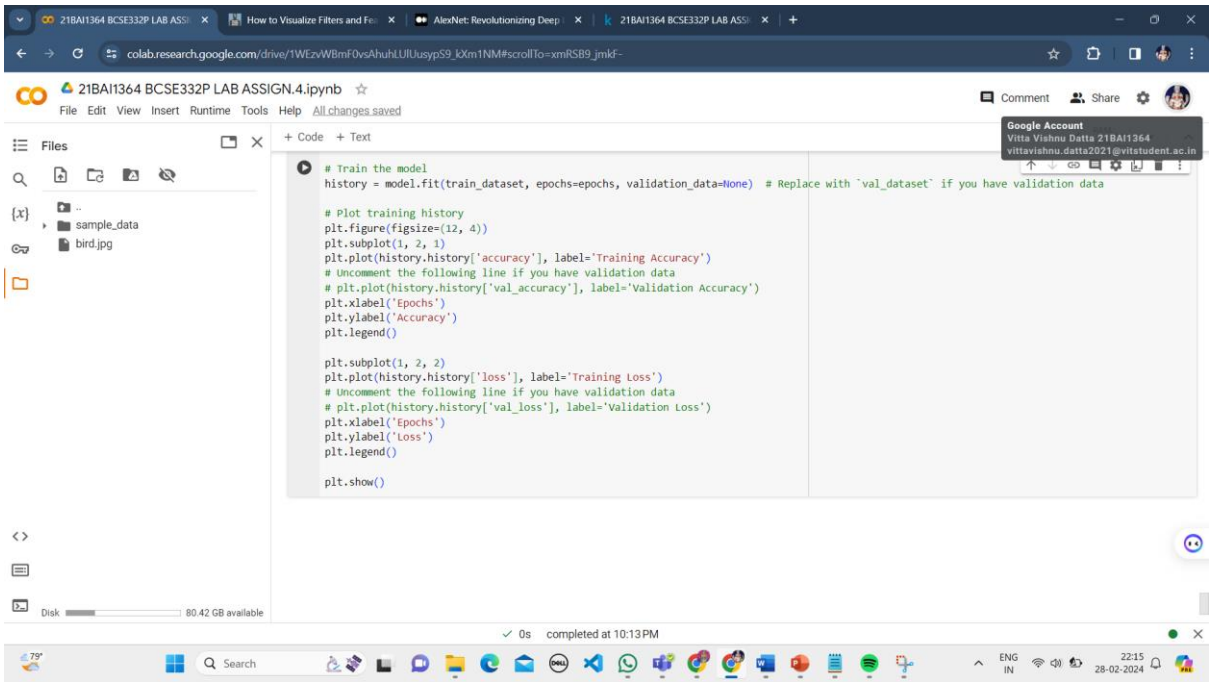


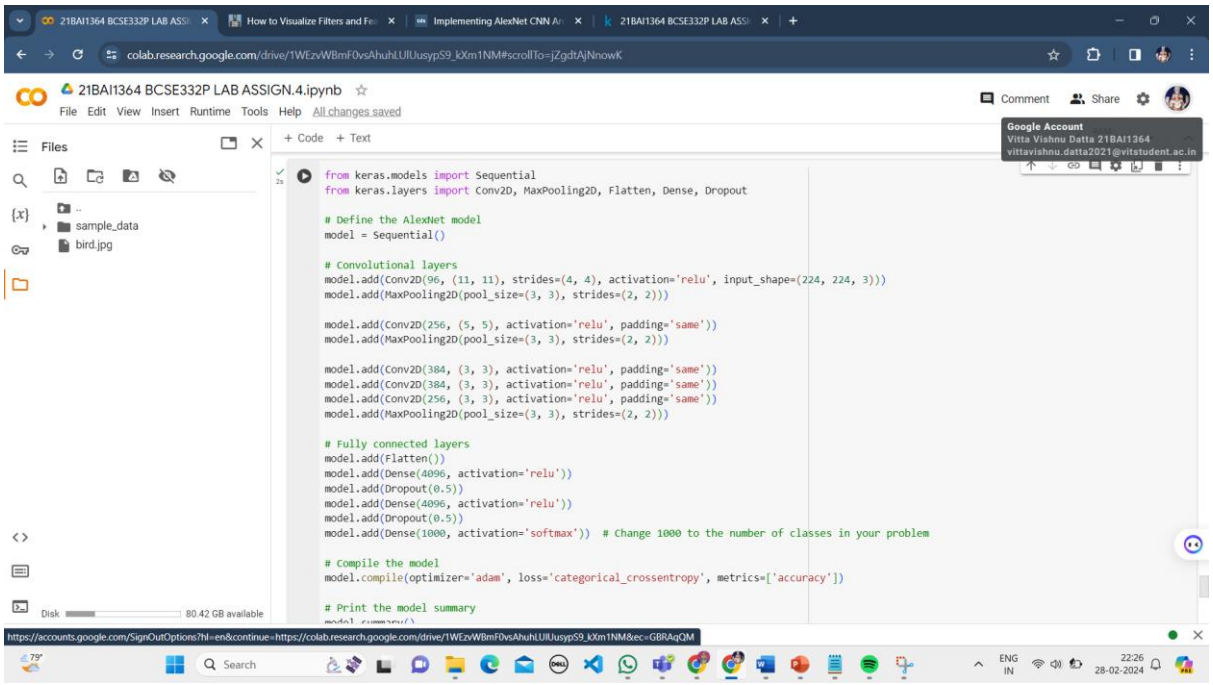
VIT®

Vellore Institute of Technology

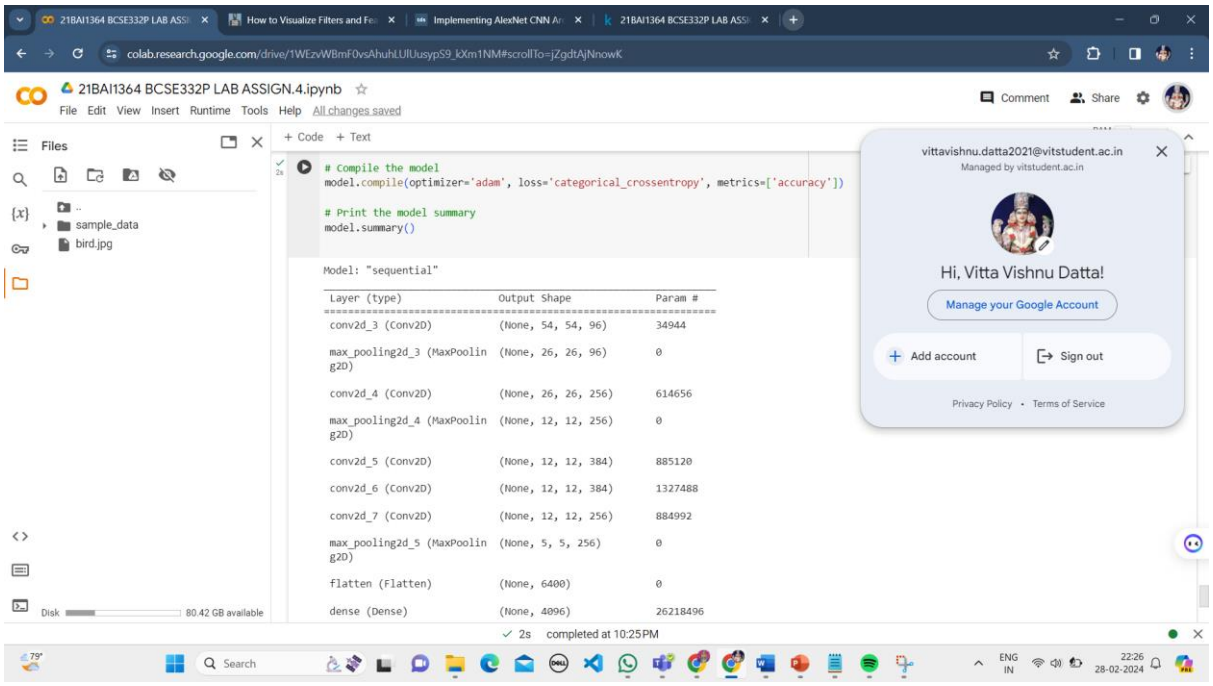
(Deemed to be University under section 3 of UGC Act, 1956)

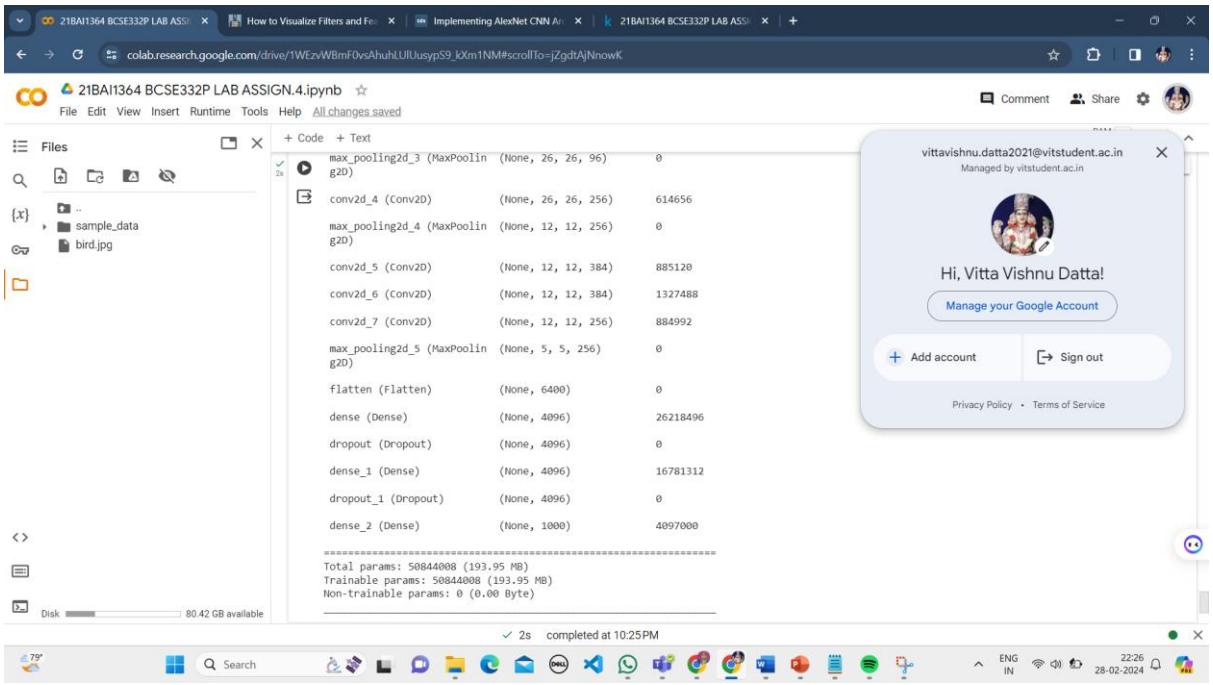
CHENNAI





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

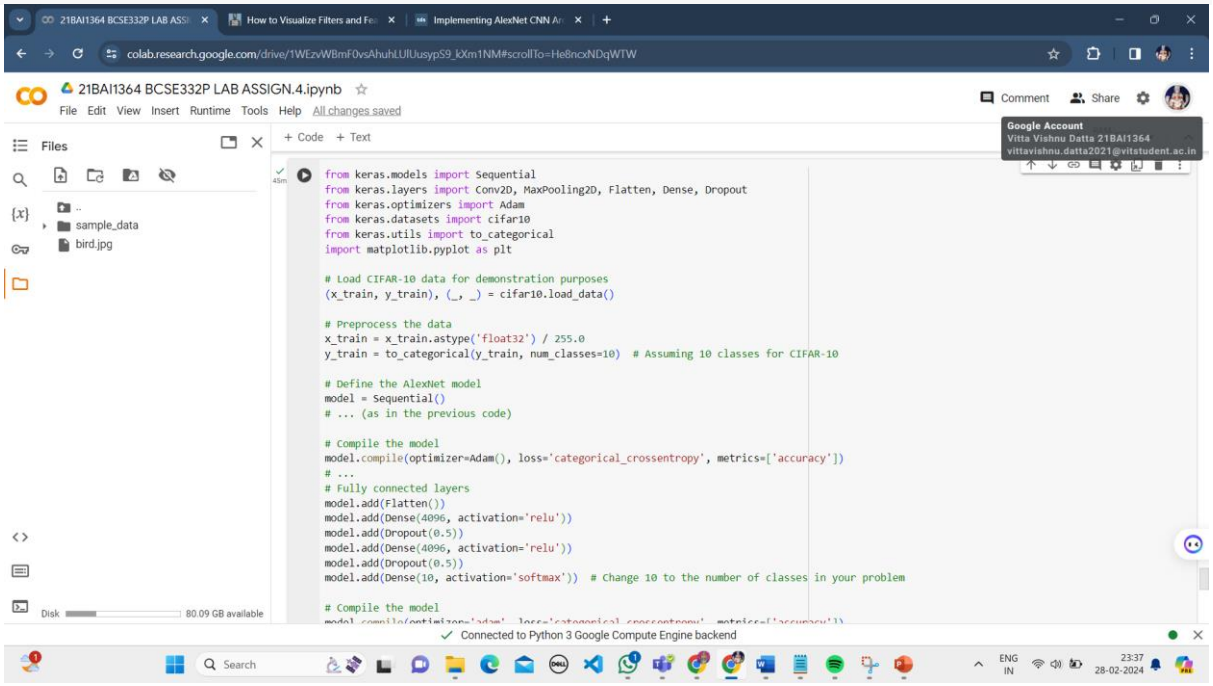





```
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
```

```
from keras.optimizers import Adam
from keras.datasets import cifar10
from keras.utils import to_categorical
import matplotlib.pyplot as plt

# Load CIFAR-10 data for demonstration purposes
(x_train, y_train), (_, _) = cifar10.load_data()

# Preprocess the data
x_train = x_train.astype('float32') / 255.0
y_train = to_categorical(y_train, num_classes=10)  # Assuming 10
classes for CIFAR-10

# Define the AlexNet model
model = Sequential()
# ... (as in the previous code)

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy',
metrics=['accuracy'])
# ...
# Fully connected layers
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))  # Change 10 to the number
of classes in your problem

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
# ...

# Train the model
history = model.fit(x_train, y_train, epochs=10, batch_size=128,
validation_split=0.2)

# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
```

```
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

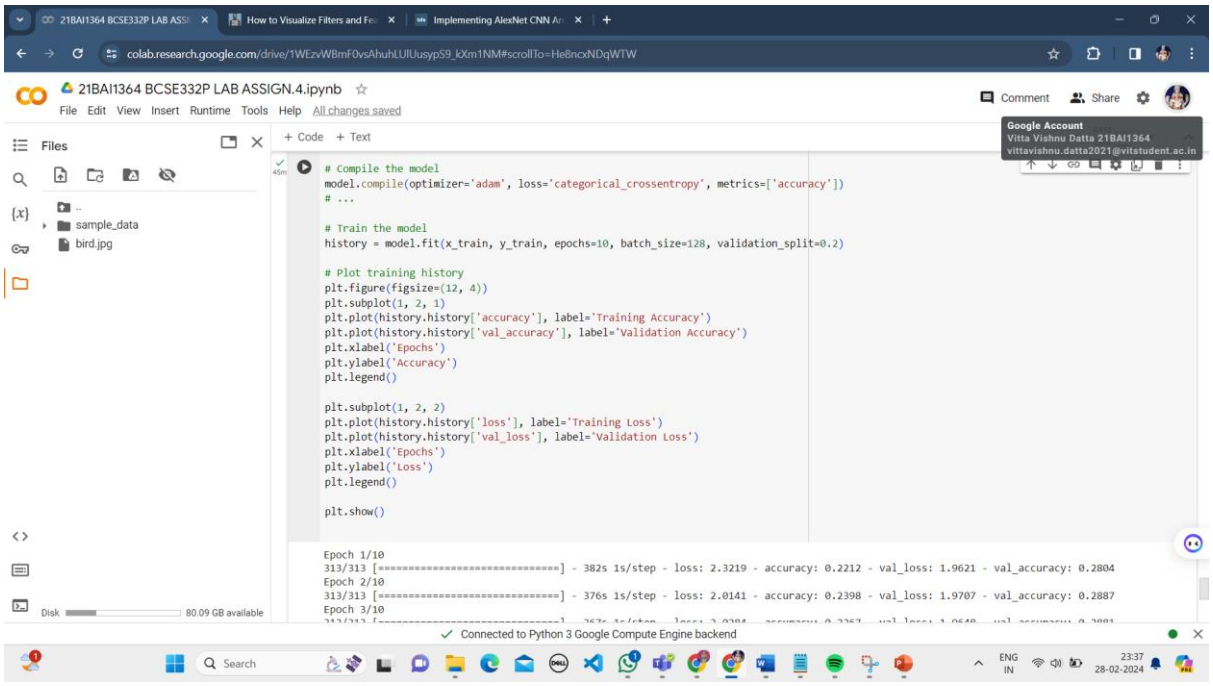


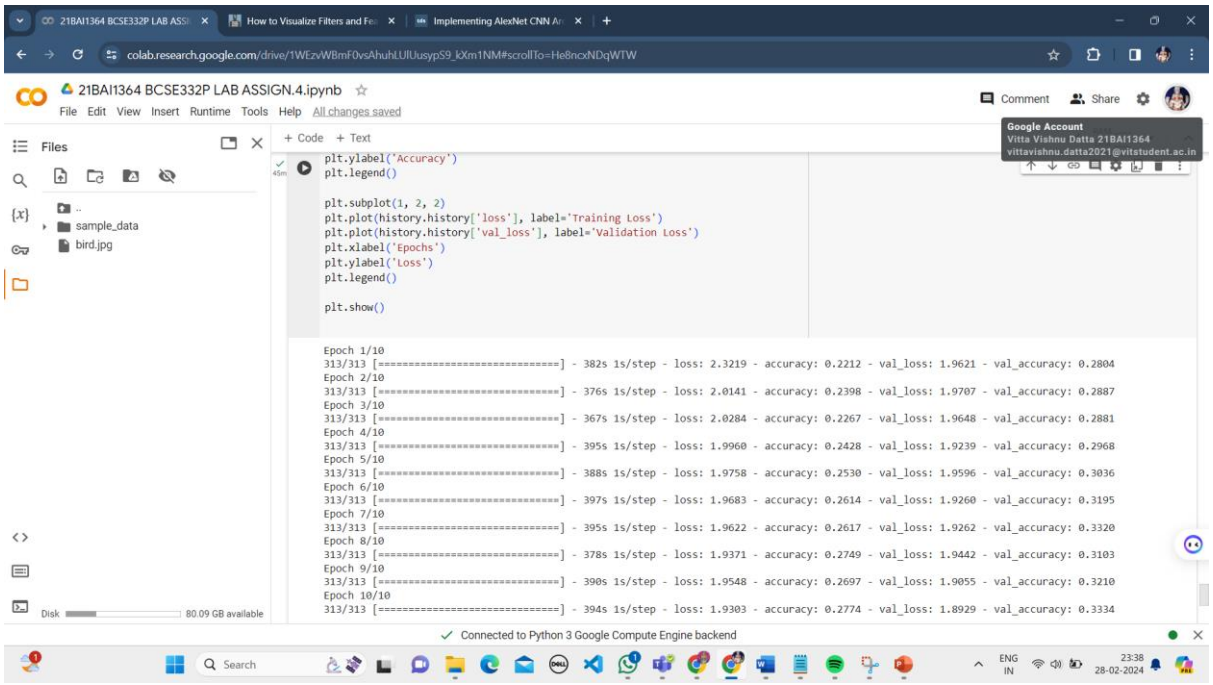
VIT®

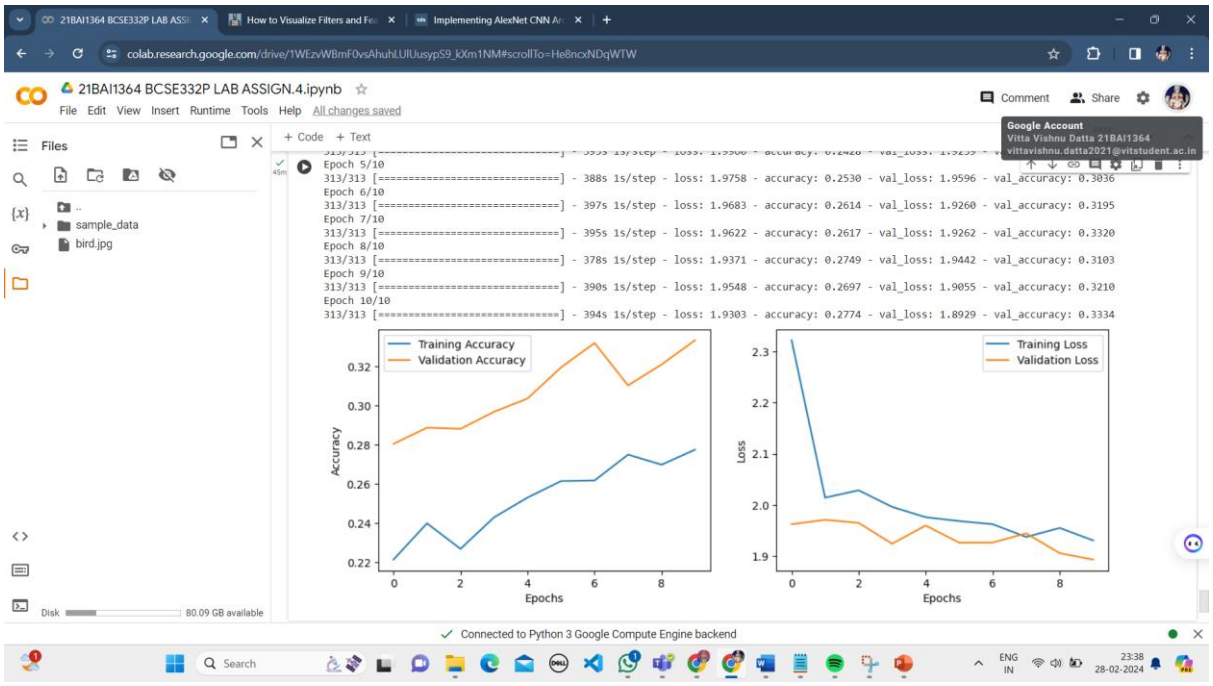
Vellore Institute of Technology

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

CHENNAI







4. Design a new CNN model with your dataset

CODE:

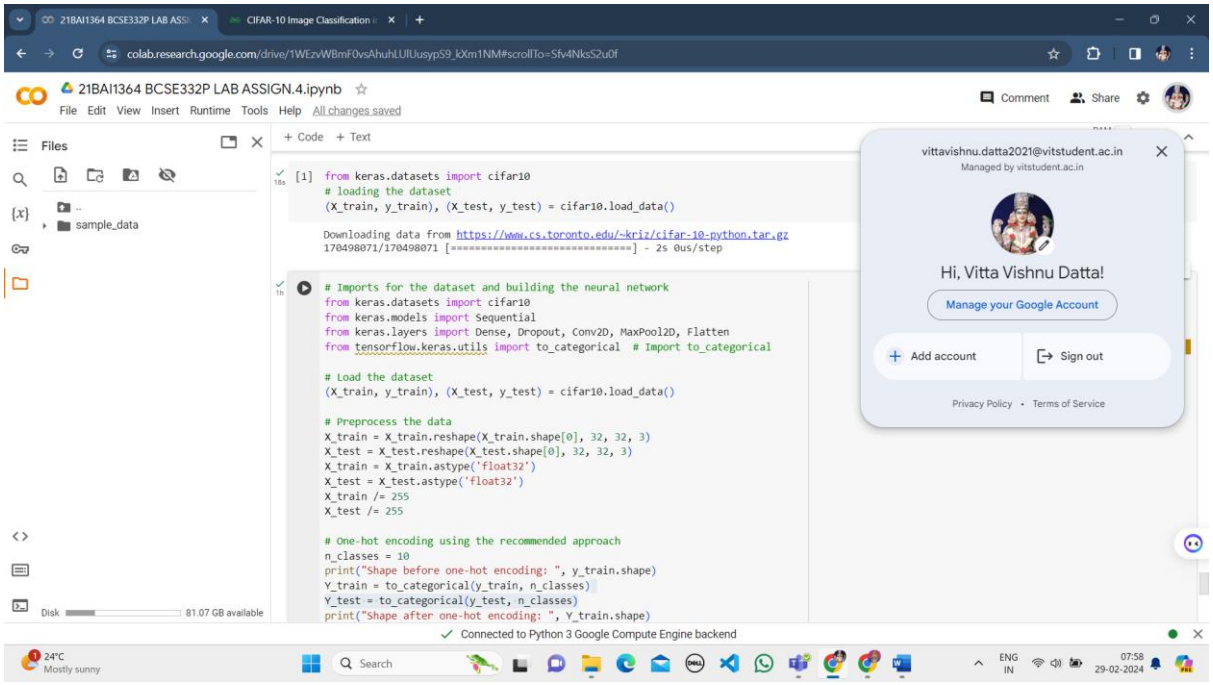
"DATASET CONSIDERED IS CIFAR10"

```
from keras.datasets import cifar10
# loading the dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
# Imports for the dataset and building the neural network
from keras.datasets import cifar10
from keras.models import Sequential
from keras.layers import Dense, Dropout, Conv2D, MaxPool2D, Flatten
from tensorflow.keras.utils import to_categorical  # Import
to_categorical

# Load the dataset
(X_train, y_train), (X_test, y_test) = cifar10.load_data()

# Preprocess the data
```

```
X_train = X_train.reshape(X_train.shape[0], 32, 32, 3)
X_test = X_test.reshape(X_test.shape[0], 32, 32, 3)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255

# One-hot encoding using the recommended approach
n_classes = 10
print("Shape before one-hot encoding: ", y_train.shape)
Y_train = to_categorical(y_train, n_classes)
Y_test = to_categorical(y_test, n_classes)
print("Shape after one-hot encoding: ", Y_train.shape)

# Build the CNN model
model = Sequential()

# convolutional layer
model.add(Conv2D(50, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu', input_shape=(32, 32, 3)))

# convolutional layer
model.add(Conv2D(75, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(125, kernel_size=(3,3), strides=(1,1), padding='same',
activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# flatten output of conv
model.add(Flatten())

# hidden layer
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(250, activation='relu'))
model.add(Dropout(0.3))

# output layer
```

```
model.add(Dense(10, activation='softmax'))

# Compile and train the model
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],
optimizer='adam')
model.fit(X_train, Y_train, batch_size=128, epochs=10,
validation_data=(X_test, Y_test))
```

SCREENSHOTS OF THE CIFAR 10 CNN MODEL:

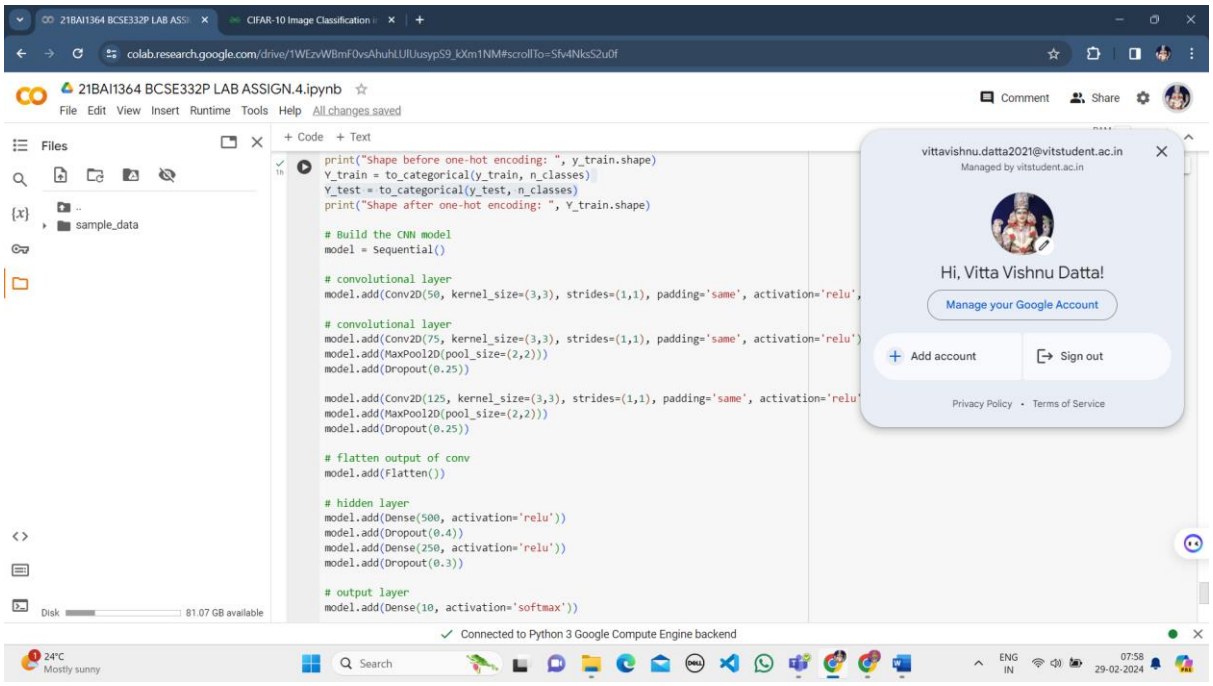


VIT®

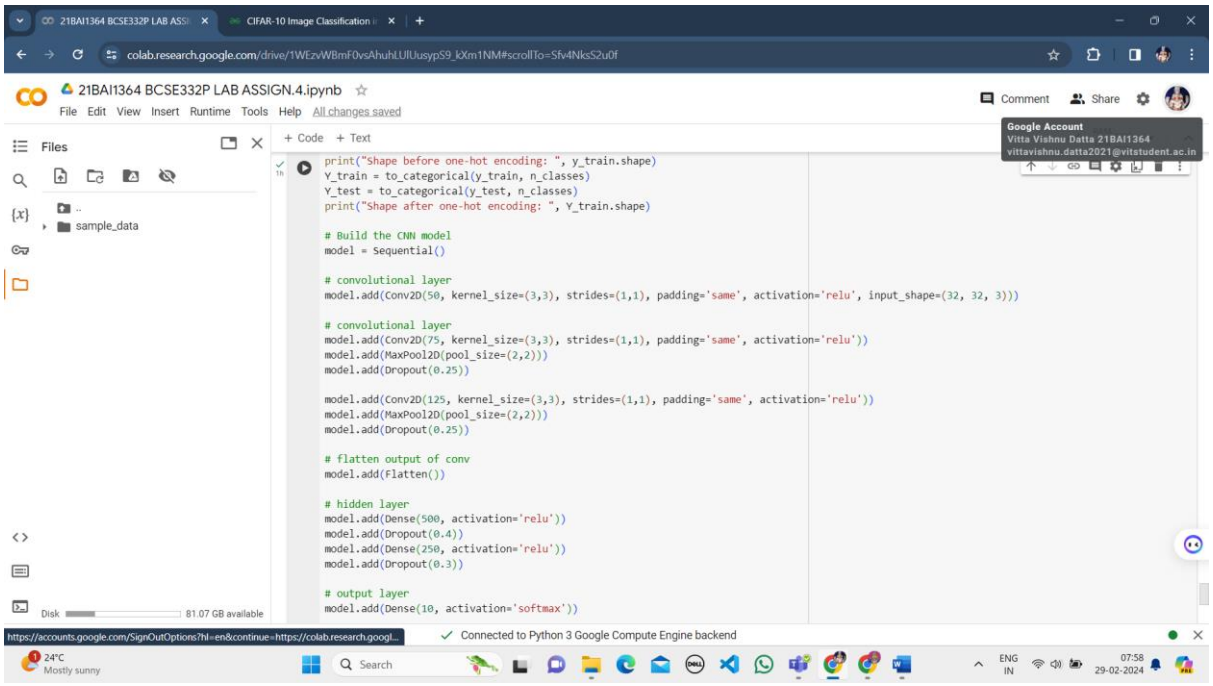
Vellore Institute of Technology

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

CHENNAI

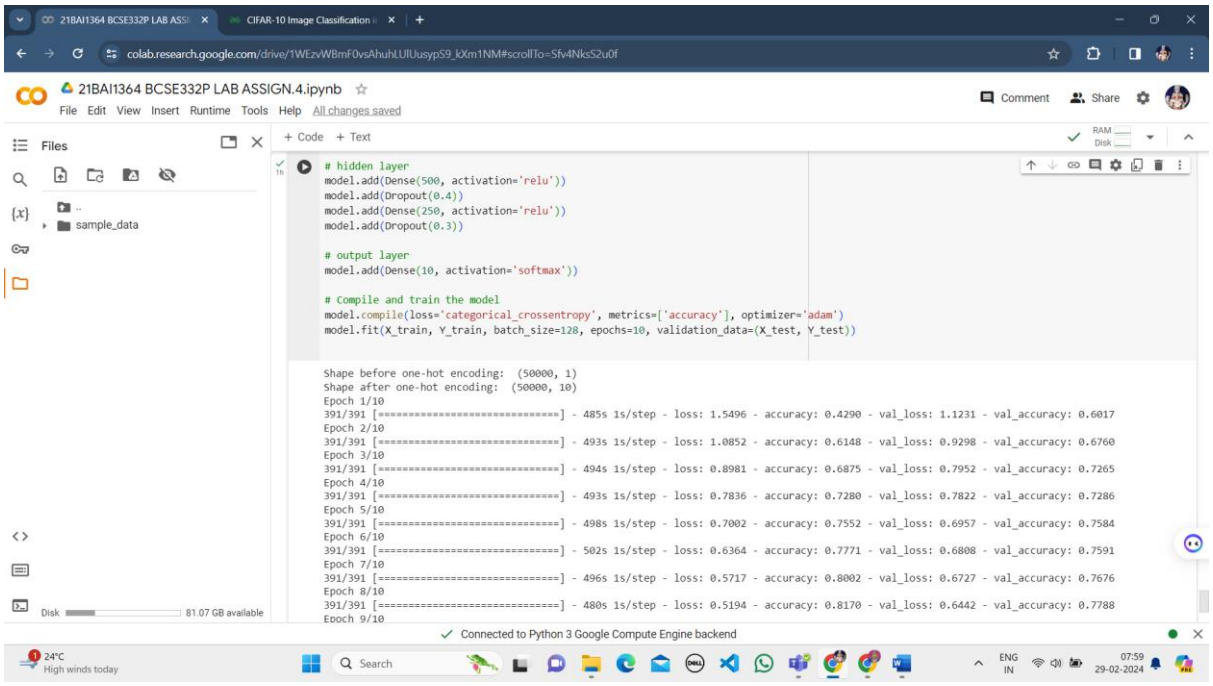


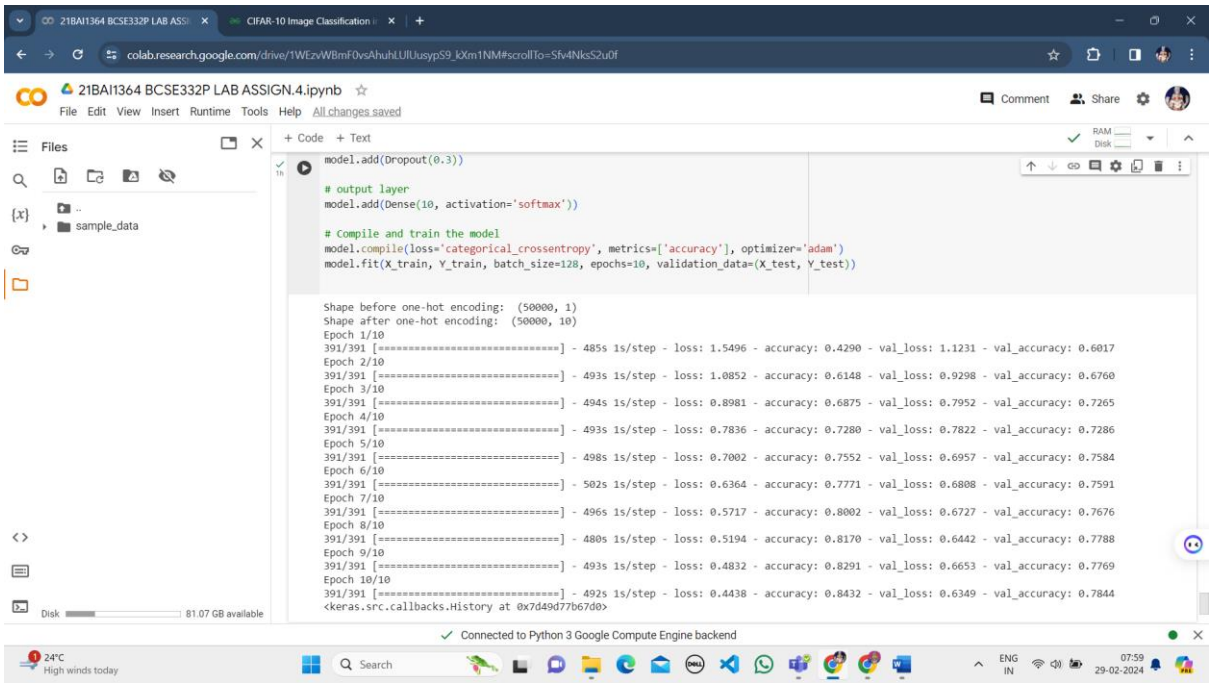

```
print("Shape before one-hot encoding: ", y_train.shape)
Y_train = to_categorical(y_train, n_classes)
Y_test = to_categorical(y_test, n_classes)
print("Shape after one-hot encoding: ", Y_train.shape)

# Build the CNN model
model = Sequential()

# convolutional layer
model.add(Conv2D(50, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu',

# convolutional layer
model.add(Conv2D(75, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu')
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(125, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# flatten output of conv
model.add(Flatten())

# hidden layer
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(250, activation='relu'))
model.add(Dropout(0.3))

# output layer
model.add(Dense(10, activation='softmax'))
```




```
print("Shape before one-hot encoding: ", y_train.shape)
Y_train = to_categorical(y_train, n_classes)
Y_test = to_categorical(y_test, n_classes)
print("Shape after one-hot encoding: ", Y_train.shape)

# Build the CNN model
model = Sequential()

# convolutional layer
model.add(Conv2D(50, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu', input_shape=(32, 32, 3)))

# convolutional layer
model.add(Conv2D(75, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(125, kernel_size=(3,3), strides=(1,1), padding='same', activation='relu'))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Dropout(0.25))

# flatten output of conv
model.add(Flatten())

# hidden layer
model.add(Dense(500, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(250, activation='relu'))
model.add(Dropout(0.3))

# output layer
model.add(Dense(10, activation='softmax'))
```

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



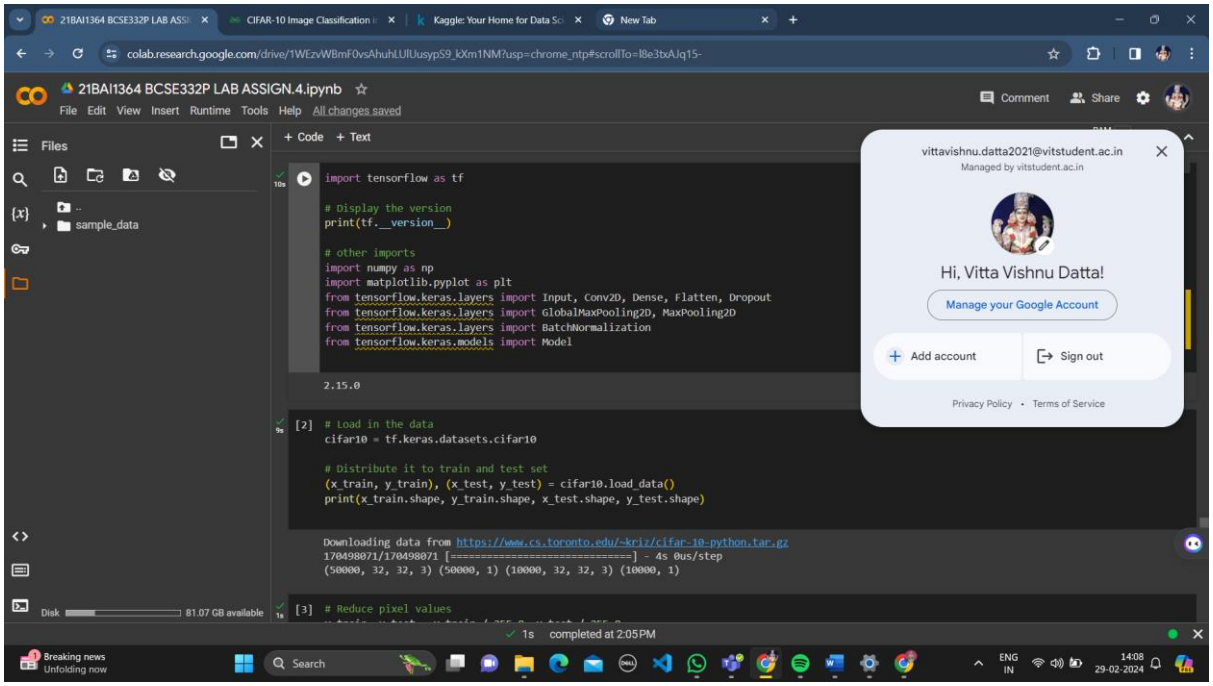


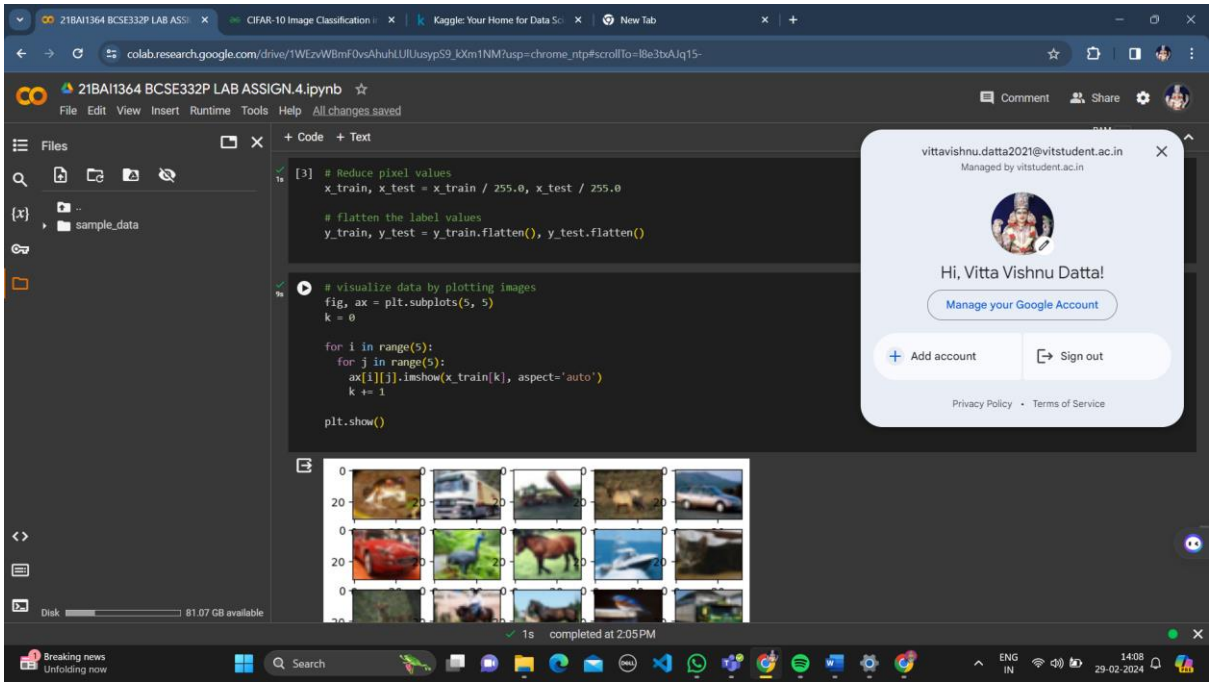
```
import tensorflow as tf

# Display the version
print(tf.__version__)

# other imports
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.layers import Input, Conv2D, Dense, Flatten,
Dropout
from tensorflow.keras.layers import GlobalMaxPooling2D, MaxPooling2D
from tensorflow.keras.layers import BatchNormalization
from tensorflow.keras.models import Model
# Load in the data
cifar10 = tf.keras.datasets.cifar10

# Distribute it to train and test set
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)

# Reduce pixel values
x_train, x_test = x_train / 255.0, x_test / 255.0

# flatten the label values
y_train, y_test = y_train.flatten(), y_test.flatten()
# visualize data by plotting images
fig, ax = plt.subplots(5, 5)
k = 0

for i in range(5):
  for j in range(5):
    ax[i][j].imshow(x_train[k], aspect='auto')
    k += 1

plt.show()
# number of classes
K = len(set(y_train))

# calculate total number of classes
# for output layer
print("number of classes:", K)
```

```
# Build the model using the functional API
# input layer
i = Input(shape=x_train[0].shape)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(i)
x = BatchNormalization()(x)
x = Conv2D(32, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(64, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = Conv2D(128, (3, 3), activation='relu', padding='same')(x)
x = BatchNormalization()(x)
x = MaxPooling2D((2, 2))(x)

x = Flatten()(x)
x = Dropout(0.2)(x)

# Hidden layer
x = Dense(1024, activation='relu')(x)
x = Dropout(0.2)(x)

# last hidden layer i.e.. output layer
x = Dense(K, activation='softmax')(x)

model = Model(i, x)

# model description
model.summary()
# Compile
model.compile(optimizer='adam',
      loss='sparse_categorical_crossentropy',
      metrics=['accuracy'])
# Fit
r = model.fit(
x_train, y_train, validation_data=(x_test, y_test), epochs=5)
```

```
# Fit with data augmentation
# Note: if you run this AFTER calling
# the previous model.fit()
# it will CONTINUE training where it left off
batch_size = 32
data_generator = tf.keras.preprocessing.image.ImageDataGenerator(
width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)

train_generator = data_generator.flow(x_train, y_train, batch_size)
steps_per_epoch = x_train.shape[0] // batch_size

r = model.fit(train_generator, validation_data=(x_test, y_test),
      steps_per_epoch=steps_per_epoch, epochs=5)

# Plot accuracy per iteration
plt.plot(r.history['accuracy'], label='acc', color='red')
plt.plot(r.history['val_accuracy'], label='val_acc', color='green')
plt.legend()
labels = '''airplane automobile bird cat deerdog frog horseship
truck'''.split()

# select the image from our test dataset
image_number = 0

# display the image
plt.imshow(x_test[image_number])

# load the image in an array
n = np.array(x_test[image_number])

# reshape it
p = n.reshape(1, 32, 32, 3)

# pass in the network for prediction and
# save the predicted label
predicted_label = labels[model.predict(p).argmax()]

# load the original label
original_label = labels[y_test[image_number]]

# display the result
print("Original label is {} and predicted label is {}".format(
```

```
  original_label, predicted_label))
```



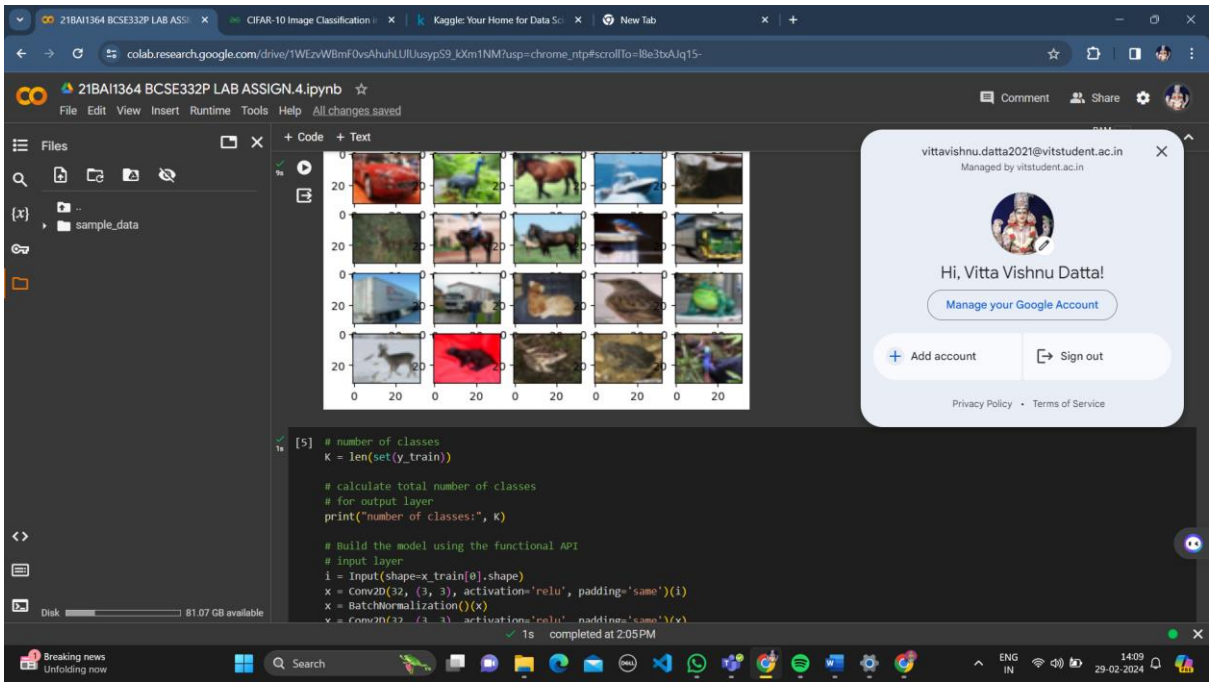


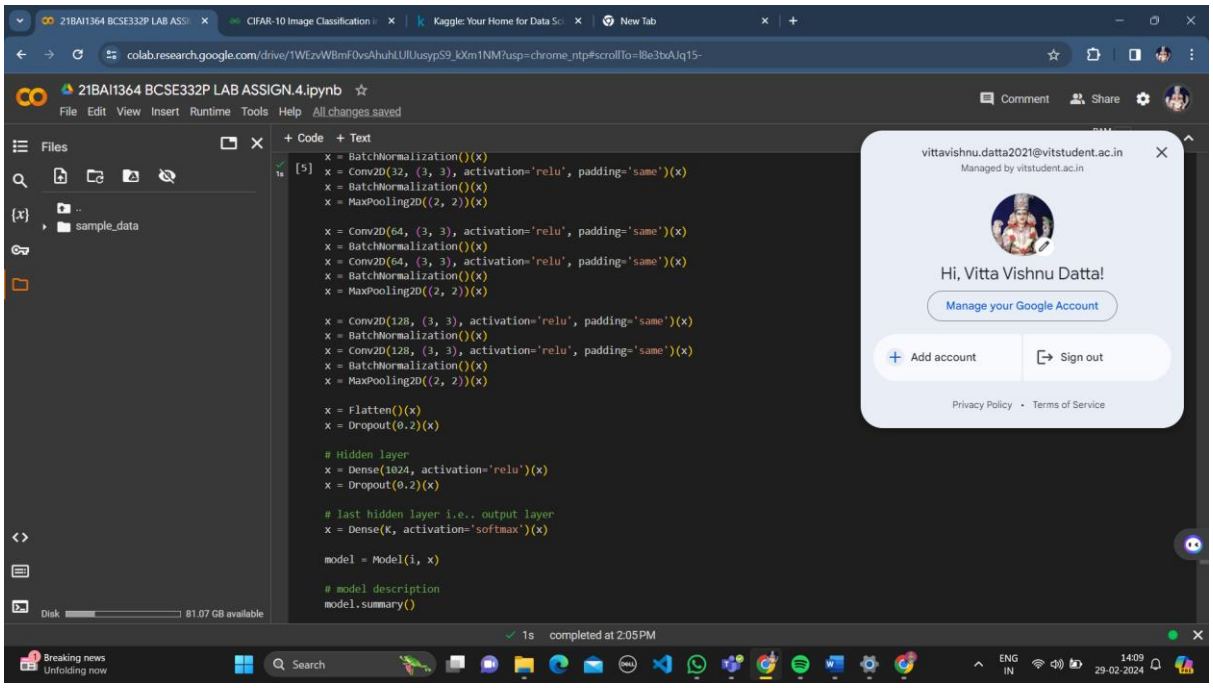
VIT®

Vellore Institute of Technology

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

CHENNAI



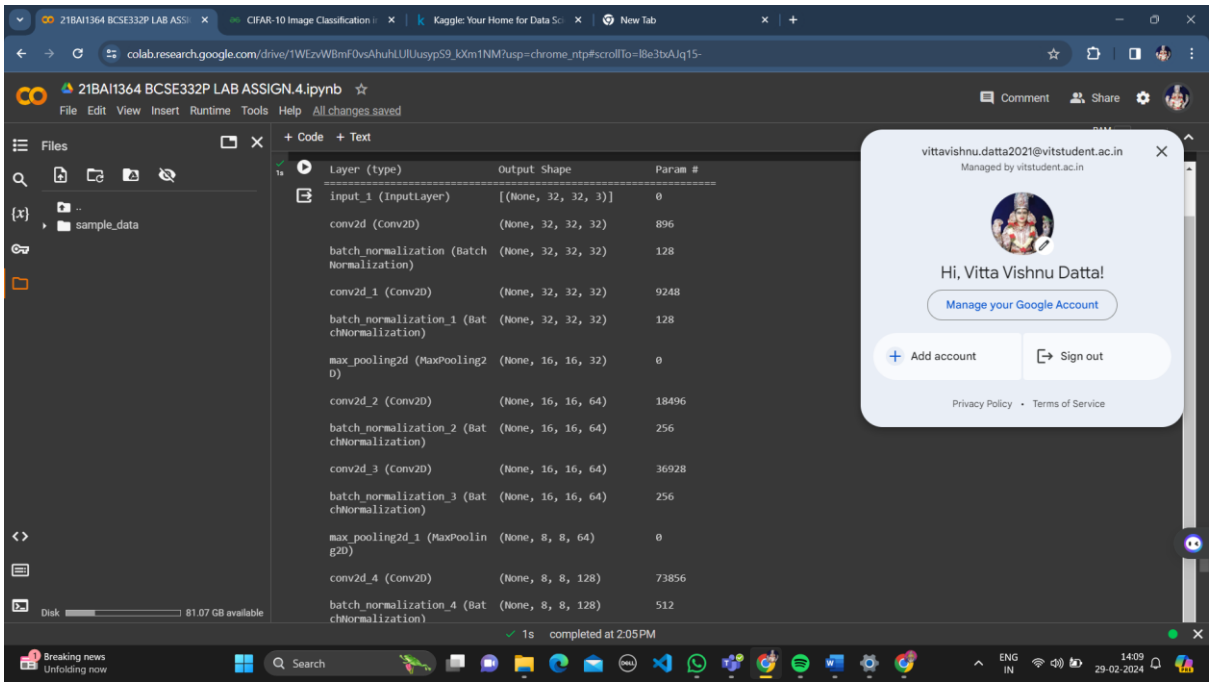


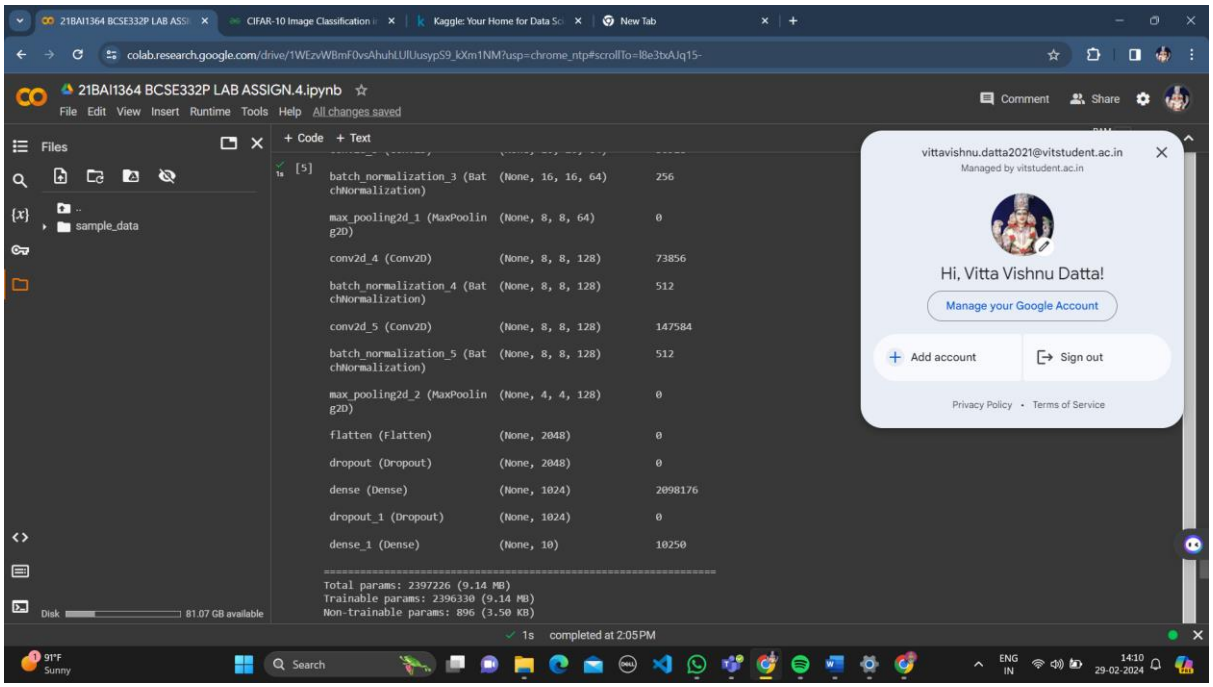
VIT®

Vellore Institute of Technology

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

CHENNAI



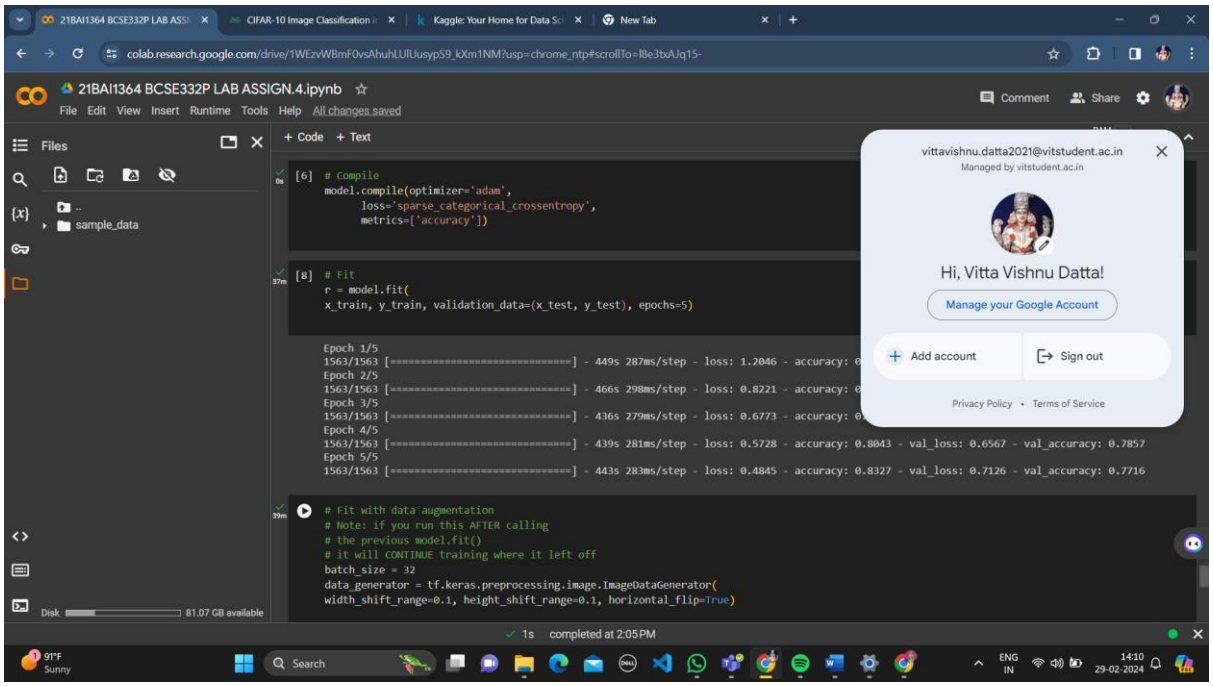


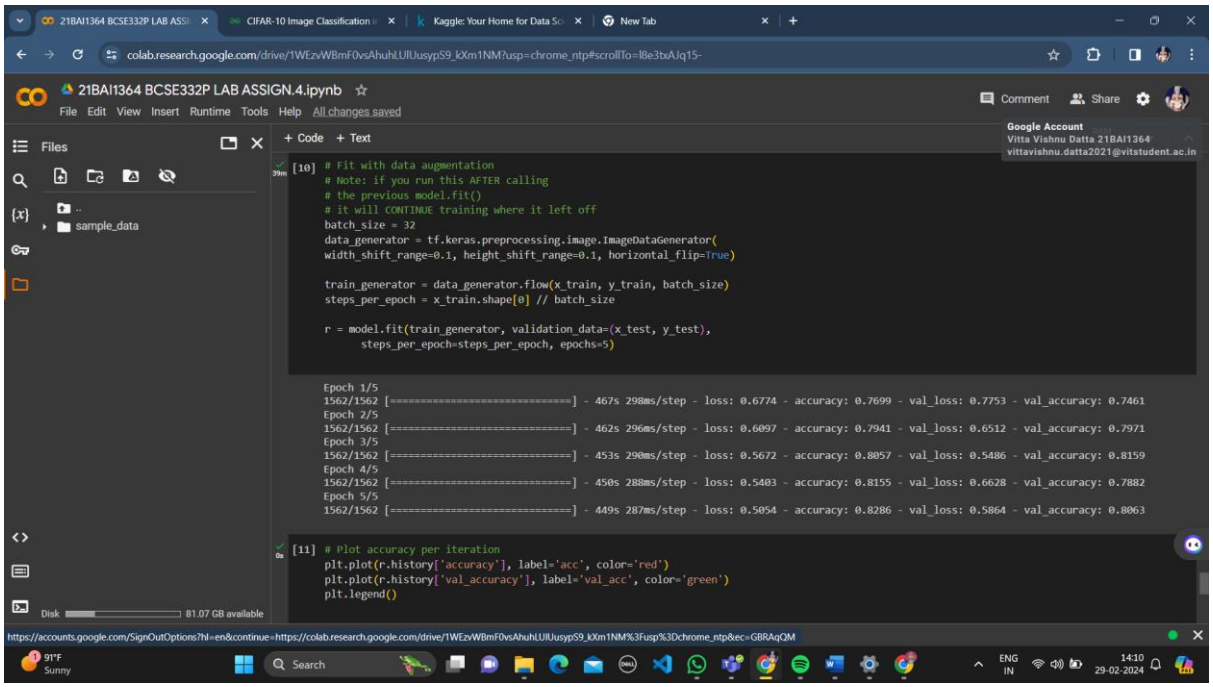
VIT®

Vellore Institute of Technology

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

CHENNAI



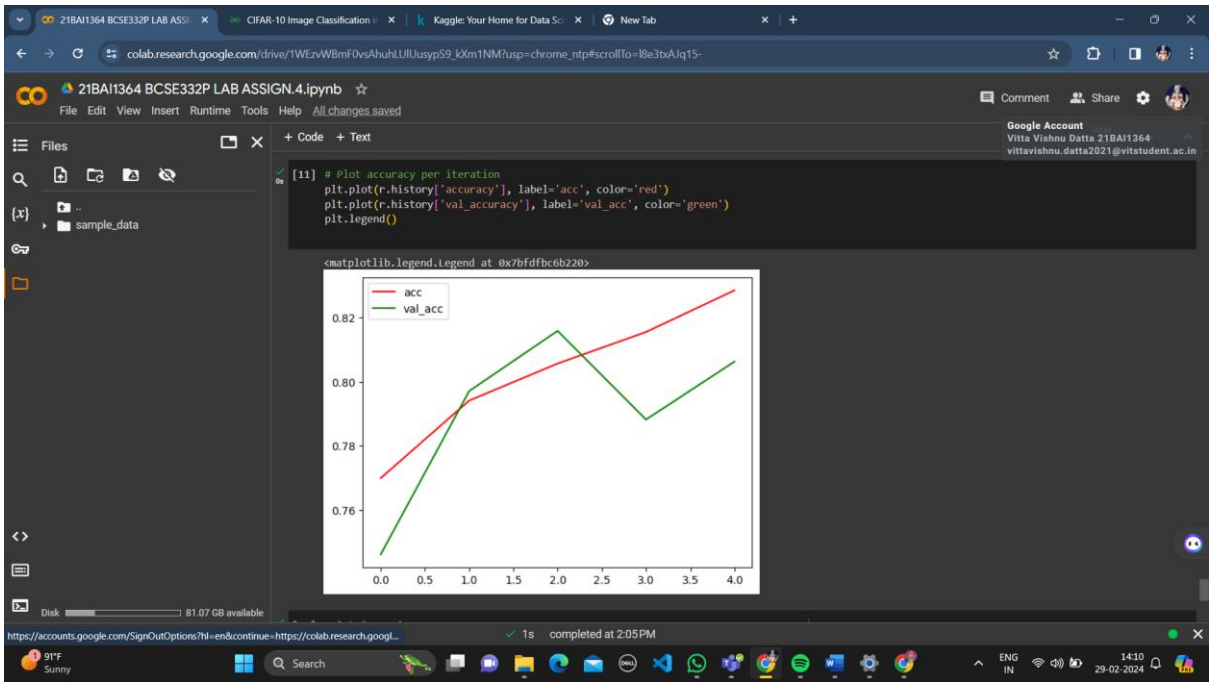


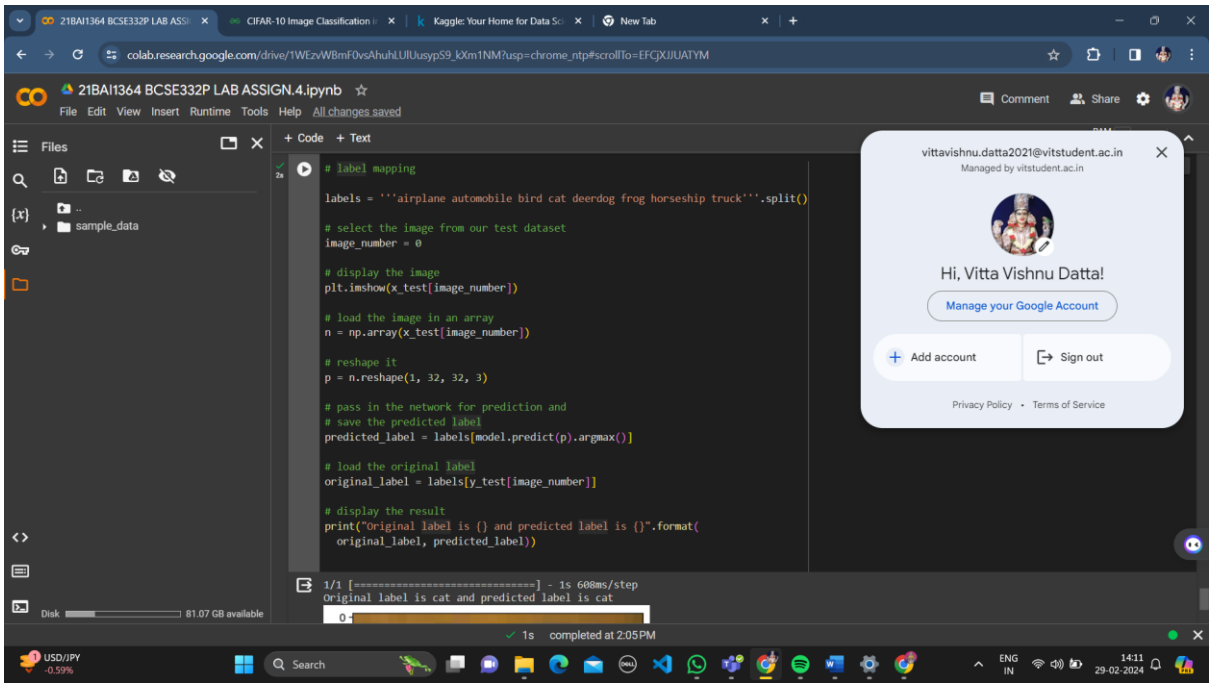
VIT®

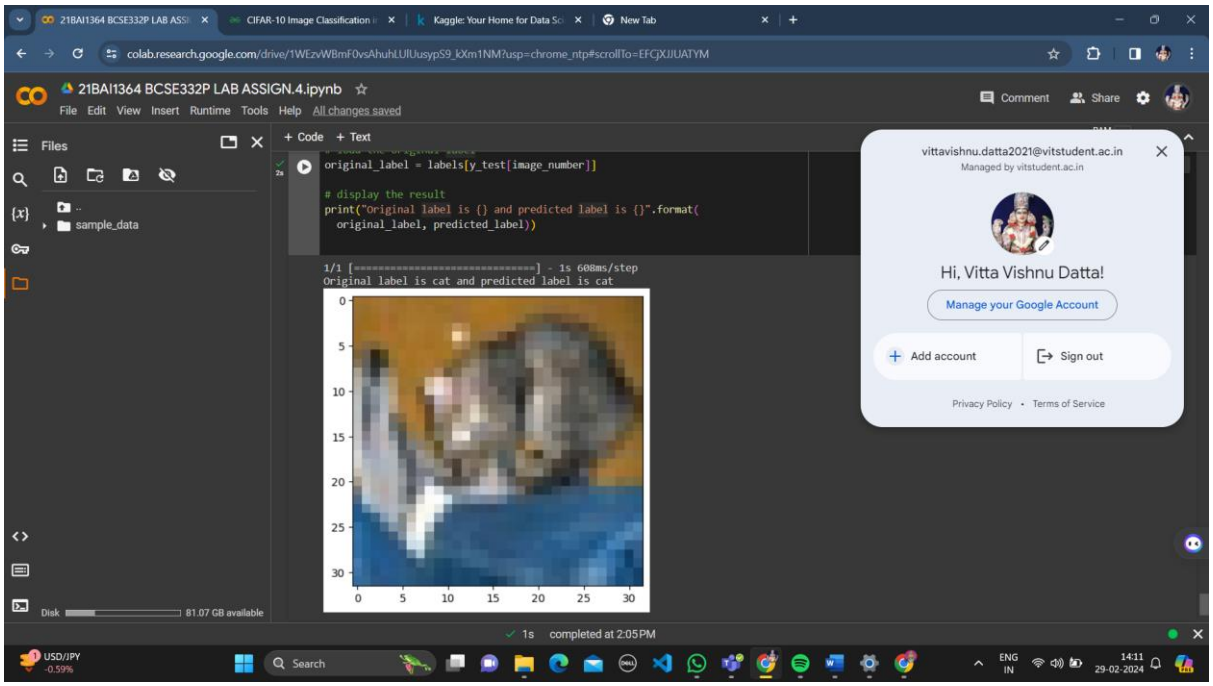
Vellore Institute of Technology

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

CHENNAI







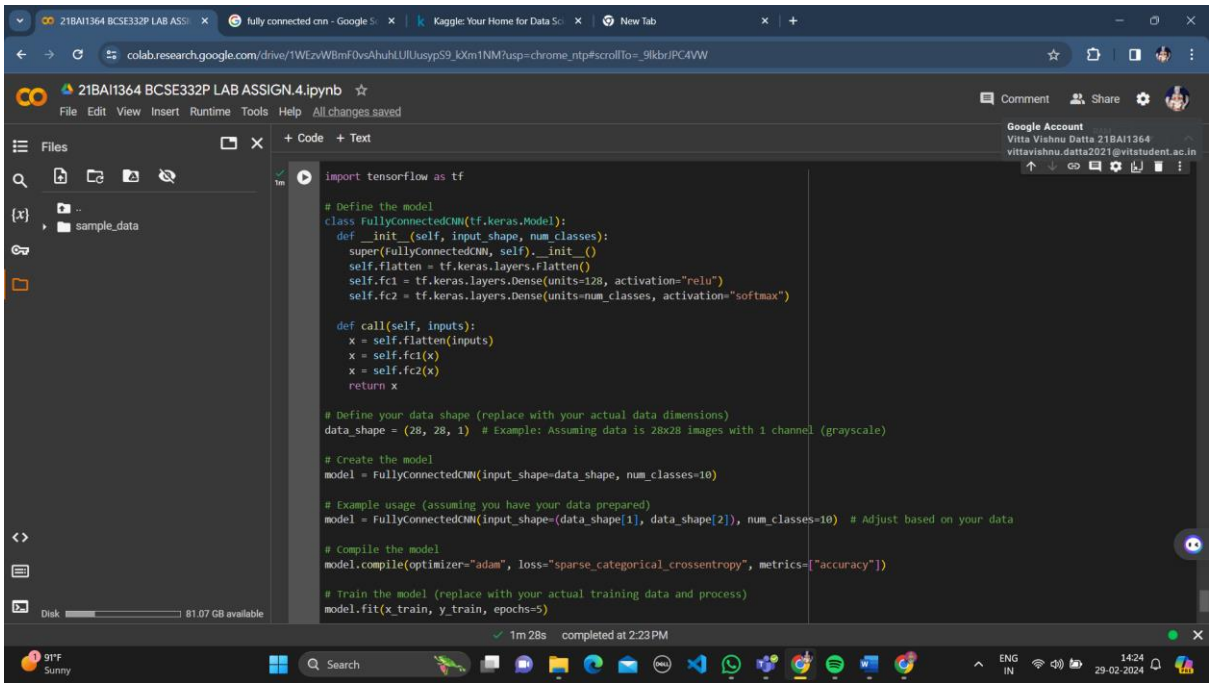
5. Design a new fully-connected CNN model.

CODE:

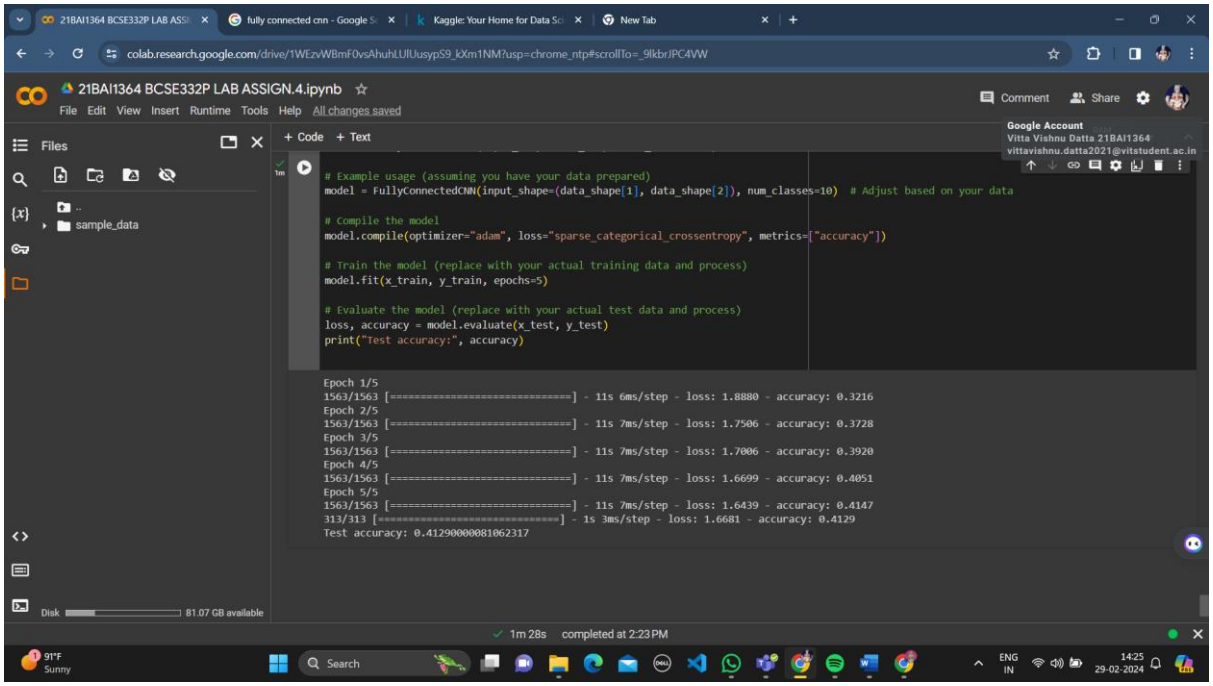
```
import tensorflow as tf

# Define the model
class FullyConnectedCNN(tf.keras.Model):
  def __init__(self, input_shape, num_classes):
    super(FullyConnectedCNN, self).__init__()
    self.flatten = tf.keras.layers.Flatten()
    self.fc1 = tf.keras.layers.Dense(units=128, activation="relu")
    self.fc2 = tf.keras.layers.Dense(units=num_classes,
activation="softmax")

  def call(self, inputs):
    x = self.flatten(inputs)
    x = self.fc1(x)
    x = self.fc2(x)
    return x
```

```
# Define your data shape (replace with your actual data dimensions)
data_shape = (28, 28, 1)  # Example: Assuming data is 28x28 images with
1 channel (grayscale)

# Create the model
model = FullyConnectedCNN(input_shape=data_shape, num_classes=10)

# Example usage (assuming you have your data prepared)
model = FullyConnectedCNN(input_shape=(data_shape[1], data_shape[2]),
num_classes=10)  # Adjust based on your data

# Compile the model
model.compile(optimizer="adam", loss="sparse_categorical_crossentropy",
metrics=["accuracy"])

# Train the model (replace with your actual training data and process)
model.fit(x_train, y_train, epochs=5)

# Evaluate the model (replace with your actual test data and process)
loss, accuracy = model.evaluate(x_test, y_test)
print("Test accuracy:", accuracy)
```

SCREENSHOTS OF THE CODE SNIPPET:



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



COMPARISON OF OUTPUT OF TWO CODES:

CODE	Test Loss	Test Accuracy	CNN Error
Code 1	0.05	99.95%	0.95%
Code 2	1.66	41.10%	2.60%

- Code 1 refers to the CNN model built by the cifar10 dataset.
- Code 2 refers to the CNN model which is a fully-connected.

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.