RAMNIRANJAN JHUNJHUNWALA COLLEGE
GHATKOPAR (W), MUMBAI - 400 086

DEPARTMENT OF
INFORMATION TECHNOLOGY

2021 - 2022

M.Sc.(I.T.) Part II (Sem - 3)

Subject : Deep Learning

Name : Sneha Ramchandra Pawar. Roll No. : 17

Hindi Vidya Prachar Samiti's

RAMNIRANJAN JHUNJHUNWALA COLLEGE

(AUTONOMOUS)

Opposite Ghatkopar Railway Station, Ghatkopar West, Mumbai-400086

CERTIFICATE

This is to certify that Miss. **Sneha Ramchandra Pawar** with Roll No. **17** has successfully completed the necessary course of experiments in the subject of Deep **Learning** during the academic year **2021 – 2022** complying with the requirements of **RAMNIRANJAN JHUNJHUNWALA COLLEGE OF ARTS, SCIENCE AND COMMERCE**, for the course of **M.Sc. (IT)** semester -I.

_____ _____

Internal Examiner **External Examiner**

_____ _____

Head Of Department **College Seal**

Index

Practical No :- 1

Aim-:Matrix Multiplication, Eigen Vectors, EigenValue Computation using TensorFlow

Matrix Multiplication-:

In mathematics, particularly in linear algebra, **matrix multiplication** is a binary operation that produces a matrix from two matrices. For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix.

Eigen Vectors-:

Eigenvectors are the vectors (non-zero) that do not change the direction when any linear transformation is applied. It changes by only a scalar factor. In a brief, we can say, if A is a linear transformation from a vector space V and **x** is a vector in V, which is not a zero vector, then v is an eigenvector of A if A(X) is a scalar multiple of **x**.

An **Eigenspace** of vector **x** consists of a set of all eigenvectors with the equivalent eigenvalue collectively with the zero vector. Though, the zero vector is not an eigenvector.
Let us say A is an "n × n" matrix and λ is an eigenvalue of matrix A, then **x**, a non-zero vector, is called as eigenvector if it satisfies the given below expression;
A**x** = λ**x**
x is an eigenvector of A corresponding to eigenvalue, λ.

EigenValue-:

Eigenvalues are the special set of scalars associated with the system of linear equations. It is mostly used in matrix equations. 'Eigen' is a German word that means 'proper' or 'characteristic'. Therefore, the term eigenvalue can be termed as characteristic value, characteristic root, proper values or latent roots as well. In simple words, the eigenvalue is a scalar that is used to transform the eigenvector. The basic equation is

$$Ax = \lambda x$$

The number or scalar value "λ" is an eigenvalue of A.
In Mathematics, an eigenvector corresponds to the real non zero eigenvalues which point in the direction stretched by the transformation whereas eigenvalue is considered as a factor by which it is stretched. In case, if the eigenvalue is negative, the direction of the transformation is negative.
For every real matrix, there is an eigenvalue. Sometimes it might be complex. The existence of the eigenvalue for the complex matrices is equal to the fundamental theorem of algebra.

What is TensorFlow?

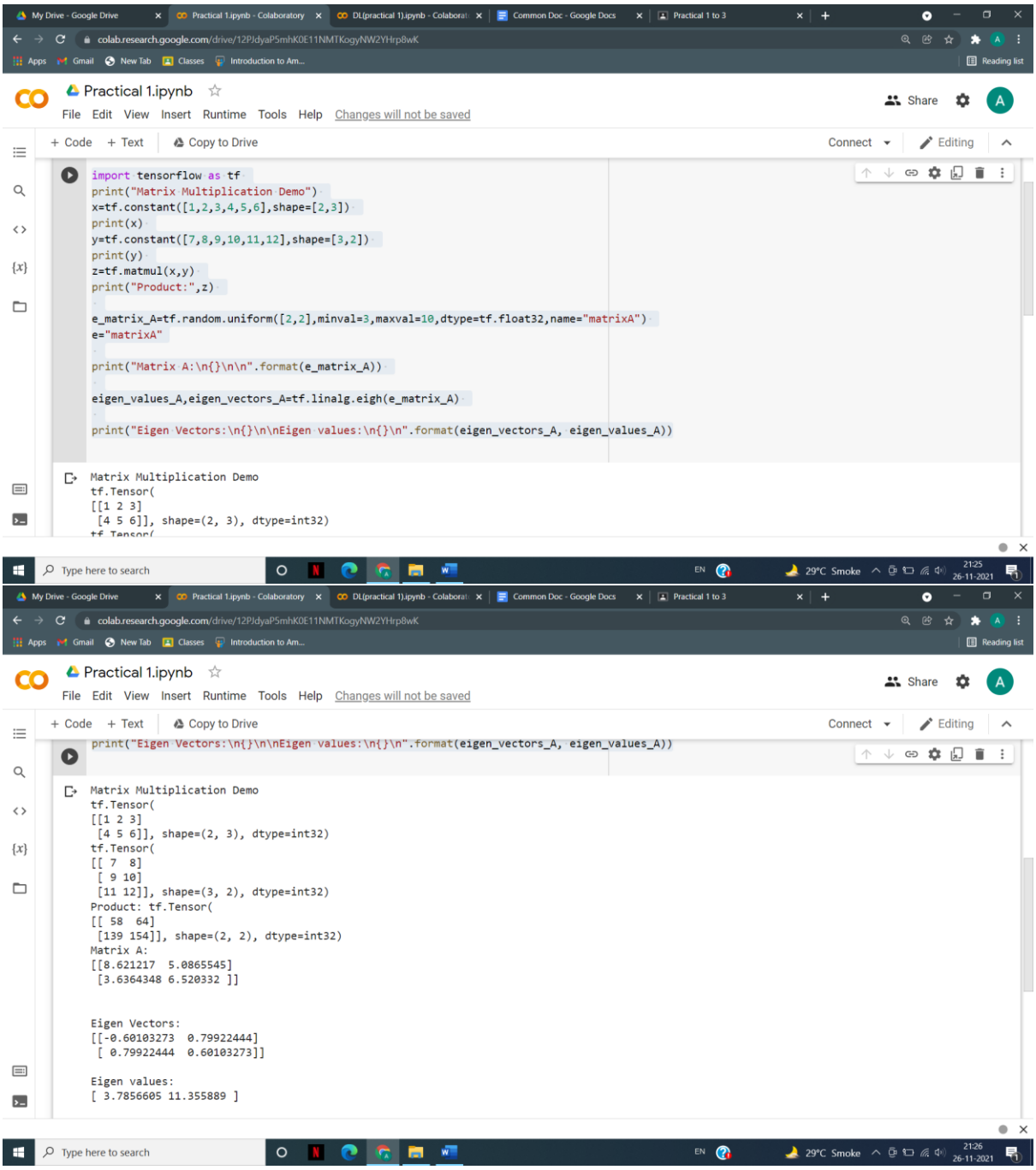
TensorFlow is an open-source end-to-end platform for creating Machine Learning applications. It is a symbolic math library that uses dataflow and differentiable programming to perform various tasks focused on training and inference of deep neural networks. It allows developers to create machine learning applications using various tools, libraries, and community resources. Currently, the most famous deep learning library in the world is Google's TensorFlow. Google product uses machine learning in all of its products to improve the search engine, translation, image captioning or recommendations.

https://colab.research.google.com/drive/12PJdyaP5mhK0E11NMTKogyNW2YHrp8wK

Code-:

```
import tensorflow as tf
print("Matrix Multiplication Demo")
x=tf.constant([1,2,3,4,5,6],shape=[2,3])
print(x)
y=tf.constant([7,8,9,10,11,12],shape=[3,2])
print(y)
z=tf.matmul(x,y)
print("Product:",z)

e_matrix_A=tf.random.uniform([2,2],minval=3,maxval=10,dtype=tf.float32,name="matrixA")

print("Matrix A:\n{}\n\n".format(e_matrix_A))

eigen_values_A,eigen_vectors_A=tf.linalg.eigh(e_matrix_A)


print("Eigen Vectors:\n{}\n\nEigen values:\n{}\n".format(eigen_vectors_A, eigen_values_A))
```

Output-:


My Drive - Google Drive
Practical 1.ipynb - Colaboratory
DL(practical 1).ipynb - Colaborat
Common Doc - Google Docs
Practical 1 to 3
colab.research.google.com/drive/12PJdyaP5mhK0E11NMTKogyNW2YHrp8wK
Practical 1.ipynb
File Edit View Insert Runtime Tools Help Changes will not be saved
Share
+ Code + Text Copy to Drive
Connect
Editing
import tensorflow as tf
print("Matrix Multiplication Demo")
x=tf.constant([1,2,3,4,5,6],shape=[2,3])
print(x)
y=tf.constant([7,8,9,10,11,12],shape=[3,2])
print(y)
z=tf.matmul(x,y)
print("Product:",z)
e_matrix_A=tf.random.uniform([2,2],minval=3,maxval=10,dtype=tf.float32,name="matrixA")
e="matrixA"
print("Matrix A:\n{}\n\n".format(e_matrix_A))
eigen_values_A,eigen_vectors_A=tf.linalg.eigh(e_matrix_A)
print("Eigen Vectors:\n{}\n\nEigen values:\n{}\n".format(eigen_vectors_A, eigen_values_A))
Matrix Multiplication Demo
tf.Tensor(
[[1 2 3]
[4 5 6]], shape=(2, 3), dtype=int32)
Type here to search
29°C Smoke
21:25
26-11-2021
Practical 1.ipynb
File Edit View Insert Runtime Tools Help Changes will not be saved
Share
+ Code + Text Copy to Drive
Connect
Editing
print("Eigen Vectors:\n{}\n\nEigen values:\n{}\n".format(eigen_vectors_A, eigen_values_A))
Matrix Multiplication Demo
tf.Tensor(
[[1 2 3]
[4 5 6]], shape=(2, 3), dtype=int32)
tf.Tensor(
[[7 8]
[9 10]
[11 12]], shape=(3, 2), dtype=int32)
Product: tf.Tensor(
[[58 64]
[139 154]], shape=(2, 2), dtype=int32)
Matrix A:
[[8.621217 5.0865545]
[3.6364348 6.520332]]
Eigen Vectors:
[[-0.60103273 0.79922444]
[0.79922444 0.60103273]]
Eigen values:
[3.7856605 11.355889]
Type here to search
29°C Smoke
21:26
26-11-2021

Practical No :- 2

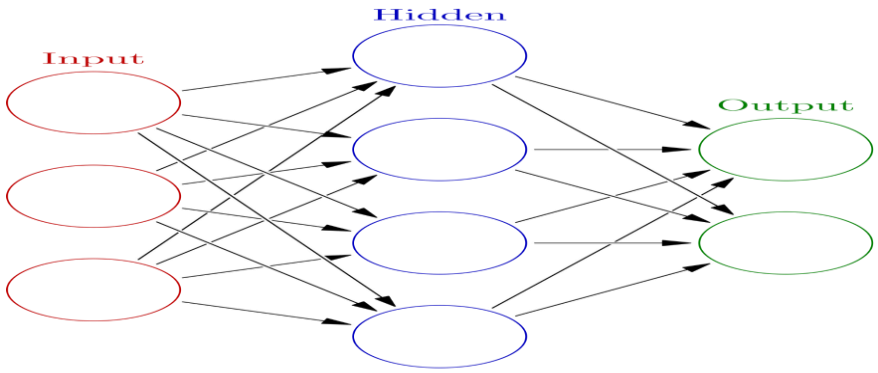
Aim-: Deep Forward Network for XOR

Deep feedforward networks, also often called feedforward neural networks, or multilayer perceptrons (MLPs), are the quintessential deep learning models.

These models are called feedforward because information flows through the function being evaluated from x, through the intermediate computations used to define f, and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself. When feedforward neural networks are extended to include feedback connections, they are called recurrent neural networks,

Feedforward networks are of extreme importance to machine learning practitioners. They form the basis of many important commercial applications. For example, the convolutional networks used for object recognition from photos are a specialized kind of feedforward network. Feedforward networks are a conceptual stepping stone on the path to recurrent networks, which power many natural language applications.

Feedforward neural networks are called networks because they are typically represented by composing together many different functions. The model is associated with a directed acyclic graph describing how the functions are composed together. For example, we might have three functions f(1), f(2), and f(3) connected in a chain, to form f(x) = f(3)(f(2)(f(1)(x))). These chain structures are the most commonly used structures of neural networks. In this case, f (1) is called the first layer of the network, f(2) is called the second layer, and so on.

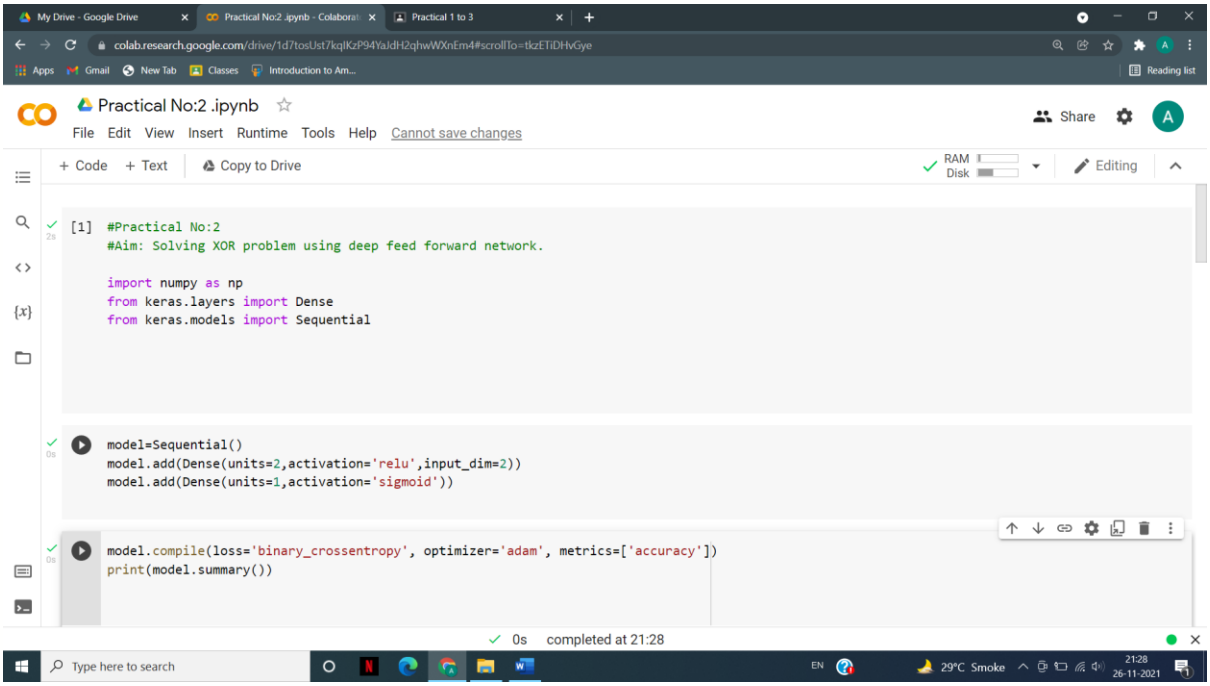


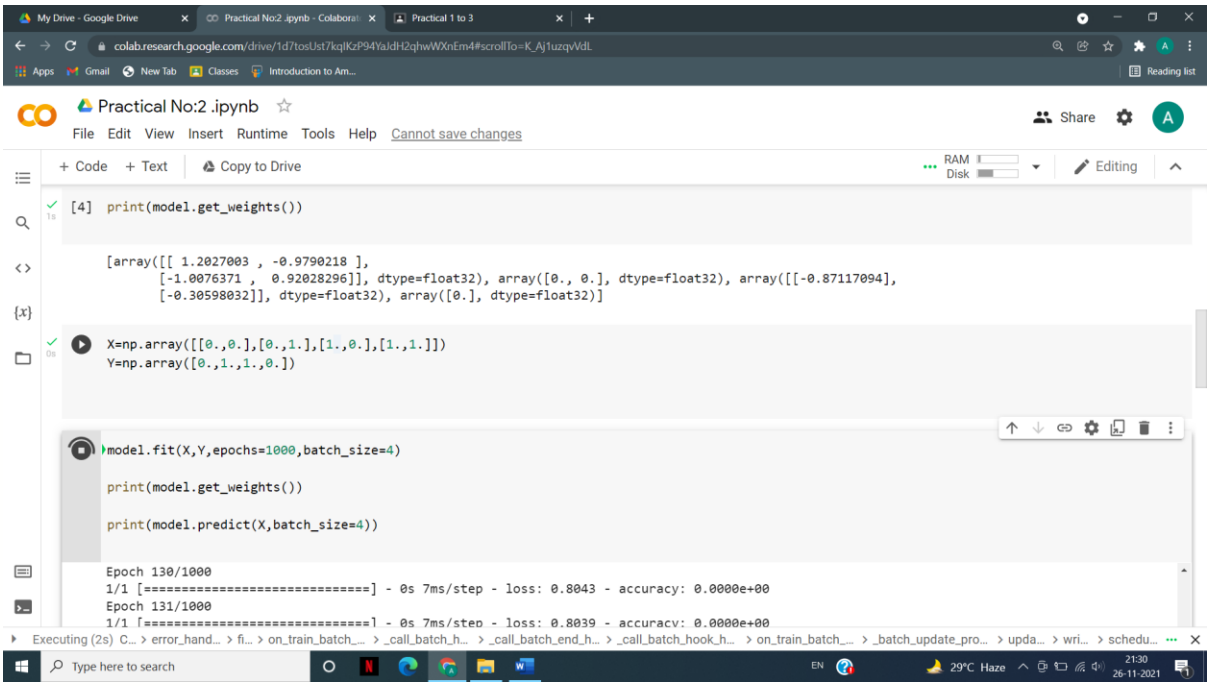
https://colab.research.google.com/drive/1d7tosUst7kqIKzP94YaJdH2qhwWXnEm4

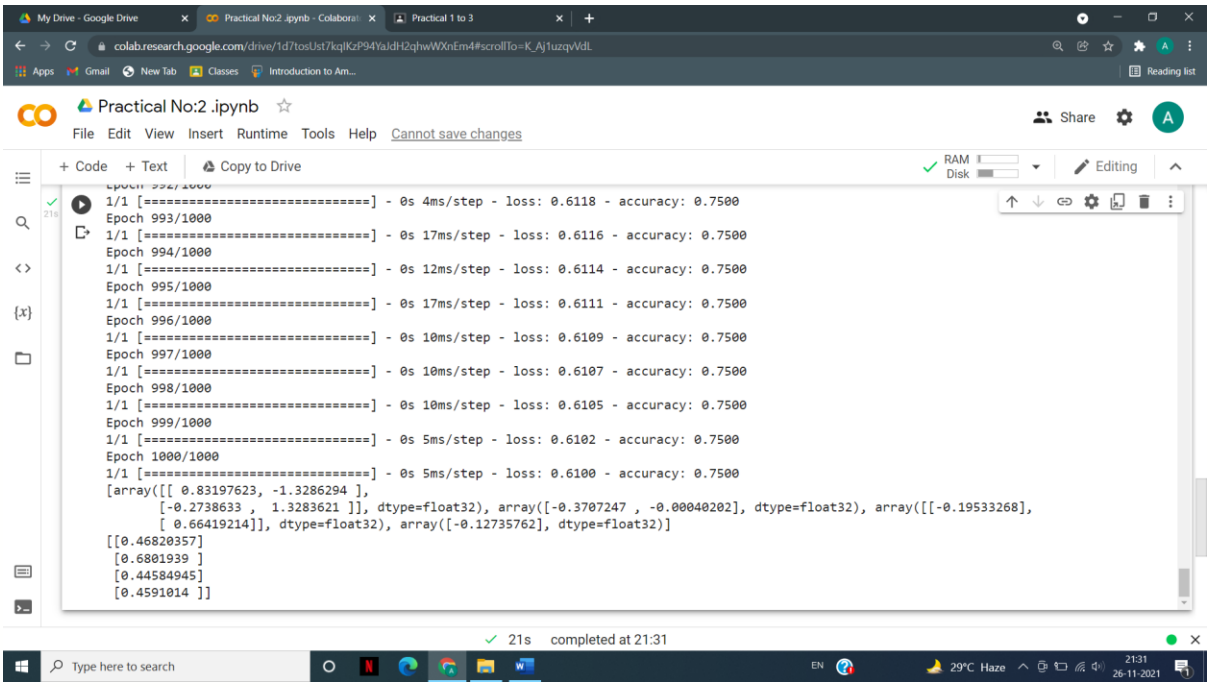
Code-:

```
import numpy as np
from keras.layers import Dense
from keras.models import Sequential
model=Sequential()
model.add(Dense(units=2,activation='relu',input_dim=2))
model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())
print(model.get_weights())
X=np.array([[0.,0.],[0.,1.],[1.,0.],[1.,1.]])
Y=np.array([0.,1.,1.,0.])
model.fit(X,Y,epochs=1000,batch_size=4)
print(model.get_weights())
print(model.predict(X,batch_size=4))
```

Output-:




Practical No:2 .ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
Editing
[4] print(model.get_weights())
[array([[1.2027003 , -0.9790218],
[-1.0076371 , 0.92028296]], dtype=float32), array([0., 0.], dtype=float32), array([[-0.87117094],
[-0.30598032]], dtype=float32), array([0.], dtype=float32)]
X=np.array([[0.,0.],[0.,1.],[1.,0.],[1.,1.]])
Y=np.array([0.,1.,1.,0.])
model.fit(X,Y,epochs=1000,batch_size=4)
print(model.get_weights())
print(model.predict(X,batch_size=4))
Epoch 130/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.8043 - accuracy: 0.0000e+00
Epoch 131/1000
1/1 [==============================] - 0s 7ms/step - loss: 0.8039 - accuracy: 0.0000e+00


Practical No:2 .ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
Editing
1/1 [==============================] - 0s 4ms/step - loss: 0.6118 - accuracy: 0.7500
Epoch 993/1000
1/1 [==============================] - 0s 17ms/step - loss: 0.6116 - accuracy: 0.7500
Epoch 994/1000
1/1 [==============================] - 0s 12ms/step - loss: 0.6114 - accuracy: 0.7500
Epoch 995/1000
1/1 [==============================] - 0s 17ms/step - loss: 0.6111 - accuracy: 0.7500
Epoch 996/1000
1/1 [==============================] - 0s 10ms/step - loss: 0.6109 - accuracy: 0.7500
Epoch 997/1000
1/1 [==============================] - 0s 10ms/step - loss: 0.6107 - accuracy: 0.7500
Epoch 998/1000
1/1 [==============================] - 0s 10ms/step - loss: 0.6105 - accuracy: 0.7500
Epoch 999/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.6102 - accuracy: 0.7500
Epoch 1000/1000
1/1 [==============================] - 0s 5ms/step - loss: 0.6100 - accuracy: 0.7500
[array([[0.83197623, -1.3286294],
[-0.2738633 , 1.3283621]], dtype=float32), array([-0.3707247 , -0.00040202], dtype=float32), array([[-0.19533268],
[0.66419214]], dtype=float32), array([-0.12735762], dtype=float32)]
[[0.46820357]
[0.6801939]
[0.44584945]
[0.4591014]]
21s completed at 21:31

Practical No :- 3

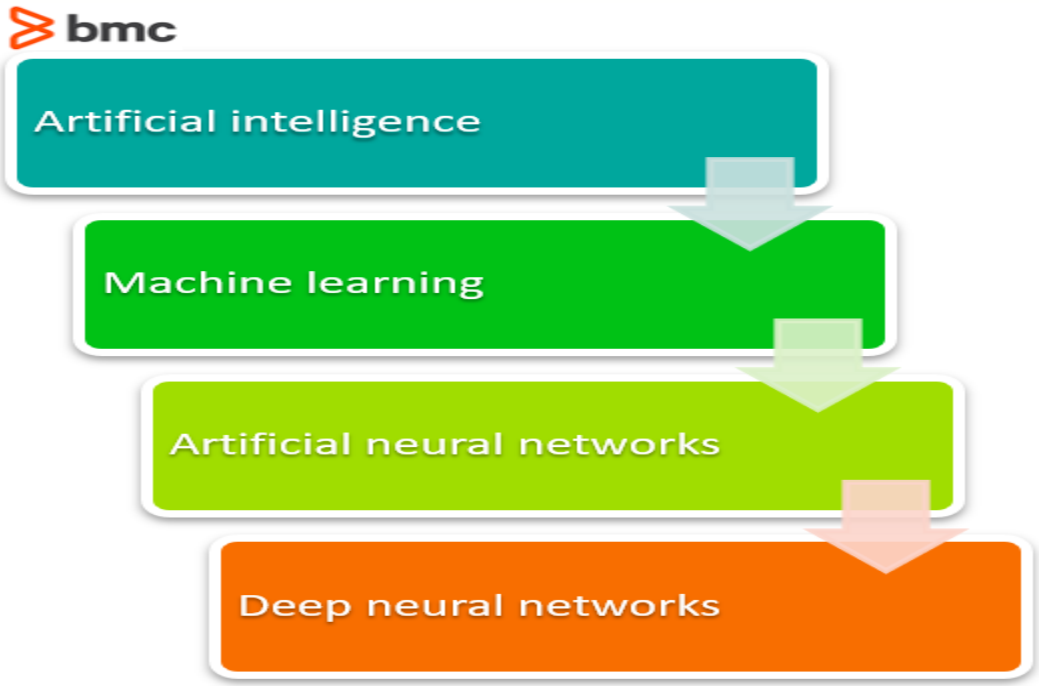
Aim-: 3a.Classification using DNN

https://colab.research.google.com/drive/19seC-t6aJplfEDSb2F7JVy2acVAUJeaH
https://colab.research.google.com/drive/1QJcZT9FU2RQvBGGzx1O6SUWQX1I2MJxn

What is a deep neural network?

At its simplest, a neural network with some level of complexity, usually at least two layers, qualifies as a deep neural network (DNN), or deep net for short. Deep nets process data in complex ways by employing sophisticated math modeling.

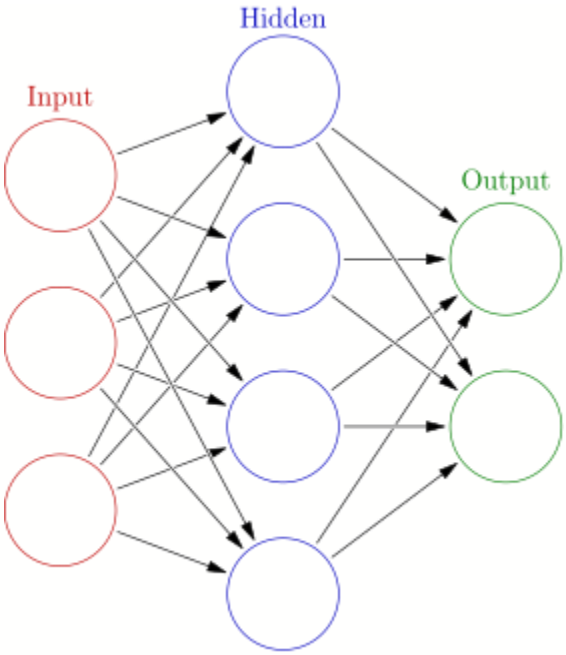
To truly understand deep neural networks, however, it's best to see it as an evolution. A few items had to be built before deep nets existed.



First, machine learning had to get developed. ML is a framework to automate (through algorithms) statistical models, like a linear regression model, to get better at making predictions. A model is a single model that makes predictions about something. Those predictions are made with some accuracy. A model that learns—machine learning—takes all its bad predictions and tweaks the weights inside the model to create a model that makes fewer mistakes.

The learning portion of creating models spawned the development of artificial neural networks. ANNs utilize the hidden layer as a place to store and evaluate how significant one of the inputs

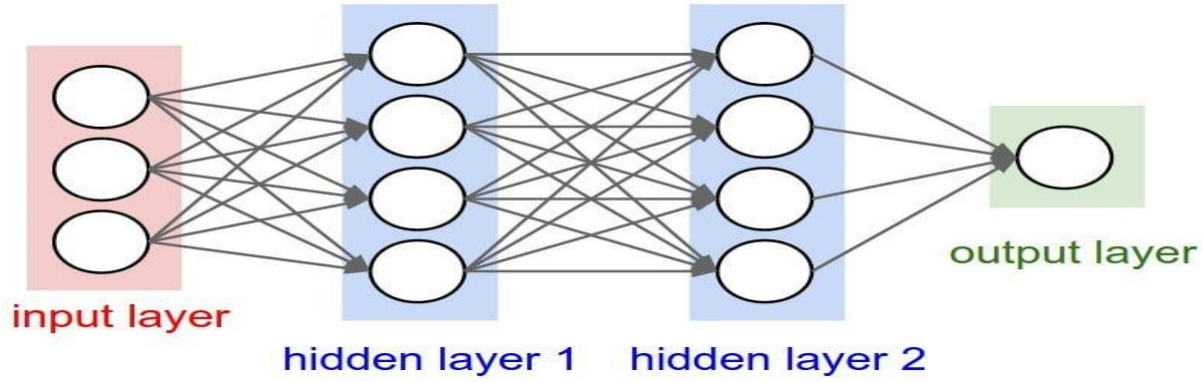
is to the output. The hidden layer stores information regarding the input's importance, and it also makes associations between the importance of combinations of inputs.



One hidden layer is considered an Artificial Neural Network (ANN)

Deep neural nets, then, capitalize on the ANN component. They say, if that works so well at improving a model—because each node in the hidden layer makes both associations and grades importance of the input to determining the output—then why not stack more and more of these upon each other and benefit even more from the hidden layer?

So, the deep net has multiple hidden layers. 'Deep' refers to a model's layers being multiple layers deep.



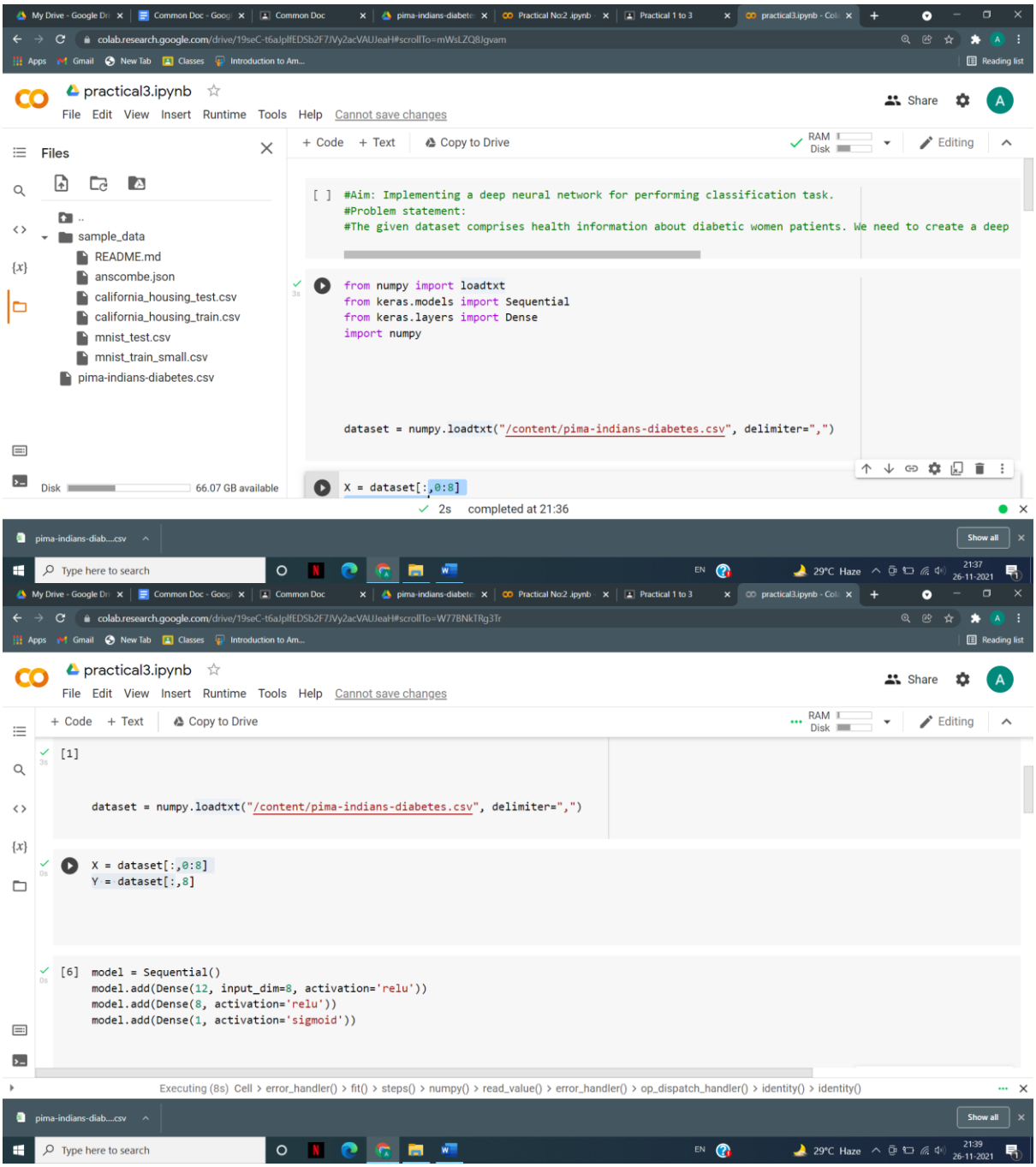
Two or more hidden layers comprise a Deep Neural Network

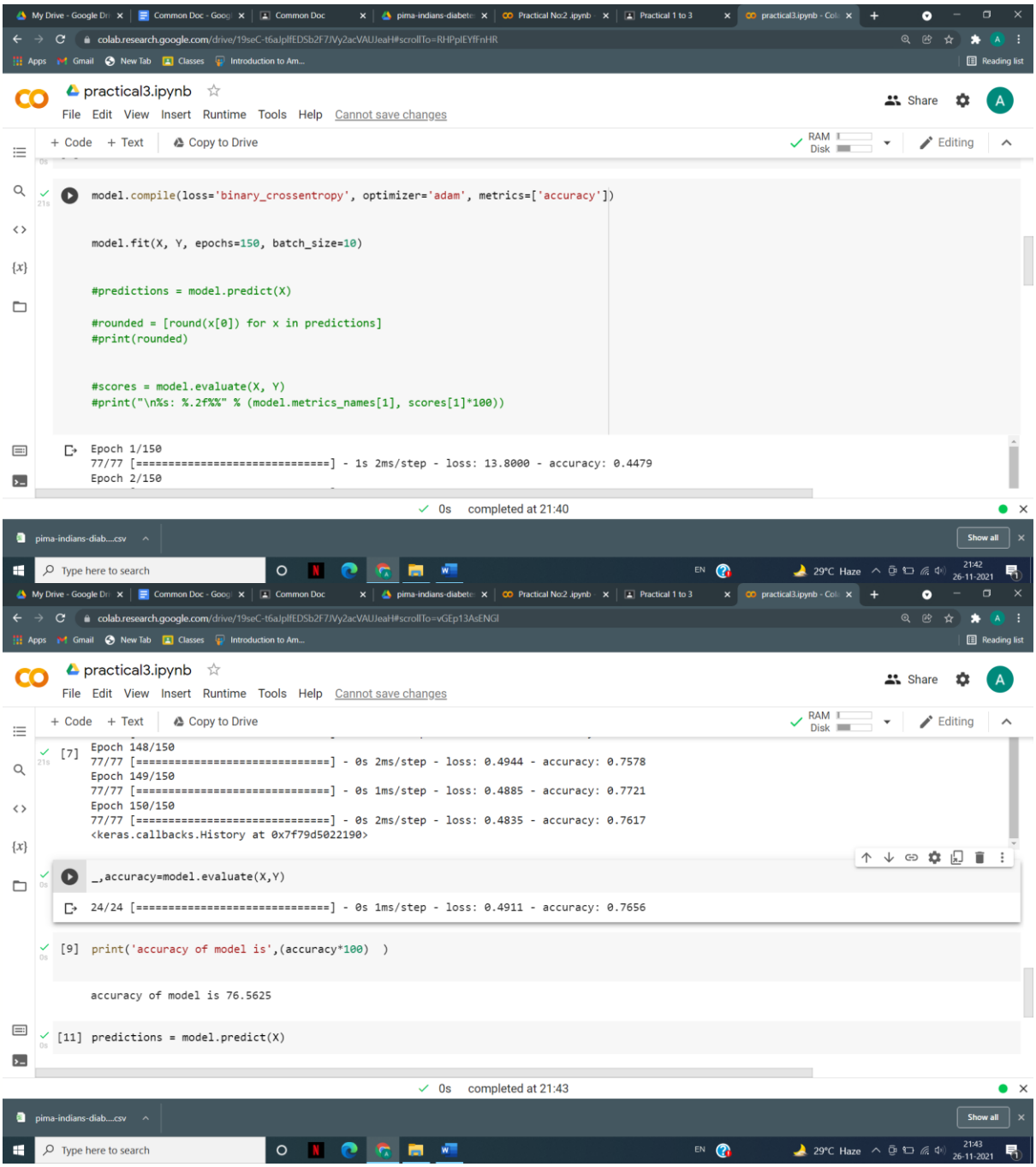
Code-:

```
from numpy import loadtxt
```

```
from keras.models import Sequential
from keras.layers import Dense
import numpy
dataset = numpy.loadtxt("/content/sample_data/pima-indians-diabetes - pima-indians-diabetes.csv",
delimiter=",")
dataset
X = dataset[:,0:8]
Y = dataset[:,8]
X


Y
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])




model.fit(X, Y, epochs=150, batch_size=10)

#predictions = model.predict(X)

#rounded = [round(x[0]) for x in predictions]
#print(rounded)


#scores = model.evaluate(X, Y)
#print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
_,accuracy=model.evaluate(X,Y)
print('accuracy of model is',(accuracy*100)  )
predictions = model.predict(X)
```

Output-:


practical3.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
Files
sample_data
README.md
anscombe.json
california_housing_test.csv
california_housing_train.csv
mnist_test.csv
mnist_train_small.csv
pima-indians-diabetes.csv
Disk 66.07 GB available
[ ] #Aim: Implementing a deep neural network for performing classification task.
#Problem statement:
#The given dataset comprises health information about diabetic women patients. We need to create a deep
from numpy import loadtxt
from keras.models import Sequential
from keras.layers import Dense
import numpy
dataset = numpy.loadtxt("/content/pima-indians-diabetes.csv", delimiter=",")
X = dataset[:,0:8]
2s completed at 21:36
practical3.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
[1]
dataset = numpy.loadtxt("/content/pima-indians-diabetes.csv", delimiter=",")
X = dataset[:,0:8]
Y = dataset[:,8]
[6] model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
Executing (8s) Cell > error_handler() > fit() > steps() > numpy() > read_value() > error_handler() > op_dispatch_handler() > identity() > identity()

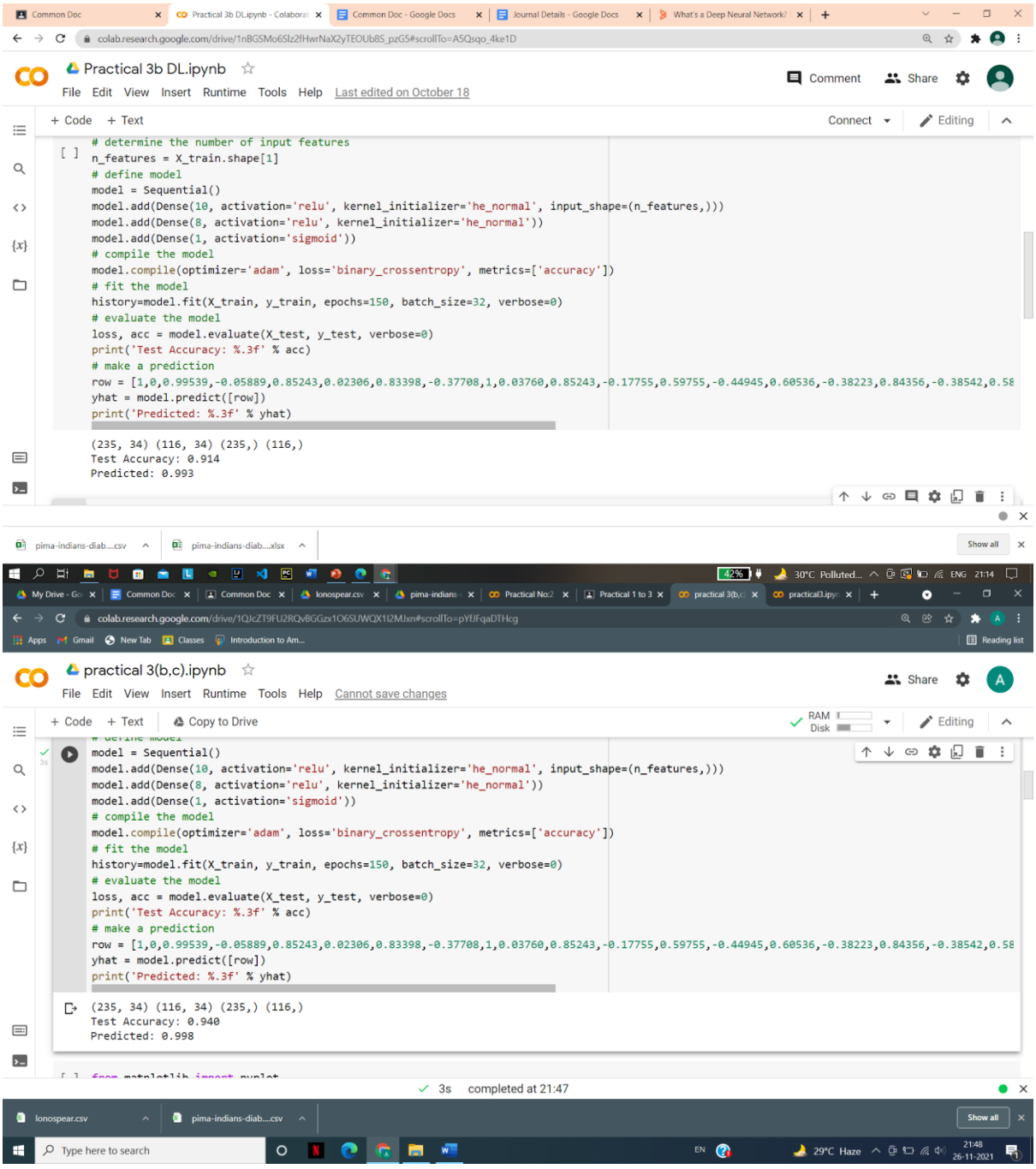


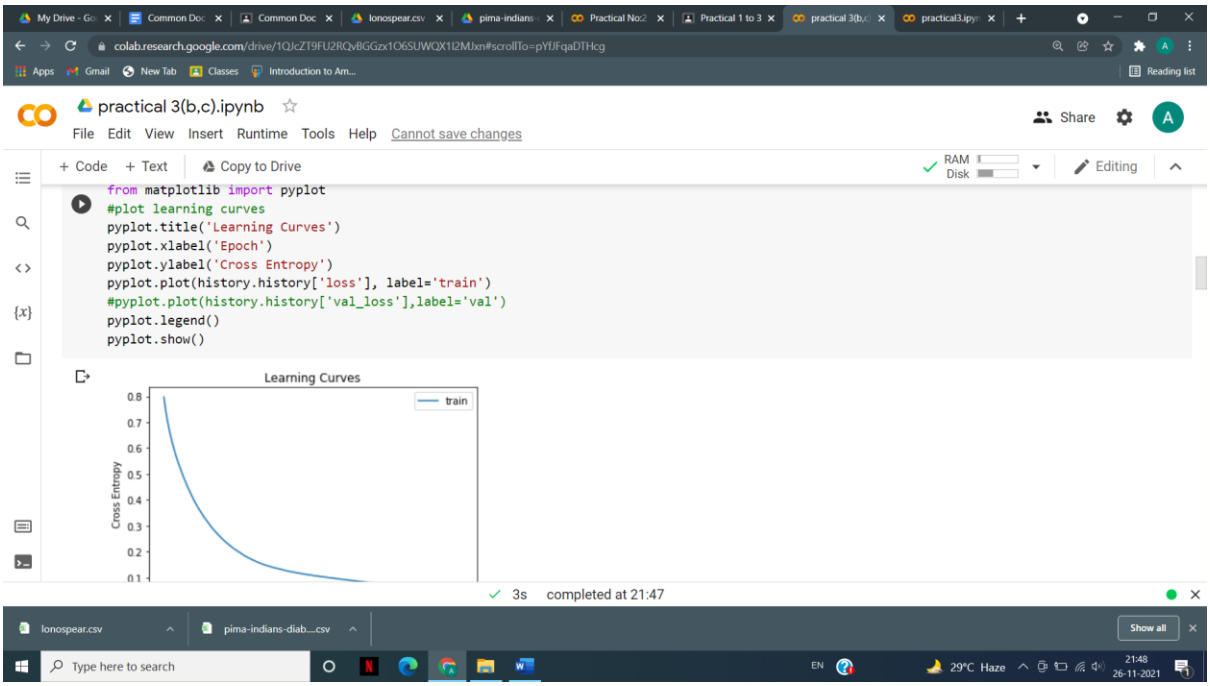
Aim-:3b.Binary Classification using MLP

Code-:

```
# mlp for binary classification
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
# load the dataset
path ='/content/sample_data/Ionospear - Ionospear.csv'
df = read_csv(path, header=None)
# split into input and output columns
X, y = df.values[:, :-1], df.values[:, -1]
# ensure all data are floating point values
X = X.astype('float32')
# encode strings to integer
y = LabelEncoder().fit_transform(y)
# split into train and test datasets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# determine the number of input features
n_features = X_train.shape[1]
# define model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))
model.add(Dense(8, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# fit the model
history=model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=0)
# evaluate the model
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print('Test Accuracy: %.3f' % acc)
# make a prediction
row = [1,0,0.99539,-0.05889,0.85243,0.02306,0.83398,-0.37708,1,0.03760,0.85243,-0.17755,0.59755,-
0.44945,0.60536,-0.38223,0.84356,-0.38542,0.58212,-0.32192,0.56971,-0.29674,0.36946,-
0.47357,0.56811,-0.51171,0.41078,-0.46168,0.21266,-0.34090,0.42267,-0.54487,0.18641,-0.45300]
yhat = model.predict([row])
print('Predicted: %.3f' % yhat)

from matplotlib import pyplot
#plot learning curves
pyplot.title('Learning Curves')
pyplot.xlabel('Epoch')
pyplot.ylabel('Cross Entropy')
pyplot.plot(history.history['loss'], label='train')
#pyplot.plot(history.history['val_loss'],label='val')
pyplot.legend()
pyplot.show()
```

Output-:

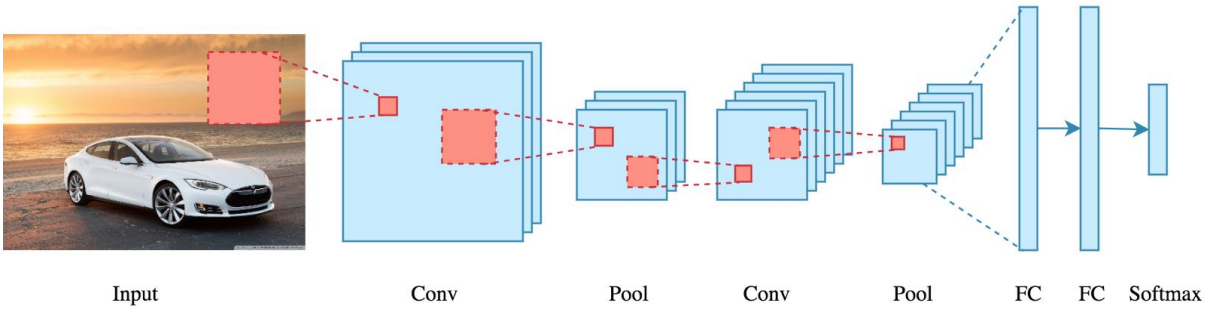

Practical 3b DL.ipynb
File Edit View Insert Runtime Tools Help Last edited on October 18
Comment Share
+ Code + Text
Connect Editing
# determine the number of input features
n_features = X_train.shape[1]
# define model
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))
model.add(Dense(8, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# fit the model
history=model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=0)
# evaluate the model
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print('Test Accuracy: %.3f' % acc)
# make a prediction
row = [1,0,0.99539,-0.05889,0.85243,0.02306,0.83398,-0.37708,1,0.03760,0.85243,-0.17755,0.59755,-0.44945,0.60536,-0.38223,0.84356,-0.38542,0.58
yhat = model.predict([row])
print('Predicted: %.3f' % yhat)
(235, 34) (116, 34) (235,) (116,)
Test Accuracy: 0.914
Predicted: 0.993
practical 3(b,c).ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal', input_shape=(n_features,)))
model.add(Dense(8, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='sigmoid'))
# compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# fit the model
history=model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=0)
# evaluate the model
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print('Test Accuracy: %.3f' % acc)
# make a prediction
row = [1,0,0.99539,-0.05889,0.85243,0.02306,0.83398,-0.37708,1,0.03760,0.85243,-0.17755,0.59755,-0.44945,0.60536,-0.38223,0.84356,-0.38542,0.58
yhat = model.predict([row])
print('Predicted: %.3f' % yhat)
(235, 34) (116, 34) (235,) (116,)
Test Accuracy: 0.940
Predicted: 0.998
3s completed at 21:47



Aim-:3c.Convolutional Neural Network

What exactly is a CNN?

In deep learning, a convolutional neural network (CNN/ConvNet) is a class of deep neural networks, most commonly applied to analyze visual imagery. Now when we think of a neural network we think about matrix multiplications but that is not the case with ConvNet. It uses a special technique called Convolution. Now in mathematics convolution is a mathematical operation on two functions that produces a third function that expresses how the shape of one is modified by the other.



But we don't really need to go behind the mathematics part to understand what a CNN is or

how it works.

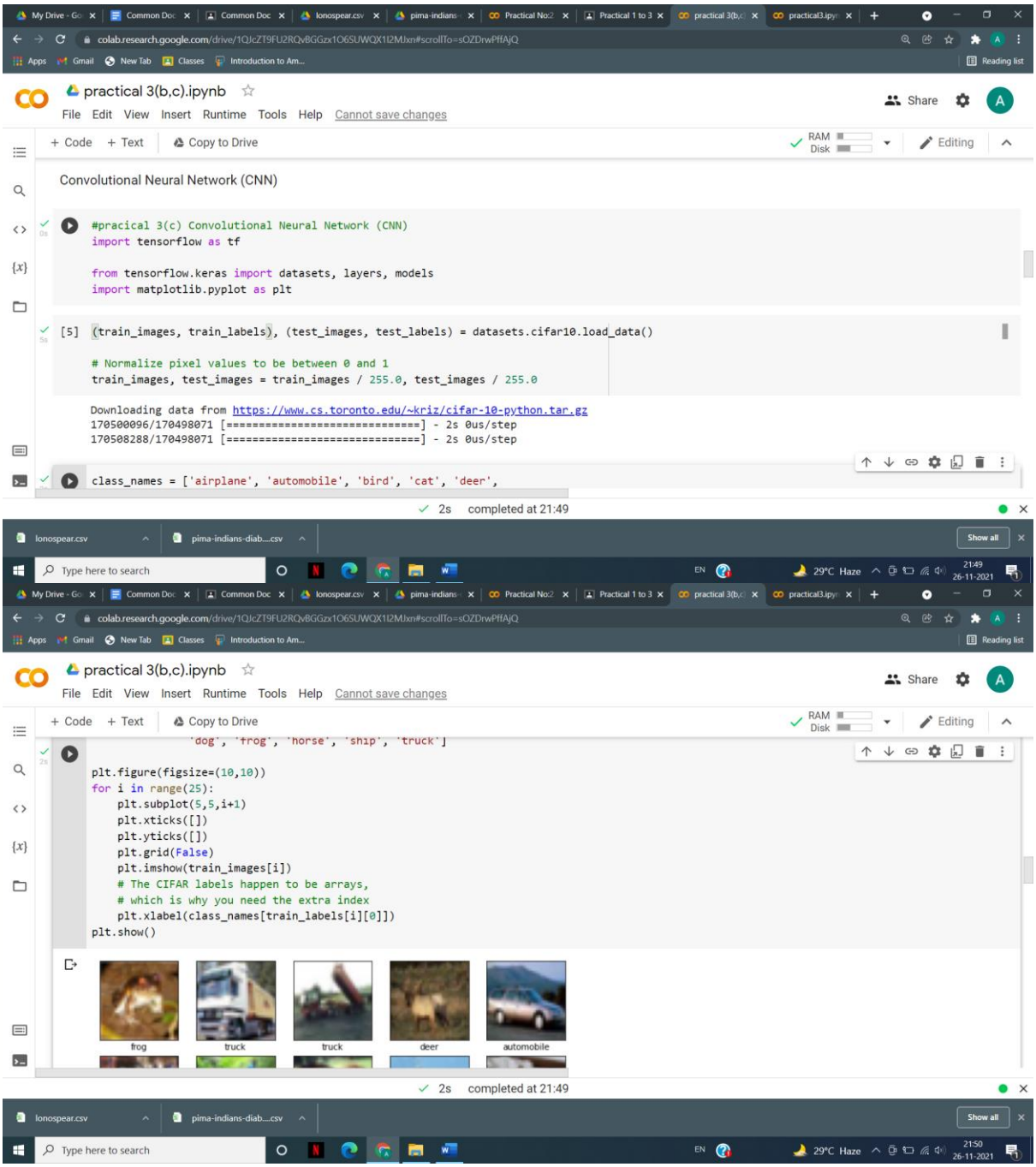
Bottom line is that the role of the ConvNet is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction.

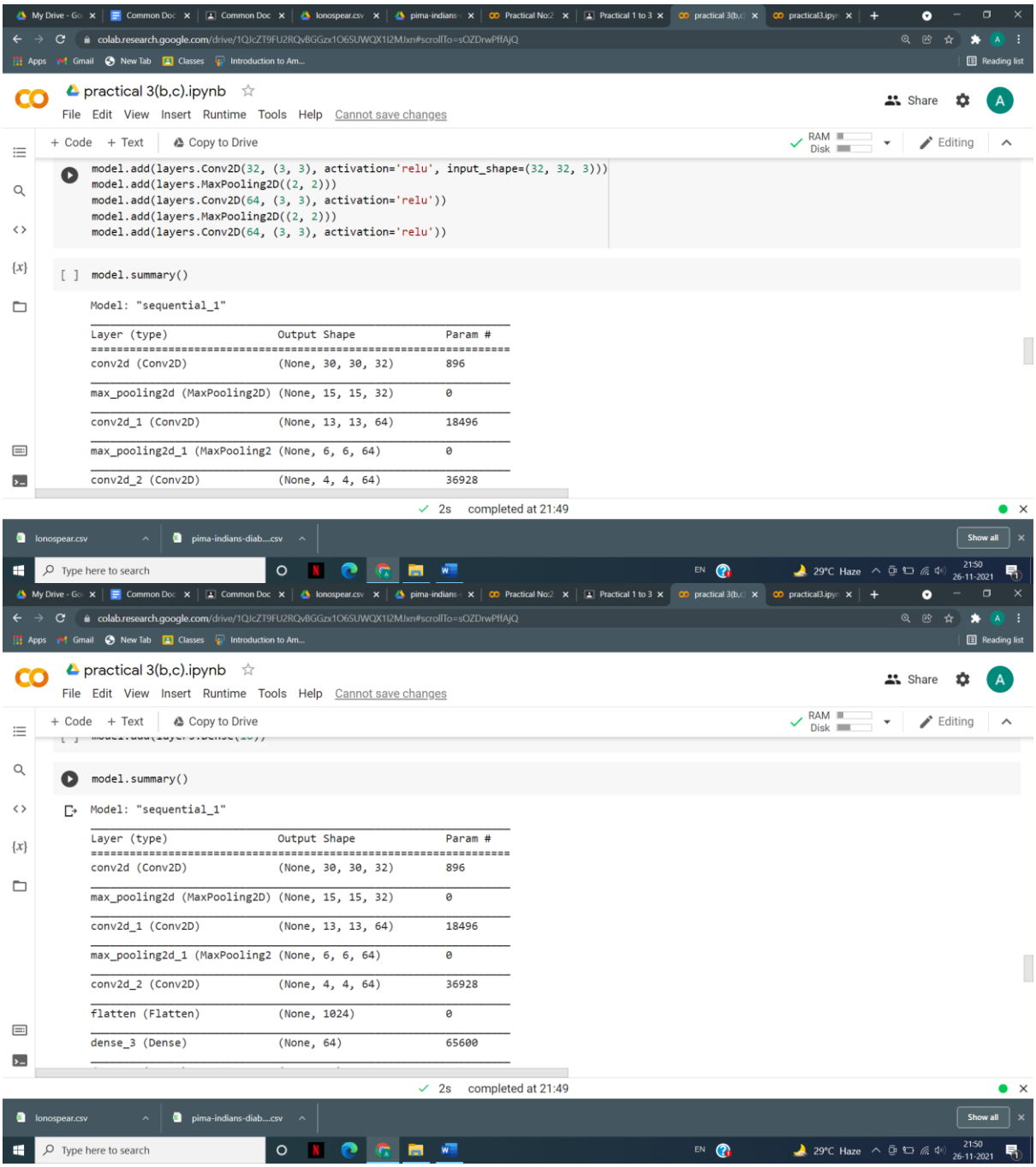
Code-:

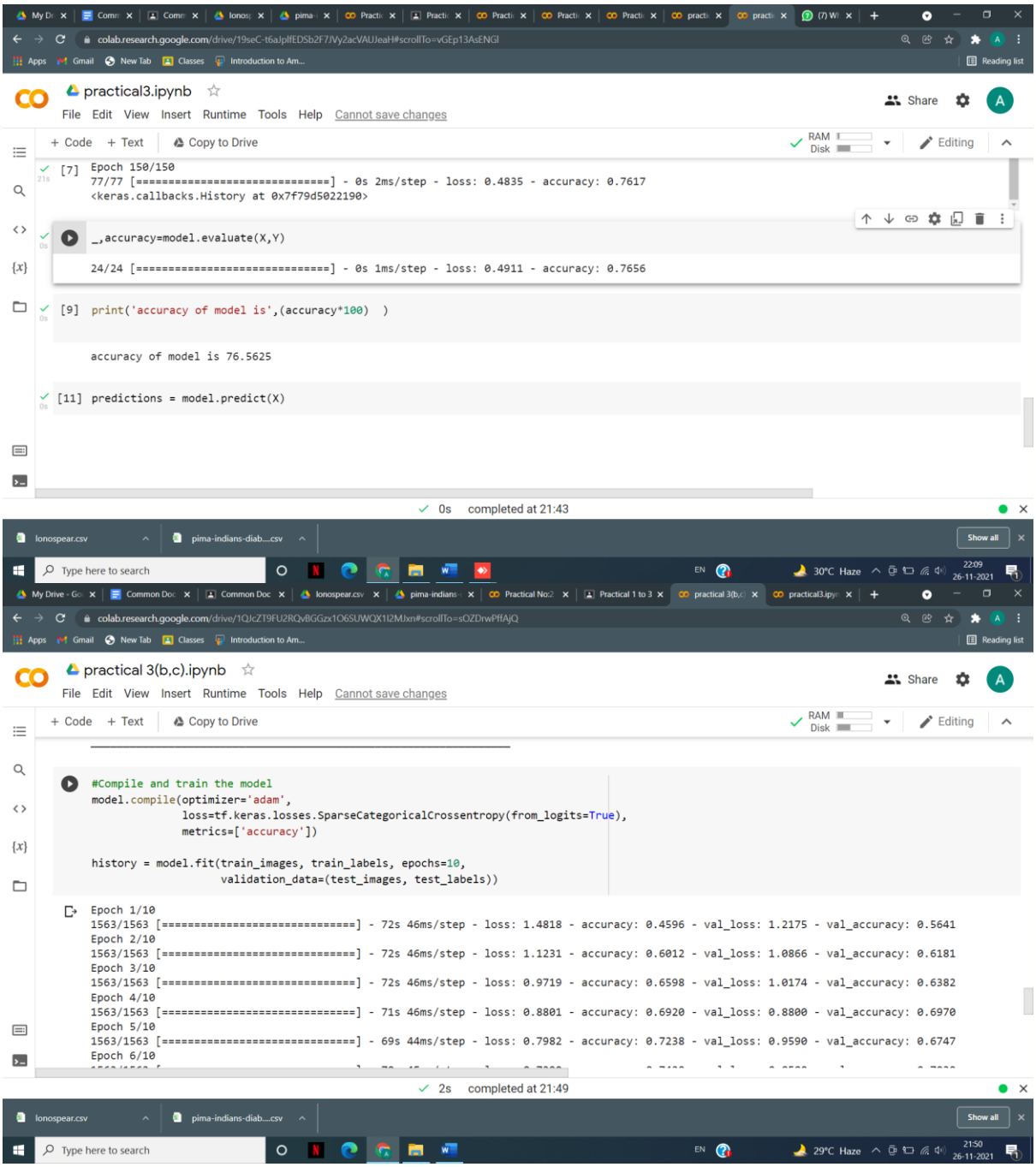
```
#Import TensorFlow
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
#Verify the data
#To verify that the dataset looks correct, let's plot the first 25 images from the training set and display the class name below each image:
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
   plt.subplot(5,5,i+1)

plt.xticks([])
   plt.yticks([])
   plt.grid(False)
   plt.imshow(train_images[i])
   # The CIFAR labels happen to be arrays,
   # which is why you need the extra index
   plt.xlabel(class_names[train_labels[i][0]])
plt.show()
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.summary()
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))
model.summary()
model.compile(optimizer='adam',
          loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
          metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
```

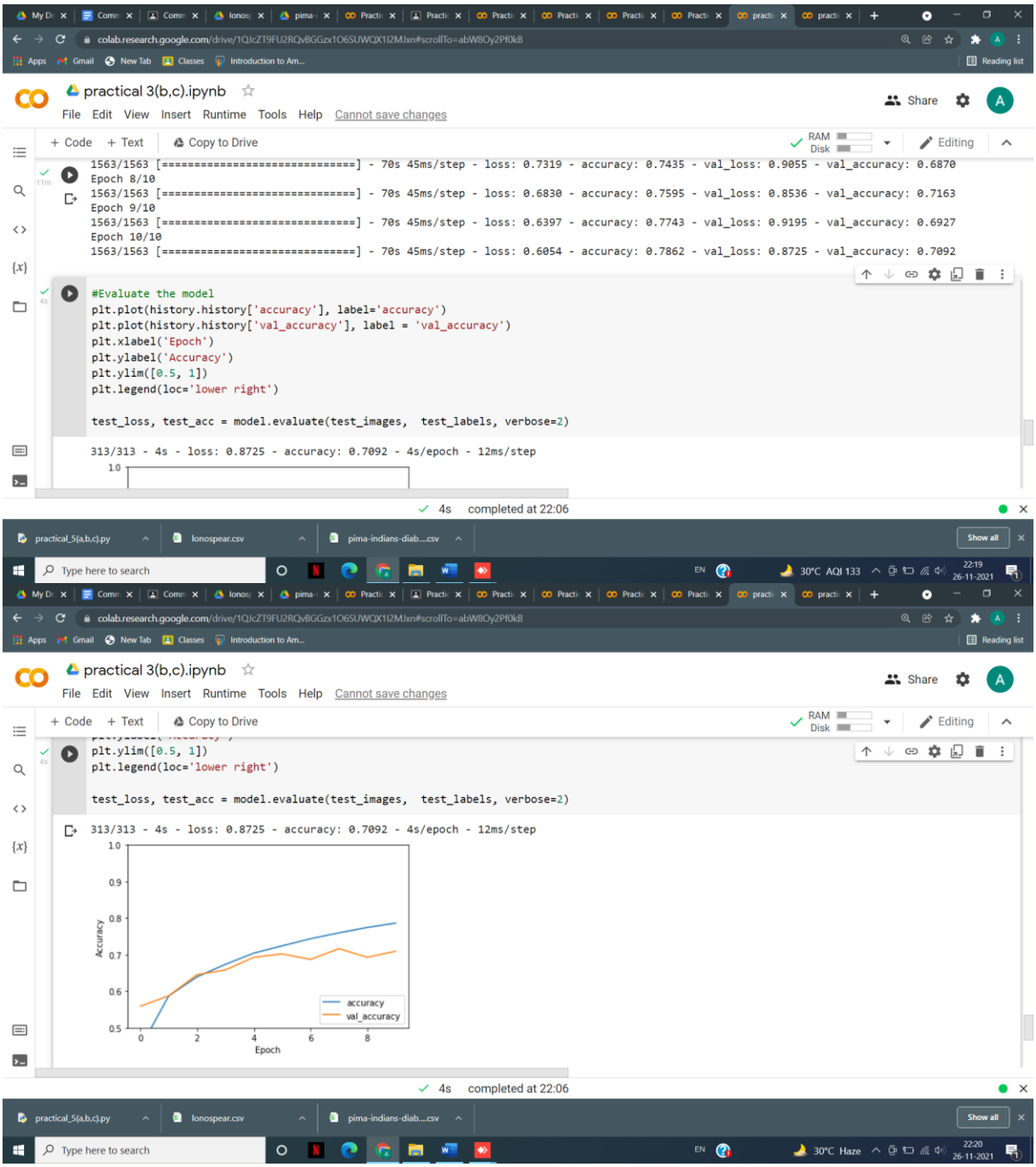
```
                validation_data=(test_images, test_labels))
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
print(test_acc)
```

Output-:


practical 3(b,c).ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
Convolutional Neural Network (CNN)
#pracical 3(c) Convolutional Neural Network (CNN)
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
[5] (train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 2s 0us/step
170508288/170498071 [==============================] - 2s 0us/step
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
2s completed at 21:49
'dog', 'frog', 'horse', 'ship', 'truck']
plt.figure(figsize=(10,10))
for i in range(25):
plt.subplot(5,5,i+1)
plt.xticks([])
plt.yticks([])
plt.grid(False)
plt.imshow(train_images[i])
# The CIFAR labels happen to be arrays,
# which is why you need the extra index
plt.xlabel(class_names[train_labels[i][0]])
plt.show()
frog
truck
truck
deer
automobile
2s completed at 21:49


practical 3(b,c).ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
[ ] model.summary()
Model: "sequential_1"
Layer (type) Output Shape Param #
conv2d (Conv2D) (None, 30, 30, 32) 896
max_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0
conv2d_1 (Conv2D) (None, 13, 13, 64) 18496
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64) 0
conv2d_2 (Conv2D) (None, 4, 4, 64) 36928
2s completed at 21:49
practical 3(b,c).ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
model.summary()
Model: "sequential_1"
Layer (type) Output Shape Param #
conv2d (Conv2D) (None, 30, 30, 32) 896
max_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0
conv2d_1 (Conv2D) (None, 13, 13, 64) 18496
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64) 0
conv2d_2 (Conv2D) (None, 4, 4, 64) 36928
flatten (Flatten) (None, 1024) 0
dense_3 (Dense) (None, 64) 65600
2s completed at 21:49


practical3.ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
[7] Epoch 150/150
77/77 [==============================] - 0s 2ms/step - loss: 0.4835 - accuracy: 0.7617
<keras.callbacks.History at 0x7f79d5022190>
_,accuracy=model.evaluate(X,Y)
24/24 [==============================] - 0s 1ms/step - loss: 0.4911 - accuracy: 0.7656
[9] print('accuracy of model is',(accuracy*100))
accuracy of model is 76.5625
[11] predictions = model.predict(X)
0s completed at 21:43
practical 3(b,c).ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
#Compile and train the model
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
history = model.fit(train_images, train_labels, epochs=10,
validation_data=(test_images, test_labels))
Epoch 1/10
1563/1563 [==============================] - 72s 46ms/step - loss: 1.4818 - accuracy: 0.4596 - val_loss: 1.2175 - val_accuracy: 0.5641
Epoch 2/10
1563/1563 [==============================] - 72s 46ms/step - loss: 1.1231 - accuracy: 0.6012 - val_loss: 1.0866 - val_accuracy: 0.6181
Epoch 3/10
1563/1563 [==============================] - 72s 46ms/step - loss: 0.9719 - accuracy: 0.6598 - val_loss: 1.0174 - val_accuracy: 0.6382
Epoch 4/10
1563/1563 [==============================] - 71s 46ms/step - loss: 0.8801 - accuracy: 0.6920 - val_loss: 0.8800 - val_accuracy: 0.6970
Epoch 5/10
1563/1563 [==============================] - 69s 44ms/step - loss: 0.7982 - accuracy: 0.7238 - val_loss: 0.9590 - val_accuracy: 0.6747
Epoch 6/10
2s completed at 21:49


practical 3(b,c).ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
1563/1563 [==========] - 70s 45ms/step - loss: 0.7319 - accuracy: 0.7435 - val_loss: 0.9055 - val_accuracy: 0.6870
Epoch 8/10
1563/1563 [==========] - 70s 45ms/step - loss: 0.6830 - accuracy: 0.7595 - val_loss: 0.8536 - val_accuracy: 0.7163
Epoch 9/10
1563/1563 [==========] - 70s 45ms/step - loss: 0.6397 - accuracy: 0.7743 - val_loss: 0.9195 - val_accuracy: 0.6927
Epoch 10/10
1563/1563 [==========] - 70s 45ms/step - loss: 0.6054 - accuracy: 0.7862 - val_loss: 0.8725 - val_accuracy: 0.7092
#Evaluate the model
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
313/313 - 4s - loss: 0.8725 - accuracy: 0.7092 - 4s/epoch - 12ms/step
4s completed at 22:06

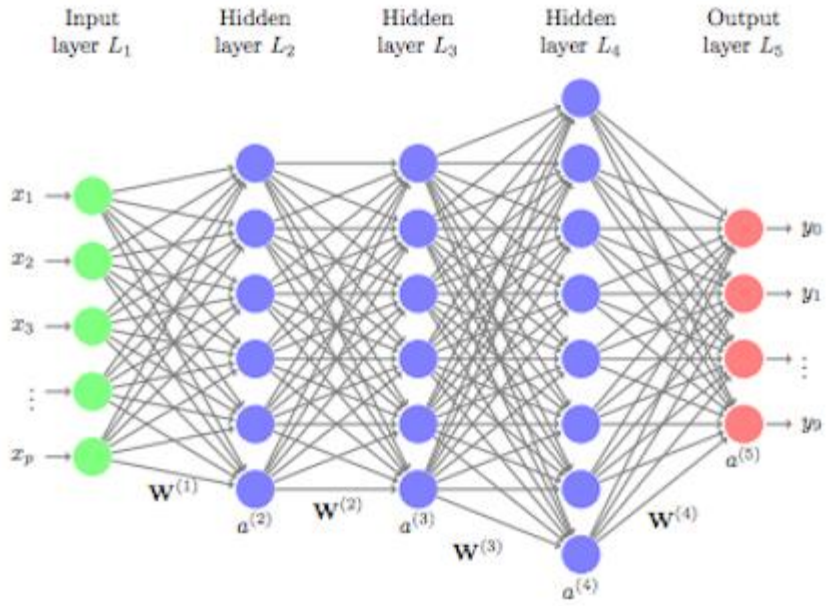
practical_5(a,b,c).py
lonospear.csv
pima-indians-diab....csv
Show all
Type here to search
30°C AQI 133
22:19
26-11-2021
colab.research.google.com/drive/1QJcZT9FU2RQvBGGzx1O6SUWQX12MJxn#scrollTo=abW8Oy2Pf0k8
practical 3(b,c).ipynb
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
plt.ylim([0.5, 1])
plt.legend(loc='lower right')
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
313/313 - 4s - loss: 0.8725 - accuracy: 0.7092 - 4s/epoch - 12ms/step
Accuracy
Epoch
accuracy
val_accuracy
4s completed at 22:06
practical_5(a,b,c).py
lonospear.csv
pima-indians-diab....csv
Show all
Type here to search
30°C Haze
22:20
26-11-2021

Practical No :- 4

Aim: Using deep feed forward network with two hidden layers for performing classification and predicting the class.

https://colab.research.google.com/drive/1IrIIB3GrdQ9L9KVSm_TDfXNZ8Bh4_dX_

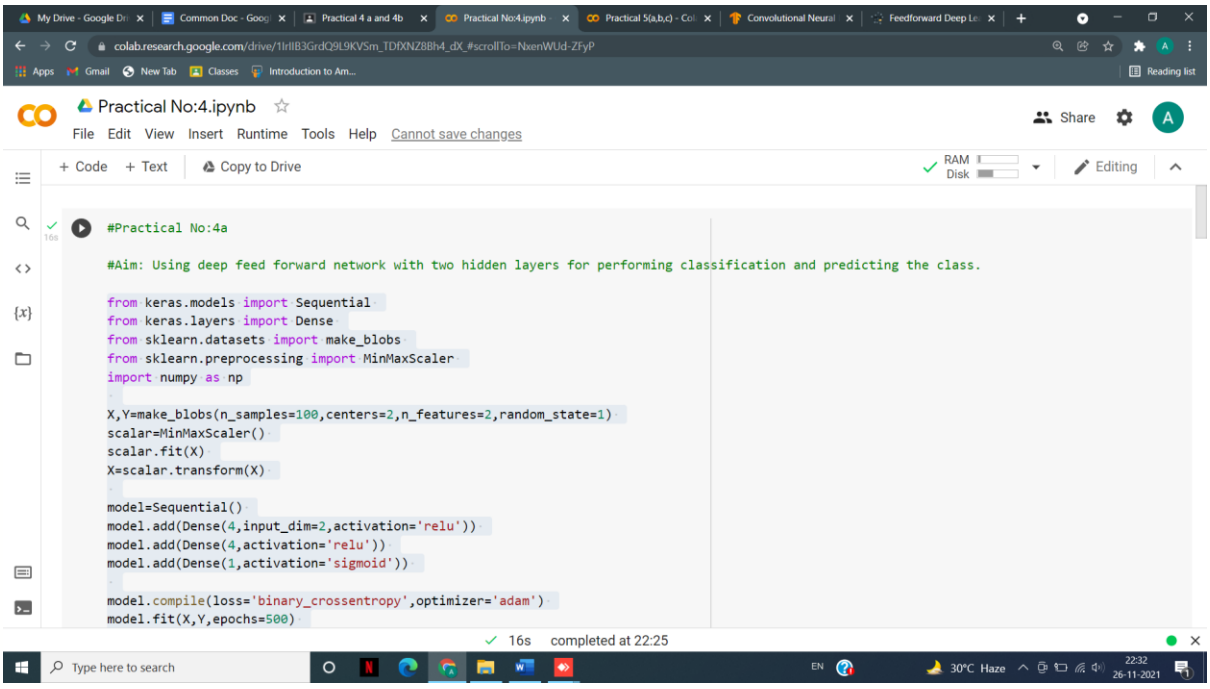
Machine learning algorithms typically search for the optimal representation of data using some feedback signal (aka objective/loss function). However, most machine learning algorithms only have the ability to use one or two layers of data transformation to learn the output representation. As data sets continue to grow in the dimensions of the feature space, finding the optimal output representation with a *shallow* model is not always possible. Deep learning provides a multi-layer approach to learn data representations, typically performed with a *multi-layer neural network*. Like other machine learning algorithms, deep neural networks (DNN) perform learning by mapping features to targets through a process of simple data transformations and feedback signals; however, DNNs place an emphasis on learning successive layers of meaningful representations. Although an intimidating subject, the overarching concept is rather simple and has proven highly successful in predicting a wide range of problems (i.e. image classification, speech recognition, autonomous driving).



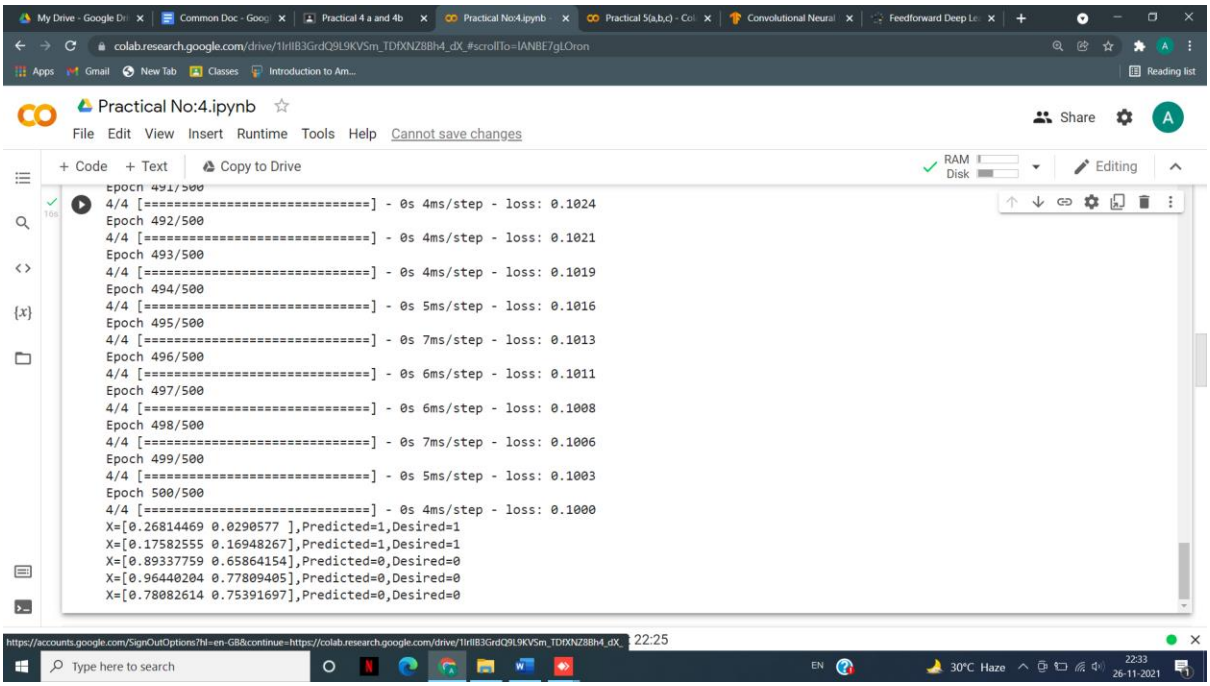
Code-:

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import make_blobs
from sklearn.preprocessing import MinMaxScaler
import numpy as np

X,Y=make_blobs(n_samples=100,centers=2,n_features=2,random_state=1)
```

```
scalar=MinMaxScaler()
scalar.fit(X)
X=scalar.transform(X)

model=Sequential()
model.add(Dense(4,input_dim=2,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam')
model.fit(X,Y,epochs=500)

Xnew,Yreal=make_blobs(n_samples=5,centers=2,n_features=2,random_state=1)

Xnew=scalar.transform(Xnew)
Ynew=model.predict(Xnew)
Ynew=np.round(Ynew).astype(int)

for i in range(len(Xnew)):
  print("X=%s,Predicted=%d,Desired=%s"%(Xnew[i],Ynew[i],Yreal[i]))
```



Output-:



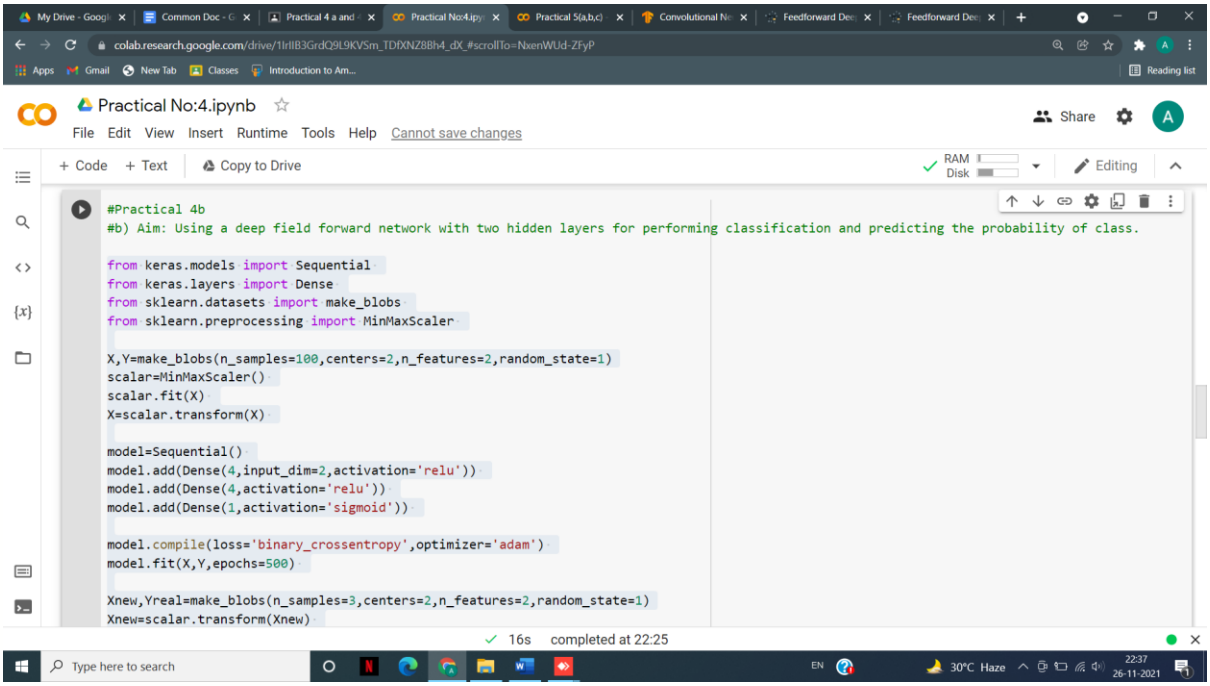
b) Aim: Using a deep field forward network with two hidden layers for performing classification and predic ting the probability of class.

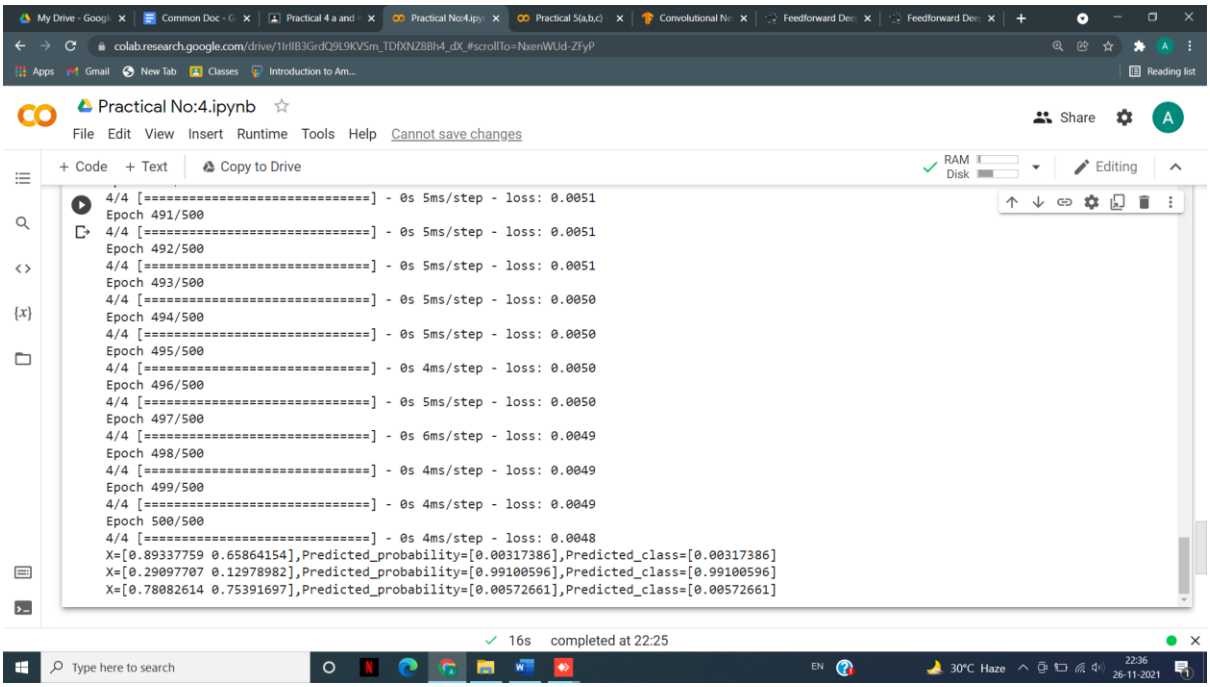
Code-:

```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.datasets import make_blobs
from sklearn.preprocessing import MinMaxScaler

X,Y=make_blobs(n_samples=100,centers=2,n_features=2,random_state=1)
scalar=MinMaxScaler()
scalar.fit(X)
X=scalar.transform(X)

model=Sequential()
model.add(Dense(4,input_dim=2,activation='relu'))
model.add(Dense(4,activation='relu'))
model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam')
model.fit(X,Y,epochs=500)

Xnew,Yreal=make_blobs(n_samples=3,centers=2,n_features=2,random_state=1)
Xnew=scalar.transform(Xnew)
Yclass=model.predict(Xnew)
```

```
Ynew=model.predict(Xnew)
for i in range(len(Xnew)):
  print("X=%s,Predicted_probability=%s,Predicted_class=%s"%(Xnew[i],Ynew[i],Yclass[i]))
```

Output-:



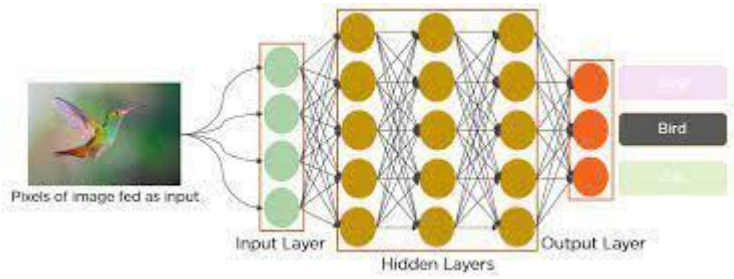


Practical No :- 5

Aim: cnn of cifer10 images

In the past few decades, Deep Learning has proved to be a very powerful tool because of its ability to handle large amounts of data. The interest to use hidden layers has surpassed traditional techniques, especially in pattern recognition. One of the most popular deep neural networks is Convolutional Neural Networks.

https://colab.research.google.com/drive/1C1hqqU2qpSzuGPKUohReCgIrMRcY-kH_#scrollTo=sjoN8Bjo4P-2

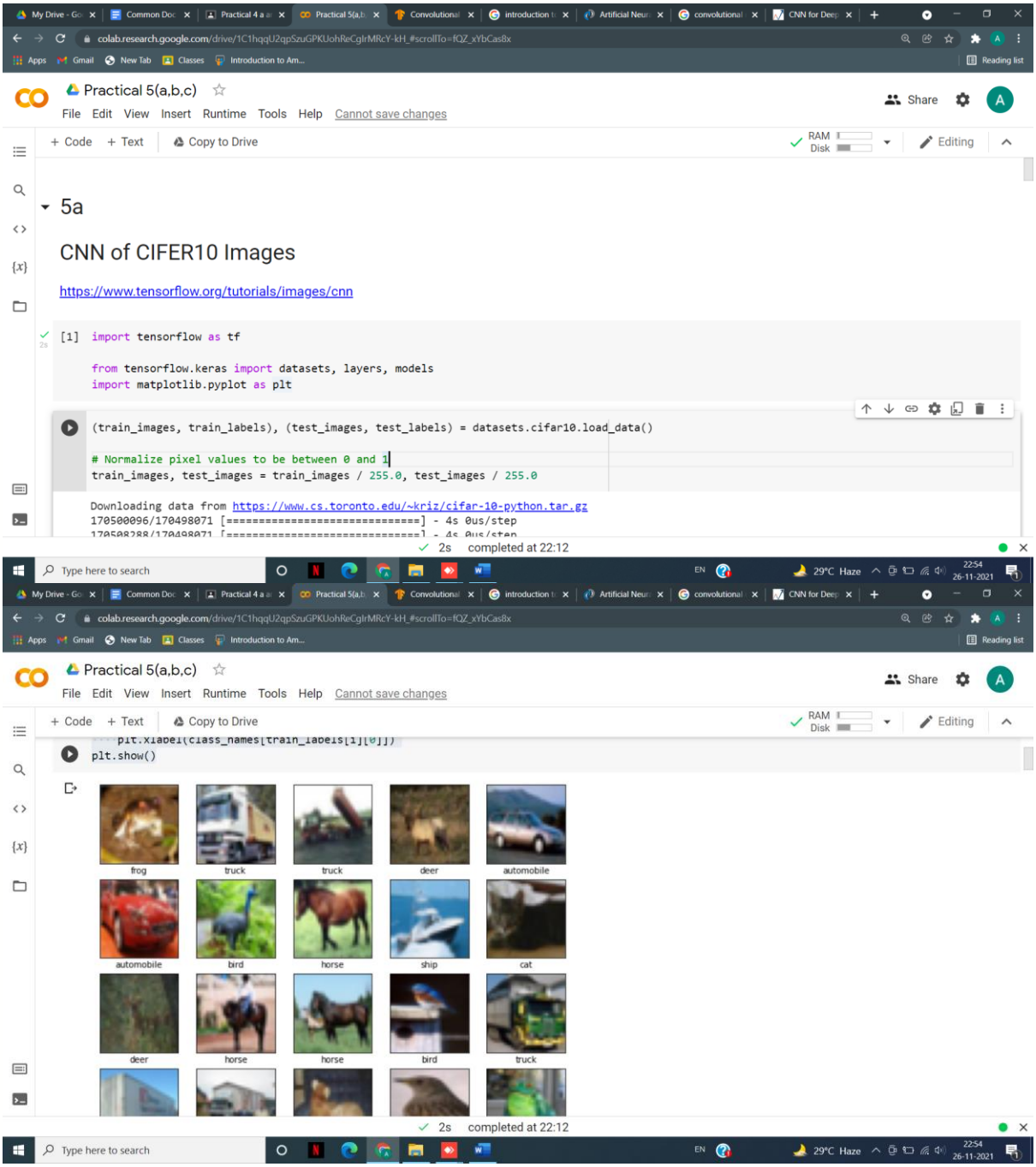


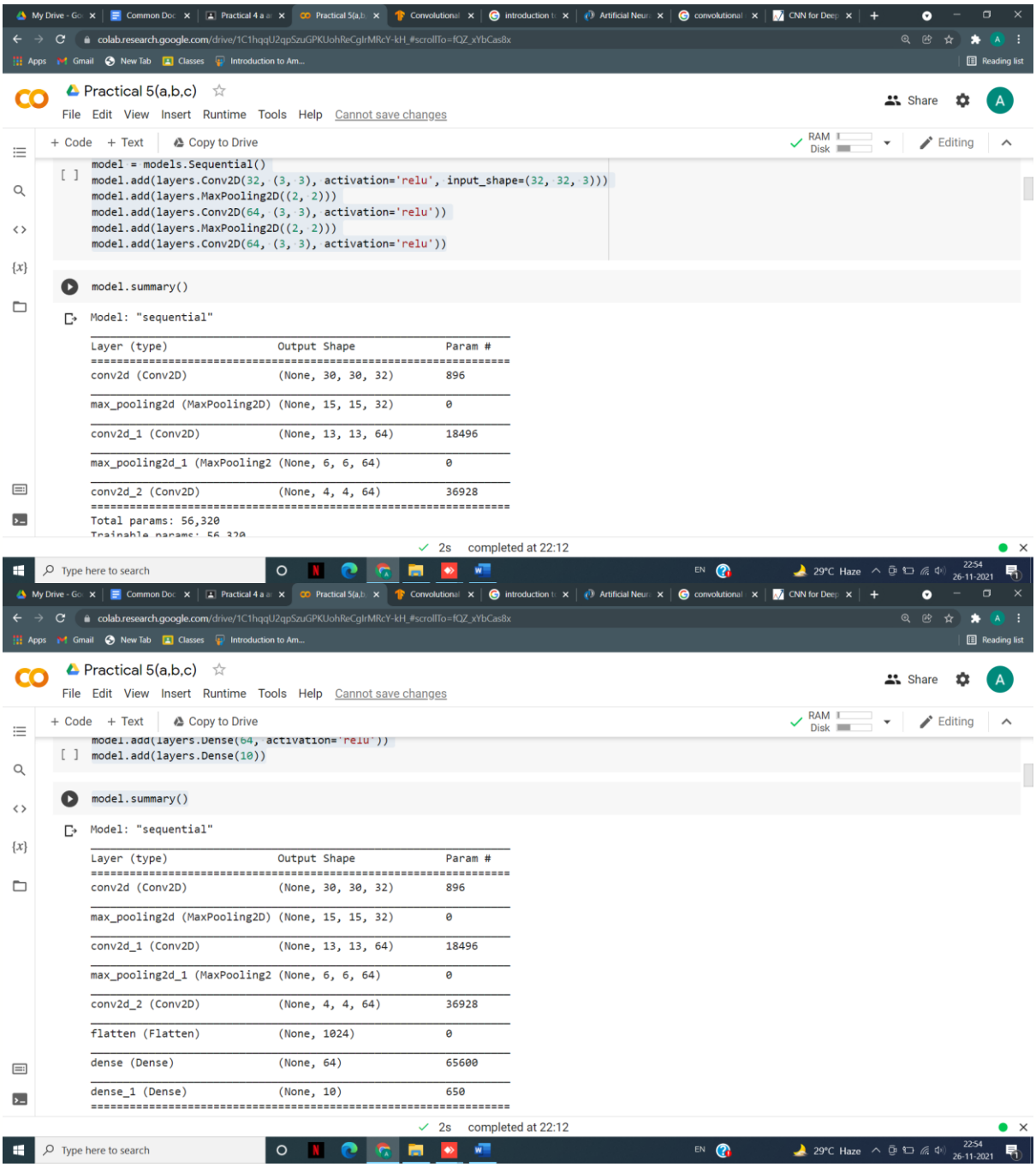
Code-:

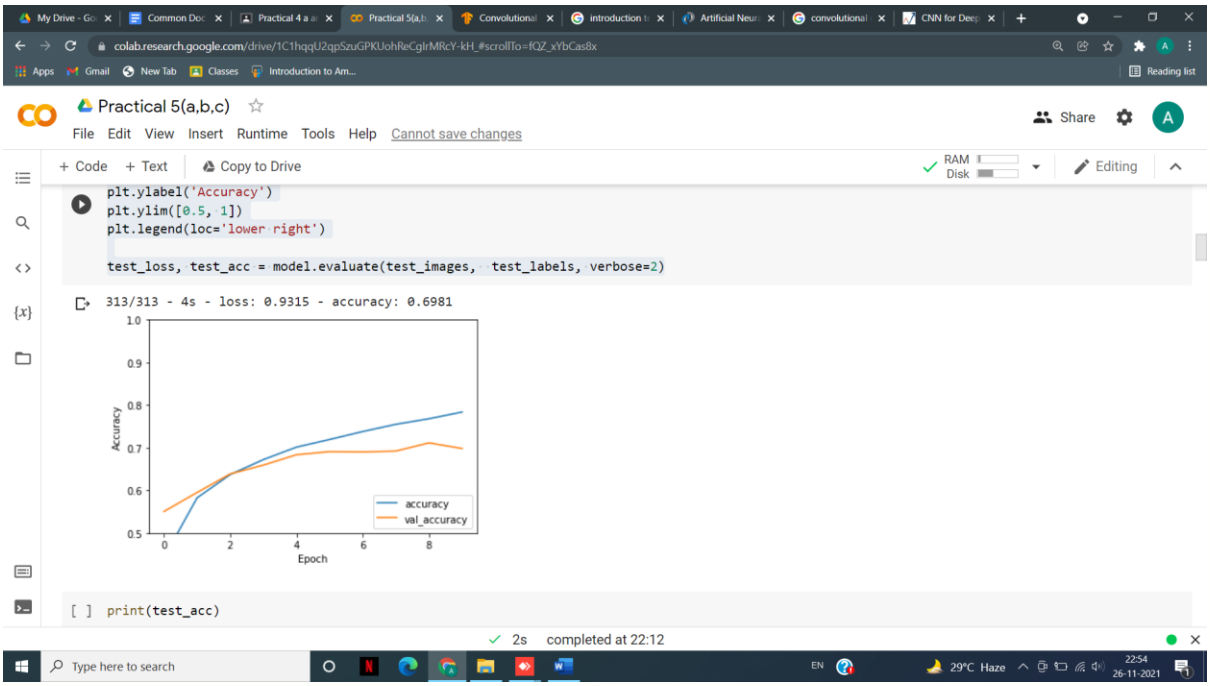
```
import tensorflow as tf
```

```
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    # The CIFAR labels happen to be arrays,
    # which is why you need the extra index
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))

model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10))

model.summary()
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0.5, 1])
plt.legend(loc='lower right')

test_loss, test_acc = model.evaluate(test_images,  test_labels, verbose=2)
```

Output-:


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
5a
CNN of CIFER10 Images
https://www.tensorflow.org/tutorials/images/cnn
[1] import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt
(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071 [==============================] - 4s 0us/step
2s completed at 22:12
plt.show()
frog
truck
truck
deer
automobile
automobile
bird
horse
ship
cat
deer
horse
horse
bird
truck
2s completed at 22:12


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
[ ] model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.summary()
Model: "sequential"
Layer (type) Output Shape Param #
conv2d (Conv2D) (None, 30, 30, 32) 896
max_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0
conv2d_1 (Conv2D) (None, 13, 13, 64) 18496
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64) 0
conv2d_2 (Conv2D) (None, 4, 4, 64) 36928
Total params: 56,320
2s completed at 22:12
Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
[ ] model.add(layers.Dense(10))
model.summary()
Model: "sequential"
Layer (type) Output Shape Param #
conv2d (Conv2D) (None, 30, 30, 32) 896
max_pooling2d (MaxPooling2D) (None, 15, 15, 32) 0
conv2d_1 (Conv2D) (None, 13, 13, 64) 18496
max_pooling2d_1 (MaxPooling2 (None, 6, 6, 64) 0
conv2d_2 (Conv2D) (None, 4, 4, 64) 36928
flatten (Flatten) (None, 1024) 0
dense (Dense) (None, 64) 65600
dense_1 (Dense) (None, 10) 650
2s completed at 22:12



b. Image classification

Code-:

```
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf

from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
 flower_photo/

 daisy/
 dandelion/
 roses/
 sunflowers/
 tulips/
import pathlib
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
"
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
data_dir = pathlib.Path(data_dir)
```

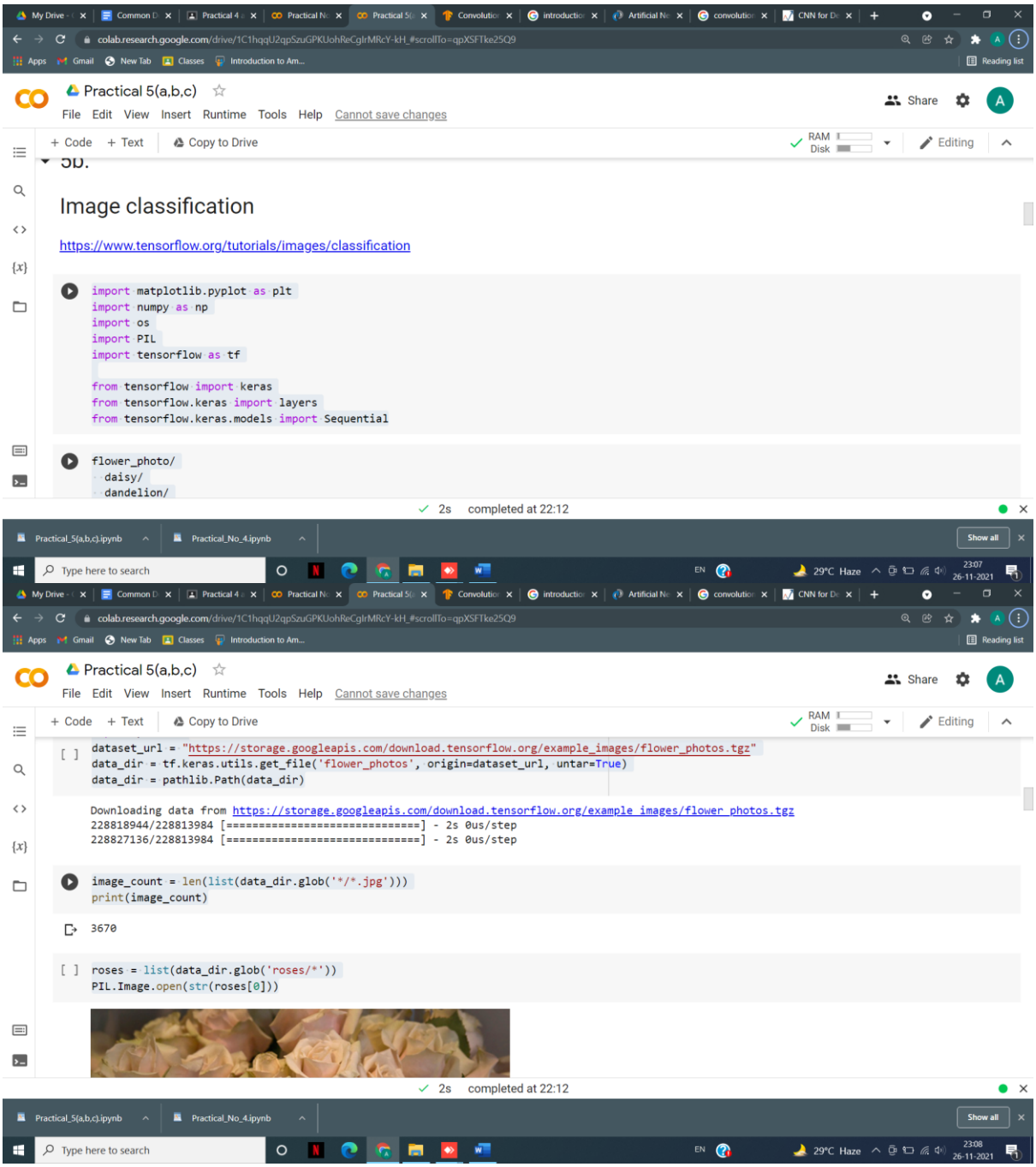
```
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[0]))
PIL.Image.open(str(roses[1]))
tulips = list(data_dir.glob('tulips/*'))
PIL.Image.open(str(tulips[0]))
PIL.Image.open(str(tulips[1]))
batch_size = 32
img_height = 180
img_width = 180
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
class_names = train_ds.class_names
print(class_names)
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
AUTOTUNE = tf.data.AUTOTUNE

train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
normalization_layer = layers.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
```

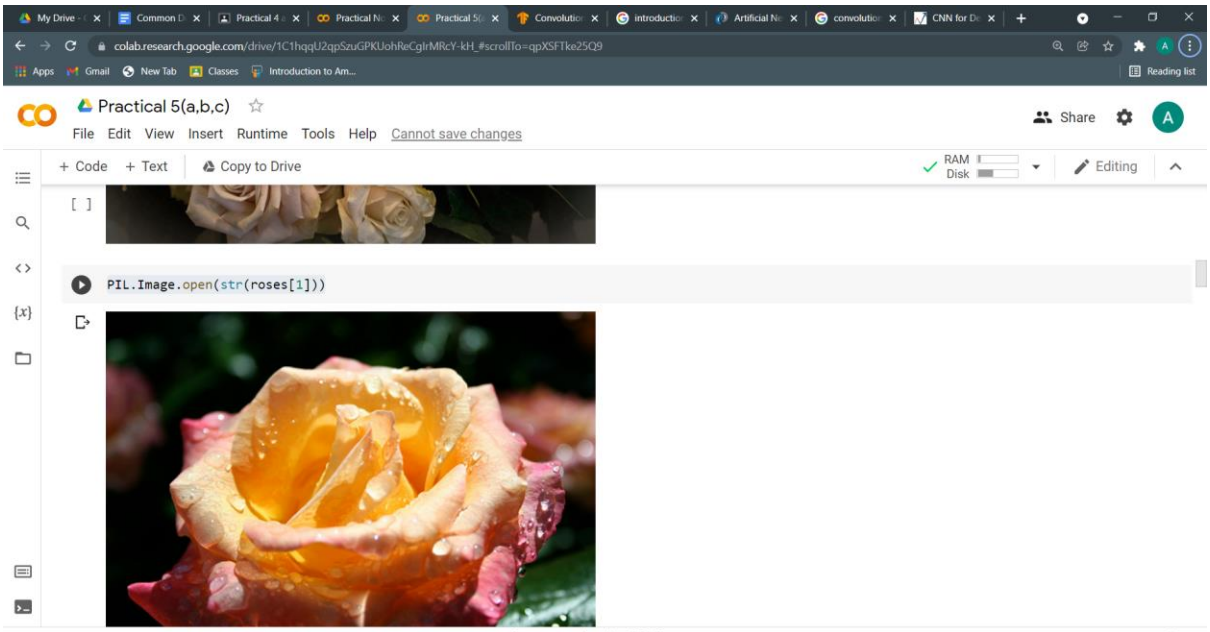
```
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
num_classes = 5

model = Sequential([
  layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
epochs=10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
)
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
data_augmentation = keras.Sequential(
```

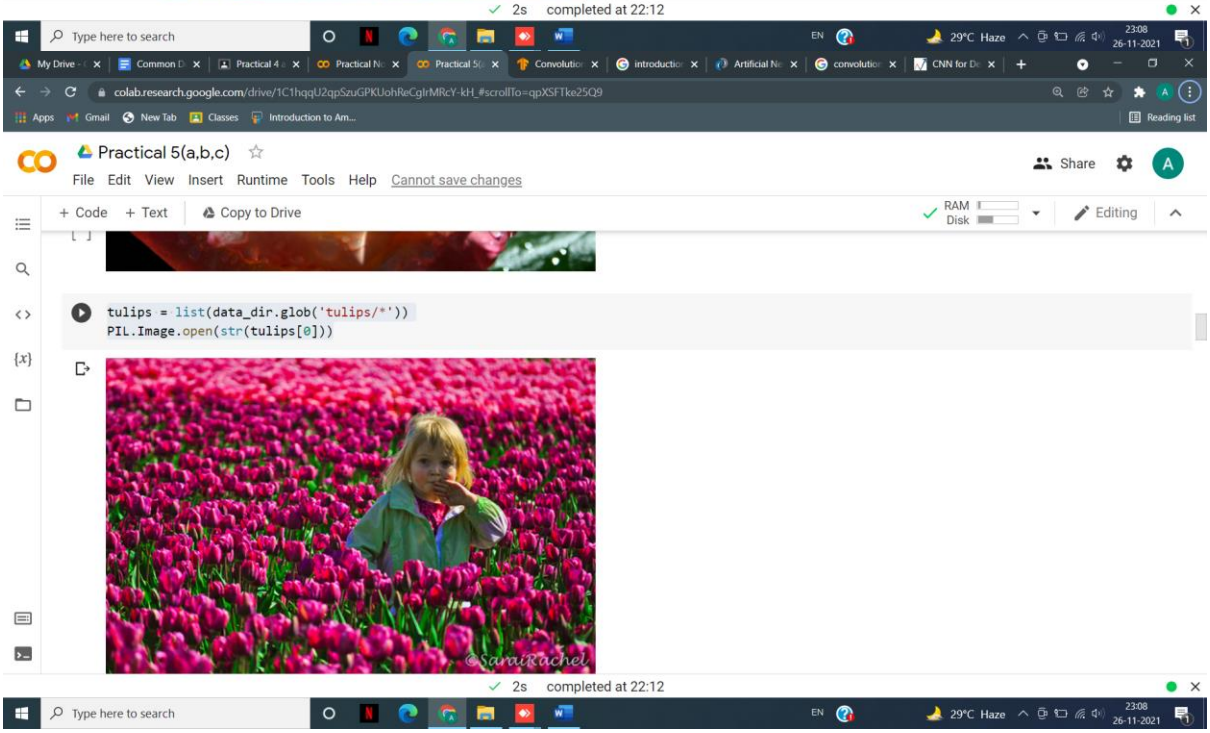
```
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
  for i in range(9):
    augmented_images = data_augmentation(images)
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(augmented_images[0].numpy().astype("uint8"))
    plt.axis("off")
model = Sequential([
  data_augmentation,
  layers.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
epochs = 10
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

loss = history.history['loss']
val_loss = history.history['val_loss']

epochs_range = range(epochs)
```

```
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
sunflower_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg"
sunflower_path = tf.keras.utils.get_file('Red_sunflower', origin=sunflower_url)

img = tf.keras.utils.load_img(
    sunflower_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)
```

Output-:


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
5b.
Image classification
https://www.tensorflow.org/tutorials/images/classification
import matplotlib.pyplot as plt
import numpy as np
import os
import PIL
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
flower_photo/
daisy/
dandelion/
2s completed at 22:12
Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
data_dir = pathlib.Path(data_dir)
Downloading data from https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
228818944/228813984 [==============================] - 2s 0us/step
228827136/228813984 [==============================] - 2s 0us/step
image_count = len(list(data_dir.glob('*/*.jpg')))
print(image_count)
3670
roses = list(data_dir.glob('roses/*'))
PIL.Image.open(str(roses[0]))
2s completed at 22:12


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
RAM Disk
Editing
Share
PIL.Image.open(str(roses[1]))
2s completed at 22:12


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
RAM Disk
Editing
Share
tulips = list(data_dir.glob('tulips/*'))
PIL.Image.open(str(tulips[0]))
2s completed at 22:12

Practical 5(a,b,c)

File Edit View Insert Runtime Tools Help Cannot save changes

+ Code + Text Copy to Drive

```
batch_size = 32
img_height = 180
img_width = 180
```

```
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)
```

2s completed at 22:12

Practical 5(a,b,c)

File Edit View Insert Runtime Tools Help Cannot save changes

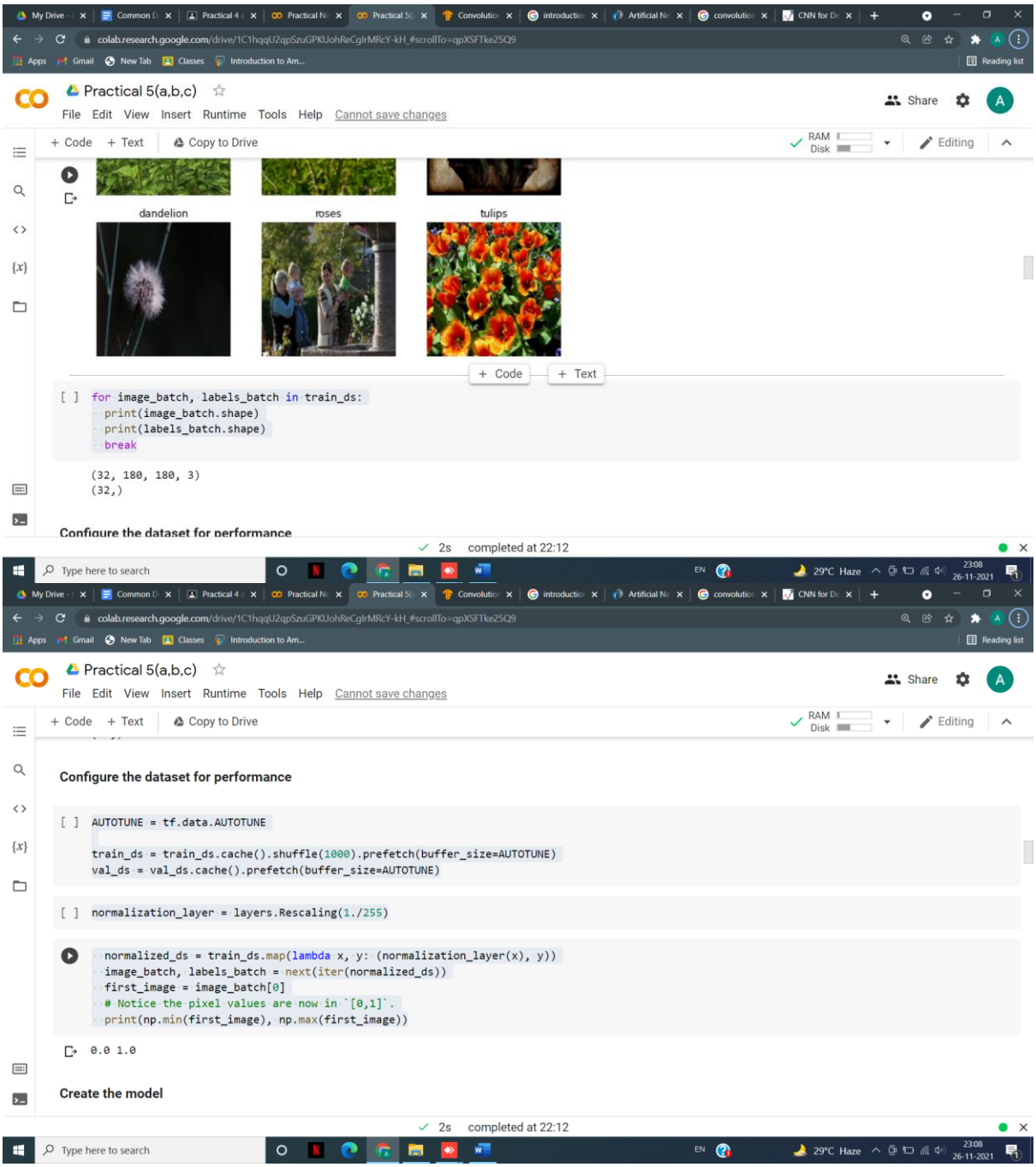
+ Code + Text Copy to Drive

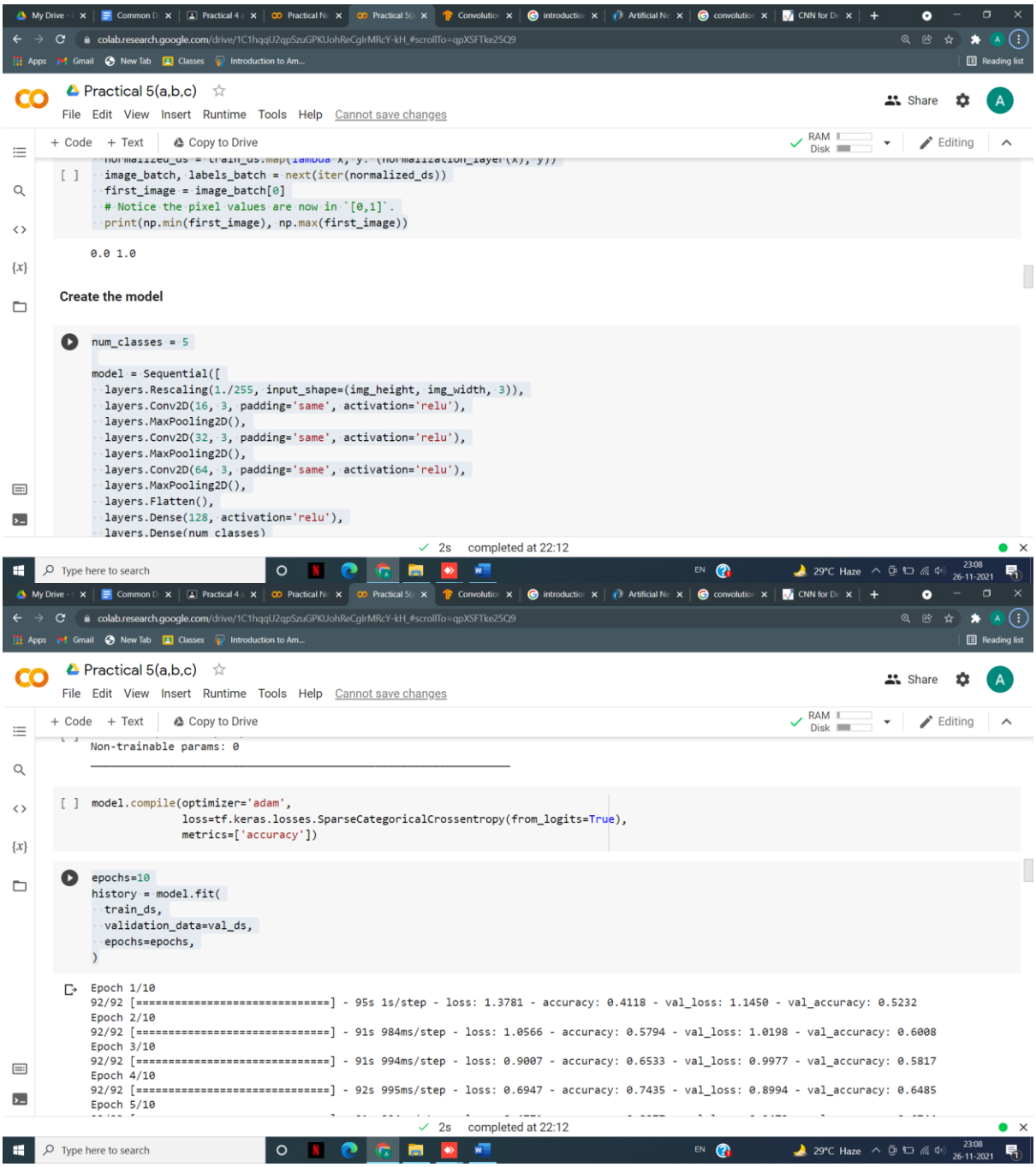
```
class_names = train_ds.class_names
print(class_names)
```

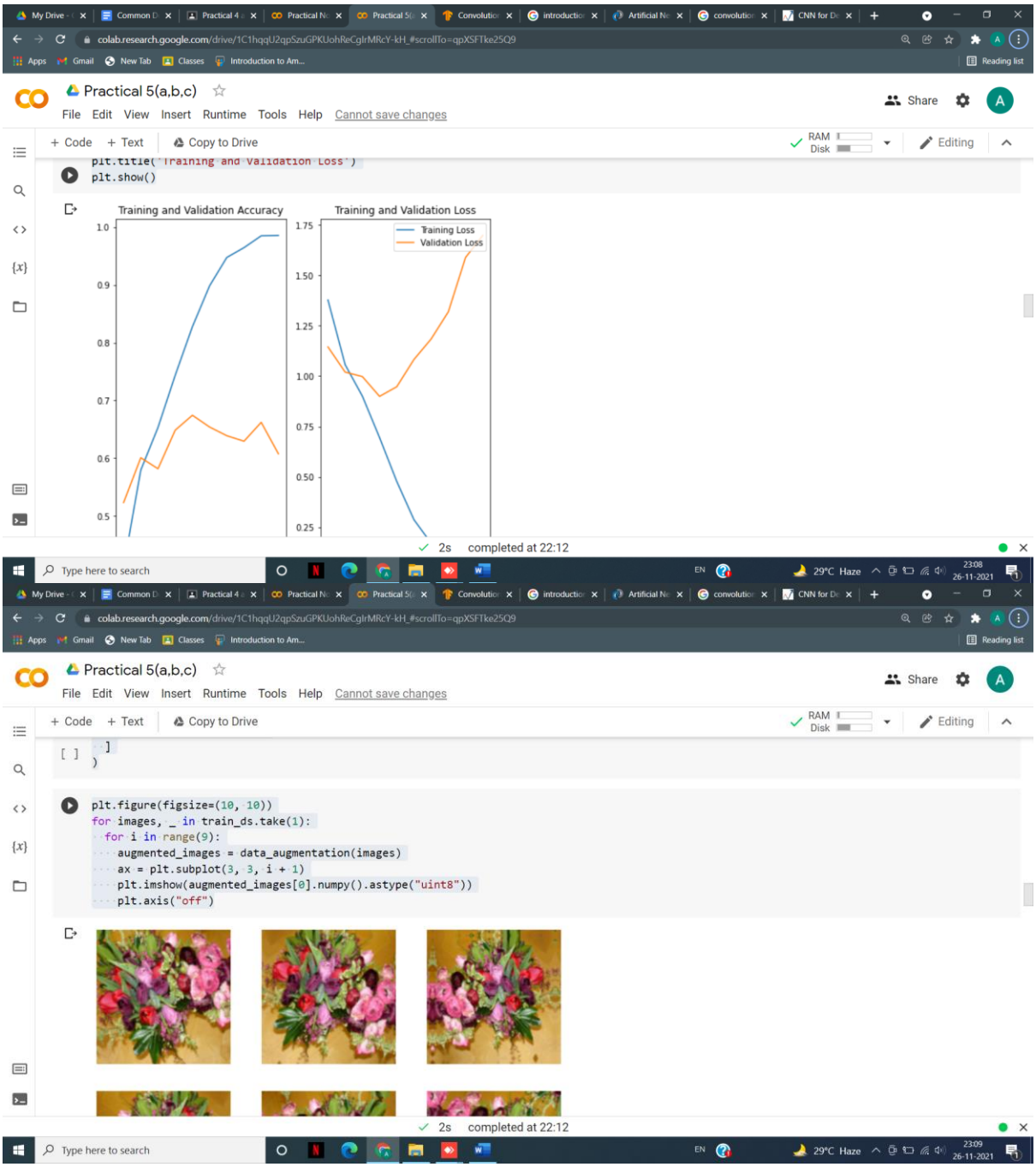
```
['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']
```

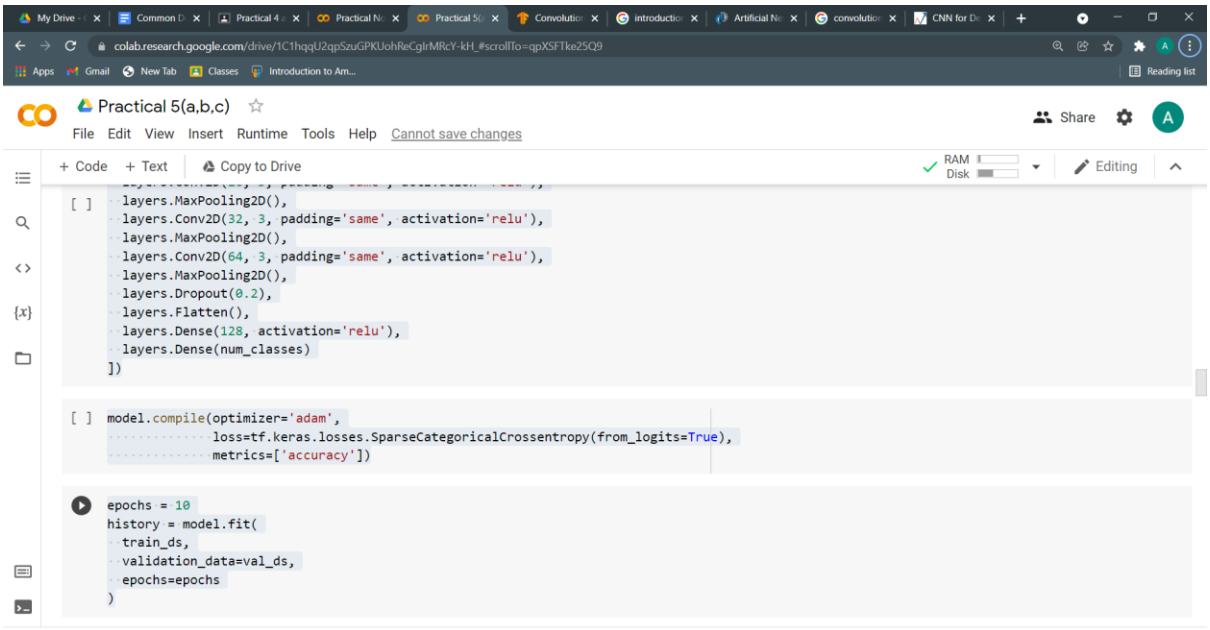
```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_ds.take(1):
  for i in range(9):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[labels[i]])
    plt.axis("off")
```

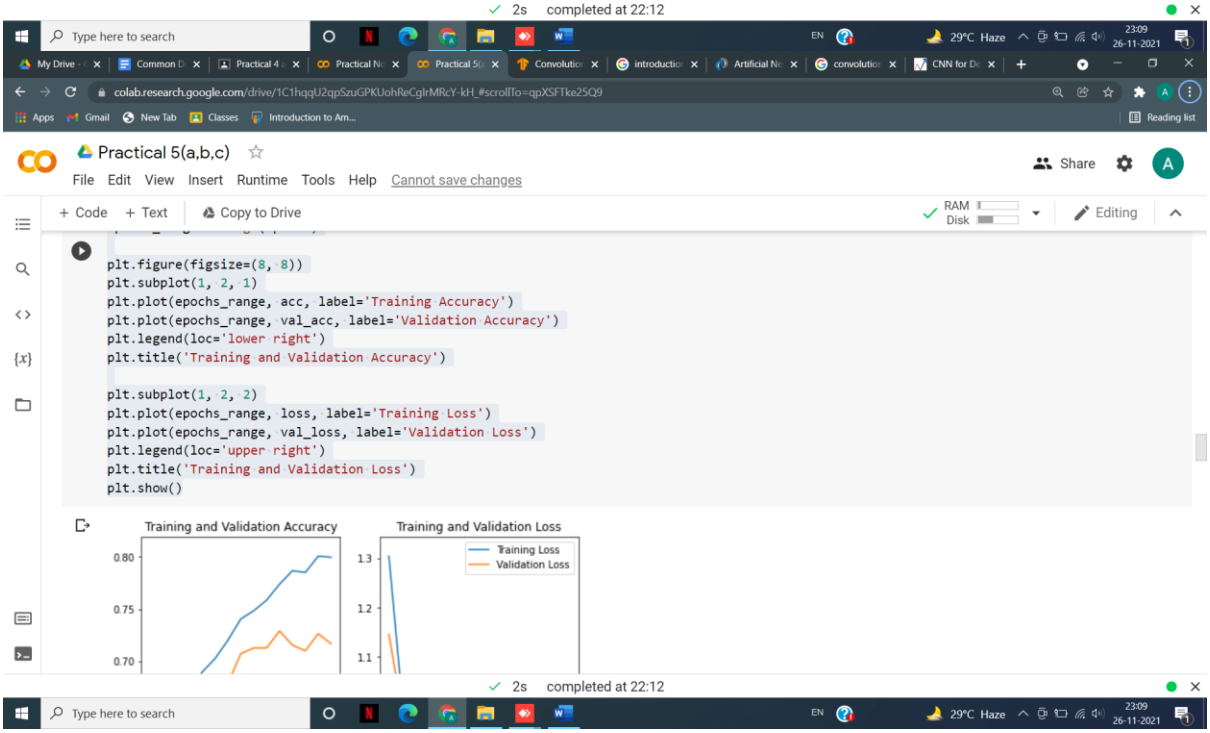
2s completed at 22:12

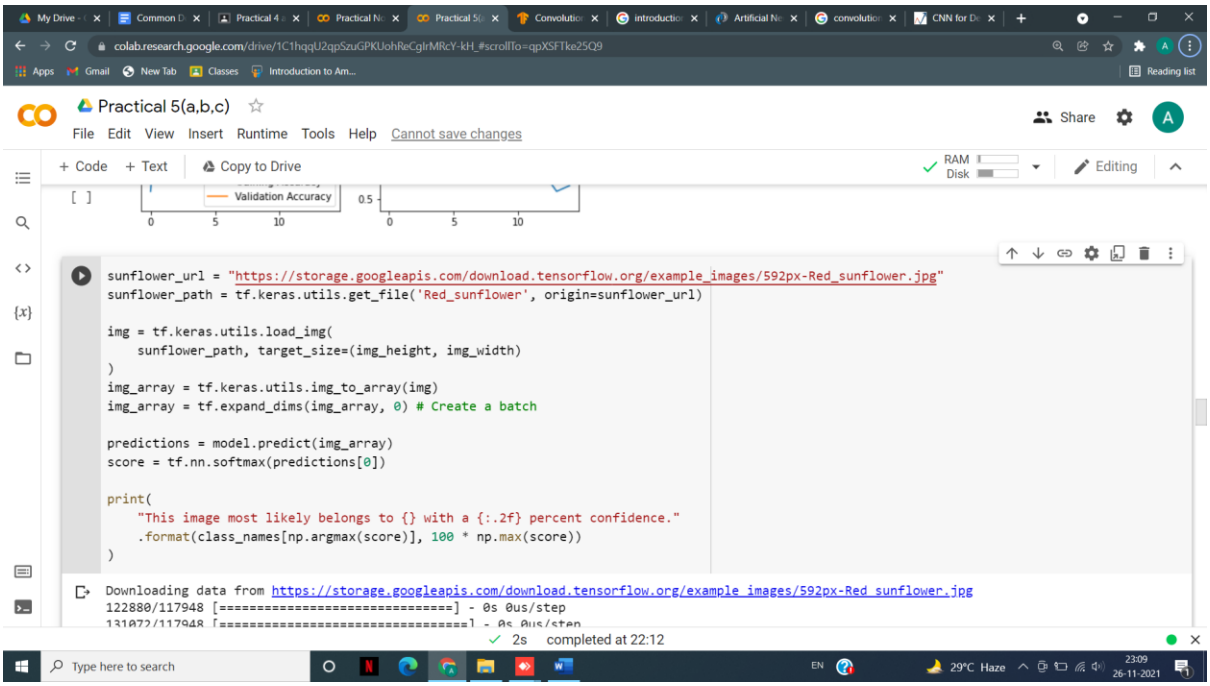

Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
dandelion
roses
tulips
+ Code + Text
for image_batch, labels_batch in train_ds:
print(image_batch.shape)
print(labels_batch.shape)
break
(32, 180, 180, 3)
(32,)
Configure the dataset for performance
2s completed at 22:12
Type here to search
29°C Haze
23:08
26-11-2021
Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
Configure the dataset for performance
AUTOTUNE = tf.data.AUTOTUNE
train_ds = train_ds.cache().shuffle(1000).prefetch(buffer_size=AUTOTUNE)
val_ds = val_ds.cache().prefetch(buffer_size=AUTOTUNE)
normalization_layer = layers.Rescaling(1./255)
normalized_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
0.0 1.0
Create the model
2s completed at 22:12
Type here to search
29°C Haze
23:08
26-11-2021


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
Share
RAM Disk
Editing
image_batch, labels_batch = next(iter(normalized_ds))
first_image = image_batch[0]
# Notice the pixel values are now in `[0,1]`.
print(np.min(first_image), np.max(first_image))
0.0 1.0
Create the model
num_classes = 5
model = Sequential([
layers.Rescaling(1./255, input_shape=(img_height, img_width, 3)),
layers.Conv2D(16, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(32, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(64, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(num_classes)
2s completed at 22:12
Type here to search
29°C Haze
23:08
26-11-2021
Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
+ Code + Text Copy to Drive
Share
RAM Disk
Editing
Non-trainable params: 0
model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
epochs=10
history = model.fit(
train_ds,
validation_data=val_ds,
epochs=epochs,
)
Epoch 1/10
92/92 [==============================] - 95s 1s/step - loss: 1.3781 - accuracy: 0.4118 - val_loss: 1.1450 - val_accuracy: 0.5232
Epoch 2/10
92/92 [==============================] - 91s 984ms/step - loss: 1.0566 - accuracy: 0.5794 - val_loss: 1.0198 - val_accuracy: 0.6008
Epoch 3/10
92/92 [==============================] - 91s 994ms/step - loss: 0.9007 - accuracy: 0.6533 - val_loss: 0.9977 - val_accuracy: 0.5817
Epoch 4/10
92/92 [==============================] - 92s 995ms/step - loss: 0.6947 - accuracy: 0.7435 - val_loss: 0.8994 - val_accuracy: 0.6485
Epoch 5/10
2s completed at 22:12
Type here to search
29°C Haze
23:08
26-11-2021


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
plt.title('Training and Validation Loss')
plt.show()
Training and Validation Accuracy
Training and Validation Loss
Training Loss
Validation Loss
2s completed at 22:12
Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
plt.figure(figsize=(10, 10))
for images, _ in train_ds.take(1):
for i in range(9):
augmented_images = data_augmentation(images)
ax = plt.subplot(3, 3, i + 1)
plt.imshow(augmented_images[0].numpy().astype("uint8"))
plt.axis("off")
2s completed at 22:12


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
[ ] layers.MaxPooling2D(),
layers.Conv2D(32, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(64, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Dropout(0.2),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(num_classes)
])
[ ] model.compile(optimizer='adam',
loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
metrics=['accuracy'])
epochs = 10
history = model.fit(
train_ds,
validation_data=val_ds,
epochs=epochs
)
2s completed at 22:12


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
RAM Disk
Editing
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()
Training and Validation Accuracy
0.80
0.75
0.70
Training and Validation Loss
1.3
1.2
1.1
Training Loss
Validation Loss
2s completed at 22:12



c.Data Augmentation

Code-:

```
import matplotlib.pyplot as plt
import numpy as np
import tensorflow as tf
import tensorflow_datasets as tfds

from tensorflow.keras import layers
(train_ds, val_ds, test_ds), metadata = tfds.load(
   'tf_flowers',
   split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
   with_info=True,
   as_supervised=True,
)
 num_classes = metadata.features['label'].num_classes
 print(num_classes)
get_label_name = metadata.features['label'].int2str

image, label = next(iter(train_ds))
_ = plt.imshow(image)
_ = plt.title(get_label_name(label))
IMG_SIZE = 180
```

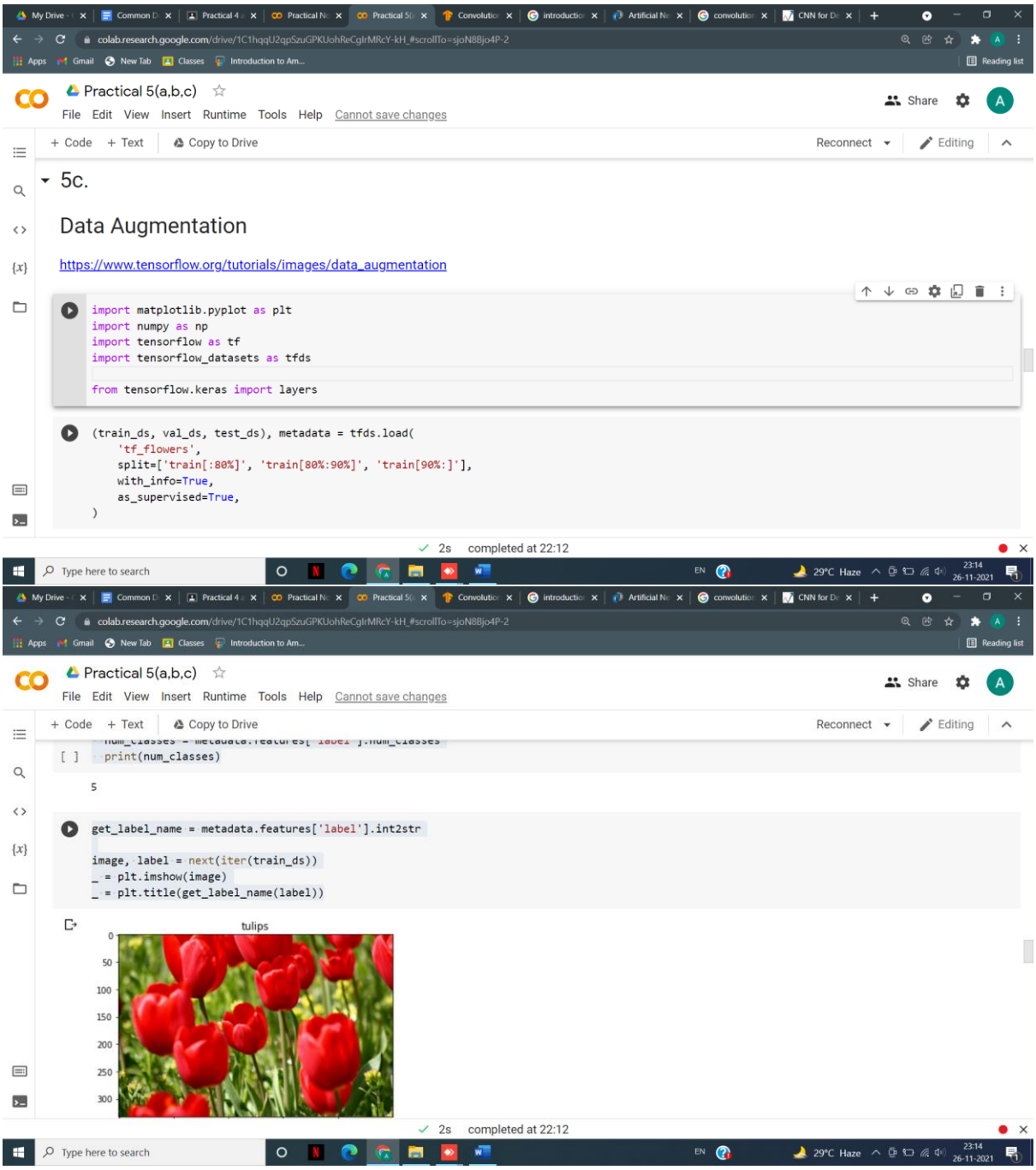
```
resize_and_rescale = tf.keras.Sequential([
  layers.Resizing(IMG_SIZE, IMG_SIZE),
  layers.Rescaling(1./255)
])
result = resize_and_rescale(image)
_ = plt.imshow(result)
print("Min and max pixel values:", result.numpy().min(), result.numpy().max())
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.2),
])
# Add the image to a batch.
image = tf.expand_dims(image, 0)
plt.figure(figsize=(10, 10))
for i in range(9):
  augmented_image = data_augmentation(image)
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(augmented_image[0])
  plt.axis("off")
model = tf.keras.Sequential([
  # Add the preprocessing layers you created earlier.
  resize_and_rescale,
  data_augmentation,
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  # Rest of your model.
])
aug_ds = train_ds.map(
  lambda x, y: (resize_and_rescale(x, training=True), y))
batch_size = 32
AUTOTUNE = tf.data.AUTOTUNE

def prepare(ds, shuffle=False, augment=False):
  # Resize and rescale all datasets.
  ds = ds.map(lambda x, y: (resize_and_rescale(x), y),
              num_parallel_calls=AUTOTUNE)

  if shuffle:
    ds = ds.shuffle(1000)

  # Batch all datasets.
  ds = ds.batch(batch_size)

  # Use data augmentation only on the training set.
  if augment:
    ds = ds.map(lambda x, y: (data_augmentation(x, training=True), y),
```

```
            num_parallel_calls=AUTOTUNE)

  # Use buffered prefetching on all datasets.
  return ds.prefetch(buffer_size=AUTOTUNE)
train_ds = prepare(train_ds, shuffle=True, augment=True)
val_ds = prepare(val_ds)
test_ds = prepare(test_ds)
model = tf.keras.Sequential([
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes)
])
model.compile(optimizer='adam',
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'])
epochs=5
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
def random_invert_img(x, p=0.5):
  if  tf.random.uniform([]) < p:
    x = (255-x)
  else:
    x
  return x
def random_invert(factor=0.5):
  return layers.Lambda(lambda x: random_invert_img(x, factor))

random_invert = random_invert()
plt.figure(figsize=(10, 10))
for i in range(9):
  augmented_image = random_invert(image)
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(augmented_image[0].numpy().astype("uint8"))
  plt.axis("off")
class RandomInvert(layers.Layer):
  def __init__(self, factor=0.5, **kwargs):
    super().__init__(**kwargs)
```

```
    self.factor = factor

  def call(self, x):
    return random_invert_img(x)
_ = plt.imshow(RandomInvert()(image)[0])
(train_ds, val_ds, test_ds), metadata = tfds.load(
    'tf_flowers',
    split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
    with_info=True,
    as_supervised=True,
)
image, label = next(iter(train_ds))
_ = plt.imshow(image)
_ = plt.title(get_label_name(label))
def visualize(original, augmented):
  fig = plt.figure()
  plt.subplot(1,2,1)
  plt.title('Original image')
  plt.imshow(original)

  plt.subplot(1,2,2)
  plt.title('Augmented image')
  plt.imshow(augmented)
# flipped = tf.image.flip_left_right(image)

visualize(image, flipped)
grayscaled = tf.image.rgb_to_grayscale(image)
visualize(image, tf.squeeze(grayscaled))
_ = plt.colorbar()
saturated = tf.image.adjust_saturation(image, 3)
visualize(image, saturated)
 Flip an image

Flip an image either vertically or horizontally with `tf.image.flip_left_right`:
for i in range(3):
  seed = (i, 0)  # tuple of size (2,)
  stateless_random_contrast = tf.image.stateless_random_contrast(
      image, lower=0.1, upper=0.9, seed=seed)
  visualize(image, stateless_random_contrast)
#### Randomly crop an image

##Randomly crop `image` using `tf.image.stateless_random_crop` by providing target `size` and `seed`. The p
ortion that gets cropped out of `image` is at a randomly chosen offset and is associated with the given `seed`.
for i in range(3):
  seed = (i, 0)  # tuple of size (2,)
  stateless_random_crop = tf.image.stateless_random_crop(
      image, size=[210, 300, 3], seed=seed)
```

visualize(image, stateless_random_crop)

Output-:



Practical 5(a,b,c)

File Edit View Insert Runtime Tools Help Cannot save changes

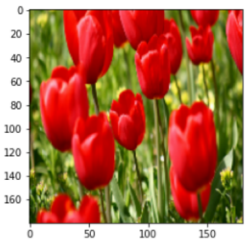
Share

+ Code + Text Copy to Drive

Reconnect Editing

```
IMG_SIZE = 180

resize_and_rescale = tf.keras.Sequential([
  layers.Resizing(IMG_SIZE, IMG_SIZE),
  layers.Rescaling(1./255)
])
```

```
result = resize_and_rescale(image)
_ = plt.imshow(result)
```



2s completed at 22:12

Practical 5(a,b,c)

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

Reconnect Editing

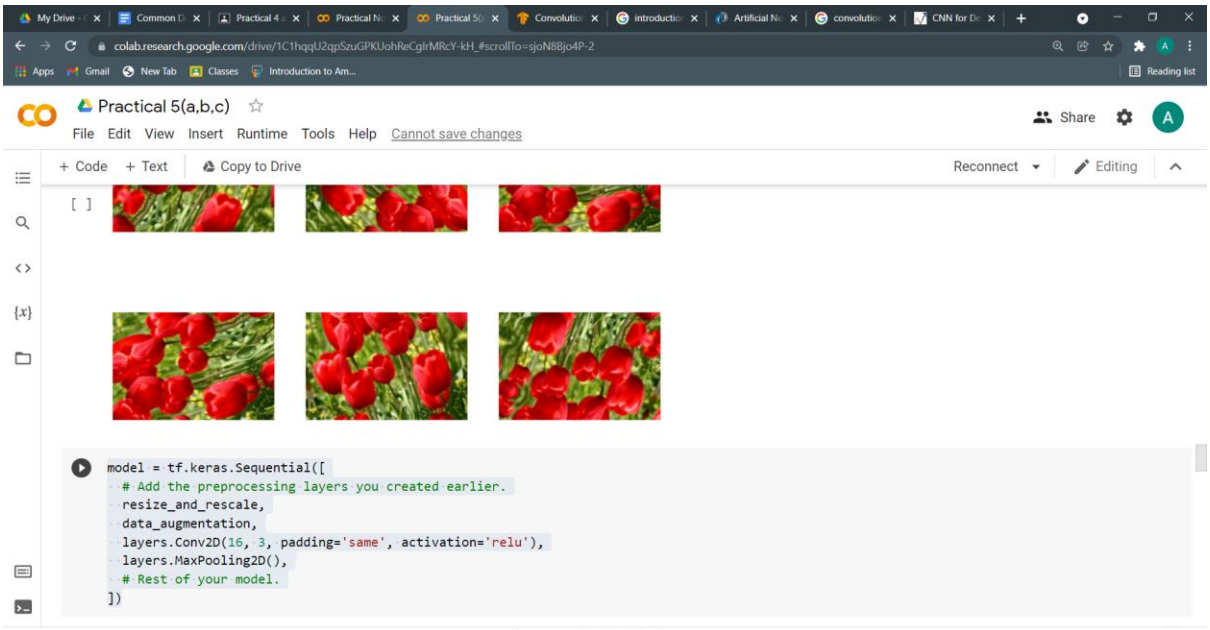
```
Min and max pixel values: 0.0 1.0
```

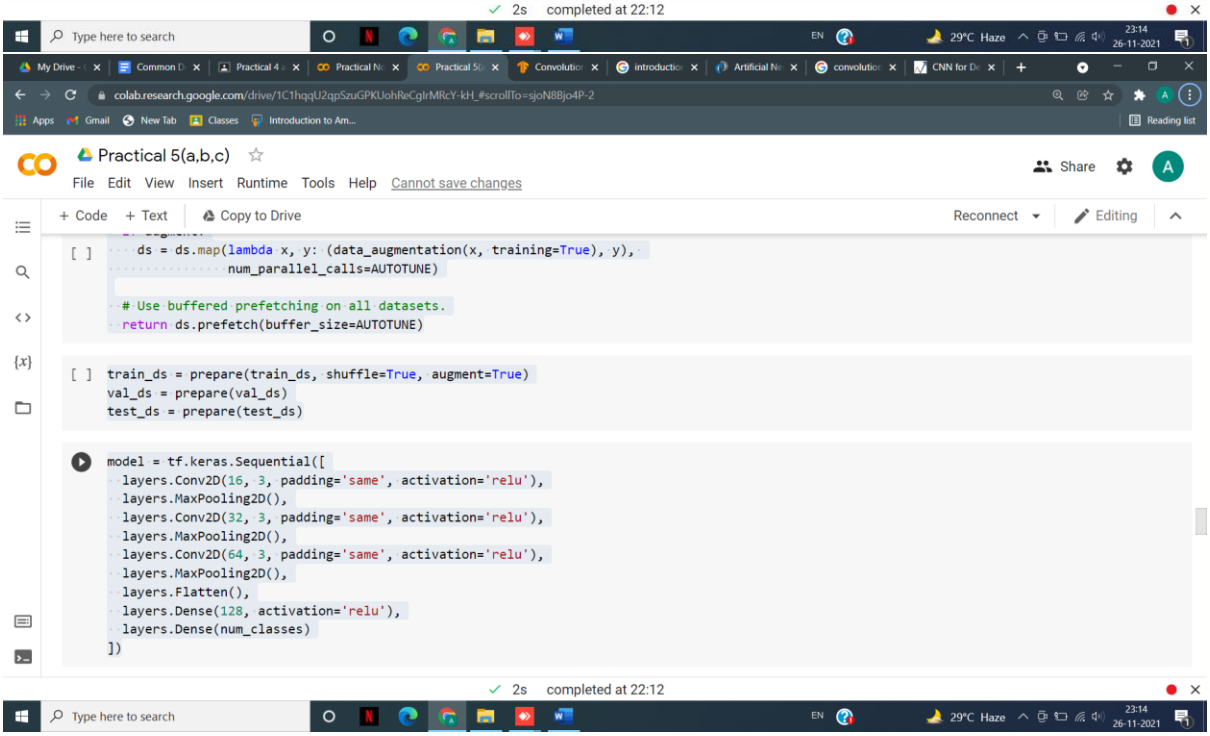
```
data_augmentation = tf.keras.Sequential([
  layers.RandomFlip("horizontal_and_vertical"),
  layers.RandomRotation(0.2),
])
```

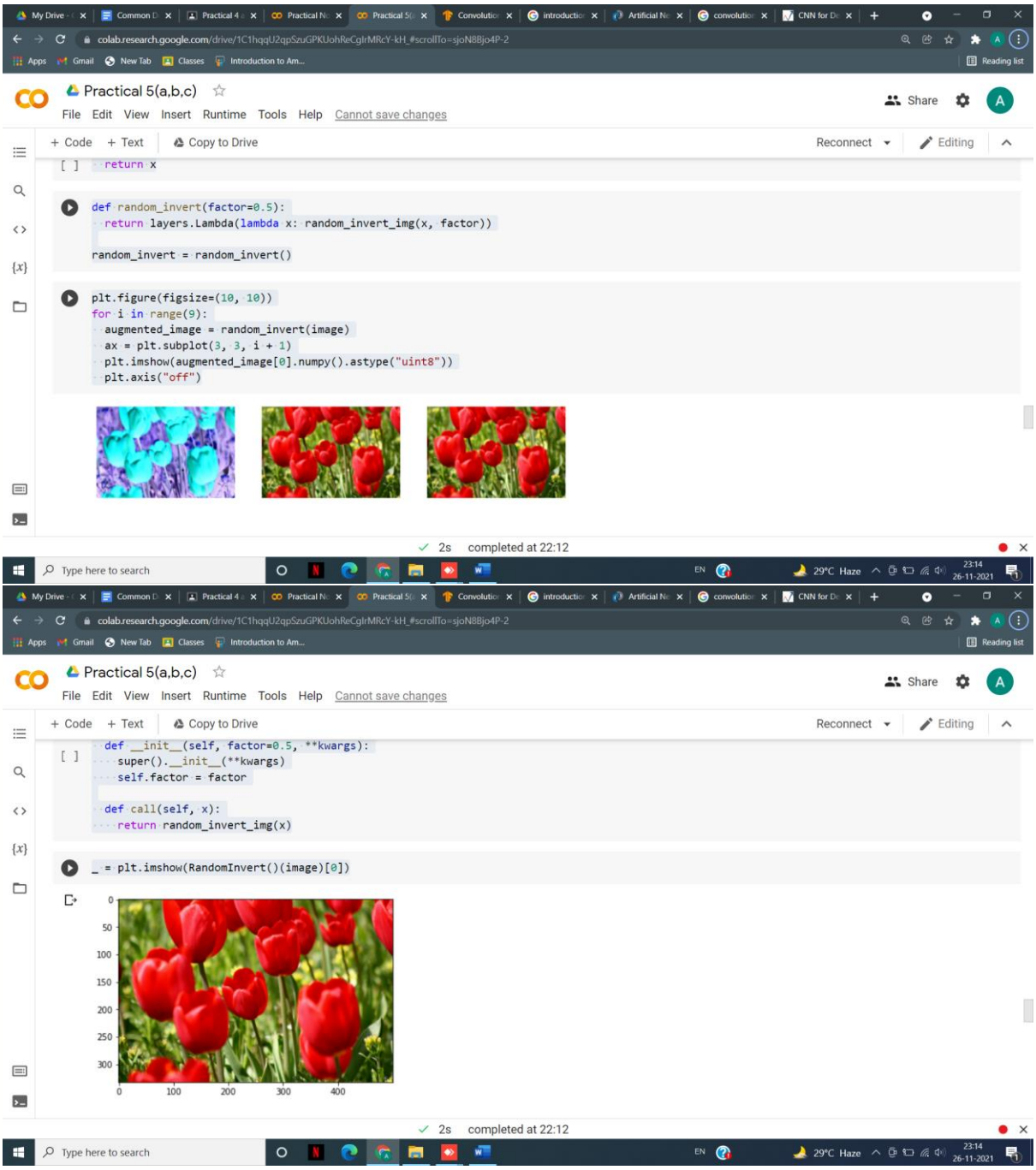
```
# Add the image to a batch.
image = tf.expand_dims(image, 0)
```

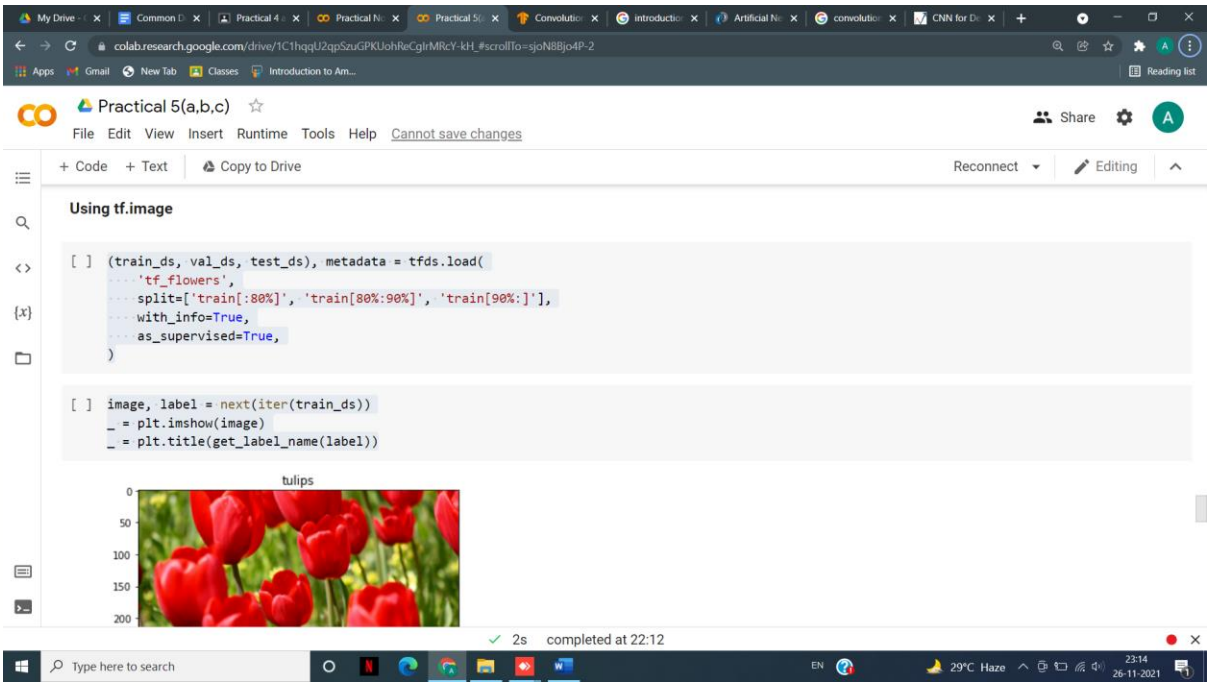
```
plt.figure(figsize=(10, 10))
for i in range(9):
  augmented_image = data_augmentation(image)
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(augmented_image[0])
  plt.axis("off")
```

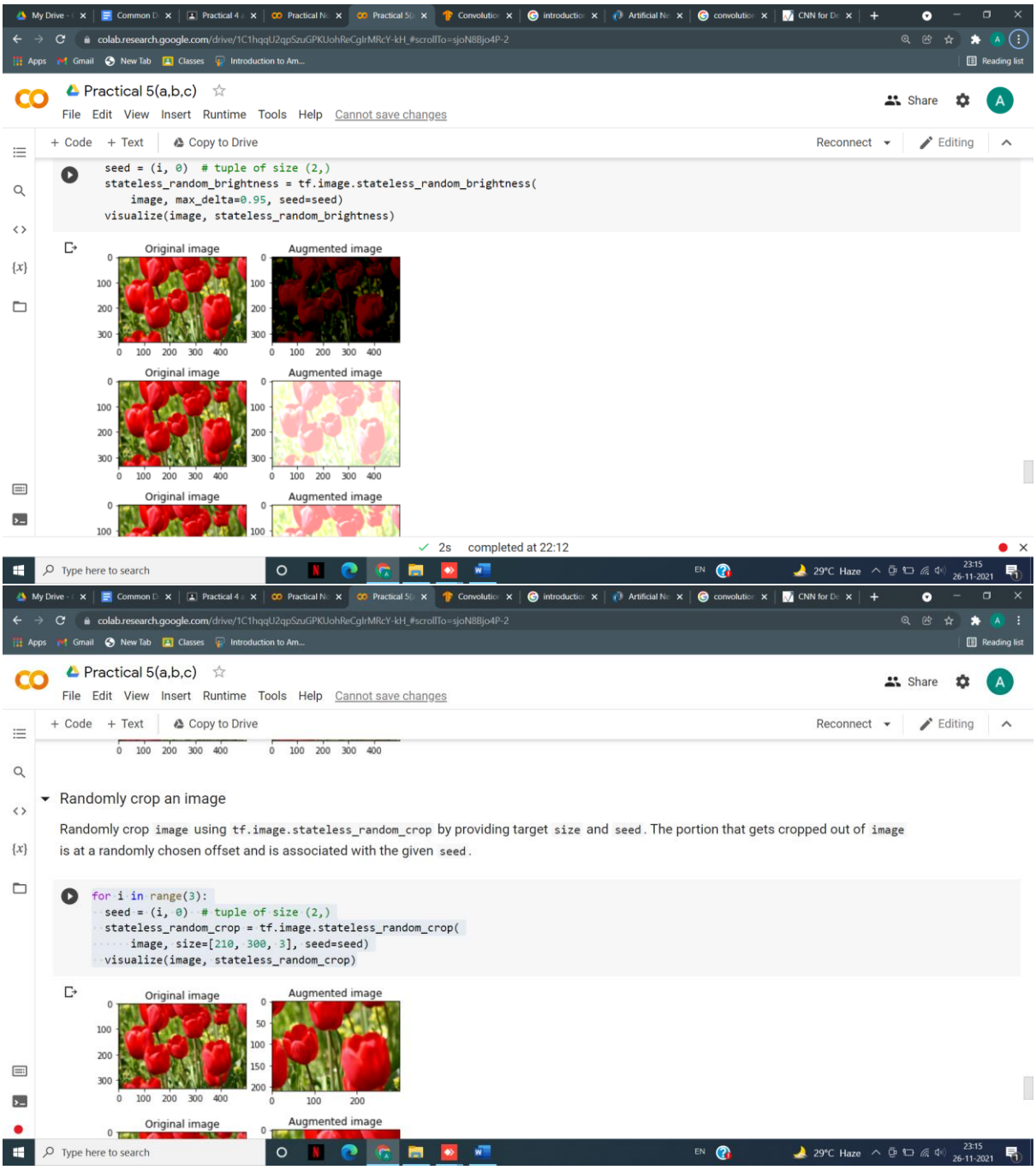
2s completed at 22:12

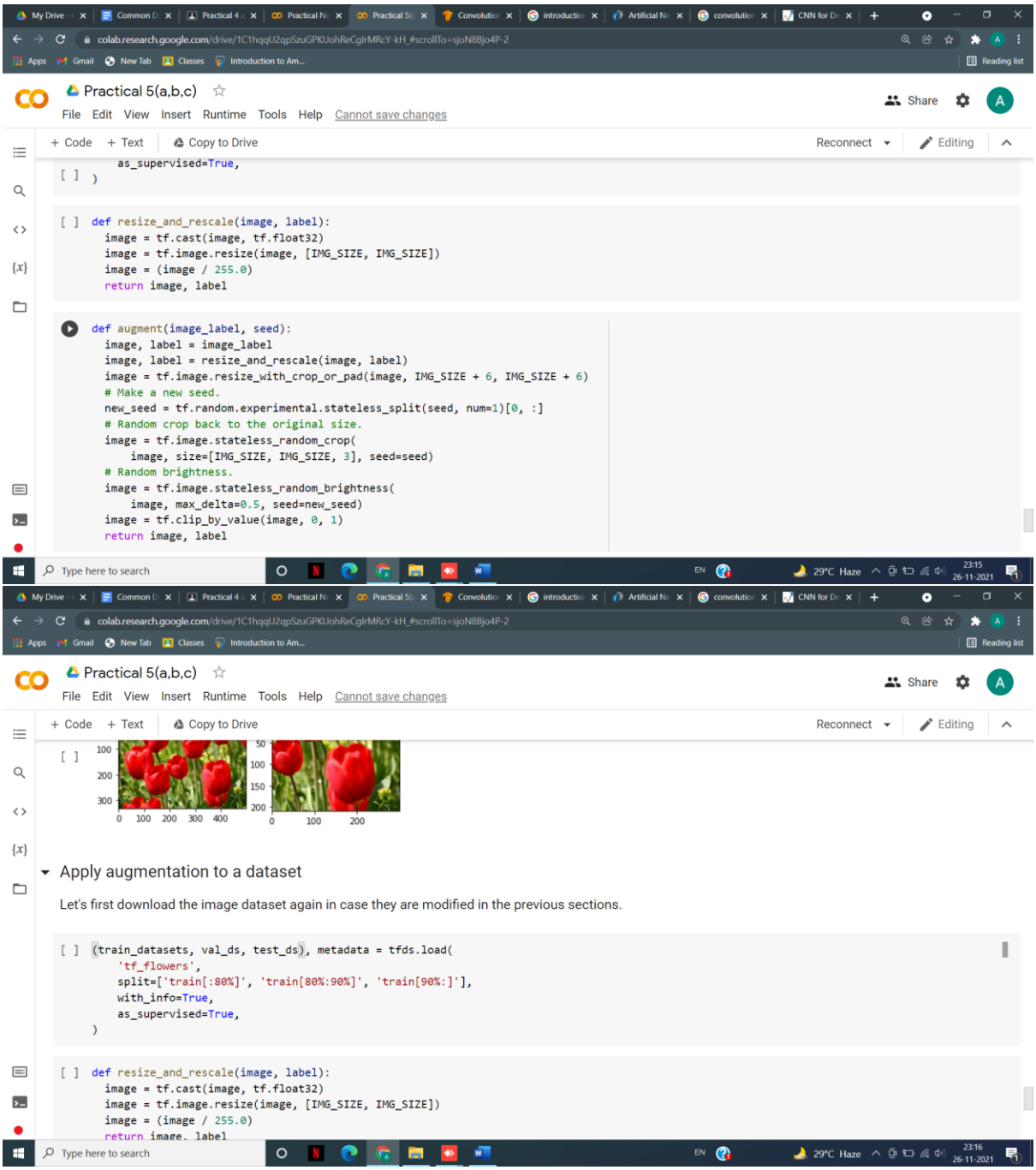

Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
Reconnect
Editing
[ ]
model = tf.keras.Sequential([
# Add the preprocessing layers you created earlier.
resize_and_rescale,
data_augmentation,
layers.Conv2D(16, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
Rest of your model.
])
2s completed at 22:12


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
Reconnect
Editing
[ ]
ds = ds.map(lambda x, y: (data_augmentation(x, training=True), y),
num_parallel_calls=AUTOTUNE)
# Use buffered prefetching on all datasets.
return ds.prefetch(buffer_size=AUTOTUNE)
[ ]
train_ds = prepare(train_ds, shuffle=True, augment=True)
val_ds = prepare(val_ds)
test_ds = prepare(test_ds)
model = tf.keras.Sequential([
layers.Conv2D(16, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(32, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Conv2D(64, 3, padding='same', activation='relu'),
layers.MaxPooling2D(),
layers.Flatten(),
layers.Dense(128, activation='relu'),
layers.Dense(num_classes)
])
2s completed at 22:12


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
Reconnect Editing
return x
def random_invert(factor=0.5):
return layers.Lambda(lambda x: random_invert_img(x, factor))
random_invert = random_invert()
plt.figure(figsize=(10, 10))
for i in range(9):
augmented_image = random_invert(image)
ax = plt.subplot(3, 3, i + 1)
plt.imshow(augmented_image[0].numpy().astype("uint8"))
plt.axis("off")
2s completed at 22:12
def __init__(self, factor=0.5, **kwargs):
super().__init__(**kwargs)
self.factor = factor
def call(self, x):
return random_invert_img(x)
_ = plt.imshow(RandomInvert()(image)[0])
0
50
100
150
200
250
300
0
100
200
300
400


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
Reconnect
Editing
Using tf.image
[ ] (train_ds, val_ds, test_ds), metadata = tfds.load(
'tf_flowers',
split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
with_info=True,
as_supervised=True,
)
[ ] image, label = next(iter(train_ds))
_ = plt.imshow(image)
_ = plt.title(get_label_name(label))
tulips
0
50
100
150
200
2s completed at 22:12
Type here to search
29°C Haze
23:14
26-11-2021


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
Reconnect Editing
seed = (i, 0) # tuple of size (2,)
stateless_random_brightness = tf.image.stateless_random_brightness(
image, max_delta=0.95, seed=seed)
visualize(image, stateless_random_brightness)
Original image
Augmented image
2s completed at 22:12
Randomly crop an image
Randomly crop image using tf.image.stateless_random_crop by providing target size and seed. The portion that gets cropped out of image is at a randomly chosen offset and is associated with the given seed.
for i in range(3):
seed = (i, 0) # tuple of size (2,)
stateless_random_crop = tf.image.stateless_random_crop(
image, size=[210, 300, 3], seed=seed)
visualize(image, stateless_random_crop)


Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
Reconnect
Editing
as_supervised=True,
)
def resize_and_rescale(image, label):
image = tf.cast(image, tf.float32)
image = tf.image.resize(image, [IMG_SIZE, IMG_SIZE])
image = (image / 255.0)
return image, label
def augment(image_label, seed):
image, label = image_label
image, label = resize_and_rescale(image, label)
image = tf.image.resize_with_crop_or_pad(image, IMG_SIZE + 6, IMG_SIZE + 6)
# Make a new seed.
new_seed = tf.random.experimental.stateless_split(seed, num=1)[0, :]
# Random crop back to the original size.
image = tf.image.stateless_random_crop(
image, size=[IMG_SIZE, IMG_SIZE, 3], seed=seed)
# Random brightness.
image = tf.image.stateless_random_brightness(
image, max_delta=0.5, seed=new_seed)
image = tf.clip_by_value(image, 0, 1)
return image, label
Type here to search
29°C Haze
23:15
26-11-2021

Practical 5(a,b,c)
File Edit View Insert Runtime Tools Help Cannot save changes
Share
+ Code + Text Copy to Drive
Reconnect
Editing
Apply augmentation to a dataset
Let's first download the image dataset again in case they are modified in the previous sections.
(train_datasets, val_ds, test_ds), metadata = tfds.load(
'tf_flowers',
split=['train[:80%]', 'train[80%:90%]', 'train[90%:]'],
with_info=True,
as_supervised=True,
)
def resize_and_rescale(image, label):
image = tf.cast(image, tf.float32)
image = tf.image.resize(image, [IMG_SIZE, IMG_SIZE])
image = (image / 255.0)
Type here to search
29°C Haze
23:16
26-11-2021

Practical No: 06

Aim: Building RNN.

What is RNN ?

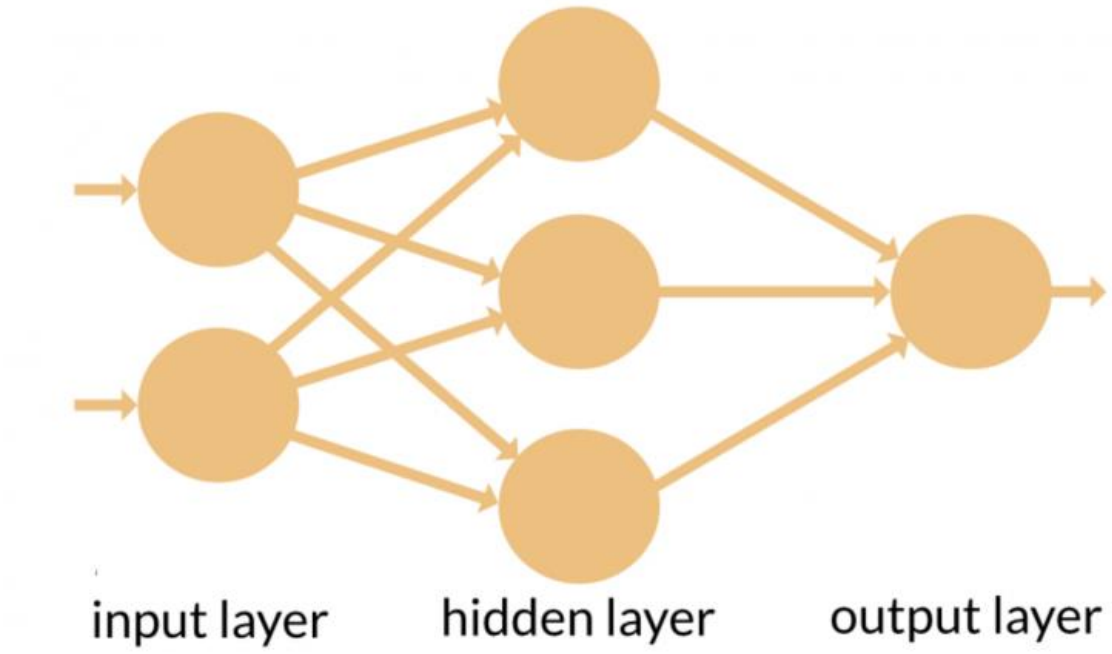
Ans. Recurrent neural networks (RNN) are the state of the art algorithm for sequential data and are used by Apple's Siri and and Google's voice search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data. It is one of the algorithms behind the scenes of the amazing achievements seen in deep learning over the past few years.

How Recurrent Neural Networks Work

To understand RNNs properly, you'll need a working knowledge of "normal" feed-forward neural networks and sequential data.

Sequential data is basically just ordered data in which related things follow each other. Examples are financial data or the DNA sequence. The most popular type of sequential data is perhaps time series data, which is just a series of data points that are listed in time order.

RNN VS. FEED-FORWARD NEURAL NETWORKS

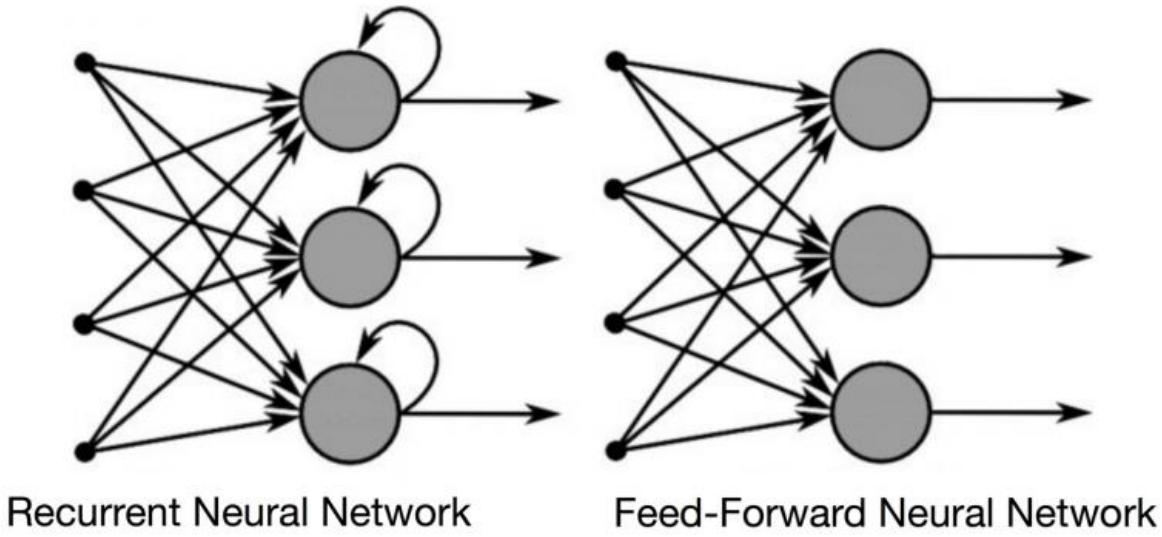


RNN's and feed-forward neural networks get their names from the way they channel information.
In a feed-forward neural network, the information only moves in one direction — from the input layer, through the hidden layers, to the output layer. The information moves straight through the network and never touches a node twice.

Feed-forward neural networks have no memory of the input they receive and are bad at predicting what's coming next. Because a feed-forward network only considers the current input, it has no notion of order in time. It simply can't remember anything about what happened in the past except its training.

In a RNN the information cycles through a loop. When it makes a decision, it considers the current input and also what it has learned from the inputs it received previously.

The two images below illustrate the difference in information flow between a RNN and a feed-forward neural network.



A usual RNN has a short-term memory. In combination with a LSTM they also have a long-term memory (more on that later).

Another good way to illustrate the concept of a recurrent neural network's memory is to explain it with an example:

Imagine you have a normal feed-forward neural network and give it the word "neuron" as an input and it processes the word character by character. By the time it reaches the character "r," it has already forgotten about "n," "e" and "u," which makes it almost impossible for this type of neural network to predict which character would come next.

A recurrent neural network, however, is able to remember those characters because of its internal memory. It produces output, copies that output and loops it back into the network.

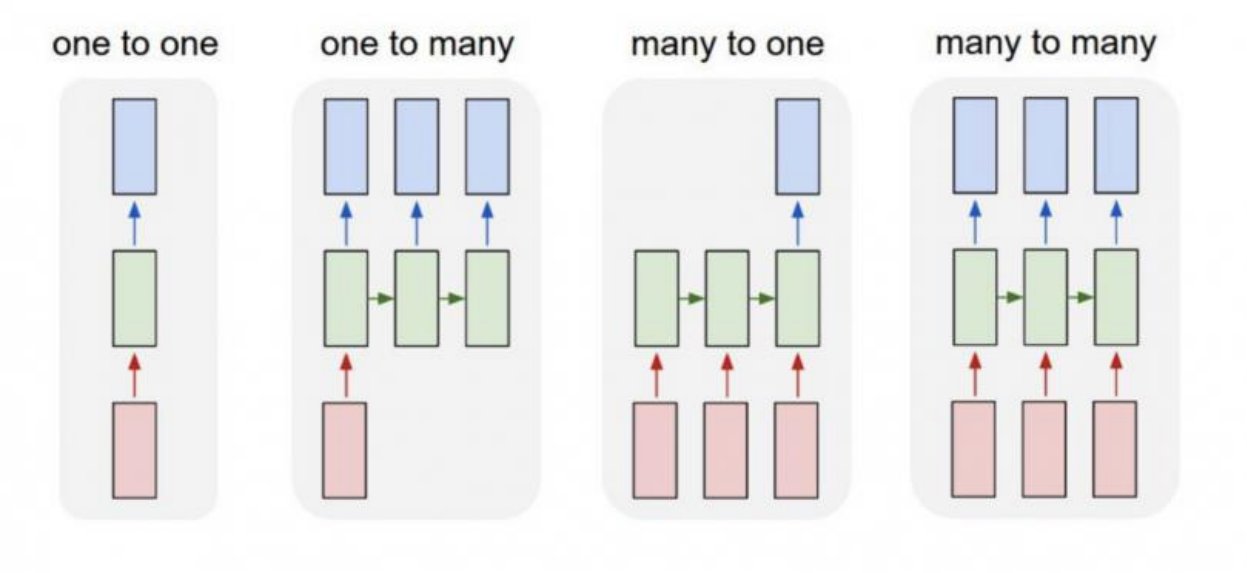
Simply put: recurrent neural networks add the immediate past to the present.
Therefore, a RNN has two inputs: the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't.

A feed-forward neural network assigns, like all other deep learning algorithms, a weight matrix to its inputs and then produces the output. Note that RNNs apply weights to the current and also to the previous input. Furthermore, a recurrent neural network will also tweak the weights for both through gradient descent and backpropagation through time (BPTT).

TYPES OF RNNS

- One to One
- One to Many
- Many to One
- Many to Many

Also note that while feed-forward neural networks map one input to one output, RNNs can map one to many, many to many (translation) and many to one (classifying a voice).



Implementation of RNN:

https://colab.research.google.com/drive/1w9mit2ABNzawqSw1zLdySgf_tLlDF4qv

```
#setup
import numpy
import tensorflow
from tensorflow import keras
from tensorflow.keras import layers


#Here is a simple example of a Sequential model that processes sequences of integers, embeds each integer i
nto a 64-dimensional vector, then processes the sequence of vectors using a LSTM layer.
model=keras.Sequential()
# Add an Embedding layer expecting input vocab of size 1000, and
# output embedding dimension of size 64.
model.add(layers.Embedding(input_dim=1000, output_dim=64))
#add a lstm layer with 128 internal units
model.add(layers.LSTM(128))
#add a dense layer with 10 units
model.add(layers.Dense(10))
model.summary()


model = keras.Sequential()
model.add(layers.Embedding(input_dim=1000, output_dim=64))

# The output of GRU will be a 3D tensor of shape (batch_size, timesteps, 256)
model.add(layers.GRU(256, return_sequences=True))

# The output of SimpleRNN will be a 2D tensor of shape (batch_size, 128)
model.add(layers.SimpleRNN(128))

model.add(layers.Dense(10))

model.summary()



encoder_vocab = 1000
decoder_vocab = 2000

encoder_input = layers.Input(shape=(None,))
encoder_embedded = layers.Embedding(input_dim=encoder_vocab, output_dim=64)(
```

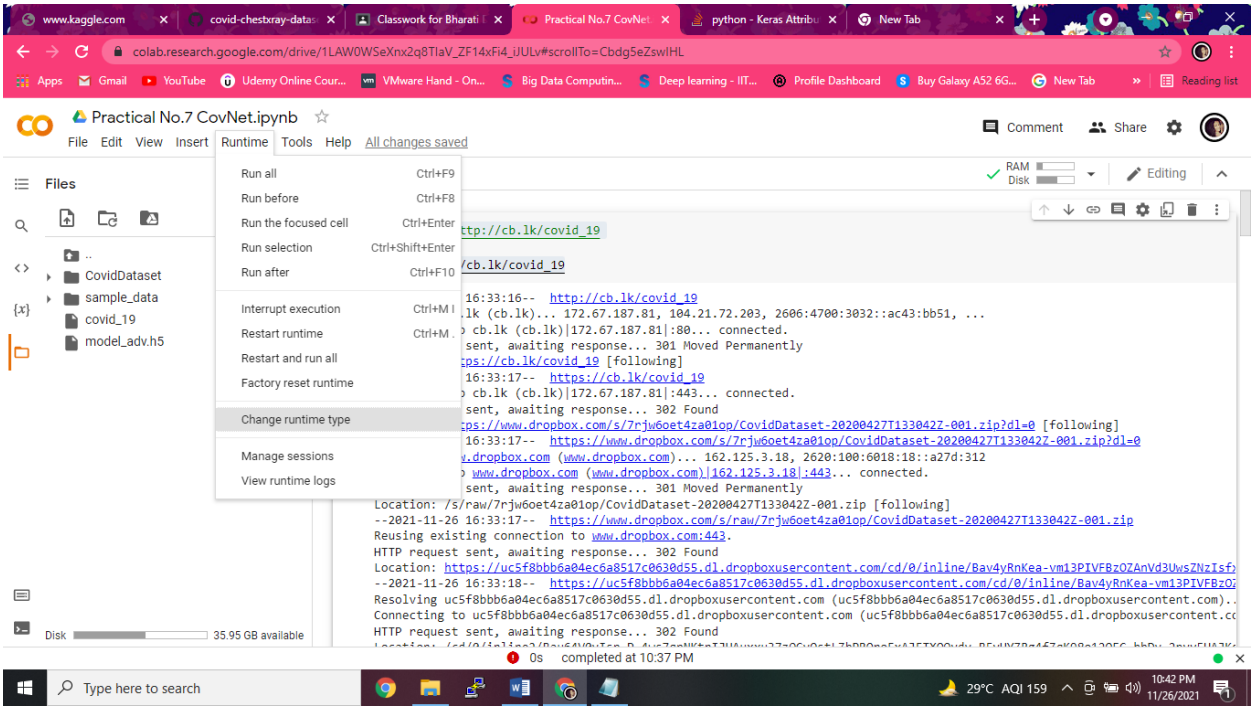
```
    encoder_input
)

# Return states in addition to output
output, state_h, state_c = layers.LSTM(64, return_state=True, name="encoder")(
    encoder_embedded
)
encoder_state = [state_h, state_c]

decoder_input = layers.Input(shape=(None,))
decoder_embedded = layers.Embedding(input_dim=decoder_vocab, output_dim=64)(
    decoder_input
)

# Pass the 2 states to a new LSTM layer, as initial state
decoder_output = layers.LSTM(64, name="decoder")(
    decoder_embedded, initial_state=encoder_state
)
output = layers.Dense(10)(decoder_output)

model = keras.Model([encoder_input, decoder_input], output)
model.summary()
```

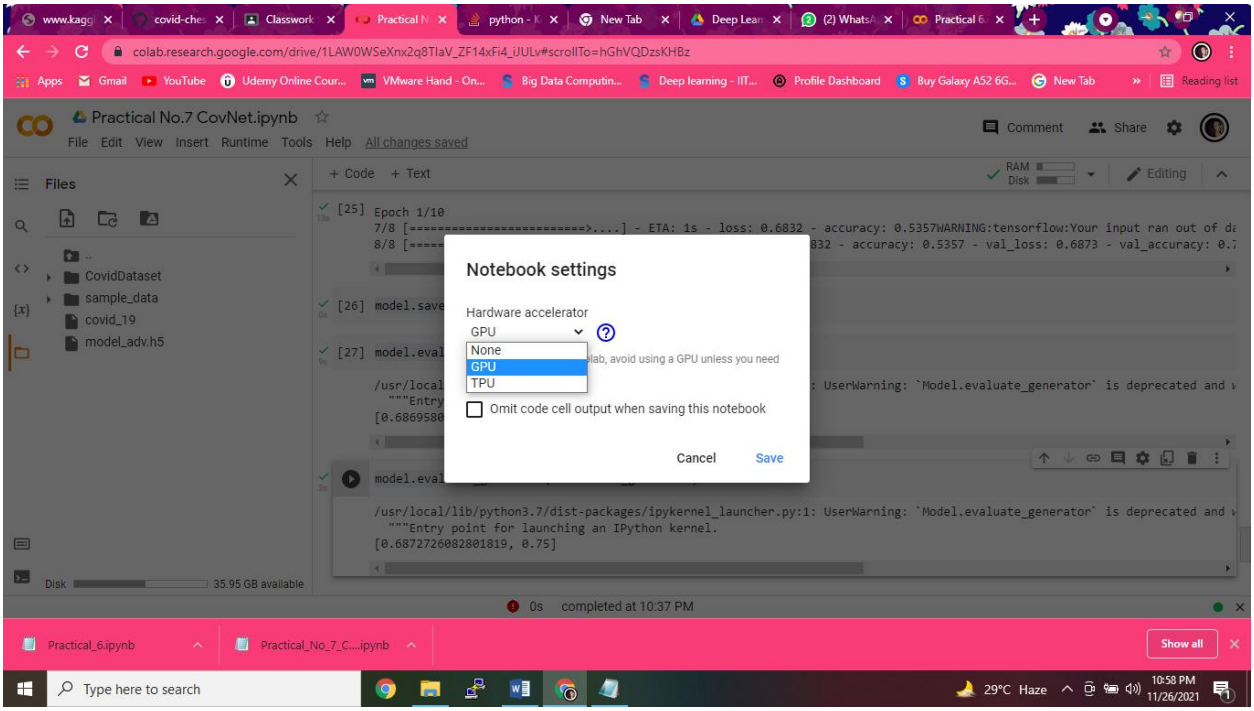
Practical No: 7

Aim: Build Deep Learning model using CovNet.

Implementation :

https://colab.research.google.com/drive/1LAW0WSeXnx2q8TIaV_ZF14xFi4_iJULv?usp=sharing





Code :

```
#Dataset : http://cb.lk/covid_19

!wget http://cb.lk/covid_19


!unzip covid_19


TRAIN_PATH = "CovidDataset/Train"
VAL_PATH = "Covid_dataset/Test"


import numpy as np
import matplotlib.pyplot as plt
import keras
from keras.layers import *
from keras.models import *
from keras.preprocessing import image


# CNN Based Model in Keras
```

```
model = Sequential()
model.add(Conv2D(32,kernel_size=(3,3),activation='relu',input_shape=(224,224,3)))
model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(64,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Conv2D(128,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(64,activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1,activation='sigmoid'))

model.compile(loss=keras.losses.binary_crossentropy,optimizer='adam',metrics=['accuracy'])
```

```
# Train from Scratch
train_datagen = image.ImageDataGenerator(
    rescale = 1./255,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
)
test_dataset = image.ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
    'CovidDataset/Train',
    target_size = (224,224),
    batch_size = 32,
    class_mode = 'binary'
)
```

```
train_generator.class_indices
```

```
validation_generator = test_dataset.flow_from_directory(
   'CovidDataset/Val',
   target_size = (224,224),
   batch_size = 32,
   class_mode = 'binary'
)


hist = model.fit_generator(
   train_generator,
   steps_per_epoch=8,
   epochs = 10,
   validation_data = validation_generator,
   validation_steps=2
)


model.save("model_adv.h5")


model.evaluate_generator(train_generator)


model.evaluate_generator(validation_generator)
```

Practical No: 8

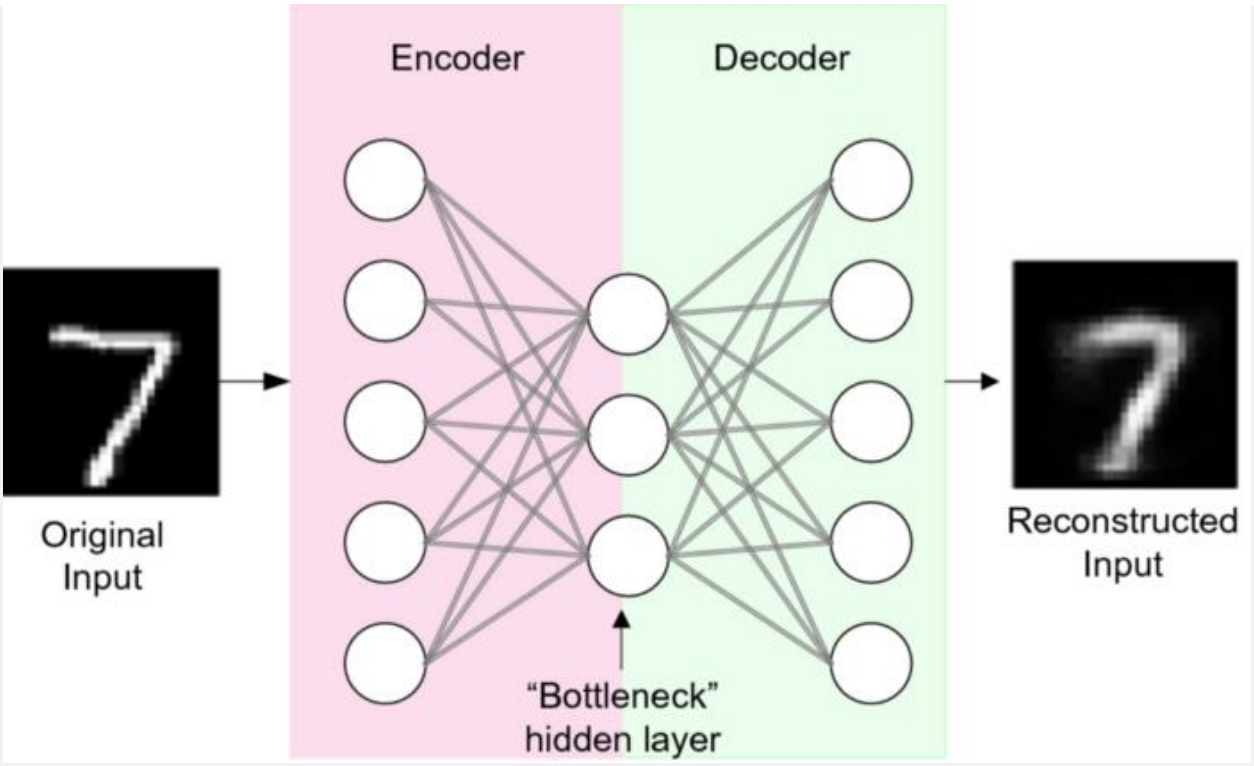
Aim: Implementing a single Auto encoder based on fully connected layer

What is an Autoencoder ?

Neural networks exist in all shapes and sizes, and are often characterized by their **input** and **output data type.** For instance, image classifiers are built with *Convolutional Neural Networks.* They take **images** as inputs, and output a **probability distribution of the classes**.
Autoencoders (AE) are a family of neural networks for which **the input is the same as the output***. They work by compressing the input into a *latent-space representation,* and then reconstructing the output from this representation.

Auto-encoder is a **complex mathematical model** which trains on **unlabeled** as well as unclassified **data** and is used to map the **input data to** another compressed **feature representation** and from that feature representation **reconstructing back the input data.**

Auto-encoder is a **complex mathematical model** which trains on **unlabeled** as well as unclassified **data** and is used to map the **input data to** another compressed **feature representation** and from that feature representation **reconstructing back the input data.**



What is fully connected layer?

Fully Connected layers in a neural networks are those layers where all the inputs from one layer are connected to every **activation unit** of the next layer. In most popular machine learning models, the last few layers are full connected layers which **compiles the data extracted** by previous layers to form the final output. It is the second most time consuming layer second to Convolution Layer.

Code:

https://colab.research.google.com/drive/1fNU9YBf5aYYEdhZJATBF5c22U-ilZE2O?usp=sharing

Single fully-connected neural layer as encoder and as decoder:

```
import keras
from keras import layers

# This is the size of our encoded representations
encoding_dim = 32  # 32 floats -> compression of factor 24.5, assuming the input is 784 floats
```

```
# This is our input image
input_img = keras.Input(shape=(784,))
# "encoded" is the encoded representation of the input
encoded = layers.Dense(encoding_dim, activation='relu')(input_img)
# "decoded" is the lossy reconstruction of the input
decoded = layers.Dense(784, activation='sigmoid')(encoded)

# This model maps an input to its reconstruction
autoencoder = keras.Model(input_img, decoded)
```

A separate encoder model:

```
# This model maps an input to its encoded representation
encoder = keras.Model(input_img, encoded)
```

decoder model:

```
# This is our encoded (32-dimensional) input
encoded_input = keras.Input(shape=(encoding_dim,))
# Retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]
# Create the decoder model
decoder = keras.Model(encoded_input, decoder_layer(encoded_input))
```

Train our autoencoder to reconstruct MNIST digits.

we'll configure our model to use a per-pixel binary crossentropy loss, and the Adam optimizer:

```
autoencoder.compile(optimizer='adam', loss='binary_crossentropy')
```

Let's prepare our input data. We're using MNIST digits, and we're discarding the labels (since we're only interested in encoding/decoding the input images).

```
from keras.datasets import mnist
import numpy as np
(x_train, _), (x_test, _) = mnist.load_data()
```

We will normalize all values between 0 and 1 and we will flatten the 28x28 images into vectors of size 784.

```
x_train = x_train.astype('float32') / 255.
x_test = x_test.astype('float32') / 255.
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))
print(x_train.shape)
```

```
print(x_test.shape)
```

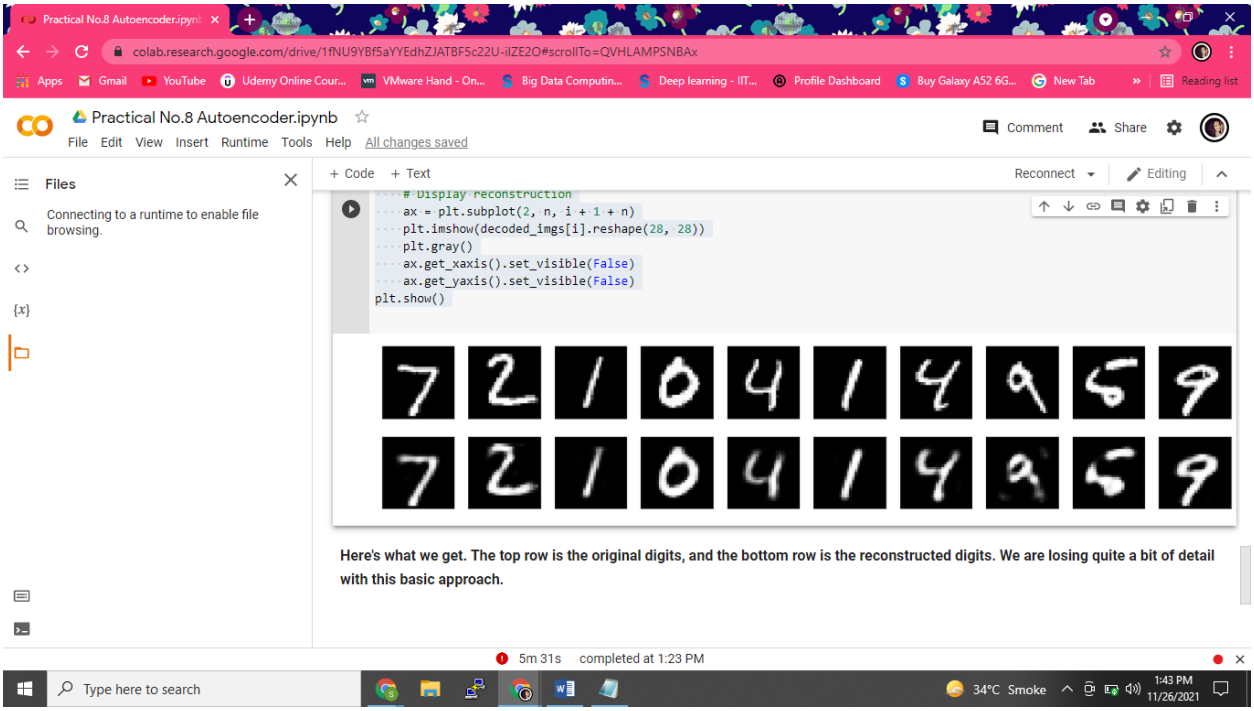
Now let's train our autoencoder for 50 epochs:

```
autoencoder.fit(x_train, x_train,
                epochs=50,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

After 50 epochs, the autoencoder seems to reach a stable train/validation loss value of about 0.09. We can try to visualize the reconstructed inputs and the encoded representations. We will use Matplotlib.

```
# Encode and decode some digits
# Note that we take them from the *test* set
encoded_imgs = encoder.predict(x_test)
decoded_imgs = decoder.predict(encoded_imgs)
```

```
# Use Matplotlib (don't ask)
import matplotlib.pyplot as plt

n = 10  # How many digits we will display
plt.figure(figsize=(20, 4))
for i in range(n):
    # Display original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)

    # Display reconstruction
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28))
    plt.gray()
    ax.get_xaxis().set_visible(False)
    ax.get_yaxis().set_visible(False)
plt.show()
```



Google Colab Link :-

https://drive.google.com/drive/folders/1raAACU-jPioJy8CNO9OwvaYWlLjljWXQ?usp=sharing