**Name: -** Shivam Borse

**Roll No.: -** BEAD20119

**Subject: -** CL IV (Deep Learning)

**Assignment No. 04**

**Problem Statement:** Design and implement CNN for image classification.

a) Select a suitable image classification dataset (medical engineering, agricultural, etc.).

b) Optimized with different hyper-parameters including learning rate, filter size, no. of layers, optimizers,

dropouts, etc.

**Code-**

import tensorflow as tf

from tensorflow.keras import datasets, layers, models

from tensorflow.keras.optimizers import Adam

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

# Load and preprocess the MNIST dataset

(train\_images, train\_labels), (test\_images, test\_labels) = datasets.mnist.load\_data()

train\_images, test\_images = train\_images / 255.0, test\_images / 255.0

# Add channel dimension to the images

train\_images = train\_images.reshape((60000, 28, 28, 1))

test\_images = test\_images.reshape((10000, 28, 28, 1))

# Split the dataset into training and validation sets

train\_images, val\_images, train\_labels, val\_labels = train\_test\_split( train\_images,

train\_labels, test\_size=0.1, random\_state=42)

# Data augmentation for training images

datagen = ImageDataGenerator(rotation\_range=10, zoom\_range=0.1, width\_shift\_range=0.1,

height\_shift\_range=0.1)

datagen.fit(train\_images)

# Create a CNN model with hyperparameter tuning and regularization

model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)))

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(128, (3, 3), activation='relu'))

model.add(layers.Flatten())

model.add(layers.Dropout(0.5))

model.add(layers.Dense(128, activation='relu'))

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

# Compile the model

model.compile(optimizer=Adam(learning\_rate=0.001), loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Train the model with data augmentation

history = model.fit(datagen.flow(train\_images, train\_labels, batch\_size=64),

                    epochs=20, validation\_data=(val\_images, val\_labels)) # Evaluate the

#model on the test set

test\_loss, test\_acc = model.evaluate(test\_images, test\_labels)

print(f"Test Accuracy: {test\_acc}")

# Plot training history

plt.plot(history.history['accuracy'], label='Train Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

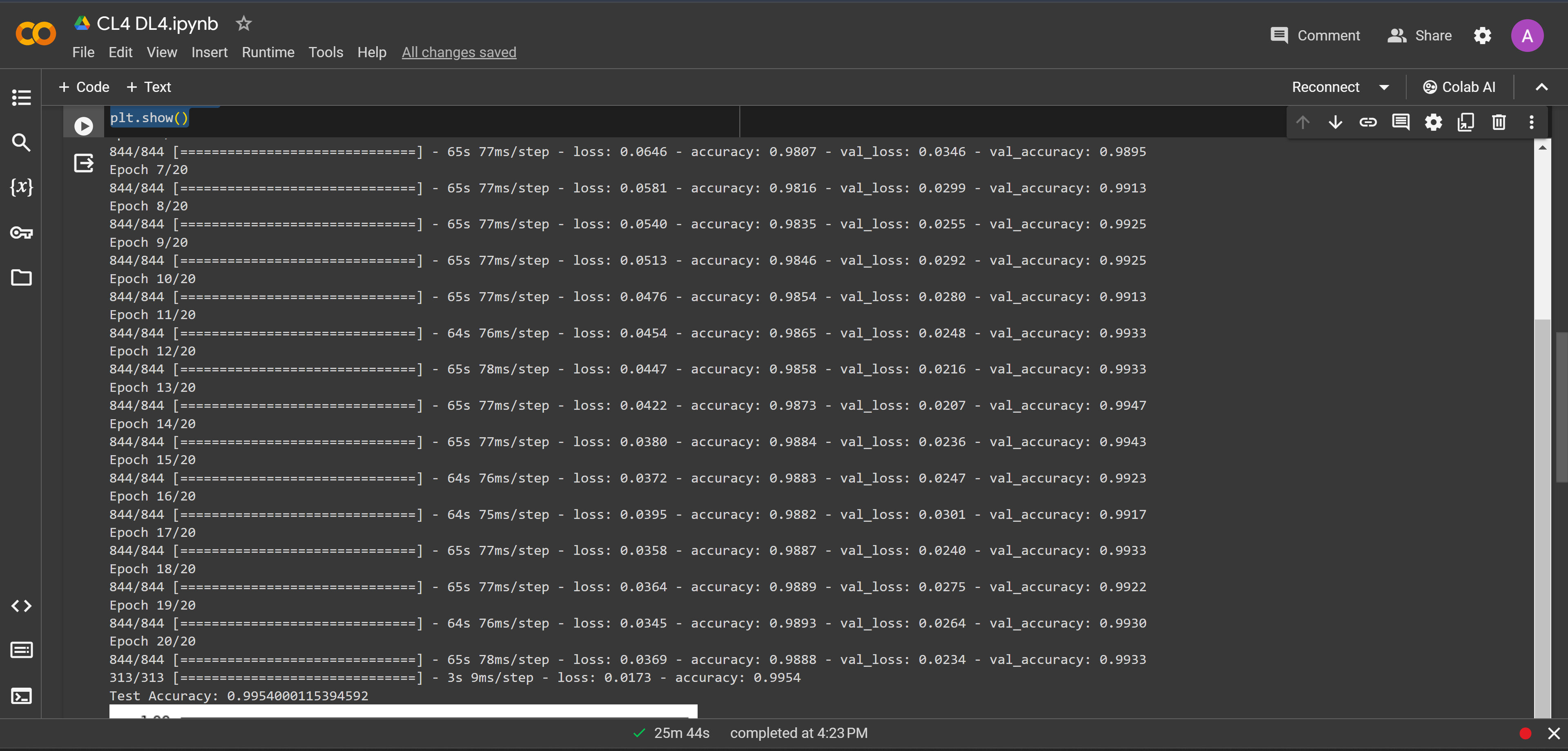
plt.xlabel('Epoch')

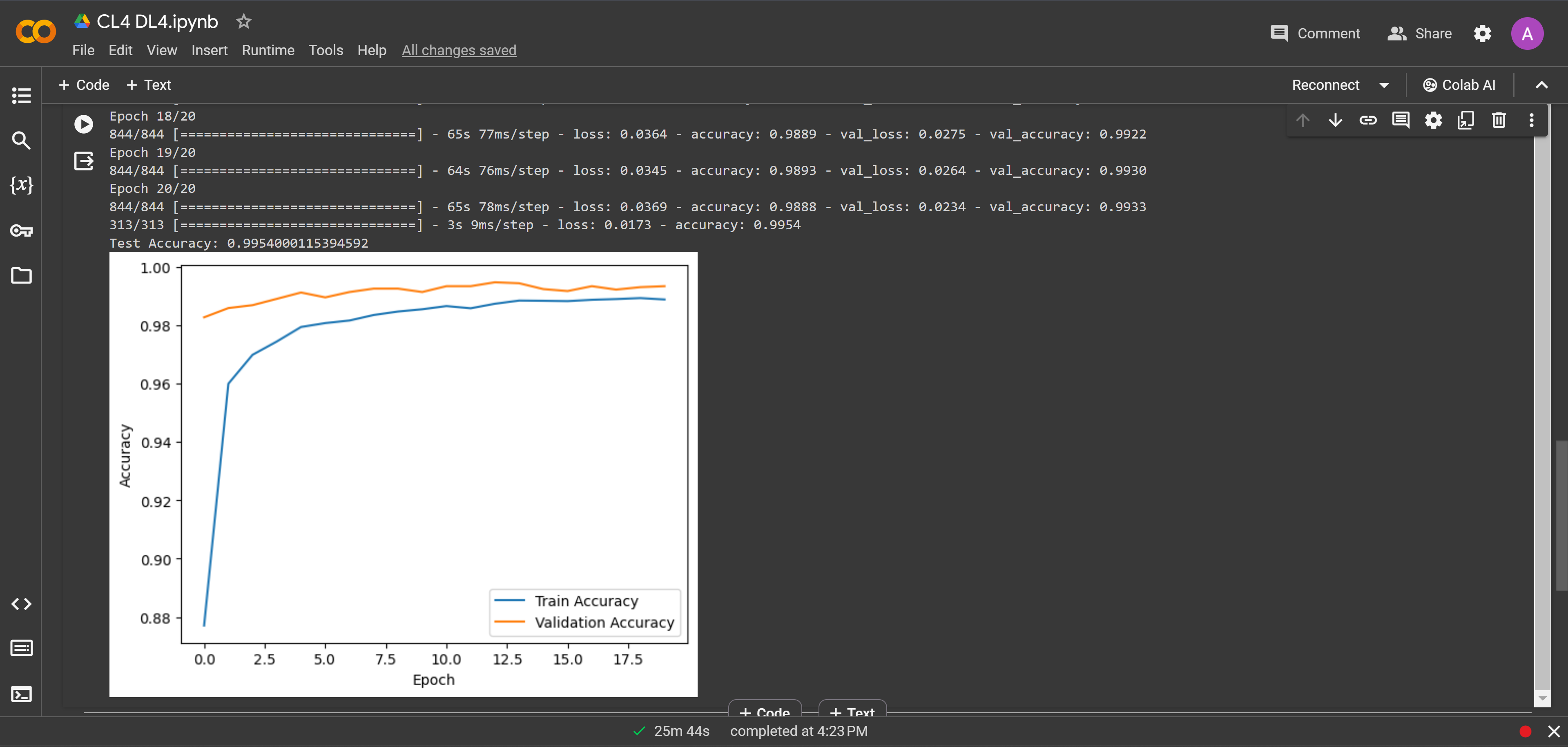
plt.ylabel('Accuracy')

plt.legend()

plt.show()

**Output-**

****

****

**Name: -** Shivam Borse

**Roll No.: -** BEAD20119

**Subject: -** CL IV (Deep Learning)

**Assignment No. 05**

**Problem Statement:** Design and implement Deep Convolutional GAN to generate images of faces/digits froma set of given images.

**Code-**

from keras.models import Model

from keras.layers import Input,Dense

import numpy as np

import pandas as pd

import keras.backend as K

import matplotlib.pyplot as plt

from keras import preprocessing

from keras.models import Sequential

from keras.layers import Conv2D,Dropout,Dense,Flatten,Conv2DTranspose,BatchNormalization,LeakyReLU,Reshape

import tensorflow as tf

from keras.layers import \*

from keras.datasets import fashion\_mnist

(train\_x, train\_y), (val\_x, val\_y) = fashion\_mnist.load\_data()

train\_x = train\_x/255.

val\_x = val\_x/255.

train\_x=train\_x.reshape(-1,28,28,1)

print(train\_x.shape)

#train\_x = train\_x.reshape(-1, 784)

#val\_x = val\_x.reshape(-1, 784)

fig,axe=plt.subplots(2,2)

idx = 0

for i in range(2):

  for j in range(2):

    axe[i,j].imshow(train\_x[idx].reshape(28,28),cmap='gray')

    idx+=1

    train\_x = train\_x\*2 - 1

    print(train\_x.max(),train\_x.min())

generator = Sequential()

generator.add(Dense(512,input\_shape=[100]))

generator.add(LeakyReLU(alpha=0.2))

generator.add(BatchNormalization(momentum=0.8))

generator.add(Dense(256))

generator.add(LeakyReLU(alpha=0.2))

generator.add(BatchNormalization(momentum=0.8))

generator.add(Dense(128))

generator.add(LeakyReLU(alpha=0.2))

generator.add(BatchNormalization(momentum=0.8))

generator.add(Dense(784))

generator.add(Reshape([28,28,1]))

generator.summary()

discriminator = Sequential()

discriminator.add(Dense(1,input\_shape=[28,28,1]))

discriminator.add(Flatten())

discriminator.add(Dense(256))

discriminator.add(LeakyReLU(alpha=0.2))

discriminator.add(Dropout(0.5))

discriminator.add(Dense(128))

discriminator.add(LeakyReLU(alpha=0.2))

discriminator.add(Dropout(0.5))

discriminator.add(Dense(64))

discriminator.add(LeakyReLU(alpha=0.2))

discriminator.add(Dropout(0.5))

discriminator.add(Dense(1,activation='sigmoid'))

discriminator.summary()

GAN =Sequential([generator,discriminator])

discriminator.compile(optimizer='adam',loss='binary\_crossentropy')

discriminator.trainable = False

GAN.compile(optimizer='adam',loss='binary\_crossentropy')

GAN.summary()

epochs = 30

batch\_size = 100

noise\_shape=100

with tf.device('/gpu:0'):

  for epoch in range(epochs):

    print(f"Currently on Epoch {epoch+1}")

    for i in range(train\_x.shape[0]//batch\_size):

      if (i+1)%100 == 0:

        print(f"\tCurrently on batch number {i+1} of {train\_x.shape[0]//batch\_size}")

      noise=np.random.normal(size=[batch\_size,noise\_shape])

      gen\_image = generator.predict\_on\_batch(noise)

      train\_dataset = train\_x[i\*batch\_size:(i+1)\*batch\_size]

      #training discriminator on real images

      train\_label=np.ones(shape=(batch\_size,1))

      train\_label=np.ones((batch\_size, 1))

      discriminator.trainable = True

      #train\_dataset=train\_x[idx]

      train\_dataset = train\_x[i\*batch\_size:(i+1)\*batch\_size]

      d\_loss\_real=discriminator.train\_on\_batch(train\_dataset,train\_label) #training discriminator on fake images

      train\_label=np.zeros(shape=(batch\_size,1))

      d\_loss\_fake=discriminator.train\_on\_batch(gen\_image,train\_label)

      #training generator

      noise=np.random.normal(size=[batch\_size,noise\_shape])

      train\_label=np.ones(shape=(batch\_size,1))

      discriminator.trainable = False

      d\_g\_loss\_batch =GAN.train\_on\_batch(noise, train\_label)

    #plotting generated images at the start and then after every 10 epoch

    if epoch % 10 == 0:samples = 10

    x\_fake = generator.predict(np.random.normal(loc=0, scale=1, size=(samples, 100)))

    for k in range(samples):

      plt.subplot(2, 5, k+1)

      plt.imshow(x\_fake[k].reshape(28, 28), cmap='gray')

      plt.xticks([])

      plt.yticks([])

    plt.tight\_layout()

    plt.show()

  print('Training is complete')

noise=np.random.normal(size=[10,noise\_shape])

gen\_image = generator.predict(noise)

plt.imshow(noise)

plt.title('How the noise looks')

fig,axe=plt.subplots(2,5)

fig.suptitle('Generated Images from Noise using GANs')

idx=0

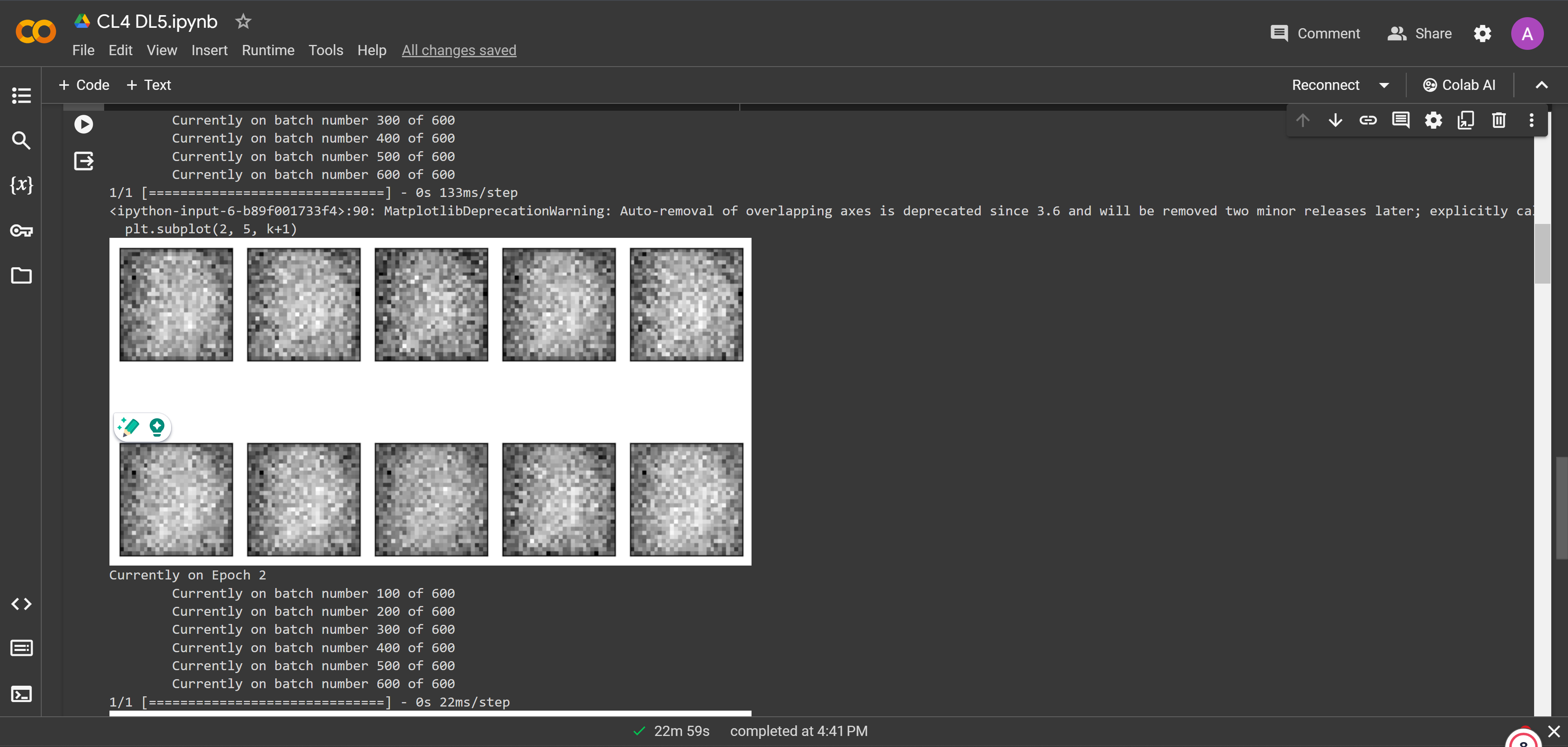
for i in range(2):

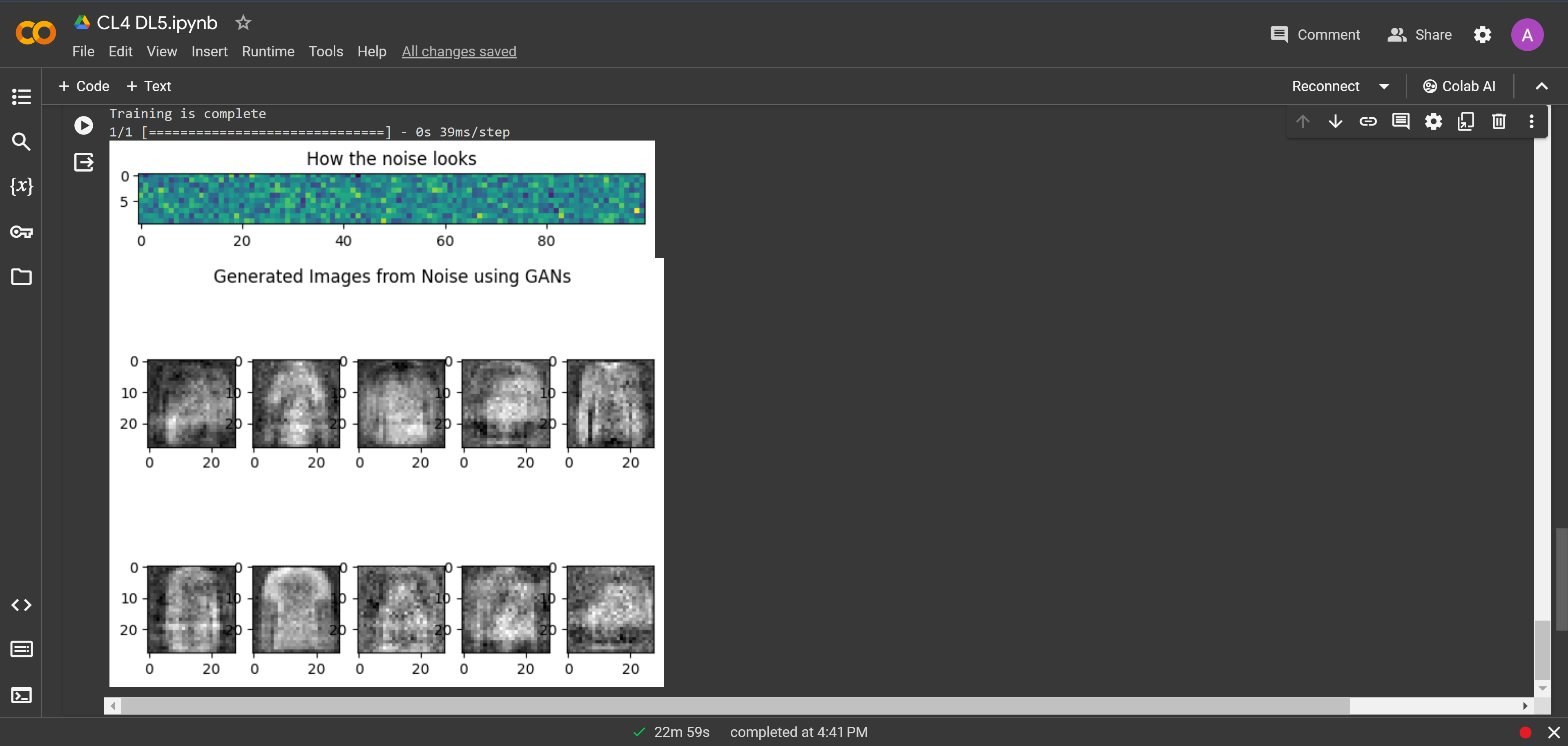
  for j in range(5):

    axe[i,j].imshow(gen\_image[idx].reshape(28,28),cmap='gray')

    idx+=1

**Output-**

****

****

**Name: -** Shivam Borse

**Roll No.: -** BEAD20119

**Subject: -** CL IV (Deep Learning)

**Assignment No. 06**

**Problem Statement:** Perform sentiment analysis with a recurrent neural networks RNN

**Code-**

import tensorflow as tf

from tensorflow.keras.datasets import imdb

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, LSTM, Dense

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Set the parameters

max\_features = 10000 # Number of words to consider as features

maxlen = 100 # Cut texts after this number of words (among top max\_features most common words)

batch\_size = 32

# Load the IMDB dataset

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=max\_features)

# Pad sequences to have a consistent length for the input to the RNN

x\_train = pad\_sequences(x\_train, maxlen=maxlen)

x\_test = pad\_sequences(x\_test, maxlen=maxlen)

# Build the RNN model with LSTM

model = Sequential()

model.add(Embedding(max\_features, 128))

model.add(LSTM(64, dropout=0.2,

recurrent\_dropout=0.2))

model.add(Dense(1, activation='sigmoid'))

# Compile the model

model.compile(loss='binary\_crossentropy',

optimizer='adam', metrics=['accuracy'])

# Train the model

model.fit(x\_train,y\_train,batch\_size=batch\_size, epochs=5,validation\_data=(x\_test, y\_test))

# Evaluate the model

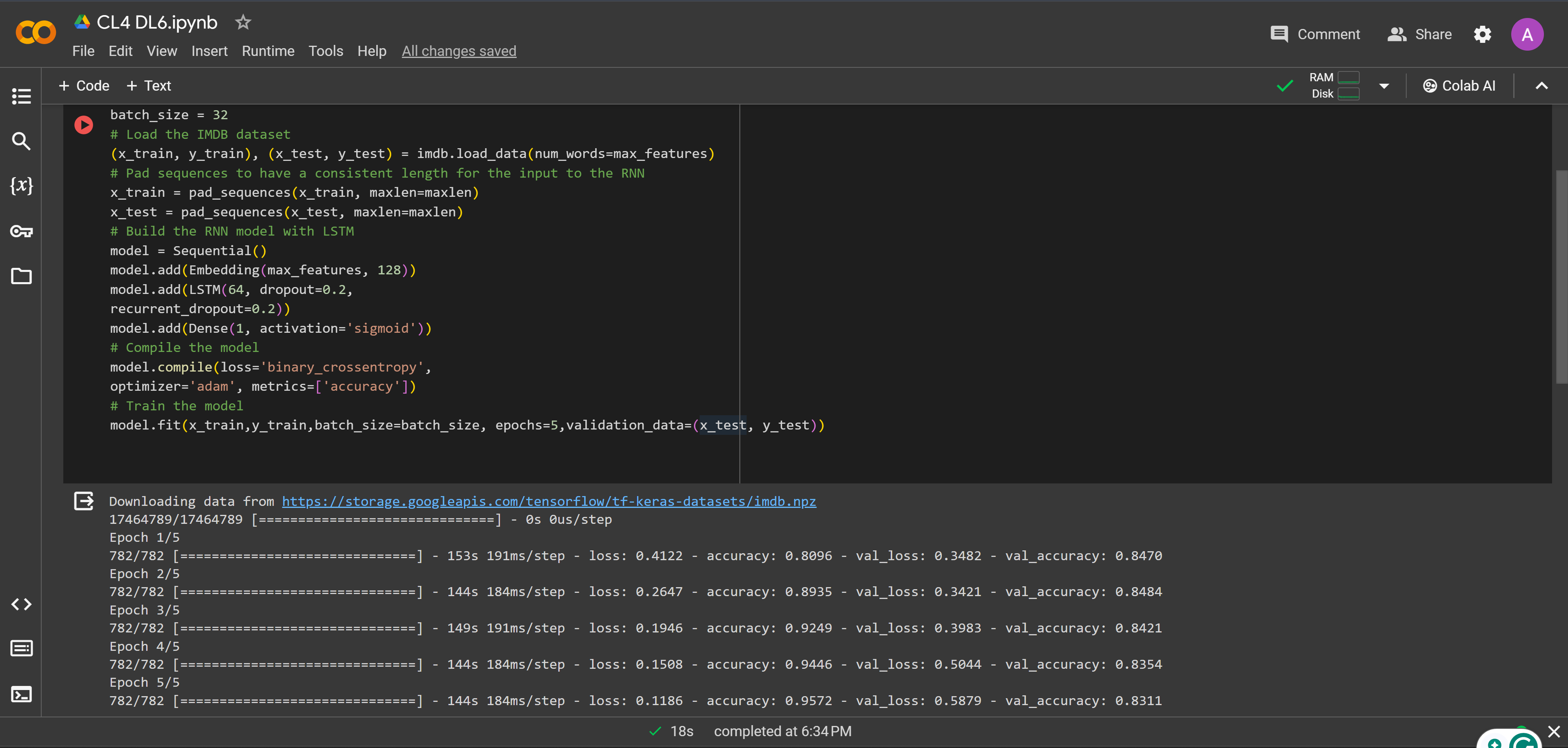
score,

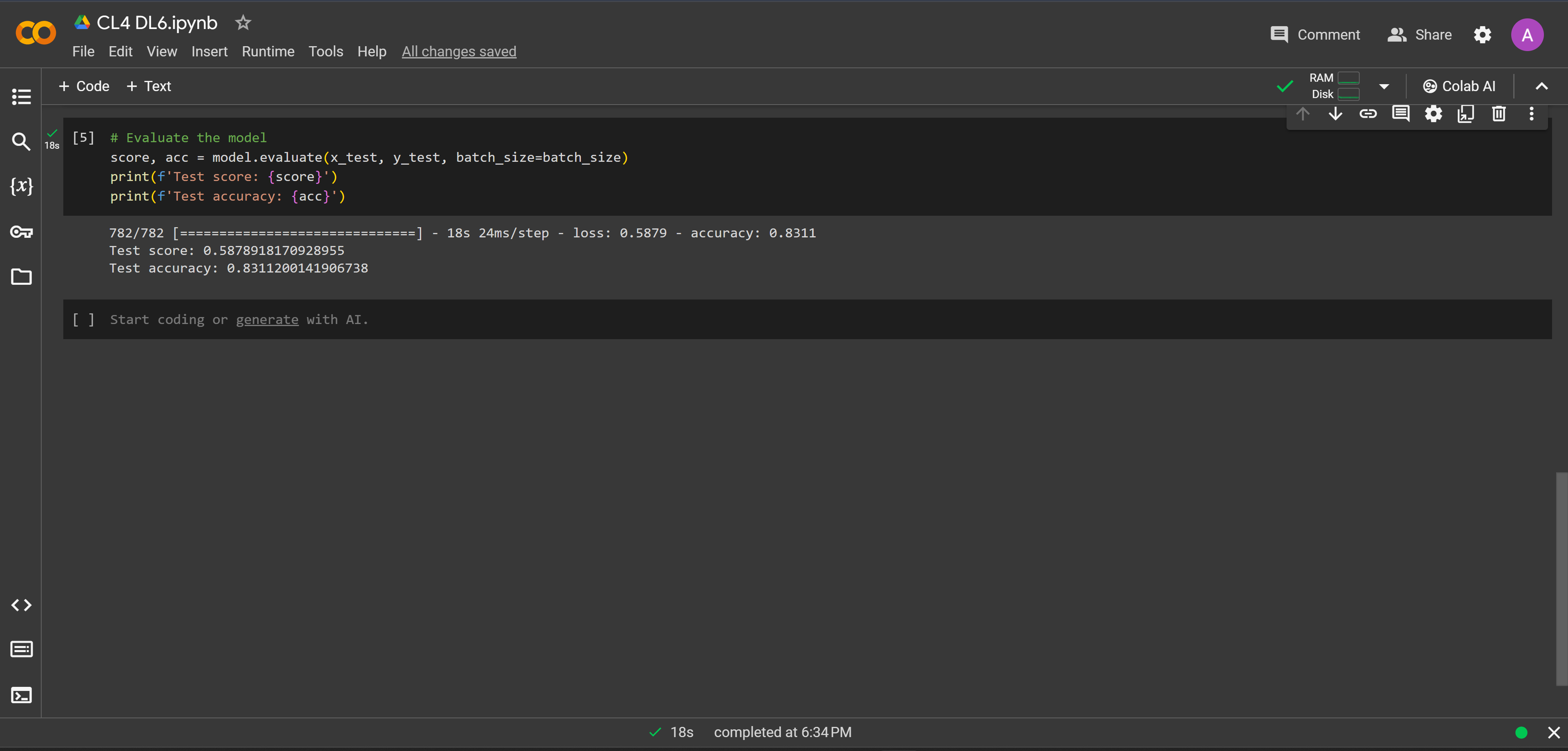
acc = model.evaluate(x\_test, y\_test, batch\_size=batch\_size)

print(f'Test score: {score}')

print(f'Test accuracy: {acc}')

**Output-**

****

****