

**RAMNIRANJAN JHUNJHUNWALA COLLEGE**

**GHATKOPAR (W), MUMBAI - 400 086**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**2021 - 2022**

**M.Sc.( I.T.) SEM II**

**Deep Learning**

**Name: Surekha Rajbhar**

**Roll No.: 13**



**CERTIFICATE**

This is to certify that Ms. **Surekha Omprakash Rajbhar** with Roll No. **13** 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 -III.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Internal Examiner External Examiner

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_

Head of Department College Seal

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No** | **Title** | **Date** | **Page No.** |
| 1. | Matrix multiplication, Eigen Vectors, Eigen Value Computation Using TensorFlow |  | 4 |
| 2. | Deep Forward Network for XOR |  | 7 |
| 3A | Classification Using DNN |  | 9 |
| 3B | Binary Classification Using MLP |  | 11 |
| 4A | Feed Forward Neural Network |  | 13 |
| 4B | Predicting the Probability of the Class |  | 15 |
| 5A | CNN for CIFAR10 Images |  | 17 |
| 5B | Image Classification |  | 21 |
| 5C | Data Augmentation |  | 26 |
| 6 | Building RNN Using Single Neuron |  | 29 |
| 7 | Using CovNet to build Deep Learning model |  | 32 |
| 8 | Implementing a single Autoencoder based on fully connected layer |  | 36 |

**Practical No. 1**

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

**Perceptron** is an algorithm for supervised learning of binary classifiers. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time.

Perceptrons are trained on examples of desired behavior. The desired behavior can be summarized by a set of input, output pairs.

**p1t1, p2t2, p3t3, p4t4...pntn**

where p is an input to the network and t is the corresponding correct (target) output. The objective is to reduce the error e, which is the difference t-a between the neuron response a, and the target vector t. The perceptron learning rule calculates desired changes to the perceptron's weights and biases given an input vector p, and the associated error e. The target vector t must contain values of either -1 or 1, as perceptrons (with signum activation functions) can only output such values.

As each iteration goes on, the perceptron has a better chance of producing the correct outputs. The perceptron rule is proven to converge on a solution in a finite number of iterations if a solution exists.

If a bias is not used, learning algorithm works to find a solution by altering only the weight vector w to point toward input vectors to be classified as 1, and away from vectors to be classified as -1. This results in a decision boundary that is perpendicular to w, and which properly classifies the input vectors.

There are three conditions that can occur for a single neuron once an input vector p is presented and the network's response a is calculated:

**CASE 1.** If an input vector is presented and the output of the neuron is correct (a = t, and e = t - a = 0), then the weight vector w is not altered.

**CASE 2.** If the neuron output is -1 and should have been 1 (a = -1 and t = 1, and e = t - a = 2), the input vector p is added to the weight vector w. This makes the weight vector point closer to the input vector, increasing the chance that the input vector will be classified as a 1 in the future.

**CASE 3.** If the neuron output is 1 and should have been -1 (a = 1 and t = -1, and e = t - a = -2), the input vector p is subtracted from the weight vector w. This makes the weight vector point farther away from the input vector, increasing the chance that the input vector is classified as a - 1 in the future

The perceptron learning rule can be written more succinctly in terms of the error e = t - a, and the change to be made to the weight vector w:

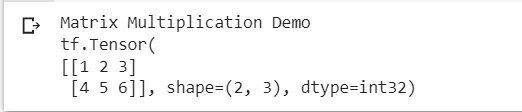
Code

import tensorflow as tf

print('Matrix Multiplication Demo')

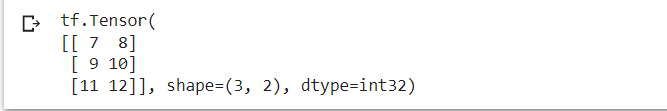
x=tf.constant([1,2,3,4,5,6], shape=[2,3])

print(x)



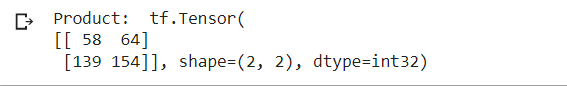
y=tf.constant([7,8,9,10,11,12], shape=[3,2])

print(y)



z=tf.matmul(x,y)

print("Product: ", z)



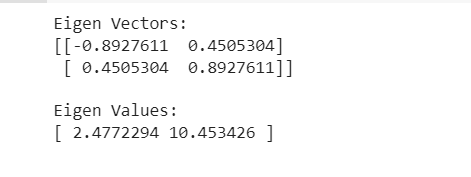
e\_matrix\_A=tf.random.uniform([2,2], minval=3, maxval=10, dtype=tf.float32, name="matrixA")

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))



**Practical 2**

**Deep Forward Network for XOR**

Solving XOR problem using Deep Forward Network

Code

import numpy as np

from keras.layers import Dense

from keras.models import Sequential

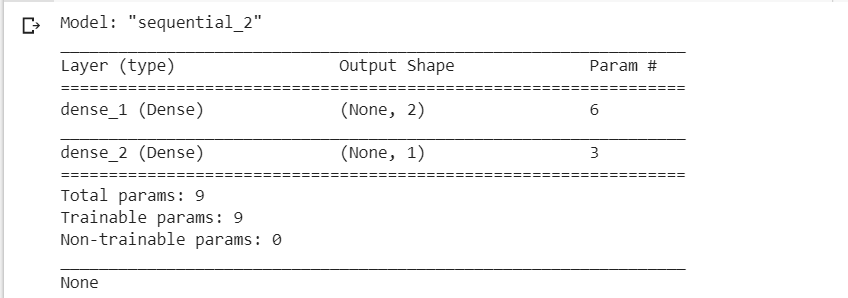
model=Sequential()

model.add(Dense(units=2, activation='relu', input\_dim=2))

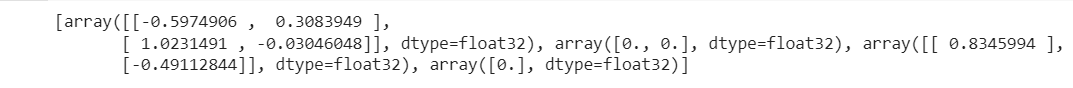
model.add(Dense(units=1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())



print(model.get\_weights())



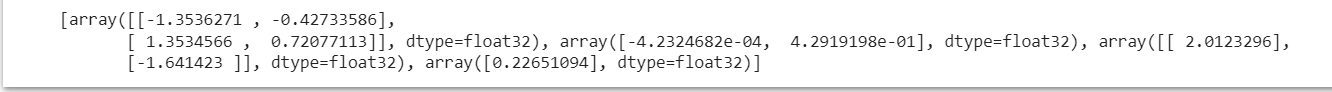
x=np.array([[0.,0.], [0.,1.], [1.,0.], [1.,1.]])

y=np.array([0.,1.,1.,0.])

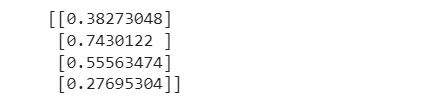
model.fit(x,y, epochs=1000, batch\_size=4)



print(model.get\_weights())



print(model.predict(x, batch\_size=4))



**Practical 3**

1. **Implement a deep neural network for performing classification task Problem statement: The given dataset comprises health information about diabitic women patients we need to create a deep forward network that will classify women suffering from diabitc melitus**

from google.colab import drive

drive.mount("/content/drive")

Location="drive/My Drive/Dataset"

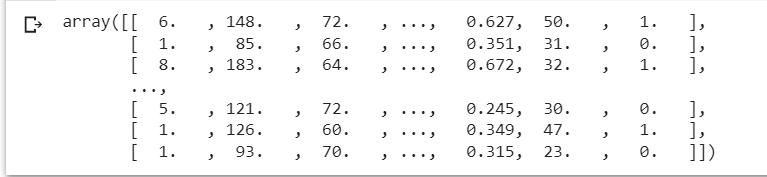
from numpy import loadtxt

from keras.models import Sequential

from keras.layers import Dense

dataset=loadtxt(Location+ '/pima-indians-diabetes.csv', delimiter=',')

dataset



x=dataset[:, 0:8]

y=dataset[:,8]

model=Sequential()

model.add(Dense(12, input\_dim=8, activation='relu'))

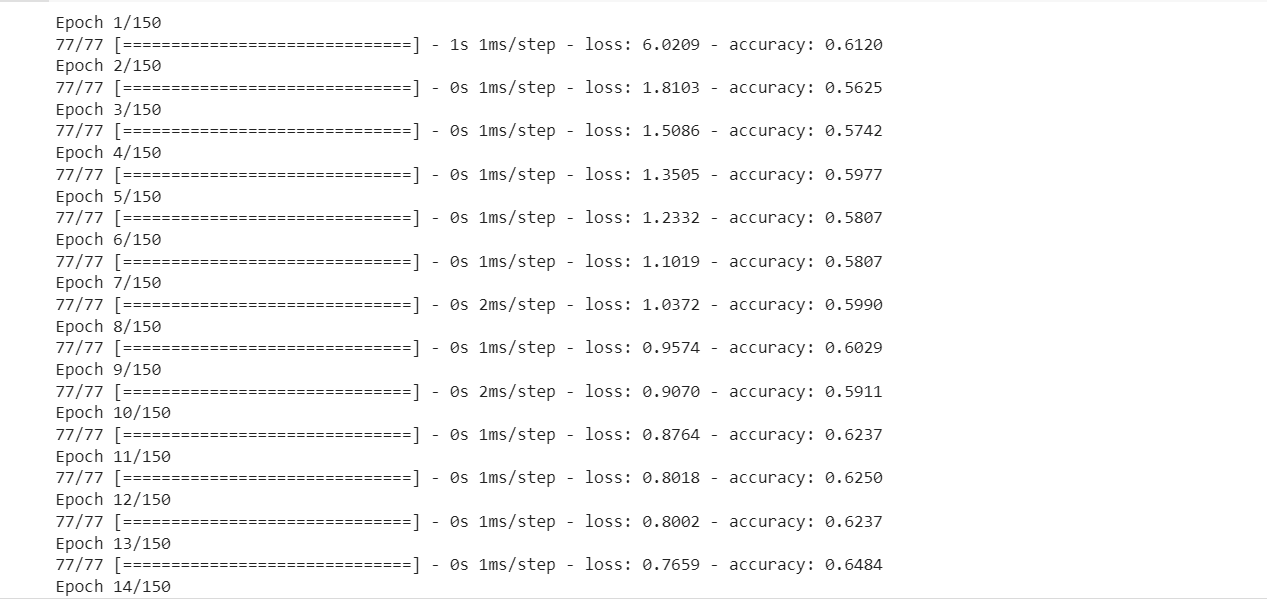
model.add(Dense(8, activation='relu'))

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

#compiling and fitting ,model

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

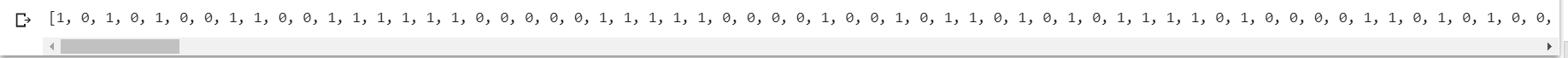
model.fit(x, y, epochs=150, batch\_size=10)



predictions=model.predict(x)

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

print(rounded)



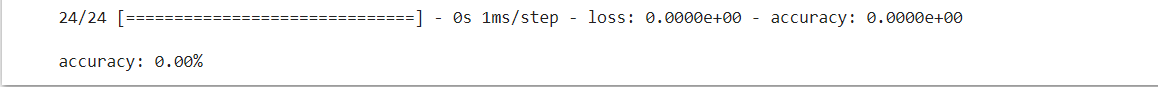
#evaluating the accuracy

accuracy=model.evaluate(x,y)

output: accuracy:0.7578

scores=model.evaluate(x)

print("\n%s: %.2f%%" %(model.metrics\_names[1], scores[1]\*100))



**Practical 3B**

**Develop a MultiLayerPerceptron (MLP) model for Ionospear Binary (two class) Classification. Also test and evaluate the model.**

Code

#mlp for binary classification

from pandas import read\_csv

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

from sklearn.preporcessing import LabelEncoder

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense

#load the dataset

path=’ionospear.csv’

df=read\_csv(path, header=None)

#split into input and output columns

x,y=df.values[:, :-1], df.values[:, -1]

#ensure all the data are floating point values

x=x.astype('float32')

#encode all string to integers

y=LabelEncoder().fit\_transform(y)

#split into train and test dataset

xtrain, xtest, ytrain, ytest=train\_test\_split(x,y, test\_size=0.33)

print(xtrain.shape, xtest.shape, ytrain.shape, ytest.shape)

#determine the number of input features

n\_features=xtrain.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

model.fit(xtrain, ytrain, epochs=150, batch\_size=32, verbose=0)

#evaluate the metrics

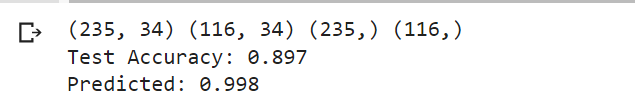
Loss, acc=model.evaluate(xtest, ytest, verbose=0)

print('Test Accuracy: %.3f' % acc)

rows=[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([rows])

print('Predicted: %.3f' % yhat)



**Practical 4**

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

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)

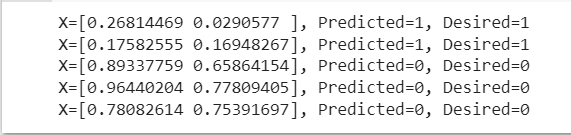
xnew=scalar.transform(xnew)

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]))



**4B. Using adeep field forward network with two hidden layers for performing classification and predicting the probability class**

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()

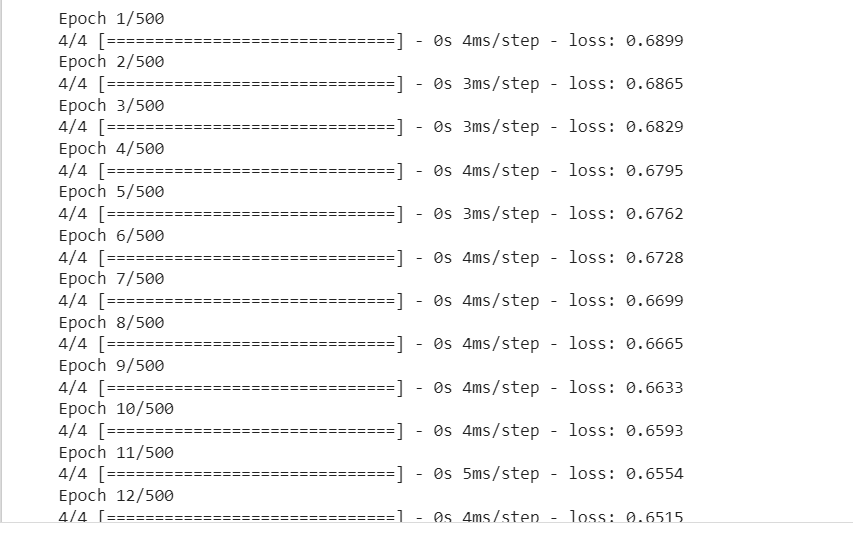
model.add(Dense(4, input\_dim=2, activation='relu'))

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)

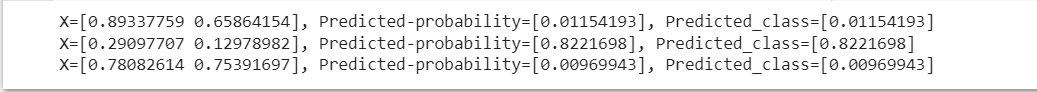
Xnew=scalar.transform(Xnew)

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]))



**Practical 5**

1. **CNN for CIFAR10 Images**

Code

import tensorflow as tf

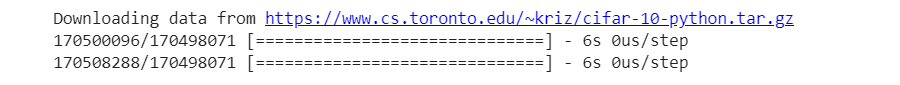
from tensorflow.keras import datasets, layers, models

import matplotlib.pyplot as plt

(train\_images, train\_labels), (test\_images, test\_labels)=datasets.cifar10.load\_data()

#Normalize pixel values to between 0 and 1

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



#Verify the data

#to verify data lets display first 25 images

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)

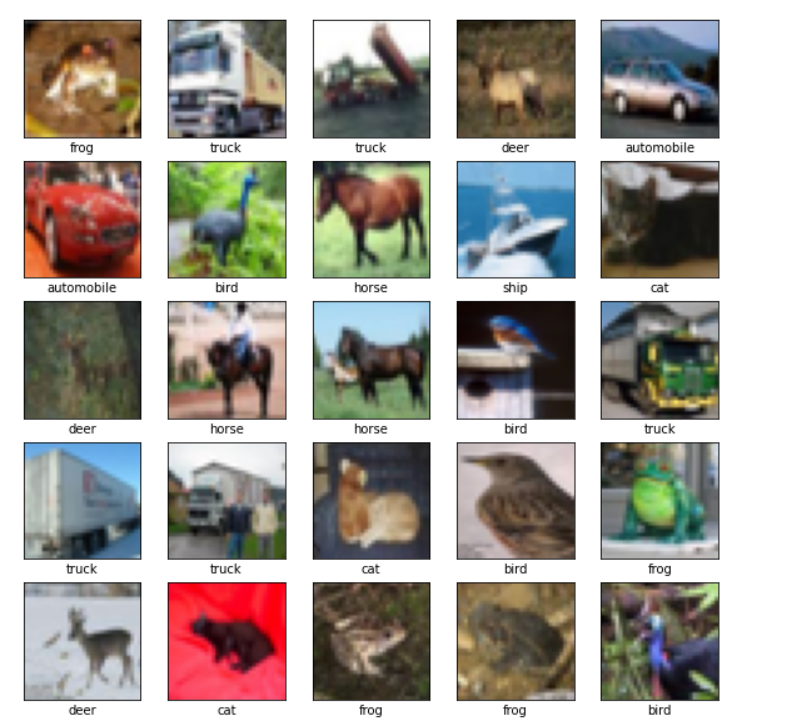
  plt.imshow(train\_images[i])

  # 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()



#create a convolutional base

model=models.Sequential()

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

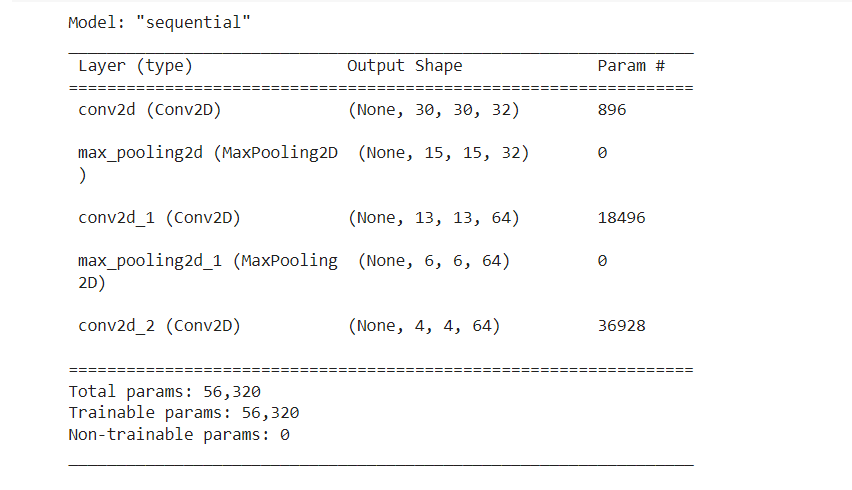
model.add(layers.MaxPooling2D((2,2)))

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()



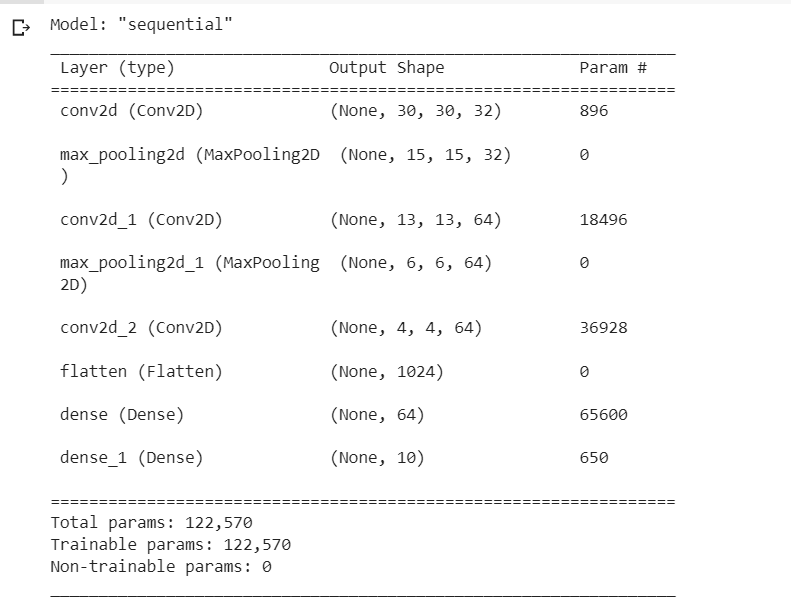
#Add Dense layers on top

model.add(layers.Flatten())

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

model.add(layers.Dense(10))

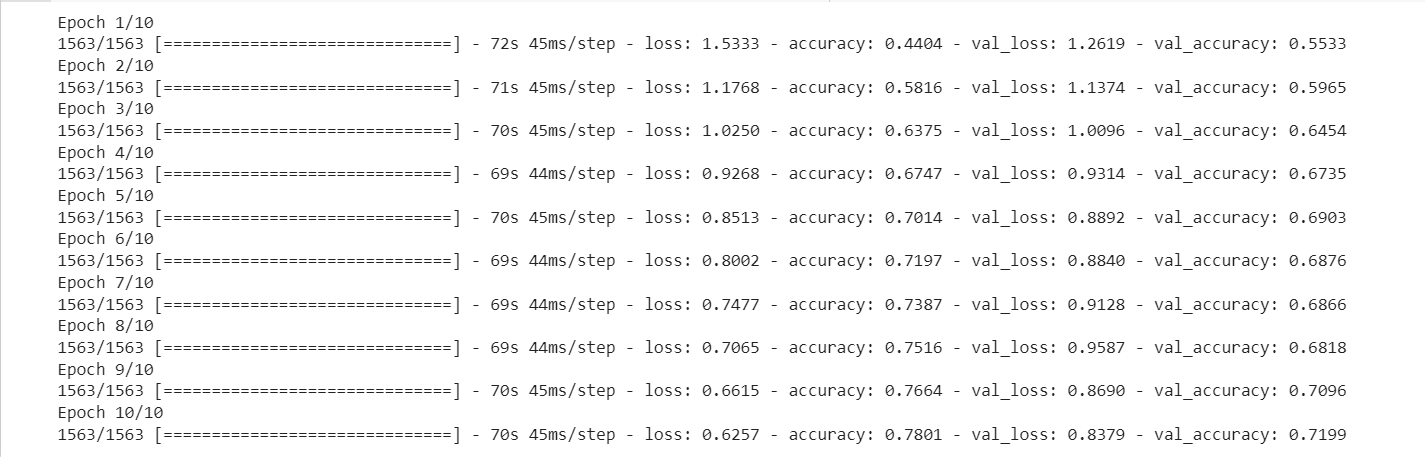
model.summary()



#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))



#Evaluate the model

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

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

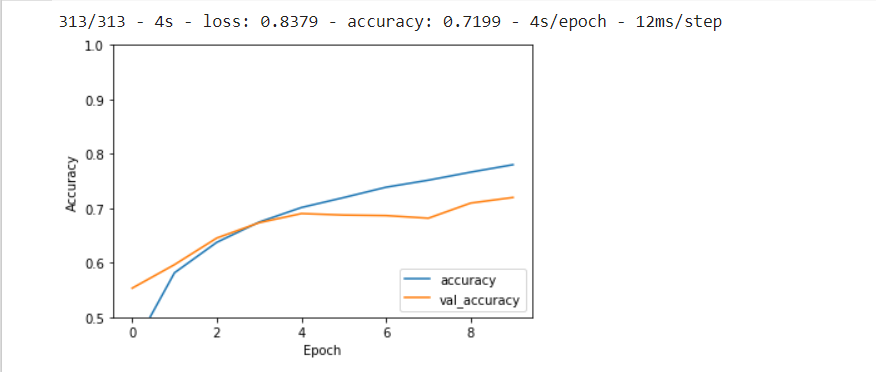
plt.xlabel('Epoch')

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)



print(test\_acc)

0.7199000120162964

**Practical 5**

1. **Image Classification**

Code

#import tensorflow and other libraries

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

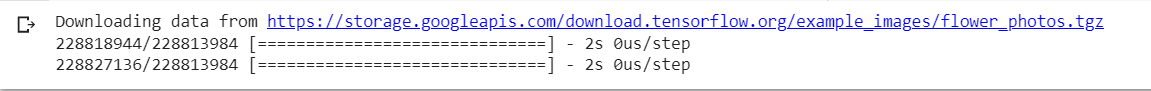
from tensorflow.keras.models import Sequential

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)

3670

#print the roses

roses = list(data\_dir.glob('roses/\*'))

PIL.Image.open(str(roses[0]))



PIL.Image.open(str(roses[1]))



#print some tulips

tulips = list(data\_dir.glob('tulips/\*'))

PIL.Image.open(str(tulips[0]))



PIL.Image.open(str(tulips[1]))



#load data using keras utility

#define some parameters for loaders

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)

Found 3670 files belonging to 5 classes

Using 2935 files for training.

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)

Found 3670 files belonging to 5 classes

Using 2935 files for validation.

class\_names = train\_ds.class\_names

print(class\_names)



#visualize the data

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



**5C. 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

)

#The flower dataset has five classes

num\_classes=metadata.features['label'].num\_classes

print(num\_classes)

5

#Let's retrieve an image from the dataset and use it to demonstrate data augmentation.

get\_label\_name=metadata.features['label'].int2str

image, label=next(iter(train\_ds))

\_=plt.imshow(image)

\_=plt.title(get\_label\_name(label))



#resizing and rescaling

img\_size=180

resize\_and\_rescale=tf.keras.Sequential([

                                        layers.Resizing(img\_size, img\_size),

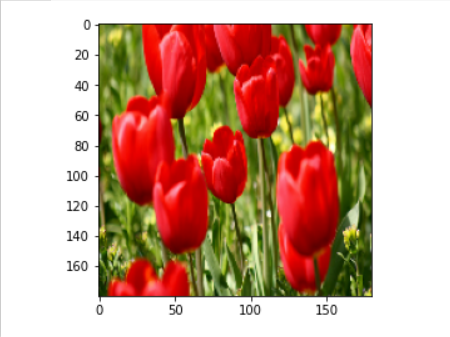
                                        layers.Rescaling(1./255)

])

#visualize he result of applying these layers to an image

result=resize\_and\_rescale(image)

\_=plt.imshow(result)



#Verify that the pixels are in the [0, 1] range

print("Min and max pixel values:", result.numpy().min(), result.numpy().max())

Min and max pixel values: 0.0 1.0

#data augmentation

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))

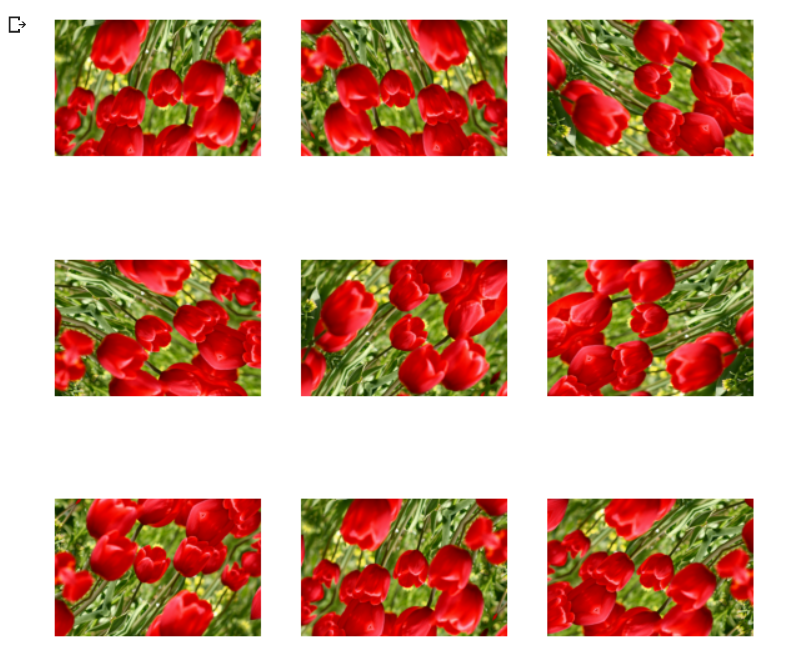
for i in range(9):

  augmented\_image=data\_augmentation(image)

  ax=plt.subplot(3,3,i+1)

  plt.imshow(augmented\_image[0])

  plt.axis('off')



**Practical 6**

**Building RNN**

Code

#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 into 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)

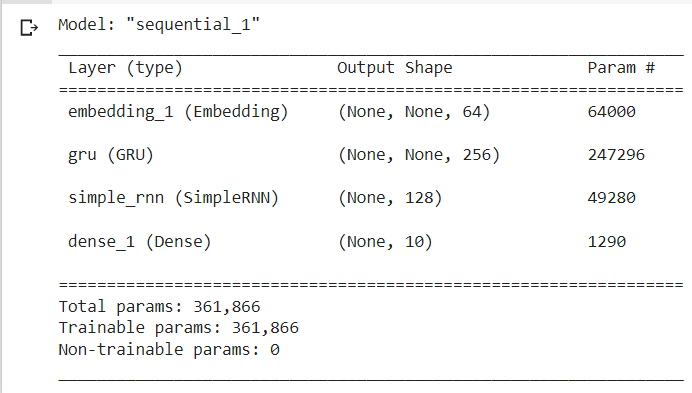
model.add(layers.GRU(256, return\_sequences=True))

# 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")(

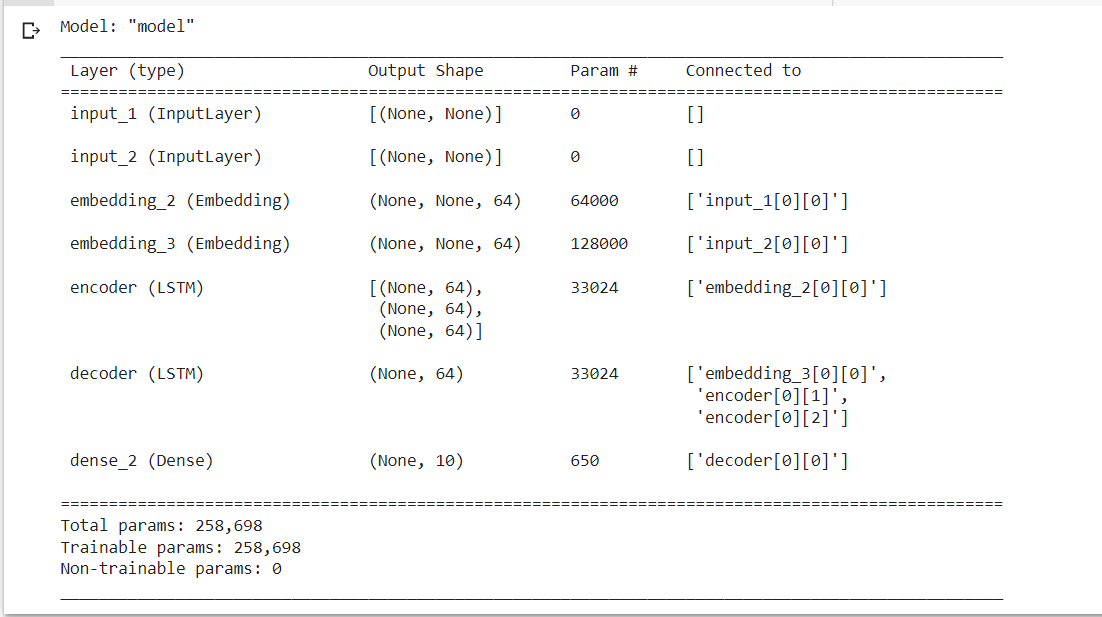
    decoder\_embedded, initial\_state=encoder\_state

)

output = layers.Dense(10)(decoder\_output)

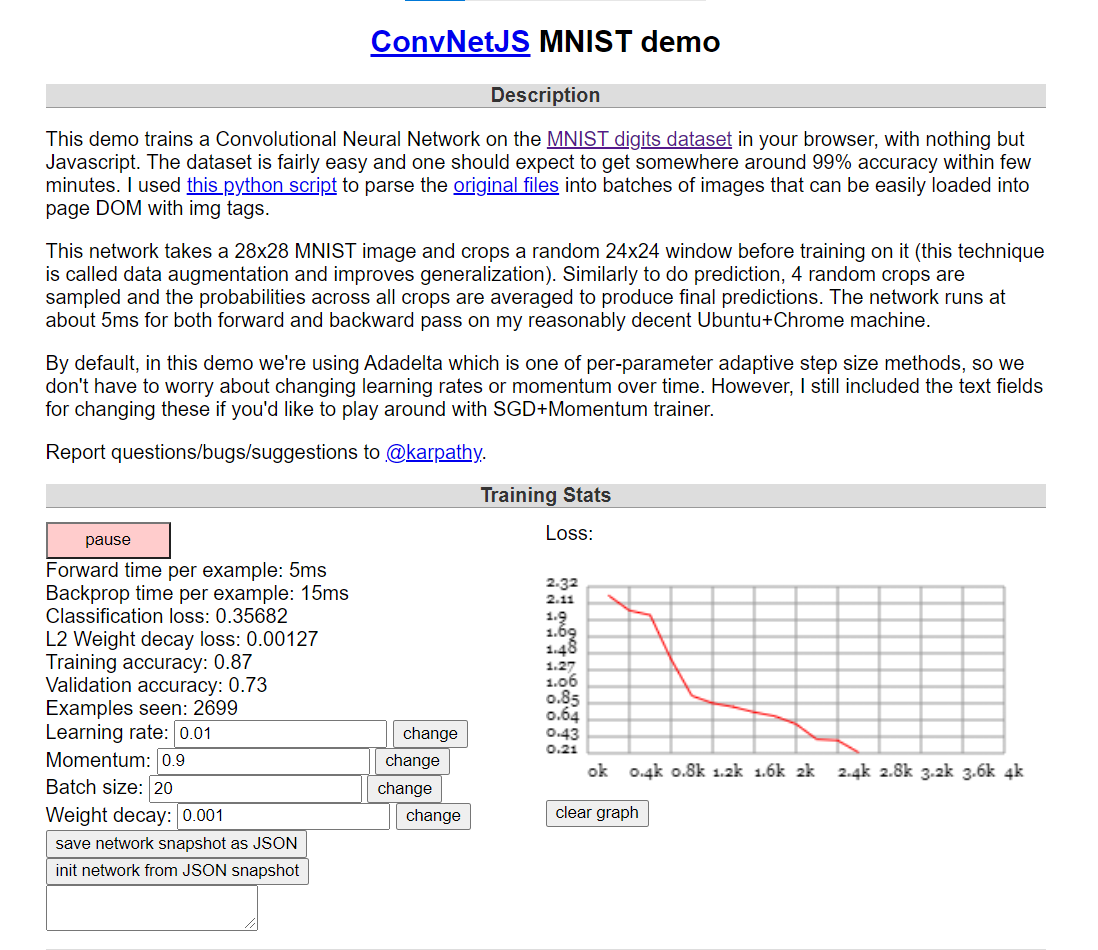
model = keras.Model([encoder\_input, decoder\_input], output)

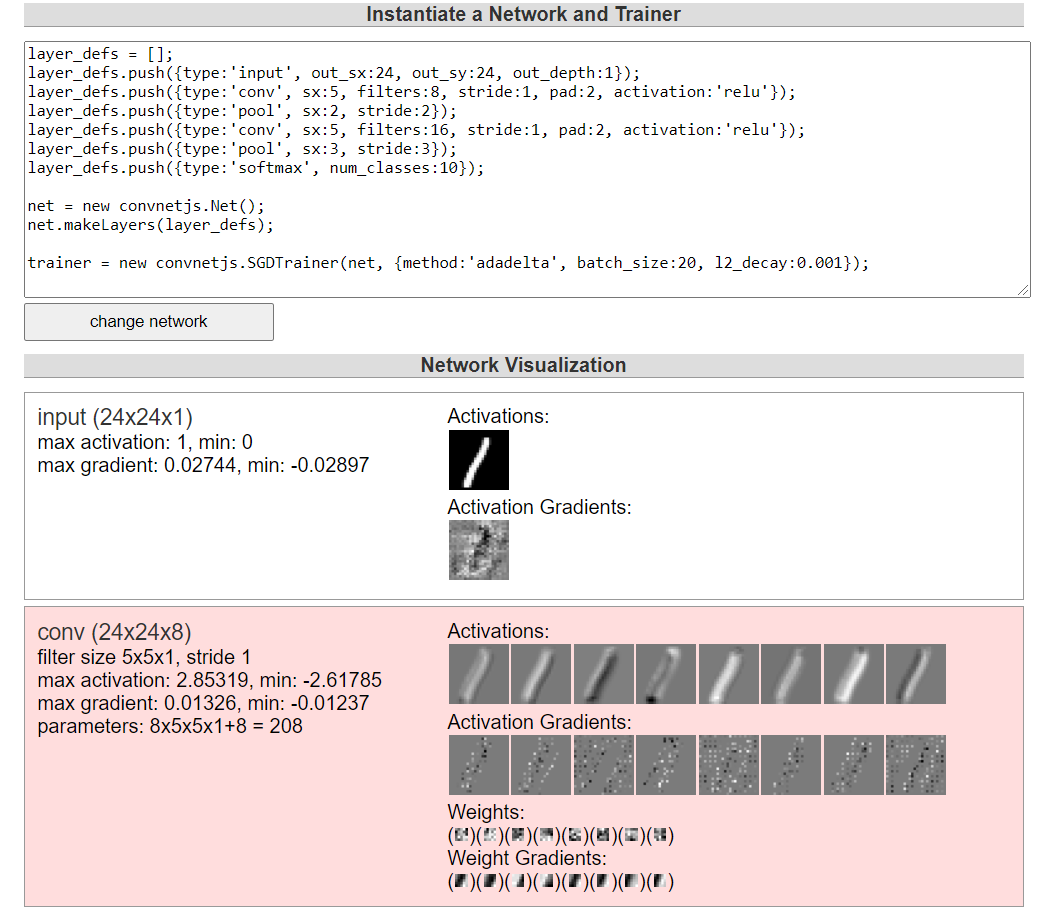
model.summary()



**Practical 7**

**Using CovNet build Deep Learning model**

****

****

****

****

**Practical No 8**

**Implement a single AutoEncoder based on Fully Connected layer**

Code

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)

#lets also create a separate encoder model

# This model maps an input to its encoded representation

encoder = keras.Model(input\_img, encoded)

# 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

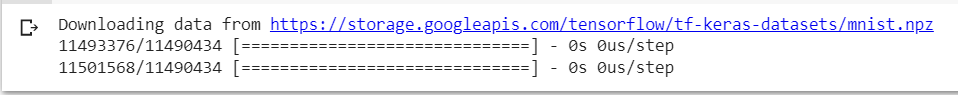
decoder = keras.Model(encoded\_input, decoder\_layer(encoded\_input))

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

from keras.datasets import mnist

import numpy as np

(x\_train, \_), (x\_test, \_) = mnist.load\_data()



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)



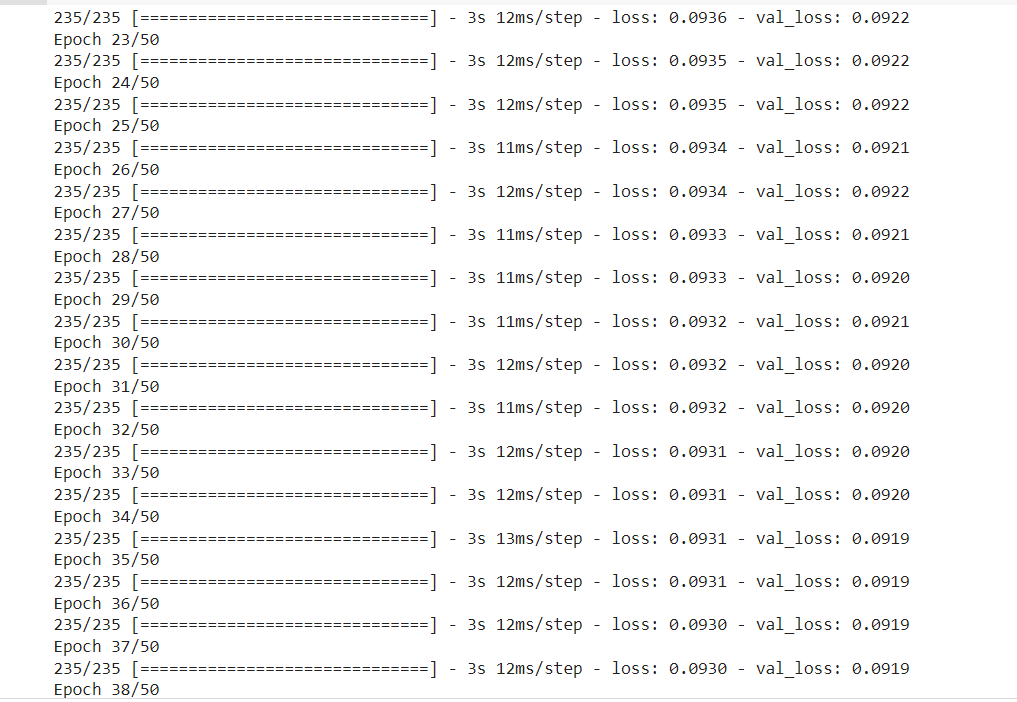
autoencoder.fit(x\_train, x\_train,

                epochs=50,

                batch\_size=256,

                shuffle=True,

                validation\_data=(x\_test, x\_test))



# 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()

