**1b. Multiplication of two: Vector, Matrix and Tensor**

import numpy as np

import tensorflow as tf

A= np.array([[1,2],[3,4],[5,6]])

print("A= ",A)

print("\n")

B=np.array([[2,5],[7,4],[4,3]])

print("B= ",B)

print("\n")

C=A\*B

print("Multiplication of two Matrix: \n",C)

print("\n")

x=np.array([1,2,3,4])

y=np.array([5,6,7,8])

z=x\*y

print("Multiplication of two Vector: ",z)

print("\n")

tensor\_A=tf.constant([[4,2]],dtype=tf.int32)

print("A: ",tensor\_A)

tensor\_B=tf.constant([[7,4]],dtype=tf.int32)

print("B: ",tensor\_B,"\n")

tensor\_multiply=tf.multiply(tensor\_A,tensor\_B)

print("Multiplication of two Tensor: ",tensor\_multiply)

print("\n")

1C)Addition of two : Vector, Matrix and Tensor

import numpy as np

import tensorflow as tf

x = np.array([1, 2, 3, 4])

y = np.array([5, 6, 7, 8])

z = x + y

print("Addition of two Arrays: ", z)

A = np.array([[1, 2], [3, 4], [5, 6]])

B = np.array([[2, 5], [7, 4], [4, 3]])

C = A \* B

print("Element-wise Multiplication of two Matrices: \n", C)

tensor\_A = tf.constant([1, 2, 3])

tensor\_B = tf.constant([4, 5, 6])

tensor\_add = tf.add(tensor\_A, tensor\_B)

print("Addition of two Tensors: ", tensor\_add)

1D) Multiply Matrix with Vector

import numpy as np

x=np.array([1,2,3,7,3,5,2])

y=np.array([[1],[3],[5],[7],[8],[8],[2]])

c=x\*y

print("Multiplication of Vector and Matrix: \n",c)

print("\n")

**1E) Matrix Dot product and Matrix Inverse Code**

import numpy as np

U = np.array([2, -3])

V = np.array([1, 3])

dotproduct = np.dot(U, V)

print("Dot product of matrix: ", dotproduct)

A = np.array([[6, 1, 1],

              [4, -2, 5],

              [2, 8, 7]])

print("\nInverse of Matrix A: \n", np.linalg.inv(A))

#### 2. Performing matrix multiplication and finding Eigen vectors and Eigen values using TensorFlow

import tensorflow as tf

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("\n")

print("Multiplying the matrices: ")

print(z)

print("\n")

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

print("\n")

print("Matrix A: \n{}\n\n".format(e\_matrix\_A))

eigan\_values\_A,eigan\_vectors\_A=tf.linalg.eigh(e\_matrix\_A)

print("Eigan Vectors: \n{}\n\nEigan Values:\n{}\n\n".format(eigan\_vectors\_A,eigan\_values\_A))

#### 3.Implementing deep neural network for performing binary classification task.

**Code:**

pip install scikeras

from pandas import read\_csv

from keras.models import Sequential from keras.layers import Dense

from scikeras.wrappers import KerasClassifier

from sklearn.model\_selection import cross\_val\_score from sklearn.preprocessing import LabelEncoder from sklearn.model\_selection import StratifiedKFold dataframe=read\_csv('sonar.csv',header=None) dataset=dataframe.values

X=dataset[:,0:60].astype(float) Y=dataset[:,60]

encoder=LabelEncoder() encoder.fit(Y) encoded\_y=encoder.transform(Y)

def create\_baseline(): model=Sequential()

model.add(Dense(60,input\_dim=60,activation='relu')) model.add(Dense(1,activation='sigmoid')) model.compile(loss='binary\_crossentropy',optimizer='adam',metrics=['accuracy']) return model

estimator=KerasClassifier(model=create\_baseline,epochs=10,batch\_size=5,verbose=0) KFold=StratifiedKFold(n\_splits=10,shuffle=True) results=cross\_val\_score(estimator,X,encoded\_y,cv=KFold)

print("\n")

print("Baseline: %.2f%% (%.2f%%)"%(results.mean()\*100,results.std()\*100))

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

import numpy as np

from sklearn.datasets import load\_wine

from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

from keras.layers import Dense, Input, concatenate, Dropout

from keras.models import Model

from tensorflow.keras.optimizers import RMSprop

# Load the wine dataset

dataset = load\_wine()

# Parameters

ensemble\_num = 10

bootstrap\_size = 0.8

training\_size = 0.8

num\_hidden\_neurons = 64

dropout = 0.25

epochs = 100

batch = 10

# Preprocessing

scaler = MinMaxScaler()

one\_hot = OneHotEncoder()

dataset['data'] = scaler.fit\_transform(dataset['data'])

dataset['target'] = one\_hot.fit\_transform(np.reshape(dataset['target'], (-1, 1))).toarray()

# Create a list to hold the data and corresponding labels

temp = []

for i in range(len(dataset.data)):

    temp.append([dataset['data'][i], np.array(dataset['target'][i])])

temp = np.array(temp, dtype=object)

np.random.shuffle(temp)

# Holdout training and test split

stop = int(training\_size \* len(dataset.data))

train\_X = np.array([x for x in temp[:stop, 0]])

train\_Y = np.array([x for x in temp[:stop, 1]])

test\_X = np.array([x for x in temp[stop:, 0]])

test\_Y = np.array([x for x in temp[stop:, 1]])

# Define sub-networks and the ensemble

sub\_net\_outputs = []

sub\_net\_inputs = []

for i in range(ensemble\_num):

    # Input layer

    net\_input = Input(shape=(train\_X.shape[1],))

    sub\_net\_inputs.append(net\_input)

    # Two hidden layers

    Y = Dense(num\_hidden\_neurons, activation='relu')(net\_input)

    Y = Dense(num\_hidden\_neurons, activation='relu')(Y)

    Y = Dropout(dropout)(Y)

    # Collect outputs of the sub-networks

    sub\_net\_outputs.append(Y)

# Concatenate all sub-networks' outputs

Y = concatenate(sub\_net\_outputs)

# Output layer with softmax activation

Y = Dense(train\_Y.shape[1], activation='softmax')(Y)

# Define the model

model = Model(inputs=sub\_net\_inputs, outputs=Y)

model.compile(optimizer=RMSprop(), loss='categorical\_crossentropy')

print('\n7\_Aditya Hadap')

print("Begin training...")

# Train the model

model.fit([train\_X] \* ensemble\_num, train\_Y, validation\_data=([test\_X] \* ensemble\_num, test\_Y), epochs=epochs, batch\_size=batch)

print("Training complete...")

# Set numpy print options

np.set\_printoptions(precision=2, suppress=True)

# Make predictions and compare with true labels

for i in range(len(test\_X)):

    prediction = model.predict([test\_X[i].reshape(1, test\_X[i].shape[0])] \* ensemble\_num)

    print(f"Prediction: {prediction} | True: {test\_Y[i]}")

**5. Evaluating feed forward deep network for multiclass Classification using K-Fold cross-validation.**

!pip install scikeras

!pip install np\_utils

import pandas as pd

from keras.models import Sequential

from keras.layers import Dense

from scikeras.wrappers import KerasClassifier

from tensorflow.keras.utils import to\_categorical

from sklearn.model\_selection import cross\_val\_score, KFold

from sklearn.preprocessing import LabelEncoder

# Load the dataset (assuming the first row contains column headers)

dataframe = pd.read\_csv("/content/iris.csv", header=0)  # Use header=0 to skip the first row if it contains headers

dataset = dataframe.values

X = dataset[:, 0:4].astype(float)

Y = dataset[:, 4]

# Encode class values as integers

encoder = LabelEncoder()

encoder.fit(Y)

encoded\_Y = encoder.transform(Y)

# Convert integers to dummy variables (i.e., one hot encoded)

dummy\_y = to\_categorical(encoded\_Y)

# Define the baseline model

def baseline\_model():

    model = Sequential()

    model.add(Dense(8, input\_dim=4, activation='relu'))

    model.add(Dense(3, activation='softmax'))

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

    return model

# Wrap the model using KerasClassifier

estimator = KerasClassifier(model=baseline\_model, epochs=200, batch\_size=5, verbose=0)

# Evaluate the model using k-fold cross-validation

kfold = KFold(n\_splits=10, shuffle=True)

results = cross\_val\_score(estimator, X, dummy\_y, cv=kfold)

# Print the accuracy

print("Accuracy: %.2f%% (%.2f%%)" % (results.mean() \* 100, results.std() \* 100))

**6) implementation of convolutional neural network to predict numbers from number images.**

from keras.datasets import mnist

from tensorflow.keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Conv2D, Flatten

import matplotlib.pyplot as plt

import numpy as np

(X\_train, Y\_train), (X\_test, Y\_test) = mnist.load\_data()

plt.imshow(X\_train[0])

plt.show

print(X\_train[0].shape)

X\_train=X\_train.reshape(60000,28,28,1)

X\_test=X\_test.reshape(10000,28,28,1)

Y\_train=to\_categorical(Y\_train)

Y\_test=to\_categorical(Y\_test)

Y\_train[0]

print(Y\_train[0])

model=Sequential()

model.add(Conv2D(64, kernel\_size=3, activation='relu', input\_shape=(28,28,1)))

model.add(Conv2D(32, kernel\_size=3, activation='relu'))

model.add(Flatten())

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

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

model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), epochs=1)

print(model.predict(X\_test[:4]))

print(Y\_test[:4])

**7. Performing encoding and decoding of images using deep autoencoder.**

import keras

from keras import layers

from keras.datasets import mnist

import numpy as np

import matplotlib.pyplot as plt

encoding\_dim=32

input\_img=keras.Input(shape=(784,))

encoded=layers.Dense(encoding\_dim,activation='relu')(input\_img)

decoded=layers.Dense(784,activation='sigmoid')(encoded)

autoencoder=keras.Model(input\_img,decoded)

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

encoded\_input=keras.Input(shape=(encoding\_dim,))

decoder\_layer=autoencoder.layers[-1]

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

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

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

x\_train=x\_train.astype('float32')/255.

x\_test=x\_test.astype('float32')/25

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=13,

                batch\_size=256,

                shuffle=True,

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

encoded\_imgs=encoder.predict(x\_test)

decoded\_imgs=decoder.predict(encoded\_imgs)

n=10

plt.figure(figsize=(20,4))

for i in range(n): 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)

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()

**8).Denoising of images using Autoencoder.**

import keras

from keras.datasets import mnist

from keras import layers

import numpy as np

from keras.callbacks import TensorBoard

import matplotlib.pyplot as plt

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

X\_train = X\_train.astype('float32') / 255.

X\_test = X\_test.astype('float32') / 255.

X\_train = np.reshape(X\_train, (len(X\_train), 28, 28, 1))

X\_test = np.reshape(X\_test, (len(X\_test), 28, 28, 1))

noise\_factor = 0.5

X\_train\_noisy = X\_train + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=X\_train.shape)

X\_test\_noisy = X\_test + noise\_factor \* np.random.normal(loc=0.0, scale=1.0, size=X\_test.shape)

X\_train\_noisy = np.clip(X\_train\_noisy, 0., 1.)

X\_test\_noisy = np.clip(X\_test\_noisy, 0., 1.)

n=10

plt.figure(figsize=(20,2))

for i in range(1,n+1):ax=plt.subplot(1,n,i)

plt.imshow(X\_test\_noisy[i].reshape(28,28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

plt.show()

input\_img = keras.Input(shape=(28, 28, 1))

x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(input\_img)

x=layers.MaxPooling2D((2,2),padding='same')(x)

x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)

encoded=layers.MaxPooling2D((2,2),padding='same')(x)

x= layers.Conv2D(32,(3,3),activation='relu',padding='same')(encoded)

x=layers.UpSampling2D((2,2))(x)

x=layers.Conv2D(32,(3,3),activation='relu',padding='same')(x)

x=layers.UpSampling2D((2,2))(x)

decoded=layers.Conv2D(1,(3,3),activation='sigmoid',padding='same')(x)

autoencoder=keras.Model(input\_img,decoded)

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

autoencoder.fit(X\_train\_noisy,X\_train,

epochs=3, batch\_size=328, shuffle=True,

validation\_data=(X\_test\_noisy,X\_test), callbacks=[TensorBoard(log\_dir='/tmp/tb',histogram\_freq=0,write\_graph=False)])

predictions=autoencoder.predict(X\_test\_noisy)

m=10

plt.figure(figsize=(20,2))

for i in range(1,m+1):ax=plt.subplot(1,m,i)

plt.imshow(predictions[i].reshape(28,28))

plt.gray()

ax.get\_xaxis().set\_visible(False)

ax.get\_yaxis().set\_visible(False)

plt.show()