**Deep learning Skill programs-list**

**NOTE:**

**1-5 programs should be completed using numpy and matplot lib**

**6-16 programs should be implemented using TensorFlow and keras**

1 .Design a one-layer Perceptron network to classify 4 classes. Assume that the data set includes 25 samples and each sample is 10 dimensional. Print the weights and biases of the model

**Solution :**

class mp:

def \_\_init\_\_(self, inodes, bias=0.2, lr=0.1):

self.w = np.array([1,1])

self.bias = bias

self.learning\_rate = lr

def get\_weight(self):

return self.w

def set\_weights(self,w):

self.w=w

def train(self, X, y, epochs):

Y\_predict = np.zeros(len(y))

for t in range(epochs):

for i, x in enumerate(X):

if (np.dot(X[i], self.w)) + self.bias <= 0:

Y\_predict[i] = 0

else:

Y\_predict[i] = 1

self.bias = self.bias + self.learning\_rate \* (y[i] - Y\_predict[i])

self.w = self.w + self.learning\_rate \* X[i] \* (y[i] - Y\_predict[i])

print("error", y[i] - Y\_predict[i], " epoch:", t, "bias:", self.bias)

return self.w

def predict(self, x\_new):

if ((np.dot(x\_new, self.w)) + self.bias) <= 0:

return 0

else:

return 1

import numpy as np

if \_\_name\_\_=="\_\_main\_\_":

X = np.array([

[0,0],

[0,1],

[1,0],

[1,1]

])

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

model=mp (2,-0.2,0.1)

print(model.get\_weight())

trained\_weights = model.train(X,y,50)

print("Trained",model.get\_weight())

print(model.predict(x\_new=[1,1]))

----------------------------------------------------------------------------------------------------------------

2.Implement a feedforward neural network and write the backpropagation code for training the network. Use numpy for all matrix/vector operations. You are not allowed to use any automatic differentiation packages. This network will be trained and tested using the Fashion-MNIST dataset with each image size as 28 x 28. Train the model to classify the images into one of 10 classes.

**Solution :**

import numpy as np

import matplotlib.pyplot as plt

class MLP:

def \_\_init\_\_(self, input\_nodes, n\_hidden\_nodes, n\_y\_nodes):

self.w1 = np.random.rand(n\_hidden\_nodes, input\_nodes)

self.w2 = np.random.rand(n\_y\_nodes, n\_hidden\_nodes)

self.b1 = np.random.rand(n\_hidden\_nodes)

self.b2 = np.random.rand(n\_y\_nodes)

def get\_weights(self):

return (self.w1,self.w2)

def sigmoid(self, z):

z = 1/(1+np.exp(-z))

return z

def forward\_prop(self, x):

h = np.dot(self.w1, x)

hout = self.sigmoid(h)

y = np.dot(self.w2, hout)

yout = self.sigmoid(y)

return h,hout,y,yout

def back\_prop(self, m, h, hout, y, yout, yp):

dz2 = yout-yp

dw2 = np.dot(dz2, hout.T)/m

dz1 = np.dot(self.w2.T, dz2) \* hout\*(1-hout)

dw1 = np.dot(dz1, x.T)/m

dw1 = np.reshape(dw1, self.w1.shape)

dw2 = np.reshape(dw2, self.w2.shape)

return dz2, dw2, dz1, dw1

def train(self, x, y, iterations, lr):

losses = []

for i in range(iterations):

h, hout, yp, yout = self.forward\_prop(x)

print("Epoch",i," : ",yout)

loss = -(1/len(x))\*np.sum(y\*np.log(yout)+(1-y)\*np.log(1-yout))

losses.append(loss)

m = 1/len(x)

da2, dw2, dz1, dw1 = self.back\_prop(m,h, hout, y, yout, y)

self.w2 = self.w2-lr\*dw2

self.w1 = self.w1-lr\*dw1

plt.plot(losses)

plt.xlabel("Iteration")

plt.ylabel("Loss")

plt.show()

def predict(self, input):

h, hout, y, yout = self.forward\_prop(input)

yout = np.squeeze(yout)

print("output : ", yout)

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

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

n\_x = 3

n\_y = 1

n\_h = 3

model = MLP(n\_x, n\_h, n\_y)

print(model.get\_weights())

model.train(x, y, 5, 0.2)

print(model.get\_weights())

model.predict(np.array([[0,1,1],[1,0,1],[0,0,1]]))

(or)

Using dataset:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from mpl\_toolkits import mplot3d

class Perceptron:

def \_\_init\_\_(self, learning\_rate=0.1):

self.learning\_rate = learning\_rate

self.\_b = 0.0 # y-intercept

self.\_w = None # weights assigned to input features

# count of errors during each iteration

self.misclassified\_samples = []

def fit(self, x: np.array, y: np.array, n\_iter=10):

"""

fit the Perceptron model on the training data """

self.\_b = 0.0

self.\_w = np.zeros(x.shape[1])

self.misclassified\_samples = []

for \_ in range(n\_iter):

# counter of the errors during this training iteration

errors = 0

for xi, yi in zip(x, y):

# for each sample compute the update value

update = self.learning\_rate \* (yi - self.predict(xi))

# and apply it to the y-intercept and weights array

self.\_b += update

self.\_w += update \* xi

errors += int(update != 0.0)

self.misclassified\_samples.append(errors)

def f(self, x: np.array) -> float:

"""

compute the output of the neuron

:param x: input features

:return: the output of the neuron

"""

return np.dot(x, self.\_w) + self.\_b

def predict(self, x: np.array):

"""

convert the output of the neuron to a binary output

:param x: input features

:return: 1 if the output for the sample is positive (or zero),

-1 otherwise

"""

return np.where(self.f(x) >= 0, 1, -1)

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'

# download and convert the csv into a DataFrame

df = pd.read\_csv(url, header=None)

df.head()

# extract the label column

y = df.iloc[:, 4].values

# extract features

x = df.iloc[:, 0:3].values

fig = plt.figure()

ax = plt.axes(projection='3d')

ax.set\_title('Iris data set')

ax.set\_xlabel("Sepal length in width (cm)")

ax.set\_ylabel("Sepal width in width (cm)")

ax.set\_zlabel("Petal length in width (cm)")

# plot the samples

ax.scatter(x[:50, 0], x[:50, 1], x[:50, 2], color='red',

marker='o', s=4, edgecolor='red', label="Iris Setosa")

ax.scatter(x[50:100, 0], x[50:100, 1], x[50:100, 2], color='blue',

marker='^', s=4, edgecolor='blue', label="Iris Versicolour")

ax.scatter(x[100:150, 0], x[100:150, 1], x[100:150, 2], color='green',

marker='x', s=4, edgecolor='green', label="Iris Virginica")

plt.legend(loc='upper left')

plt.show()

x = x[0:100, 0:2] # reduce the dimensionality of the data

y = y[0:100]

# plot Iris Setosa samples

plt.scatter(x[:50, 0], x[:50, 1], color='red', marker='o', label='Setosa')

# plot Iris Versicolour samples

plt.scatter(x[50:100, 0], x[50:100, 1], color='blue', marker='x',

label='Versicolour')

# show the legend

plt.xlabel("Sepal length")

plt.ylabel("Petal length")

plt.legend(loc='upper left')

# show the plot

plt.show()

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

# map the labels to a binary integer value

y = np.where(y == 'Iris-setosa', 1, -1)

# standardization of the input features

// standarization = (mean -input)/ standard deviation

plt.hist(x[:, 0], bins=100)

plt.title("Features before standardization")

plt.savefig("./before.png", dpi=300)

plt.show()

x[:, 0] = (x[:, 0] - x[:, 0].mean()) / x[:, 0].std()

x[:, 1] = (x[:, 1] - x[:, 1].mean()) / x[:, 1].std()

plt.hist(x[:, 0], bins=100)

plt.title("Features after standardization")

plt.show()

# split the data

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.25,

random\_state=0)

# train the model

classifier = Perceptron(learning\_rate=0.01)

classifier.fit(x\_train, y\_train)

# plot the number of errors during each iteration

plt.plot(range(1, len(classifier.misclassified\_samples) + 1),

classifier.misclassified\_samples, marker='o')

plt.xlabel('Epoch')

plt.ylabel('Errors')

plt.show()

from matplotlib.colors import ListedColormap

def plot\_decision\_regions(x, y):

resolution = 0.001

# define a set of markers

markers = ('o', 'x')

# define available colors

cmap = ListedColormap(('red', 'blue'))

# select a range of x containing the scaled test set

x1\_min, x1\_max = x[:, 0].min() - 0.5, x[:, 0].max() + 0.5

x2\_min, x2\_max = x[:, 1].min() - 0.5, x[:, 1].max() + 0.5

# create a grid of values to test the classifier on

xx1, xx2 = np.meshgrid(np.arange(x1\_min, x1\_max, resolution),

np.arange(x2\_min, x2\_max, resolution))

Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)

Z = Z.reshape(xx1.shape)

# plot the decision region...

plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)

plt.xlim(xx1.min(), xx1.max())

plt.ylim(xx2.min(), xx2.max())

# ...and the points from the test set

for idx, c1 in enumerate(np.unique(y)):

plt.scatter(x=x[y == c1, 0],

y=x[y == c1, 1],

alpha=0.8,

c=cmap(idx),

marker=markers[idx],

label=c1)

plt.show()

plot\_decision\_regions(x\_test, y\_test)

**--------------------------------------------------------------------------------------------------------------**

3) Implement a 2-class classification neural network with a single hidden layer

Use units with a non-linear activation function, such as tanh

Compute the cross entropy loss, Implement forward and backward propagation using functions Using Planar data classification with one hidden layer

**Solution:**

import numpy as np

def cross\_entropy\_loss(y\_predict,y):

loss=-np.sum(y\*np.log(y\_predict))

return loss/float(y\_predict.shape[0])

def accuracy(y\_pred,y\_true,y\_size):

if y\_size>1:

acc=y\_pred.argmax(axis=1)==y\_true.argmax(axis=1)

else:

acc=y\_pred==y\_true

return acc.mean()

def tanh(x):

return ((np.exp(x)-np.exp(-x))/(np.exp(x)+np.exp(-x)))

class MLP\_1:

def \_\_init\_\_(self,input\_size,hidden\_size,output\_size,learning\_rate=0.1):

self.input\_size=input\_size

self.hidden\_size=hidden\_size

self.output\_size=output\_size

self.lr=learning\_rate

self.W1=np.random.normal(scale=0.5,size=(input\_size,hidden\_size))

self.W2=np.random.normal(scale=0.5,size=(hidden\_size,output\_size))

def predict(self,x):

Z1=np.dot(x,self.W1)

A1=tanh(Z1)

Z2=np.dot(A1,self.W2)

A2=tanh(Z2)

print(A2.shape)

return A2

def train(self,X\_train,Y\_train,epochs):

results=pd.DataFrame(columns=["cel","accuracy"])

for itr in range(epochs):

Z1=np.dot(X\_train,self.W1)

A1=tanh(Z1)

print(A1.shape,self.W2.shape)

Z2=np.dot(A1,self.W2)

A2=tanh(Z2)

N=len(X\_train)

cel=cross\_entropy\_loss(A2,Y\_train)

acc=accuracy(A2,Y\_train,self.output\_size)

results=results.append({"cel":cel,"accuracy":acc},ignore\_index=True)

E1=A2-Y\_train

dW1=E1\*A2\*(1-A2)

E2=np.dot(dW1,self.W2.T)

dW2=E2\*A1\*(1-A1)

W2\_update=np.dot(A1.T,dW1)/N

W1\_update=np.dot(X\_train.T,dW2)/N

self.W2=self.W2-self.lr\*W2\_update

self.W1=self.W1-self.lr\*W1\_update

return results

from sklearn.datasets import load\_iris

import pandas as pd

import numpy as np

data=load\_iris()

x=data.data

y=data.target

#print(y[125])

y=pd.get\_dummies(y).values

y\_train=y

x\_train=x

ml\_iris=MLP\_1(4,2,3,0.1)

model\_results=ml\_iris.train(x\_train,y\_train,10)

-----------------------------------------------------------------------------------------------------------

# 4. Implement Random -mini-batch evaluations on the above program.

# **Solution :**

**import** warnings  
warnings.filterwarnings(**'ignore'**)

**import** numpy **as** np  
import seaborn **as** sns  
import matplotlib.pyplot **as** plt  
import pandas **as** pd  
import math

**def** sigmoid(x):  
 **return** 1/(1+np.exp(-x))  
  
def mean\_squared\_error(y\_pred,y\_true):  
 **return** ((y\_pred -y\_true)\*\*2).sum()/(2\*y\_pred.size)  
  
def mse\_graph\_generate(mse):  
 plt.figure()  
 plt.plot(mse,label=**"Mean Squared Error"**,color=**"red"**,linestyle=**"dashed"**)  
 plt.title(**"Epochs vs Mean squared error"**,color=**"darkred"**,size=13)  
 plt.ylabel(**"Mean squared error"**)  
 plt.xlabel(**"Epochs"**)  
 plt.legend()  
 plt.show()  
   
def cross\_graph(cross):  
 plt.figure()  
 plt.plot(cross,label=**"Cross Entropy Error"**,color=**"red"**,linestyle=**"dashed"**)  
 plt.title(**"Epochs vs Cross Entropy error"**,color=**"darkred"**,size=13)  
 plt.ylabel(**"Cross Entropy Error"**)  
 plt.xlabel(**"Epochs"**)  
 plt.legend()  
 plt.show()

**class** MNN:  
 **def** \_\_init\_\_(self,input\_nodes,hidden\_nodes,output\_nodes,learning\_rate):  
 self.input\_nodes=input\_nodes  
 self.hidden\_nodes=hidden\_nodes  
 self.output\_nodes=output\_nodes  
 self.learning\_rate=learning\_rate  
 self.w1=np.random.rand(input\_nodes,hidden\_nodes)  
 self.w2=np.random.rand(hidden\_nodes,output\_nodes)  
   
 **def** get\_weights(self):  
 **return** self.w1,self.w2  
   
 **def** prediction(self,x):  
 hout=sigmoid(np.dot(x,self.w1))  
 out=sigmoid(np.dot(hout,self.w2))  
 print(out)  
 **return** out  
   
 **def** batch\_train(self,x\_train,y\_train):  
 add\_mse=0  
 add\_cross=0  
   
 epoch\_list=[]  
 cross\_list=[]  
 mse\_list=[]  
   
# for i in range(epochs):  
 hin=sigmoid(np.dot(x\_train,self.w1))  
 oin=sigmoid(np.dot(hin,self.w2))  
   
 length=len(x\_train)  
   
# epoch\_list.append(i+1)  
 mse=mean\_squared\_error(oin,y\_train)  
 mse\_list.append(mse)  
   
 add\_mse=add\_mse+mse  
 closs=cross\_entropy\_loss(oin,y\_train)  
 cross\_list.append(closs)  
   
 add\_cross=add\_cross+closs  
# BackPropagation  
 e1=oin-y\_train  
 dw1=e1\*oin\*(1-oin)  
   
 e2=np.dot(dw1,np.transpose(self.w2))  
 dw2=e2\*hin\*(1-hin)  
   
 update\_w2=np.dot(np.transpose(hin),dw1)/length  
 update\_w1=np.dot(np.transpose(x\_train),dw2)/length  
   
 self.w2=self.w2-self.learning\_rate-update\_w2  
 self.w1=self.w1-self.learning\_rate-update\_w1  
# print(closs)  
 **return** mse,closs  
   
 **def** train(self,x\_train,y\_train,epochs):  
   
 list\_mse=[]  
 list\_cross=[]  
 list\_epoch=[]  
 list\_mini\_batch=[]  
 **for** i **in** range(epochs):  
 batch\_size=64  
 losses=np.zeros(epochs)  
 num\_batch=math.ceil(x\_train.shape[0]/batch\_size)  
 list\_epoch.append(i+1)  
# print()  
 closs\_sum=0  
# print("Epoch : ",i+1)  
 **for** mini\_batch **in** zip(batch\_x,batch\_y):  
 x\_mini,y\_mini=mini\_batch  
 m=len(x\_mini)  
 mse,closs=self.batch\_train(x\_mini,y\_mini)  
 closs\_sum=closs\_sum+closs  
# print("Mini batch")  
 list\_mse.append(mse)  
 list\_cross.append(closs\_sum)  
 list\_mini\_batch.append(mini\_batch)  
 print(**"Mean Square Error : "**,mse)  
 print(**"Cross Entropy Error : "**,closs)

**if** \_\_name\_\_==**"\_\_main\_\_"**:  
 print(**"\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*"**)  
 print(**"Main Menu"**)  
 print(**"1. Credict Card Dataset"**)  
 print(**"2. Heart Dataset"**)  
 print(**"\*-\*-\*-\*-\*-\*-\*-\*-\*-\*-\*"**)  
 val=int(input(**"Enter your choice : "**))  
 lab=pd.read\_csv(**"Dataset/heart.csv"**)  
 lab[**'target'**].unique()  
 x=lab.iloc[:,:-1]  
 y=lab[**"target"**]  
 model=MNN(13,5,1,0.1)  
 print(model.get\_weights())  
 model.train(x,y,80)  
 model.prediction(x)

# --------------------------------------------------------------------------------------------------------------

5.   Implement adam and SGD algorithms with only numpy library and use it to implement DNN classification model

**Solution:**

import numpy as np

# import math.pyplot as plt

def cost\_fun(theta,x,y):

m=len(y)

prediction=x.dot(theta)

error=(1/2\*m)\*np.sum(np.square((prediction-y)))

return error

def SGD(x,y,theta,lr,epochs):

m=len(y)

cal\_history=np.zeros([epochs])

for i in range(epochs):

cost=0

# print("epoch",i)

for j in range(m):

rand\_x=np.random.randint(0,m)

x[j]=x[rand\_x,:]

y[j]=y[rand\_x].reshape(1,1)

predict\_y=np.dot(x[j],theta)

theta=theta-(1/m)\*(lr\*np.dot((predict\_y-y),x[j].T))

cost+=cost\_fun(theta,x,y)

# print(cost)

cal\_history[i]=cost

return cal\_history,theta

l=0.01

theta=0.1

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

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

cal\_history,theta=SGD(x,y,theta,l,10)

print(cal\_history)

print(theta)

----------------------------------------------------------------------------------------------------------------

# 6. Implement keras Regression using car price dataset with drop out, normalization layers. Use early stopping to overcome overfit.

# Solution:

# import numpy as np

# import pandas as pd

# from keras.models import Sequential

# from keras.layers import Dense, Dropout, BatchNormalization

# from keras.callbacks import EarlyStopping

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

# 

# data = pd.read\_csv('car\_price\_data.csv')

# 

# data = pd.get\_dummies(data, columns=['CarName','fueltype','aspiration','doornumber','carbody','drivewheel','enginelocation','enginetype','cylindernumber','fuelsystem'])

# 

# 

# X = data.drop(['price'], axis=1)

# y = data['price']

# 

# X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2)

# 

# model = Sequential()

# 

# model.add(Dense(64, input\_dim=X\_train.shape[1], activation='relu'))

# model.add(BatchNormalization())

# model.add(Dropout(0.5))

# model.add(Dense(32, activation='relu'))

# model.add(BatchNormalization())

# model.add(Dropout(0.5))

# model.add(Dense(1))

# 

# model.compile(loss='mean\_squared\_error', optimizer='adam')

# 

# early\_stopping = EarlyStopping(monitor='val\_loss', patience=10) # regularization technique

# 

# model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_data=(X\_test, y\_test), callbacks=[early\_stopping])

# ----------------------------------------------------------------------------------------------------------------

7. Use the standard train/test split of fashion\_mnist (use (X\_train, y\_train), (X\_test, y\_test) = fashion\_mnist.load\_data()). Consider 10% of the training data aside as validation data find out which hyperparameter gives you best result from the below list

* number of epochs: 5, 10
* number of hidden layers: 3, 4, 5
* size of every hidden layer: 32, 64, 128
* weight decay (L2 regularisation): 0, 0.0005, 0.5
* learning rate: 1e-3, 1 e-4
* optimizer: sgd, momentum, nesterov, rmsprop, adam, nadam
* batch size: 16, 32, 64
* weight initialisation: random, Xavier
* activation functions: sigmoid, tanh, ReLu

**Solution:**

import keras

from keras.datasets import fashion\_mnist

from keras.models import Sequential

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

from keras.optimizers import SGD, RMSprop, Adam

from keras.regularizers import l2

from keras.initializers import RandomNormal, glorot\_normal

from keras.activations import sigmoid, tanh, relu

from keras.utils import to\_categorical

import numpy as np

# Load the Fashion MNIST dataset

(X\_train, y\_train), (X\_test, y\_test) = fashion\_mnist.load\_data()

# Preprocess the data

X\_train = X\_train.reshape(60000, 28, 28, 1) / 255.0

X\_test = X\_test.reshape(10000, 28, 28, 1) / 255.0

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

# Split 10% of the training data as validation data

X\_val = X\_train[-6000:]

y\_val = y\_train[-6000:]

X\_train = X\_train[:-6000]

y\_train = y\_train[:-6000]

# Define a list of hyperparameters to try

epochs = [5, 10]

hidden\_layers = [3, 4, 5]

hidden\_layer\_size = [32, 64, 128]

weight\_decay = [0, 0.0005, 0.5]

learning\_rate = [1e-3, 1e-4]

optimizers = [SGD(), RMSprop(), Adam()]

batch\_size = [16, 32, 64]

weight\_initialization = [RandomNormal(seed=0), glorot\_normal(seed=0)]

activations = [sigmoid, tanh, relu]

# Initialize the best hyperparameters and accuracy

best\_params = None

best\_accuracy = 0.0

# Loop through all possible combinations of hyperparameters

for e in epochs:

for hl in hidden\_layers:

for hls in hidden\_layer\_size:

for wd in weight\_decay:

for lr in learning\_rate:

for opt in optimizers:

for bs in batch\_size:

for wi in weight\_initialization:

for act in activations:

# Define the model architecture

model = Sequential()

model.add(Flatten(input\_shape=(28, 28, 1)))

for i in range(hl):

model.add(Dense(hls, activation=act, kernel\_initializer=wi, kernel\_regularizer=l2(wd)))

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

# Compile the model

opt.learning\_rate = lr

model.compile(optimizer=opt, loss='cross\_entropy',metrics=['Accuracy'])

----------------------------------------------------------------------------------------------------------------

8., Create a convolutional neural network via Keras and it should have the following layers:

input of (32, 32, 3) ,conv2D, 16 kernels, kernel size = 3, valid padding, ReLu actvation ,conv2D, 16 kernels, kernel size = 3, valid padding, ReLu actvation, maxpooling kernel size = 2\*2, conv2D, 32 kernels, kernel size = 3, valid padding, ReLu actvation, conv2D, 32 kernels, kernel size = 3, valid padding, ReLu actvation, maxpooling kernel size = 2\*2, flatten,Dense, 10 neurons, softmax activation

And Fit the neural network for the training data.( Download the CIFAR10 dataset)

* use Adam optimizer with its default settings ,use batch size of 64, use accuracy as a metric, use categorical\_crossentropy loss
* print the metric after each epoch for both the train and the test set!
* norm the images to have the pixel values between 0-1 (instead of 0-255), convert the labels to one-hot-encoded variables (see to\_categorical), train the neural network for 5 epochs

**Solution:**

import keras

from keras.datasets import cifar10

from keras.models import Sequential

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

from keras.optimizers import Adam

from keras.utils import to\_categorical

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

X\_train = X\_train / 255.0 # normalizing values

X\_test = X\_test / 255.0

# Convert labels to one-hot-encoded variables categorical data to numerical data

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

model = Sequential()

model.add(Conv2D(16, (3,3), padding='valid', activation='relu', input\_shape=(32,32,3)))

model.add(Conv2D(16, (3,3), padding='valid', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Conv2D(32, (3,3), padding='valid', activation='relu'))

model.add(Conv2D(32, (3,3), padding='valid', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Flatten()) # fully connected layer

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

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

history = model.fit(X\_train, y\_train, batch\_size=64, epochs=5, validation\_data=(X\_test, y\_test))

for i in range(5):

print("Epoch ", i+1)

print("Train accuracy: ", history.history['accuracy'][i])

print("Test accuracy: ", history.history['val\_accuracy'][i])

----------------------------------------------------------------------------------------------------------------

9. , Improve convolutional neural network for the above program that can achieve 70% accuracy on the test set. Use regularization techniques

**Solution:**

from keras.datasets import cifar10

from keras.utils import to\_categorical

from keras.models import Sequential

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

from keras.optimizers import Adam

(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()

X\_train = X\_train / 255.0 # normalize values

X\_test = X\_test / 255.0

# Convert the labels to one-hot-encoded variables

y\_train = to\_categorical(y\_train, 10)

y\_test = to\_categorical(y\_test, 10)

model = Sequential()

model.add(Conv2D(16, (3, 3), padding='valid', activation='relu', input\_shape=(32, 32, 3))) # muliple kernels

# adding batch normalization layer

# for different sizes of feature map

# we are considering 16 kernels in input layer . we do get different sizes of feature maps. to normalize it we are using batchNormalization layer

model.add(BatchNormalization())

model.add(Conv2D(16, (3, 3), padding='valid', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(32, (3, 3), padding='valid', activation='relu'))

model.add(Conv2D(32, (3, 3), padding='valid', activation='relu'))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

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

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

model.add(Dropout(0.5)) # regularization added to improve accuracy

history = model.fit(X\_train, y\_train, batch\_size=64, epochs=5, validation\_data=(X\_test, y\_test))

for i in range(5):

print('Epoch:', i + 1)

print('Train Accuracy:', history.history['accuracy'][i])

print('Test Accuracy:', history.history['val\_accuracy'][i])

----------------------------------------------------------------------------------------------------------------

10. Implement a VGG model by building 3 ‘blocks’ of 2 convolutional layers each, MaxPooling after each block ,The first block should use at least 32 filters, later blocks should use morefilter size is 3x3 with zero-padding . No of hidden nodes at dense layer are 128. use ReLU activations. Plot error graph and compare it with CNN model

**Solution:**

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.datasets import cifar10

# Load the CIFAR-10 dataset

(x\_train, y\_train), (x\_val, y\_val) = cifar10.load\_data()

# Normalize the input data

x\_train = x\_train.astype("float32") / 255.0

x\_val = x\_val.astype("float32") / 255.0

# Define the VGG model

model = keras.Sequential(

[

# Block 1

layers.Conv2D(32, (3, 3), padding="same", activation="relu", input\_shape=(32, 32, 3)),

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

layers.MaxPooling2D((2, 2)),

# Block 2

layers.Conv2D(64, (3, 3), padding="same", activation="relu"),

layers.Conv2D(64, (3, 3), padding="same", activation="relu"),

layers.MaxPooling2D((2, 2)),

# Block 3

layers.Conv2D(128, (3, 3), padding="same", activation="relu"),

layers.Conv2D(128, (3, 3), padding="same", activation="relu"),

layers.MaxPooling2D((2, 2)),

# Dense layers

layers.Flatten(),

layers.Dense(128, activation="relu"),

layers.Dense(128, activation="relu"),

layers.Dense(10, activation="softmax"),

]

)

# Compile the model

model.compile(

loss="sparse\_categorical\_crossentropy",

optimizer="adam",

metrics=["accuracy"],

)

# Train the model

history = model.fit(

x\_train, y\_train, batch\_size=32, epochs=3,validation\_data=(x\_val,y\_val)

)

# Plot the training and validation loss

import matplotlib.pyplot as plt

plt.plot(history.history["loss"], label="training loss")

plt.plot(history.history["val\_loss"], label="validation loss")

plt.legend()

plt.show()

**----------------------------------------------------------------------------------------------------------------**

11. Perform image augmentation (rotation, shift, shear, zoom, flip, using the ImageDataGenerator  and compare the result, before augumentation and after augumentaion.

# 12. Implement RNN for IMDV review classification

# 13. Implement LSTM. Perform LSTM on stock market analysis

# 14. Implement Region-based CNN object detection with TensorFlow using custom dataset.

# 15 Implement seq2seq models using lstm and autoencoders

# 16 Denoising autoenecoders with Keras and TensorFlow

# 17 . Project selection , Abstract preparation

# 18. Review

# 19 Project Analysis, module design

# 20. Review

# 21, Implementation of Project

# 22. Review

# 23. Document Preparation and submission

# 24. Review