Ex 4

import random

import matplotlib.pyplot as plt

import matplotlib

import numpy as np

import tensorflow as tf

from tensorflow.keras import layers

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.datasets import mnist

from tensorflow.keras.datasets import fashion\_mnist

from tensorflow.keras.utils import to\_categorical

from matplotlib.ticker import (MultipleLocator, FormatStrFormatter)

plt.rcParams['axes.titlesize'] = 16

plt.rcParams['axes.labelsize'] = 14

plt.rcParams['image.cmap'] = 'gray'

SEED\_VALUE = 42

# Fix seed to make training deterministic.

random.seed(SEED\_VALUE)

np.random.seed(SEED\_VALUE)

tf.random.set\_seed(SEED\_VALUE)

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

X\_valid = X\_train\_all[:10000]

X\_train = X\_train\_all[10000:]

y\_valid = y\_train\_all[:10000]

y\_train = y\_train\_all[10000:]

print(X\_train.shape)

print(X\_valid.shape)

print(X\_test.shape)

plt.figure(figsize=(18, 5))

for i in range(3):

plt.subplot(1, 3, i + 1)

plt.axis(True)

plt.imshow(X\_train[i], cmap='gray')

plt.subplots\_adjust(wspace=0.2, hspace=0.2)

X\_train = X\_train.reshape((X\_train.shape[0], 28 \* 28))

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

X\_test = X\_test.reshape((X\_test.shape[0], 28 \* 28))

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

X\_valid = X\_valid.reshape((X\_valid.shape[0], 28 \* 28))

X\_valid = X\_valid.astype("float32") / 255

# Load the Fashion MNIST dataset.

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

# The labels in the Fashion MNIST dataset are encoded as integers.

print(y\_train\_fashion[0:9])

# Convert the labels to one-hot encoding.

y\_train\_onehot = to\_categorical(y\_train\_fashion[0:9])

print(y\_train\_onehot)

Ex 5

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.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)

Ex no 6

# Generate a synthetic dataset for illustration

import numpy as np

from sklearn.datasets import make\_classification

# Generate a synthetic dataset for illustration

X, y = make\_classification(n\_samples=1000, n\_features=20, n\_informative=10,

n\_classes=2, random\_state=42)

# Rest of your code (including the RandomizedSearchCV part)

from scipy.stats import randint

from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import RandomizedSearchCV

param\_dist = {

"max\_depth": [3, None],

"max\_features": randint(1, 9),

"min\_samples\_leaf": randint(1, 9),

"criterion": ["gini", "entropy"]

}

tree = DecisionTreeClassifier()

tree\_cv = RandomizedSearchCV(tree, param\_dist, cv=5)

tree\_cv.fit(X, y)

print("Tuned Decision Tree Parameters: {}".format(tree\_cv.best\_params\_))

print("Best score is {}".format(tree\_cv.best\_score\_))