Python

**Question 1: -**

**Answer:**

def word\_frequency(input\_string):

# split the input string into words

words = input\_string.split()

# create a dictionary to store the frequency of each word

freq = {}

# iterate over each word and update the frequency in the dictionary

for word in words:

if word in freq:

freq[word] += 1

else:

freq[word] = 1

# find the word with the highest frequency

max\_freq = max(freq.values())

# return the length of the word with the highest frequency

for word, freq in freq.items():

if freq == max\_freq:

return len(word)

**Question 2:** -

**Answer**: def is\_valid(s):

# create a frequency map to keep track of the characters and their frequencies

freq = {}

# iterate over each character in the string and update the frequency map

for c in s:

if c in freq:

freq[c] += 1

else:

freq[c] = 1

# check if the frequency map has any characters with frequency less than 1

for \_, v in freq.items():

if v < 1:

return False

# if all characters have frequency greater than or equal to 1, the string is valid

return True

**Question 3:**

**Answer:**

import requests

import json

import openpyxl

# Download the data from the provided link

response=requests.get('https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json.in')

# Read the data from the response

data = response.text

# Convert the JSON data into a Python dictionary

pokedex\_data = json.loads(data)

# Create a new Excel workbook and worksheet

workbook = openpyxl.Workbook()

worksheet = workbook.active

# Set the column headers

worksheet['A1'] = 'Pokemon Number'

worksheet['B1'] = 'Pokemon Name'

worksheet['C1'] = 'Pokemon Type 1'

worksheet['D1'] = 'Pokemon Type 2'

worksheet['E1'] = 'Total'

worksheet['F1'] = 'HP'

worksheet['G1'] = 'Attack'

worksheet['H1'] = 'Defense'

worksheet['I1'] = 'Sp. Attack'

worksheet['J1'] = 'Sp. Defense'

worksheet['K1'] = 'Speed'

# Loop through each pokemon in the pokedex data and write it to the Excel worksheet

for index, pokemon in enumerate(pokedex\_data):

# Get the pokemon data

number = pokemon['num']

name = pokemon['name']

type\_1 = pokemon['type\_1']

type\_2 = pokemon['type\_2']

total = pokemon['total']

hp = pokemon['hp']

attack = pokemon['attack']

defense = pokemon['defense']

sp\_attack = pokemon['sp\_attack']

sp\_defense = pokemon['sp\_defense']

speed = pokemon['speed']

# Write the data to the worksheet

worksheet.append([number, name, type\_1, type\_2, total, hp, attack, defense, sp\_attack, sp\_defense, speed])

# Set the column widths

worksheet.column\_dimensions['A'].width = 15

worksheet.column\_dimensions['B'].width = 30

worksheet.column\_dimensions['C'].width = 15

worksheet.column\_dimensions['D'].width = 15

worksheet.column\_dimensions['E'].width = 10

worksheet.column\_dimensions['F'].width = 10

worksheet.column\_dimensions['G'].width = 10

worksheet.column\_dimensions['H'].width = 10

worksheet.column\_dimensions['I'].width = 15

worksheet.column\_dimensions['J'].width = 15

worksheet.column\_dimensions['K'].width = 1

**Question 4:**

Answer:

import requests

import json

import csv

# Download the data from the given URL

response = requests.get('https://data.nasa.gov/resource/y77d-th95.json')

# Check if the request was successful

if response.status\_code == 200:

# Load the JSON data into a variable

data = response.json()

# Create a new list to hold the converted data

converted\_data = []

# Loop through each row in the JSON data

for row in data:

# Create a new dictionary to hold the converted data for this row

converted\_row = {}

# Convert each column in the row into the proper data type

converted\_row['name'] = row['name']

converted\_row['height'] = float(row['height'])

converted\_row['mass'] = float(row['mass'])

converted\_row['fall'] = row['fall'] == 'true'

converted\_row['year'] = int(row['year'])

# Add the converted row to the list of converted data

converted\_data.append(converted\_row)

# Create a new CSV file and write the converted data to it

with open('planets.csv', 'w', newline='') as csv\_file:

csv\_writer = csv.writer(csv\_file)

csv\_writer.writerow(['name', 'height', 'mass', 'fall', 'year'])

csv\_writer.writerows(converted\_data)

**Question 5:**

**Answer:**

import requests

import json

# API endpoint

url = "http://api.tvmaze.com/singlesearch/shows?q=westworld&embed=episodes"

# Send a GET request to the API endpoint

response = requests.get(url)

# Check if the request was successful

if response.status\_code == 200:

# Load the response data into a variable

data = response.json()

# Print the name of the TV show

print("TV Show: " + data["name"])

# Loop through each episode in the "episodes" array

for episode in data["\_embedded"]["episodes"]:

# Print the name of the episode

print("Episode: " + episode["name"])

# Print the air date of the episode

print("Air Date: " + episode["airdate"])

# Print the season number of the episode

print("Season: " + episode["season"])

# Print the episode number of the episode

print("Number: " + episode["number"])

# Print a separator to separate the data for each episode

print("---")

# If the request was not successful, print an error message

else:

print("Error: " )

**Question 10:**

**Answer:**

**import re**

**# Define a function to count the number of occurrences of a word in a phrase**

**def count\_word(phrase, word):**

**# Use the re.findall function to find all occurrences of the word in the phrase**

**matches = re.findall(r'\b' + word + r'\b', phrase)**

**# Return the number of matches**

**return len(matches)**

**# Get the phrase from the user**

**phrase = input("Enter a phrase: ")**

**# Count the number of occurrences of each word type**

**nouns = count\_word(phrase, "noun")**

**verbs = count\_word(phrase, "verb")**

**adjectives = count\_word(phrase, "adjective")**

**pronouns = count\_word(phrase, "pronoun")**

**# Print the results**

**print("\nNouns:", nouns)**

**print("Verbs:", verbs)**

**print("Adjectives:", adjectives)**

**print("Pronouns:", pronouns)**

**Statistics**

**Question 1:**

**Answer:**

A correlation coefficient of 0.7 indicates a strong positive relationship between SAT scores and college GPA. The correlation coefficient ranges from -1 to +1, where 0 represents no correlation, -1 represents a perfect negative correlation, and +1 represents a perfect positive correlation.

In this case, a correlation coefficient of 0.7 suggests that there is a strong tendency for students with higher SAT scores to have higher college GPAs, and vice versa. This positive correlation indicates that as SAT scores increase, college GPAs tend to increase as well.

However, it's important to note that correlation does not imply causation. While there is a strong association between SAT scores and college GPA, it does not necessarily mean that high SAT scores directly cause high college GPAs. Other factors, such as study habits, motivation, and personal circumstances, can also influence a student's academic performance in college**.**

**Question 2:**

**Answer:**

a. To find the percentage of individuals with heights between 160 cm and 180 cm, we need to calculate the z-scores for these values and then find the corresponding area under the standard normal distribution curve.

The z-score formula is given by:

z = (x - μ) / σ

Where:

x = the value (height in this case)

μ = the mean height

σ = the standard deviation

For 160 cm:

z1 = (160 - 170) / 10 = -1

For 180 cm:

z2 = (180 - 170) / 10 = 1

Using a standard normal distribution table or a statistical software, we can find the area under the curve between z1 and z2.

The area between z1 and z2 represents the percentage of individuals with heights between 160 cm and 180 cm.

b. To find the probability that the average height of a random sample of 100 individuals is greater than 175 cm, we need to calculate the standard error of the mean (SEM) and then find the corresponding area under the standard normal distribution curve.

The SEM formula is given by:

SEM = σ / sqrt(n)

Where:

σ = the standard deviation

n = the sample size

For n = 100, we have:

SEM = 10 / sqrt(100) = 1

To find the probability that the average height is greater than 175 cm, we need to find the area to the right of 175 cm under the normal distribution curve using the SEM.

c. To find the z-score corresponding to a height of 185 cm, we can use the z-score formula:

z = (x - μ) / σ

Where:

x = the value (height in this case)

μ = the mean height

σ = the standard deviation

For x = 185 cm, μ = 170 cm, and σ = 10 cm, we can calculate the z-score.

d. To find the approximate height corresponding to the threshold of 5% of the dataset, we can use the z-score formula and the standard normal distribution table.

We need to find the z-score that corresponds to an area of 0.05 in the left tail of the standard normal distribution. Using the z-score, we can calculate the height.

e. The coefficient of variation (CV) is a measure of relative variability and is calculated as the ratio of the standard deviation to the mean, expressed as a percentage.

CV = (σ / μ) \* 100

Where:

σ = the standard deviation

μ = the mean height

f. Skewness is a measure of the asymmetry of a distribution. A skewness of approximately zero indicates that the dataset is approximately symmetric.

Skewness can be calculated using the formula:

Skewness = (3 \* (Mean - Median)) / Standard Deviation

If the skewness is positive, it means the tail of the distribution is skewed to the right. If the skewness is negative, it means the tail is skewed to the left. A skewness close to zero indicates that the distribution is approximately symmetric.

**Question 4:**

**Answer:**

The favorable outcomes in this case are 1, 4, 9, and 16, which are the perfect square numbers between 1 and 20.

The total number of possible outcomes is 20 because there are 20 slips of paper in the hat.

Therefore, the probability is calculated as:

Probability = Number of Favorable Outcomes / Total Number of Possible Outcomes

Probability = 4 / 20

Probability = 1 / 5

So, the probability that the number on the slip of paper is a perfect square is 1/5 or 0.2, which is equivalent to 20%.

**Question 5:**

**Answer:**

Let's define the events as follows:

A: The taxi belongs to Company A.

B: The taxi is late.

We need to find P(A|B), which represents the probability that the taxi belongs to Company A given that it is late.

According to Bayes' theorem:

P(A|B) = (P(B|A) \* P(A)) / P(B)

P(B|A) is the probability of the taxi being late given that it belongs to Company A. In this case, it is 1 - the success rate of Company A, which is 1 - 0.95 = 0.05.

P(A) is the probability of selecting a taxi from Company A, which is given as 80% or 0.8.

P(B) is the probability of the taxi being late, regardless of the company it belongs to. We can calculate this by taking the weighted average of the probabilities of being late for both companies.

P(B) = P(B|A) \* P(A) + P(B|~A) \* P(~A)

P(B|~A) is the probability of the taxi being late given that it does not belong to Company A. In this case, it is 1 - the success rate of Company B, which is 1 - 0.90 = 0.10.

P(~A) is the probability of selecting a taxi from Company B, which is given as 20% or 0.2.

P(B) = (0.05 \* 0.8) + (0.10 \* 0.2)

Now we can substitute these values into Bayes' theorem to find P(A|B):

P(A|B) = (0.05 \* 0.8) / [(0.05 \* 0.8) + (0.10 \* 0.2)]

Calculating the expression:

P(A|B) = 0.04 / (0.04 + 0.02)

Simplifying the expression:

P(A|B) = 0.04 / 0.06

P(A|B) = 2/3 or approximately 0.67

Therefore, the probability that a randomly selected late taxi belongs to Company A is 2/3 or approximately 0.67, which is equivalent to 67%.

**Question 10:**

**Answer:**

The binomial distribution formula is given by:

P(X = k) = C(n, k) \* p^k \* (1 - p)^(n - k)

Where:

P(X = k) is the probability of having exactly k successes (defective bulbs).

C(n, k) is the number of combinations of n items taken k at a time.

p is the probability of success (defective bulb).

n is the number of trials (number of light bulbs in this case).

k is the number of successful trials (number of defective bulbs in this case).

(1 - p) is the probability of failure (non-defective bulb).

a. To find the probability that exactly 20 bulbs are defective, we can use the binomial distribution formula:

P(X = 20) = C(500, 20) \* (0.05)^20 \* (1 - 0.05)^(500 - 20)

b. To find the probability that at least 10 bulbs are defective, we need to calculate the probability of having 10, 11, 12, ..., up to 500 defective bulbs and sum up all these probabilities:

P(X ≥ 10) = P(X = 10) + P(X = 11) + ... + P(X = 500)

c. To find the probability that at most 15 bulbs are defective, we need to calculate the probability of having 0, 1, 2, ..., up to 15 defective bulbs and sum up all these probabilities:

P(X ≤ 15) = P(X = 0) + P(X = 1) + ... + P(X = 15)

d. On average, the number of defective bulbs you would expect in a batch of 500 can be calculated using the expected value (mean) of a binomial distribution:

E(X) = n \* p

Where:

E(X) is the expected value of X (number of defective bulbs).

n is the number of trials (number of light bulbs in this case).

p is the probability of success (defective bulb).

E(X) = 500 \* 0.05

**Deep Learning**

**Question1:**

**Answer:**

import tensorflow as tf

from tensorflow.keras import layers

# Load and preprocess the MNIST dataset

mnist = tf.keras.datasets.mnist

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

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

x\_train = x\_train[..., tf.newaxis]

x\_test = x\_test[..., tf.newaxis]

# Set the desired accuracy threshold

accuracy\_threshold = 0.96

# Function to create a CNN model

def create\_model():

model = tf.keras.Sequential()

model.add(layers.Conv2D(16, (3, 3), activation='relu', input\_shape=(28, 28, 1)))

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

model.add(layers.Flatten())

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

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

return model

# Function to train and evaluate the model

def train\_and\_evaluate\_model(model):

model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

model.fit(x\_train, y\_train, epochs=10, verbose=0)

\_, test\_accuracy = model.evaluate(x\_test, y\_test, verbose=0)

return test\_accuracy

# Architecture 1

model\_1 = create\_model()

accuracy\_1 = train\_and\_evaluate\_model(model\_1)

print("Architecture 1 accuracy:", accuracy\_1)

# Architecture 2

model\_2 = create\_model()

model\_2.add(layers.Dense(64, activation='relu'))

accuracy\_2 = train\_and\_evaluate\_model(model\_2)

print("Architecture 2 accuracy:", accuracy\_2)

# Architecture 3

model\_3 = create\_model()

model\_3.add(layers.Dense(128, activation='relu'))

accuracy\_3 = train\_and\_evaluate\_model(model\_3)

print("Architecture 3 accuracy:", accuracy\_3)

# Create a comparison table

print("Architecture\t\tAccuracy")

print("=================================")

print("Architecture 1\t\t{:.2f}%".format(accuracy\_1 \* 100))

print("Architecture 2\t\t{:.2f}%".format(accuracy\_2 \* 100))

print("Architecture 3\t\t{:.2f}%".format(accuracy\_3 \* 100))

**Question2:**

**Answer:**

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

# Set the device (GPU if available, else CPU)

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

# Define the transforms for data preprocessing

transform = transforms.Compose([

transforms.RandomHorizontalFlip(),

transforms.RandomCrop(32, padding=4),

transforms.ToTensor(),

transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))

])

# Load and preprocess the CIFAR-10 dataset

trainset = torchvision.datasets.CIFAR10(root='./data', train=True, download=True, transform=transform)

testset = torchvision.datasets.CIFAR10(root='./data', train=False, download=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch\_size=128, shuffle=True, num\_workers=2)

testloader = torch.utils.data.DataLoader(testset, batch\_size=128, shuffle=False, num\_workers=2)

# Set the desired accuracy threshold

accuracy\_threshold = 0.96

# Function to train and evaluate the model

def train\_and\_evaluate\_model(model, criterion, optimizer, epochs):

model.to(device)

for epoch in range(epochs):

running\_loss = 0.0

correct = 0

total = 0

for i, (inputs, labels) in enumerate(trainloader, 0):

inputs, labels = inputs.to(device), labels.to(device)

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

\_, predicted = outputs.max(1)

total += labels.size(0)

correct += predicted.eq(labels).sum().item()

train\_loss = running\_loss / len(trainloader)

train\_accuracy = correct / total

test\_loss, test\_accuracy = evaluate\_model(model)

print('Epoch:', epoch + 1)

print('Train Loss: {:.3f} | Train Accuracy: {:.2f}%'.format(train\_loss, train\_accuracy \* 100))

print('Test Loss: {:.3f} | Test Accuracy: {:.2f}%\n'.format(test\_loss, test\_accuracy \* 100))

if test\_accuracy >= accuracy\_threshold:

print('Desired accuracy reached. Stopping training.')

break

def evaluate\_model(model):

model.eval()

test\_loss = 0.0

correct = 0

total = 0

with torch.no\_grad():

for inputs, labels in testloader:

inputs, labels = inputs.to(device), labels.to(device)

outputs = model(inputs)

loss = criterion(outputs, labels)

test\_loss += loss.item()

\_, predicted = outputs.max(1)

total += labels.size(0)

correct += predicted.eq(labels).sum().item()

test\_loss /= len(testloader)

test\_accuracy = correct / total

return test\_loss, test\_accuracy

# Define the CNN architectures

class Net1(nn.Module):

def \_init\_(self):

super(Net1, self).\_init\_()

self.conv1 = nn.Conv2d(3, 16, kernel\_size=3, stride=1, padding=1)

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.fc1 = nn.Linear(16 \* 16 \* 16, 64)

self.fc2 = nn.Linear(64, 10)

def forward(self, x):

x = self.pool(torch.relu(self.conv1(x)))

x = x.view(-1, 16 \* 16 \* 16)

x = torch.relu(self.fc1(x))

x = self.fc2(x)

return x

class Net2(nn.Module):

def \_init\_(self):

super(Net2, self).\_init\_()

self.conv1 = nn.Conv2d(3, 8, kernel\_size=3, stride=1, padding=1)

self.conv2 = nn.Conv2d(8, 16, kernel\_size=3, stride=1, padding=1)

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.fc1 = nn.Linear(16 \* 8 \* 8, 64)

self.fc2 = nn.Linear(64, 10)

def forward(self, x):

x = self.pool(torch.relu(self.conv1(x)))

x = self.pool(torch.relu(self.conv2(x)))

x = x.view(-1, 16 \* 8 \* 8)

x = torch.relu(self.fc1(x))

x = self.fc2(x)

return x

class Net3(nn.Module):

def \_init\_(self):

super(Net3, self).\_init\_()

self.conv1 = nn.Conv2d(3, 8, kernel\_size=3, stride=1, padding=1)

self.conv2 = nn.Conv2d(8, 16, kernel\_size=3, stride=1, padding=1)

self.conv3 = nn.Conv2d(16, 32, kernel\_size=3, stride=1, padding=1)

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.fc1 = nn.Linear(32 \* 4 \* 4, 128)

self.fc2 = nn.Linear(128, 10)

def forward(self, x):

x = self.pool(torch.relu(self.conv1(x)))

x = self.pool(torch.relu(self.conv2(x)))

x = self.pool(torch.relu(self.conv3(x)))

x = x.view(-1, 32 \* 4 \* 4)

x = torch.relu(self.fc1(x))

x = self.fc2(x)

return x

class Net4(nn.Module):

def \_init\_(self):

super(Net4, self).\_init\_()

self.conv1 = nn.Conv2d(3, 8, kernel\_size=3, stride=1, padding=1)

self.conv2 = nn.Conv2d(8, 16, kernel\_size=3, stride=1, padding=1)

self.conv3 = nn.Conv2d(16, 32, kernel\_size=3, stride=1, padding=1)

self.conv4 = nn.Conv2d(32, 64, kernel\_size=3, stride=1, padding=1)

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.fc1 = nn.Linear(64 \* 2 \* 2, 256)

self.fc2 = nn.Linear(256, 10)

def forward(self, x):

x = self.pool(torch.relu(self.conv1(x)))

x = self.pool(torch.relu(self.conv2(x)))

x = self.pool(torch.relu(self.conv3(x)))

x = self.pool(torch.relu(self.conv4(x)))

x = x.view(-1, 64 \* 2 \* 2)

x = torch.relu(self.fc1(x))

x = self.fc2(x)

return x

class Net5(nn.Module):

def \_init\_(self):

super(Net5, self).\_init\_()

self.conv1 = nn.Conv2d(3, 16, kernel\_size=3, stride=1, padding=1)

self.conv2 = nn.Conv2d(16, 32, kernel\_size=3, stride=1, padding=1)

self.conv3 = nn.Conv2d(32, 64, kernel\_size=3, stride=1, padding=1)

self.conv4 = nn.Conv2d(64, 128, kernel\_size=3, stride=1, padding=1)

self.pool = nn.MaxPool2d(kernel\_size=2, stride=2)

self.fc1 = nn.Linear(128 \* 4 \* 4, 256)

self.fc2 = nn.Linear(256, 10)

def forward(self, x):

x = self.pool(torch.relu(self.conv1(x)))

x = self.pool(torch.relu(self.conv2(x)))

x = self.pool(torch.relu(self.conv3(x)))

x = self.pool(torch.relu(self.conv4(x)))

x = x.view(-1, 128 \* 4 \* 4)

x = torch.relu(self.fc1(x))

x = self.fc2(x)

return x

# Create instances of the CNN architectures

net1 = Net1()

net2 = Net2()

net3 = Net3()

net4 = Net4()

net5 = Net5()

# Define the criterion and optimizer

criterion = nn.CrossEntropyLoss()

optimizer1 = optim.SGD(net1.parameters(), lr=0.001, momentum=0.9)

optimizer2 = optim.SGD(net2.parameters(), lr=0.001, momentum=0.9)

optimizer3 = optim.SGD(net3.parameters(), lr=0.001, momentum=0.9)

optimizer4 = optim.SGD(net4.parameters(), lr=0.001, momentum=0.9)

optimizer5 = optim.SGD(net5.parameters(), lr=0.001, momentum=0.9)

# Train and evaluate each model

print('Model 1:')

train\_and\_evaluate\_model(net1, criterion, optimizer1, epochs=50)

print('Model 2:')

train\_and\_evaluate\_model(net2, criterion, optimizer2, epochs=50)

print('Model 3:')

train\_and\_evaluate\_model(net3, criterion, optimizer3, epochs=50)

print('Model 4:')

train\_and\_evaluate\_model(net4, criterion, optimizer4, epochs=50)

print('Model 5:')

train\_and\_evaluate\_model(net5, criterion, optimizer5, epochs=50)

**Question3:**

**Answer:**

import torch

import torch.nn as nn

import torch.optim as optim

from torchvision import datasets, transforms

# Set device

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# Define the CNN architecture

class Net(nn.Module):

def \_init\_(self):

super(Net, self).\_init\_()

self.conv1 = nn.Conv2d(1, 8, kernel\_size=3)

self.conv2 = nn.Conv2d(8, 16, kernel\_size=3)

self.fc1 = nn.Linear(16 \* 5 \* 5, 128)

self.fc2 = nn.Linear(128, 10)

def forward(self, x):

x = torch.relu(self.conv1(x))

x = torch.relu(self.conv2(x))

x = x.view(-1, 16 \* 5 \* 5)

x = torch.relu(self.fc1(x))

x = self.fc2(x)

return x

# Load the MNIST dataset

transform = transforms.Compose([

transforms.ToTensor(),

transforms.Normalize((0.1307,), (0.3081,))

])

train\_dataset = datasets.MNIST(root='./data', train=True, download=True, transform=transform)

test\_dataset = datasets.MNIST(root='./data', train=False, download=True, transform=transform)

# Set the batch size

batch\_size = 64

# Create data loaders

train\_loader = torch.utils.data.DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)

test\_loader = torch.utils.data.DataLoader(test\_dataset, batch\_size=batch\_size, shuffle=False)

# Create an instance of the CNN

model = Net().to(device)

# Define the loss function and optimizer

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop

num\_epochs = 10

for epoch in range(num\_epochs):

model.train()

for batch\_idx, (data, targets) in enumerate(train\_loader):

data = data.to(device)

targets = targets.to(device)

# Forward pass

scores = model(data)

loss = criterion(scores, targets)

# Backward pass and optimization

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# Validation loop

model.eval()

correct = 0

total = 0

with torch.no\_grad():

for data, targets in test\_loader:

data = data.to(device)

targets = targets.to(device)

scores = model(data)

\_, predictions = scores.max(1)

correct += (predictions == targets).sum()

total += targets.size(0)

accuracy = 100.0 \* correct / total

print(f'Epoch {epoch + 1}/{num\_epochs}, Validation Accuracy: {accuracy:.2f}%')

# Check if the minimum validation accuracy is reached

if accuracy >= 99.40:

break

TensorFlow:

python

Copy code

import tensorflow as tf

from tensorflow.keras import layers, models, optimizers

# Load the MNIST dataset

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

# Normalize and reshape the images

train\_images = train\_images.reshape(-1, 28, 28, 1) / 255.0

test\_images = test\_images.reshape(-1, 28, 28, 1) / 255.0

# Create a pure CNN model

model = models.Sequential()

model.add(layers.Conv2D(8, kernel\_size=3, activation='relu', input\_shape=(28, 28, 1)))

model.add(layers.Conv2D(16, kernel\_size=3, activation='relu'))

model.add(layers.Flatten())

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

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

# Print the model summary

model.summary()

# Compile the model

model.compile(optimizer=optimizers.Adam(lr=0.001),

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Train the model

history = model.fit(train\_images, train\_labels, epochs=10, batch\_size=64, validation\_data=(test\_images, test\_labels))

# Check the validation accuracy

val\_accuracy = history.history['val\_accuracy'][-1]

print(f'Validation Accuracy: {val\_accuracy:.2f}%')

# Check if the minimum validation accuracy is reached

if val\_accuracy >= 0.994:

print("Minimum validation accuracy reached.")

**Computer Vision**

**Question 1:**

**Answer:**

import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader

import torchvision.transforms as transforms

import torchvision.datasets as datasets

from torch.utils.tensorboard import SummaryWriter

import albumentations as A

Define the dataset and data loaders using torchvision.datasets.ImageFolder and torch.utils.data.DataLoader.

python

Copy code

# Define the transformations for data augmentation

train\_transform = transforms.Compose([

A.Resize(256, 256),

A.RandomCrop(224, 224),

A.HorizontalFlip(),

A.Normalize(),

transforms.ToTensor(),

])

test\_transform = transforms.Compose([

A.Resize(256, 256),

A.CenterCrop(224, 224),

A.Normalize(),

transforms.ToTensor(),

])

# Define the datasets and data loaders

train\_dataset = datasets.ImageFolder('path/to/train/dataset', transform=train\_transform)

train\_loader = DataLoader(train\_dataset, batch\_size=64, shuffle=True, num\_workers=4)

test\_dataset = datasets.ImageFolder('path/to/test/dataset', transform=test\_transform)

test\_loader = DataLoader(test\_dataset, batch\_size=64, shuffle=False, num\_workers=4)

Define the deep learning model architecture using PyTorch's nn.Module and customize it according to your requirements.

python

Copy code

class VegetableClassifier(nn.Module):

def \_init\_(self, num\_classes):

super(VegetableClassifier, self).\_init\_()

# Define your model architecture here

def forward(self, x):

# Implement the forward pass of your model

return x

Define the training loop and functions for training, validation, and testing.

python

Copy code

def train(model, train\_loader, optimizer, criterion, device):

# Training logic here

def validate(model, test\_loader, criterion, device):

# Validation logic here

def test(model, test\_loader, device):

# Testing logic here

Initialize the model, optimizer, criterion, device, and other required variables.

python

Copy code

# Initialize the model

model = VegetableClassifier(num\_classes=10)

# Move the model to the desired device

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

model.to(device)

# Define the optimizer and criterion

optimizer = optim.Adam(model.parameters(), lr=0.001)

criterion = nn.CrossEntropyLoss()

# Initialize the TensorBoard writer for logging

writer = SummaryWriter(log\_dir='logs')

# Define the number of epochs and other hyperparameters

num\_epochs = 10

Implement the distributed parallel training using torch.nn.DataParallel.

python

Copy code

# Wrap the model with DataParallel

model = nn.DataParallel(model)

Start the training loop and log the training progress using TensorBoard.

python

Copy code

for epoch in range(num\_epochs):

train(model, train\_loader, optimizer, criterion, device)

validation\_loss, validation\_acc = validate(model, test\_loader, criterion, device)

writer.add\_scalar('Validation Loss', validation\_loss, epoch)

writer.add\_scalar('Validation Accuracy', validation\_acc, epoch)

# Close the TensorBoard writer

writer.close()

Evaluate the trained model on the test dataset.

python

Copy code

test(model, test\_loader, device

**Question 2:**

**Answer:**

. a. To find the percentage of individuals with heights between 160 cm and 180 cm, we need to calculate the z-scores for these values and then find the corresponding area under the standard normal distribution curve.

The z-score formula is given by:

z = (x - μ) / σ

Where:

x = the value (height in this case)

μ = the mean height

σ = the standard deviation

For 160 cm:

z1 = (160 - 170) / 10 = -1

For 180 cm:

z2 = (180 - 170) / 10 = 1

Using a standard normal distribution table or a statistical software, we can find the area under the curve between z1 and z2.

The area between z1 and z2 represents the percentage of individuals with heights between 160 cm and 180 cm.

b. To find the probability that the average height of a random sample of 100 individuals is greater than 175 cm, we need to calculate the standard error of the mean (SEM) and then find the corresponding area under the standard normal distribution curve.

The SEM formula is given by:

SEM = σ / sqrt(n)

Where:

σ = the standard deviation

n = the sample size

For n = 100, we have:

SEM = 10 / sqrt(100) = 1

To find the probability that the average height is greater than 175 cm, we need to find the area to the right of 175 cm under the normal distribution curve using the SEM.

c. To find the z-score corresponding to a height of 185 cm, we can use the z-score formula:

z = (x - μ) / σ

Where:

x = the value (height in this case)

μ = the mean height

σ = the standard deviation

For x = 185 cm, μ = 170 cm, and σ = 10 cm, we can calculate the z-score.

d. To find the approximate height corresponding to the threshold of 5% of the dataset, we can use the z-score formula and the standard normal distribution table.

We need to find the z-score that corresponds to an area of 0.05 in the left tail of the standard normal distribution. Using the z-score, we can calculate the height.

e. The coefficient of variation (CV) is a measure of relative variability and is calculated as the ratio of the standard deviation to the mean, expressed as a percentage.

CV = (σ / μ) \* 100

Where:

σ = the standard deviation

μ = the mean height

f. Skewness is a measure of the asymmetry of a distribution. A skewness of approximately zero indicates that the dataset is approximately symmetric.

Skewness can be calculated using the formula:

Skewness = (3 \* (Mean - Median)) / Standard Deviation

If the skewness is positive, it means the tail of the distribution is skewed to the right. If the skewness is negative, it means the tail is skewed to the left. A skewness close to zero indicates that the distribution is approximately symmetric.