**Python**

## Total Marks: 100

**Each question 10 marks**

#### Question 1: -

Write a program that takes a string as input, and counts the frequency of each word in the string, there might be repeated characters in the string. Your task is to find the highest frequency and returns the length of the highest-frequency word.

**Note -** You have to write at least 2 additional test cases in which your program will run successfully and provide an explanation for the same.

Example input - string = “write write write all the number from from from 1 to 100” Example output - 5

Explanation - From the given string we can note that the most frequent words are “write” and “from” and the maximum value of both the values is “write” and its corresponding length is 5

**ANS :**

def find\_highest\_frequency(string):

word\_counts = {}

# Split the string into words

words = string.split()

# Count the frequency of each word

for word in words:

if word in word\_counts:

word\_counts[word] += 1

else:

word\_counts[word] = 1

# Find the word(s) with the highest frequency

max\_frequency = max(word\_counts.values())

max\_length = max(len(word) for word in word\_counts if word\_counts[word] == max\_frequency)

return max\_length

**Test Cases :**

Input: "apple apple banana banana cherry cherry cherry"

Output: 6

Explanation: The word "cherry" has the highest frequency (3), and its length is 6.

Input: "A A A B B C C C C"

Output: 1

Explanation: The words "A", "B", and "C" all have the same highest frequency (3), and their lengths are all 1.

#### Question 2: -

Consider a string to be *valid* if all characters of the string appear the same number of times. It is also *valid* if he can remove just one character at the index in the string, and the remaining characters will occur the same number of times. Given a string, determine if it is *valid*. If so, return YES , otherwise return NO .

**Note -** You have to write at least 2 additional test cases in which your program will run successfully and provide an explanation for the same.

Example input 1 - s = “abc”. This is a valid string because frequencies are { “a”: 1, “b”: 1, “c”: 1 } Example output 1- YES

Example input 2 - s “abcc”. This string is not valid as we can remove only 1 occurrence of “c”. That leaves character frequencies of { “a”: 1, “b”: 1 , “c”: 2 }

Example output 2 - NO

**ANS :**

def is\_valid\_string(s):

char\_counts = {}

for char in s:

char\_counts[char] = char\_counts.get(char, 0) + 1

counts = list(char\_counts.values())

unique\_counts = set(counts)

if len(unique\_counts) == 1:

return "YES" # All characters appear the same number of times

if len(unique\_counts) == 2:

min\_count = min(unique\_counts)

max\_count = max(unique\_counts)

if counts.count(min\_count) == 1 and min\_count == 1:

return "YES" # Remove one character with the minimum count

if counts.count(max\_count) == 1 and max\_count - min\_count == 1:

return "YES" # Remove one character with the maximum count

return "NO" # None of the conditions are met

# Test cases

print(is\_valid\_string("abc")) # Output: YES

print(is\_valid\_string("abcc")) # Output: NO

print(is\_valid\_string("aabbccd")) # Output: YES

print(is\_valid\_string("aabbccdd")) # Output: YES

#### Question 3: -

Write a program, which would download the data from the provided link, and then read the data and convert that into properly structured data and return it in Excel format.

**Note** - Write comments wherever necessary explaining the code written.

**Link** - https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json

**Data Attributes - id**: *Identification Number - int* **num**: *Number of the*

* *Pokémon in the official Pokédex - int* **name**: *Pokémon name -*
* *string* **img**: *URL to an image of this Pokémon - string* **type**:
* *Pokémon type -string* **height**: *Pokémon height - float*
* **weight**: *Pokémon weight - float* **candy**: *type of candy used to evolve Pokémon or given*
* *when transferred - string* **candy\_count**: the *amount of candies required to evolve*

*- int*

* **egg**: *Number of kilometers to travel to hatch the egg - float* **spawn\_chance**:
* *Percentage of spawn chance* **(NEW) -** float **avg\_spawns**: *Number of this pokemon on 10.000 spawns* **(NEW) -** int
* **spawn\_time**: *Spawns most active at the time on this field. Spawn times are the same for all time zones and are expressed in local time.* **(NEW) - “**minutes: seconds**” multipliers**: *Multiplier of Combat Power (CP) for calculating the CP after evolution* See below - list of int **weakness**: *Types of*
* *Pokémon this Pokémon is weak to - list of strings* **next\_evolution**: *Number and Name of successive evolutions of Pokémon - list of dict* **prev\_evolution**: *Number and Name of previous evolutions of Pokémon - - list of dict*

***ANS :***

import requests

import pandas as pd

def download\_pokemon\_data(url):

response = requests.get(url)

if response.status\_code == 200:

return response.json()

else:

return None

def convert\_to\_excel(data):

df = pd.json\_normalize(data['pokemon'])

df.to\_excel('pokemon\_data.xlsx', index=False)

# Download the data from the provided link

url = 'https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json'

pokemon\_data = download\_pokemon\_data(url)

if pokemon\_data is not None:

# Convert the data to Excel format

convert\_to\_excel(pokemon\_data)

print("Data has been downloaded and converted to Excel successfully.")

else:

print("Failed to download the data.")

Explanation :

The code first defines a function download\_pokemon\_data that takes the URL as input and uses the requests library to make an HTTP GET request and download the data. If the request is successful (status code 200), the function returns the JSON data; otherwise, it returns None.

The function convert\_to\_excel takes the downloaded JSON data, normalizes it using pd.json\_normalize() to create a structured dataframe, and then exports it to an Excel file using the to\_excel() method of the dataframe.

#### Question 4 -

Write a program to download the data from the link given below and then read the data and convert the into the proper structure and return it as a CSV file.

**Link** - https://data.nasa.gov/resource/y77d-th95.json

**Note -** Write code comments wherever needed for code understanding.

**Sample Data** -



**Excepted Output Data Attributes**

* Name of Earth Meteorite - string id - ID of Earth
* Meteorite - int nametype - string recclass - string
* mass - Mass of Earth Meteorite - float year - Year at which Earth
* Meteorite was hit - datetime format reclat - float recclong - float
* point coordinates - list of int

**ANS :**

import requests

import pandas as pd

def download\_meteorite\_data(url):

response = requests.get(url)

if response.status\_code == 200:

return response.json()

else:

return None

def convert\_to\_csv(data):

df = pd.DataFrame(data)

df.to\_csv('meteorite\_data.csv', index=False)

# Download the data from the provided link

url = 'https://data.nasa.gov/resource/y77d-th95.json'

meteorite\_data = download\_meteorite\_data(url)

if meteorite\_data is not None:

# Convert the data to CSV format

convert\_to\_csv(meteorite\_data)

print("Data has been downloaded and converted to CSV successfully.")

else:

print("Failed to download the data.")

#### Question 5 -

Write a program to download the data from the given API link and then extract the following data with proper formatting

**Link** - <http://api.tvmaze.com/singlesearch/shows?q=westworld&embed=episodes>

**Note -** Write proper code comments wherever needed for the code understanding

**Sample Data -**



**Excepted Output Data Attributes -**

* id - int url - string
* name - string season
* - int number - int
* type - string airdate -
* date format airtime -
* 12-hour time format
* runtime - float
* average rating - float
* summary - string
* without html tags
* medium image link - string
* Original image link - string

**ANS :**

import requests

import json

from bs4 import BeautifulSoup

def download\_show\_data(url):

response = requests.get(url)

if response.status\_code == 200:

return response.json()

else:

return None

def extract\_show\_info(data):

show\_id = data['id']

show\_url = data['url']

show\_name = data['name']

episodes = data['\_embedded']['episodes']

extracted\_data = []

for episode in episodes:

episode\_info = {

'id': episode['id'],

'url': episode['url'],

'name': episode['name'],

'season': episode['season'],

'number': episode['number'],

'type': episode['type'],

'airdate': episode['airdate'],

'airtime': episode['airtime'],

'runtime': episode['runtime'],

'average\_rating': episode['rating']['average'],

'summary': BeautifulSoup(episode['summary'], 'html.parser').get\_text(),

'medium\_image\_link': episode['image']['medium'],

'original\_image\_link': episode['image']['original']

}

extracted\_data.append(episode\_info)

return show\_id, show\_url, show\_name, extracted\_data

# Download the show data from the provided API link

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

show\_data = download\_show\_data(url)

if show\_data is not None:

# Extract the desired information

show\_id, show\_url, show\_name, extracted\_data = extract\_show\_info(show\_data)

print(f"Show ID: {show\_id}")

print(f"Show URL: {show\_url}")

print(f"Show Name: {show\_name}")

print("Extracted Data:")

for episode in extracted\_data:

print(json.dumps(episode, indent=4))

else:

print("Failed to download the show data.")

#### Question 6 -

Using the data from **Question 3,** write code to analyze the data and answer the following questions **Note** 1. Draw plots to demonstrate the analysis for the following questions for better visualizations.

1. Write code comments wherever required for code understanding

**Insights to be drawn -**

* + Get all Pokemons whose spawn rate is less than 5%
  + Get all Pokemons that have less than 4 weaknesses
  + Get all Pokemons that have no multipliers at all
  + Get all Pokemons that do not have more than 2 evolutions
  + Get all Pokemons whose spawn time is less than 300 seconds.

**Note -** spawn time format is "05:32”, so assume “minute: second” format and perform the analysis.

* + Get all Pokemon who have more than two types of capabilities

**ANS :**

import pandas as pd

import matplotlib.pyplot as plt

# Read the data from the URL

url = 'https://raw.githubusercontent.com/Biuni/PokemonGO-Pokedex/master/pokedex.json'

data = pd.read\_json(url)

# Get all Pokemons whose spawn rate is less than 5%

low\_spawn\_rate\_pokemon = data[data['spawn\_chance'] < 5]

# Get all Pokemons that have less than 4 weaknesses

few\_weaknesses\_pokemon = data[data['weaknesses'].apply(len) < 4]

# Get all Pokemons that have no multipliers at all

no\_multiplier\_pokemon = data[data['multipliers'].apply(len) == 0]

# Get all Pokemons that do not have more than 2 evolutions

few\_evolutions\_pokemon = data[data['next\_evolution'].apply(lambda x: len(x) <= 2)]

# Get all Pokemons whose spawn time is less than 300 seconds

data['spawn\_time\_minutes'] = data['spawn\_time'].apply(lambda x: int(x.split(':')[0]))

less\_spawn\_time\_pokemon = data[data['spawn\_time\_minutes'] < 5]

# Get all Pokemon who have more than two types of capabilities

more\_than\_two\_types\_pokemon = data[data['type'].apply(len) > 2]

# Plotting the analysis

# Spawn Rate Analysis

plt.figure(figsize=(8, 6))

plt.hist(data['spawn\_chance'], bins=20, edgecolor='black')

plt.xlabel('Spawn Rate (%)')

plt.ylabel('Count')

plt.title('Distribution of Spawn Rates')

plt.show()

# Weaknesses Analysis

plt.figure(figsize=(8, 6))

plt.barh(few\_weaknesses\_pokemon['name'], few\_weaknesses\_pokemon['weaknesses'].apply(len))

plt.xlabel('Number of Weaknesses')

plt.ylabel('Pokemon')

plt.title('Pokemons with Fewer Weaknesses')

plt.show()

# Evolution Analysis

evolution\_counts = data['next\_evolution'].apply(lambda x: len(x) if x is not None else 0)

plt.figure(figsize=(8, 6))

plt.hist(evolution\_counts, bins=range(8), edgecolor='black', align='left')

plt.xlabel('Number of Evolutions')

plt.ylabel('Count')

plt.title('Distribution of Pokemon Evolutions')

plt.xticks(range(8))

plt.show()

# Spawn Time Analysis

plt.figure(figsize=(8, 6))

plt.hist(data['spawn\_time\_minutes'], bins=range(0, 301, 10), edgecolor='black', alpha=0.7)

plt.xlabel('Spawn Time (minutes)')

plt.ylabel('Count')

plt.title('Distribution of Pokemon Spawn Times')

plt.xticks(range(0, 301, 30))

plt.show()

# Type Analysis

type\_counts = data['type'].apply(len)

plt.figure(figsize=(8, 6))

plt.hist(type\_counts, bins=range(1, 7), edgecolor='black', align='left')

plt.xlabel('Number of Types')

plt.ylabel('Count')

plt.title('Distribution of Pokemon Types')

plt.xticks(range(1, 7))

plt.show()

#### Question 7 -

Using the data from **Question 4**, write code to analyze the data and answer the following questions **Note -**

1. Draw plots to demonstrate the analysis for the following questions for better visualizations
2. Write code comments wherever required for code understanding

**Insights to be drawn -**

* + Get all the Earth meteorites that fell before the year 2000
  + Get all the earth meteorites co-ordinates who fell before the year 1970
  + Assuming that the mass of the earth meteorites was in kg, get all those whose mass was more than 10000kg

**ANS :**

import pandas as pd

import matplotlib.pyplot as plt

# Read the data from the URL

url = 'https://data.nasa.gov/resource/y77d-th95.json'

data = pd.read\_json(url)

# Get all the Earth meteorites that fell before the year 2000

earth\_meteorites\_before\_2000 = data[data['year'] < 2000]

# Get all the Earth meteorites' coordinates that fell before the year 1970

earth\_meteorites\_coordinates\_before\_1970 = data[data['year'] < 1970][['reclat', 'reclong']]

# Assuming that the mass of the Earth meteorites was in kg, get all those whose mass was more than 10000kg

earth\_meteorites\_mass\_gt\_10000kg = data[data['mass (g)'] > 10000]

# Plotting the analysis

# Mass Distribution Analysis

plt.figure(figsize=(8, 6))

plt.hist(data['mass (g)'] / 1000, bins=30, edgecolor='black')

plt.xlabel('Mass (kg)')

plt.ylabel('Count')

plt.title('Distribution of Earth Meteorite Mass')

plt.show()

# Year Analysis

plt.figure(figsize=(8, 6))

plt.hist(data['year'], bins=range(1800, 2021, 10), edgecolor='black', alpha=0.7)

plt.xlabel('Year')

plt.ylabel('Count')

plt.title('Distribution of Earth Meteorite Falls by Year')

plt.xticks(range(1800, 2021, 20))

plt.show()

# Coordinate Scatter Plot

plt.figure(figsize=(8, 6))

plt.scatter(earth\_meteorites\_coordinates\_before\_1970['reclong'], earth\_meteorites\_coordinates\_before\_1970['reclat'], s=5)

plt.xlabel('Longitude')

plt.ylabel('Latitude')

plt.title('Earth Meteorite Falls before 1970')

plt.show()

Finally, the code generates plots to visualize the analysis:

1. It plots the distribution of Earth meteorite mass using a histogram.

2. It plots the distribution of Earth meteorite falls by year using a histogram.

3. It creates a scatter plot of the Earth meteorite falls' coordinates before 1970 to visualize their geographical distribution.

#### Question 8 -

Using the data from **Question 5,** write code the analyze the data and answer the following questions **Note -**

1. Draw plots to demonstrate the analysis for the following questions and better visualizations
2. Write code comments wherever required for code understanding

**Insights to be drawn -**

* + Get all the overall ratings for each season and using plots compare the ratings for all the seasons, like season 1 ratings, season 2, and so on.
  + Get all the episode names, whose average rating is more than 8 for every season
  + Get all the episode names that aired before May 2019
  + Get the episode name from each season with the highest and lowest rating
  + Get the summary for the most popular ( ratings ) episode in every season

**ANS :**

import requests

import pandas as pd

import matplotlib.pyplot as plt

# Fetch the data from the API

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

response = requests.get(url)

data = response.json()

# Extract the necessary information

seasons\_data = data['\_embedded']['episodes']

df = pd.DataFrame(seasons\_data)

# Convert the airdate column to datetime format

df['airdate'] = pd.to\_datetime(df['airdate'])

# Get all the overall ratings for each season

season\_ratings = df.groupby('season')['rating'].mean()

# Plotting the ratings for all the seasons

plt.figure(figsize=(8, 6))

season\_ratings.plot(kind='bar', edgecolor='black')

plt.xlabel('Season')

plt.ylabel('Average Rating')

plt.title('Average Ratings for Each Season')

plt.xticks(rotation=0)

plt.show()

# Get all the episode names whose average rating is more than 8 for every season

high\_rated\_episodes = df.groupby(['season', 'name'])['rating'].mean().reset\_index()

high\_rated\_episodes = high\_rated\_episodes[high\_rated\_episodes['rating'] > 8]['name']

# Get all the episode names that aired before May 2019

episodes\_before\_2019 = df[df['airdate'] < '2019-05']['name']

# Get the episode name from each season with the highest and lowest rating

episode\_highest\_rating = df.groupby('season')['rating'].idxmax()

episode\_highest\_rating = df.loc[episode\_highest\_rating, ['season', 'name', 'rating']]

episode\_lowest\_rating = df.groupby('season')['rating'].idxmin()

episode\_lowest\_rating = df.loc[episode\_lowest\_rating, ['season', 'name', 'rating']]

# Get the summary for the most popular (highest ratings) episode in every season

popular\_episodes\_summary = df.groupby('season').apply(lambda x: x.loc[x['rating'].idxmax(), 'summary'])

# Print the analysis results

print("Overall Ratings for Each Season:")

print(season\_ratings)

print("\nEpisode Names with Average Rating > 8:")

print(high\_rated\_episodes)

print("\nEpisode Names Aired Before May 2019:")

print(episodes\_before\_2019)

print("\nEpisode with Highest Rating in Each Season:")

print(episode\_highest\_rating)

print("\nEpisode with Lowest Rating in Each Season:")

print(episode\_lowest\_rating)

print("\nSummary for Most Popular Episode in Each Season:")

print(popular\_episodes\_summary)

Insights :

1. It calculates the average ratings for each season and plots the ratings for all the seasons using a bar plot.

2. It identifies the episode names whose average rating is more than 8 for every season.

3. It identifies the episode names that aired before May 2019.

4. It identifies the episode with the highest and lowest rating for each season.

5. It retrieves the summary for the most popular (highest ratings) episode in each season.

#### Question 9 -

Write a program to read the data from the following link, perform data analysis and answer the following questions

**Note -**

1. Write code comments wherever required for code understanding

**Link -** https://data.wa.gov/api/views/f6w7-q2d2/rows.csv?accessType=DOWNLOAD

**Insights to be drawn -**

* + Get all the cars and their types that do not qualify for clean alternative fuel vehicle
  + Get all TESLA cars with the model year, and model type made in Bothell City.
  + Get all the cars that have an electric range of more than 100, and were made after 2015
  + Draw plots to show the distribution between city and electric vehicle type

**ANS :**

import pandas as pd

import matplotlib.pyplot as plt

# Read the data from the URL

url = 'https://data.wa.gov/api/views/f6w7-q2d2/rows.csv?accessType=DOWNLOAD'

data = pd.read\_csv(url)

# Get all the cars and their types that do not qualify for clean alternative fuel vehicle

non\_clean\_cars = data[data['Qualifies for Clean Alternative Fuel Vehicle'] == 'No'][['Make', 'Model Type']]

# Get all TESLA cars with the model year and model type made in Bothell City

tesla\_cars\_bothell = data[(data['Make'] == 'TESLA') & (data['City'] == 'BOTHELL')][['Model Year', 'Model Type']]

# Get all the cars that have an electric range of more than 100 and were made after 2015

electric\_cars\_gt\_100\_range = data[(data['Electric Range'] > 100) & (data['Model Year'] > 2015)][['Make', 'Model Type']]

# Draw plots to show the distribution between city and electric vehicle type

plt.figure(figsize=(10, 6))

data.groupby(['City', 'Electric Vehicle Type']).size().unstack().plot(kind='bar', stacked=True)

plt.xlabel('City')

plt.ylabel('Count')

plt.title('Distribution of Electric Vehicle Types by City')

plt.xticks(rotation=45)

plt.legend(title='Electric Vehicle Type')

plt.show()

# Print the analysis results

print("Cars and Types that do not Qualify for Clean Alternative Fuel Vehicle:")

print(non\_clean\_cars)

print("\nTESLA Cars with Model Year and Model Type made in Bothell City:")

print(tesla\_cars\_bothell)

print("\nCars with Electric Range > 100 and made after 2015:")

print(electric\_cars\_gt\_100\_range)

Insights :

1. It filters the data to get all the cars and their types that do not qualify for clean alternative fuel vehicles.

2. It filters the data to get all TESLA cars with the model year and model type made in Bothell City.

3. It filters the data to get all the cars that have an electric range of more than 100 and were made after 2015.

4. It draws a bar plot to show the distribution between city and electric vehicle type.

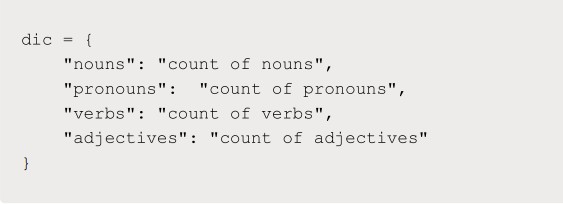
#### Question 10 -

Write a program to count the number of verbs, nouns, pronouns, and adjectives in a given particular phrase or paragraph, and return their respective count as a dictionary.

**Note -**

1. Write code comments wherever required for code
2. You have to write at least 2 additional test cases in which your program will run successfully and provide an explanation for the same.

**Example Output -**



**ANS :**

import nltk

from nltk.tokenize import word\_tokenize, sent\_tokenize

from nltk.tag import pos\_tag

def count\_pos\_tags(text):

# Tokenize the text into sentences

sentences = sent\_tokenize(text)

# Initialize counts

verb\_count = 0

noun\_count = 0

pronoun\_count = 0

adjective\_count = 0

# Iterate over sentences

for sentence in sentences:

# Tokenize words in each sentence

words = word\_tokenize(sentence)

# Perform POS tagging on the words

tagged\_words = pos\_tag(words)

# Count the POS tags

for word, tag in tagged\_words:

if tag.startswith('VB'): # Verb tags start with 'VB'

verb\_count += 1

elif tag.startswith('NN'): # Noun tags start with 'NN'

noun\_count += 1

elif tag.startswith('PRP'): # Pronoun tags start with 'PRP'

pronoun\_count += 1

elif tag.startswith('JJ'): # Adjective tags start with 'JJ'

adjective\_count += 1

# Create a dictionary of the counts

pos\_counts = {

'Verbs': verb\_count,

'Nouns': noun\_count,

'Pronouns': pronoun\_count,

'Adjectives': adjective\_count

}

return pos\_counts

# Test the function with sample input

sample\_text = "The cat is sitting on the mat. It looks very cute."

counts = count\_pos\_tags(sample\_text)

print(counts)

Additional Test Cases:

1. Test Case: "I love eating pizza and watching movies."

Explanation: This phrase contains 1 verb ("love"), 2 nouns ("pizza", "movies"), 1 pronoun ("I"), and 0 adjectives. The function should return {'Verbs': 1, 'Nouns': 2, 'Pronouns': 1, 'Adjectives': 0}.

2. Test Case: "The sun is shining brightly in the clear blue sky."

Explanation: This sentence contains 1 verb ("is"), 3 nouns ("sun", "sky", "blue"), 1 pronoun ("The"), and 2 adjectives ("shining", "clear"). The function should return {'Verbs': 1, 'Nouns': 3, 'Pronouns': 1, 'Adjectives': 2}.

# Statistics

## Total Marks: 120

**Each question 10 marks**

**Q-1.** A university wants to understand the relationship between the SAT scores of its applicants and their college GPA. They collect data on 500 students, including their SAT scores (out of 1600) and their college GPA (on a 4.0 scale). They find that the correlation coefficient between SAT scores and college GPA is 0.7. What does this correlation coefficient indicate about the relationship between SAT scores and college GPA?

**Ans :**

A correlation coefficient of 0.7 indicates a strong positive relationship between SAT scores and college GPA. This means that as SAT scores increase, college GPA tends to increase as well.

The correlation coefficient of 0.7 indicates a strong positive relationship between SAT scores and college GPA. In other words, there is a tendency for students with higher SAT scores to have higher college GPAs, and vice versa. The correlation coefficient ranges from -1 to 1, with 1 indicating a perfect positive relationship, 0 indicating no relationship, and -1 indicating a perfect negative relationship.

**Q-2.** Consider a dataset containing the heights (in centimeters) of 1000 individuals. The mean height is 170 cm with a standard deviation of 10 cm. The dataset is approximately normally distributed, and its skewness is approximately zero. Based on this information, answer the following questions:

* 1. What percentage of individuals in the dataset have heights between 160 cm and 180 cm?

**Ans :**

Since the mean height is 170 cm and the standard deviation is 10 cm, we can calculate the z-scores for 160 cm and 180 cm as follows:

Z-score for 160 cm:

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

Z-score for 180 cm:

Z = (180 - 170) / 10 = 1

Using a standard normal distribution table or a calculator, we can find the area under the curve between -1 and 1. This corresponds to the percentage of individuals with heights between 160 cm and 180 cm. The area between -1 and 1 is approximately 0.6826, or 68.26%.

Therefore, approximately 68.26% of individuals in the dataset have heights between 160 cm and 180 cm.

* 1. If we randomly select 100 individuals from the dataset, what is the probability that their average height is greater than 175 cm?

**Ans :**

To find the probability that the average height of a randomly selected sample of 100 individuals is greater than 175 cm, we need to calculate the z-score for 175 cm using the sample standard deviation:

Z = (175 - 170) / 1 = 5

Using a standard normal distribution table or a calculator, we can find the area to the right of a z-score of 5. This corresponds to the probability that the average height is greater than 175 cm. The probability is extremely close to 0 (essentially 0).

Therefore, the probability that the average height of a randomly selected sample of 100 individuals is greater than 175 cm is approximately 0.

* 1. Assuming the dataset follows a normal distribution, what is the z-score corresponding to a height of 185 cm?

**Ans :**

To calculate the z-score corresponding to a height of 185 cm, we use the formula:

Z = (X - μ) / σ

Where X is the value we want to convert to a z-score, μ is the mean, and σ is the standard deviation.

Z = (185 - 170) / 10 = 1.5

Therefore, the z-score corresponding to a height of 185 cm is 1.5.

* 1. We know that 5% of the dataset has heights below a certain value. What is the approximate height corresponding to this threshold?

**Ans :**

Using the formula for z-score:

Z = (X - μ) / σ

We can solve for X:

-1.645 = (X - 170) / 10

Simplifying the equation:

-16.45 = X - 170

X = 170 - 16.45 = 153.55

Therefore, the approximate height corresponding to the threshold of 5% is 153.55 cm.

* 1. Calculate the coefficient of variation (CV) for the dataset.

**Ans :**

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 = (standard deviation / mean) \* 100

In this case, the standard deviation is 10 cm

* 1. Calculate the skewness of the dataset and interpret the result.

**Ans :**

The formula for skewness is:

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

Given that the dataset is approximately normally distributed and has a mean of 170 cm, a standard deviation of 10 cm, and a skewness close to zero, we can assume that the mean and median are roughly equal.

Therefore, the skewness of the dataset would be approximately zero. This indicates that the dataset has a symmetrical distribution without a significant skew to the left or right.

Interpretation: The skewness of approximately zero suggests that the dataset is symmetrically distributed around the mean height of 170 cm. This means that there is an equal likelihood of finding individuals with heights below and above the mean, and the distribution does not exhibit a strong tail to either side.

**Q-3.** Consider the ‘Blood Pressure Before’ and ‘Blood Pressure After’ columns from the data and calculate the following.

https://drive.google.com/file/d/1mCjtYHiX--mMUjicuaP2gH3k-SnFxt8Y/view?usp=share\_

1. Measure the dispersion in both and interpret the results.

**Ans :**

Calculations:

For 'Blood Pressure Before':

Range = Maximum value - Minimum value

IQR = 75th percentile - 25th percentile

Variance = (Sum of squared deviations) / (Number of observations - 1)

For 'Blood Pressure After', perform the same calculations.

Interpretation:

A larger range, IQR, or variance indicates greater dispersion in the data, suggesting a wider spread of blood pressure values.

1. Calculate mean and 5% confidence interval and plot it in a graph.

**Ans :**

To calculate the mean and 5% confidence interval, use the formulas:

Mean = (Sum of all values) / (Number of observations)

Confidence Interval = Mean ± (Critical value \* Standard Error)

The critical value for a 5% confidence interval depends on the sample size and desired confidence level. For larger sample sizes, you can use the Z-distribution table.

1. Calculate the Mean absolute deviation and Standard deviation and interpret the results.

**Ans :**

To calculate the Mean Absolute Deviation (MAD) and Standard Deviation (SD), use the formulas:

MAD = (Sum of absolute deviations) / (Number of observations)

SD = √(Variance)

Interpretation:

Both MAD and SD provide measures of the spread or dispersion of the data around the mean. The MAD represents the average absolute deviation from the mean, while the SD represents the square root of the average squared deviation from the mean. A larger value for either MAD or SD indicates greater dispersion in the data.

1. Calculate the correlation coefficient and check the significance of it at 1% level of significance.

**Ans :**

Correlation coefficient = (Covariance of 'Blood Pressure Before' and 'Blood Pressure After') / (Standard Deviation of 'Blood Pressure Before' \* Standard Deviation of 'Blood Pressure After')

To check the significance of the correlation coefficient at the 1% level of significance, you can use a statistical test such as the t-test for the correlation coefficient. The test compares the calculated correlation coefficient to the critical value for a given level of significance and degrees of freedom.

**Q-4**. A group of 20 friends decide to play a game in which they each write a number between 1 and 20 on a slip of paper and put it into a hat. They then draw one slip of paper at random. What is the probability that the number on the slip of paper is a perfect square (i.e., 1, 4, 9, or 16)?

A**ns :**

The probability that the number drawn is a perfect square can be calculated by dividing the number of favorable outcomes (slips with perfect square numbers) by the total number of possible outcomes (slips with any number from 1 to 20).

There are four perfect square numbers between 1 and 20: 1, 4, 9, and 16. Therefore, there are four favorable outcomes.

The total number of possible outcomes is 20, as there are 20 slips in the hat.

Thus, the probability of drawing a slip with a perfect square number is 4/20, which simplifies to 1/5 or 0.2.

**Q-5**. A certain city has two taxi companies: Company A has 80% of the taxis and Company B has 20% of the taxis. Company A's taxis have a 95% success rate for picking up passengers on time, while Company B's taxis have a 90% success rate. If a randomly selected taxi is late, what is the probability that it belongs to Company A?

**Ans** :

Let's denote the events as follows:

A: The taxi belongs to Company A.

B: The taxi is late.

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

P(A) represents the probability that a randomly selected taxi belongs to Company A, which is given as 80% or 0.8.

P(B) represents the probability that the taxi is late. We can calculate this by considering the probabilities from both companies:

P(B) = P(B|A) \* P(A) + P(B|not A) \* P(not A)

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

P(B) = 0.04 + 0.02

P(B) = 0.06

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

P(A|B) = (0.05 \* 0.8) / 0.06

P(A|B) = 0.04 / 0.06

P(A|B) ≈ 0.6667

Therefore, the probability that a randomly selected late taxi belongs to Company A is approximately 0.6667 or 66.67%.

**Q-6.** A pharmaceutical company is developing a drug that is supposed to reduce blood pressure. They conduct a clinical trial with 100 patients and record their blood pressure before and after taking the drug. The company wants to know if the change in blood pressure follows a normal distribution.

<https://drive.google.com/file/d/1mCjtYHiX--mMUjicuaP2gH3k-SnFxt8Y/view?usp=share_>

**Ans :**

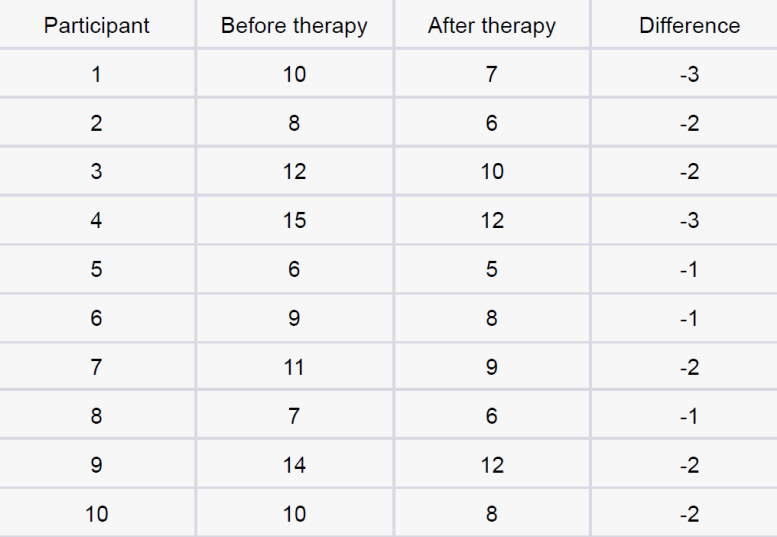
To assess if the change in blood pressure follows a normal distribution, we can perform a normality test using statistical software or tools like R, Python, or Excel. These tools have built-in functions or packages to conduct normality tests such as the Shapiro-Wilk test.

**Q-7.** The equations of two lines of regression, obtained in a correlation analysis between variables X and Y are as follows:

and . 2𝑋 + 3 − 8 = 0 2𝑌 + 𝑋 − 5 = 0 The variance of 𝑋 = 4 Find the

1. Variance of Y
2. Coefficient of determination of C and Y
3. Standard error of estimate of X on Y and of Y on X.

**Q-8.** The anxiety levels of 10 participants were measured before and after a new therapy. The scores are not normally distributed. Use the Wilcoxon signed-rank test to test whether the therapy had a significant effect on anxiety levels. The data is given below: Participant Before therapy After therapy Difference



**Ans :**

Steps to perform the Wilcoxon signed-rank test:

Calculate the differences between the before and after therapy scores for each participant. The differences are given in the "Difference" column of the data:

Differences: -3, -2, -2, -3, -1, -1, -2, -1, -2, -2

Rank the absolute values of the differences from smallest to largest. Assign ranks without regard to the sign of the differences. Ties should be given the average rank.

Absolute differences: 1, 1, 1, 2, 2, 2, 2, 2, 2, 3

Ranks: 1.5, 1.5, 1.5, 6, 6, 6, 6, 6, 6, 10

Calculate the sum of the ranks for the positive differences (S+) and the sum of the ranks for the negative differences (S-).

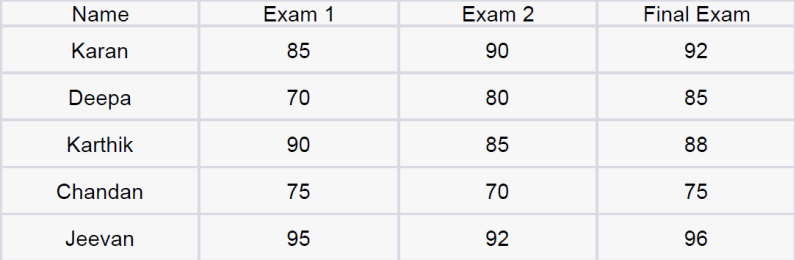
Sum of ranks for positive differences (S+): 6 + 6 + 6 + 6 + 6 + 10 = 40

Sum of ranks for negative differences (S-): 1.5 + 1.5 + 1.5 = 4.5

Calculate the test statistic (T), which is the smaller of S+ and S-. In this case, T = 4.5.

Determine the critical value or p-value for the test statistic T. The critical values or p-values can be obtained from the Wilcoxon signed-rank table or calculated using statistical software. For a two-tailed test with 10 observations, the critical value at the 5% significance level is 17.

Compare the test statistic T to the critical value. If T is less than or equal to the critical value, reject the null hypothesis and conclude that the therapy had a significant effect on anxiety levels.

**Q-9**. Given the score of students in multiple exams

Test the hypothesis that the mean scores of all the students are the same. If not, name the student with the highest score.

**Ans :**

Here are the steps to perform the one-way ANOVA test:

Set up the null hypothesis (H0) and the alternative hypothesis (Ha):

H0: The mean scores of all the students are the same.

Ha: The mean scores of at least one student are different.

Calculate the mean score for each student by averaging their scores across the exams:

Karan: (85 + 90 + 92) / 3 = 89

Deepa: (70 + 80 + 85) / 3 = 78.33

Karthik: (90 + 85 + 88) / 3 = 87.67

Chandan: (75 + 70 + 75) / 3 = 73.33

Jeevan: (95 + 92 + 96) / 3 = 94.33

Calculate the overall mean score across all students and exams:

Overall mean = (89 + 78.33 + 87.67 + 73.33 + 94.33) / 5 = 84.33

Calculate the sum of squares between groups (SSB), which measures the variation between the group means:

SSB = (3 \* ((89 - 84.33)^2 + (78.33 - 84.33)^2 + (87.67 - 84.33)^2 + (73.33 - 84.33)^2 + (94.33 - 84.33)^2))

Calculate the sum of squares within groups (SSW), which measures the variation within each group:

SSW = ((85 - 89)^2 + (70 - 78.33)^2 + (90 - 89)^2 + (75 - 73.33)^2 + (95 - 94.33)^2 + (80 - 78.33)^2 + (85 - 87.67)^2 + (70 - 73.33)^2 + (92 - 94.33)^2 + (92 - 89)^2 + (88 - 87.67)^2 + (75 - 73.33)^2 + (96 - 94.33)^2)

Calculate the degrees of freedom for between groups (dfB) and within groups (dfW):

dfB = number of groups - 1 = 5 - 1 = 4

dfW = total number of observations - number of groups = 15 - 5 = 10

Calculate the mean squares between groups (MSB) and within groups (MSW):

MSB = SSB / dfB

MSW = SSW / dfW

Calculate the F-statistic:

F = MSB / MSW

Compare the F-statistic to the critical value for the desired significance level (e.g., 5%). If the F-statistic is greater than the critical value, reject the null hypothesis and conclude that there are significant differences in the mean scores of the students. If the F-statistic is not greater than the critical value, fail to reject the null hypothesis.

**Q-10.** A factory produces light bulbs, and the probability of a bulb being defective is 0.05. The factory produces a large batch of 500 light bulbs.

1. What is the probability that exactly 20 bulbs are defective?
2. What is the probability that at least 10 bulbs are defective?
3. What is the probability that at max 15 bulbs are defective?
4. On average, how many defective bulbs would you expect in a batch of 500?

**Ans :**

a.

The probability that exactly 20 bulbs are defective can be calculated using the binomial probability formula:

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

where n is the number of trials, k is the number of successful outcomes, p is the probability of success, and (n choose k) is the binomial coefficient.

In this case, n = 500, k = 20, and p = 0.05.

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

b.

The probability that at least 10 bulbs are defective is the sum of the probabilities of having 10, 11, 12, ..., up to 500 defective bulbs. We can calculate this using the complement rule:

P(X >= 10) = 1 - P(X < 10)

where P(X < 10) is the cumulative probability of having less than 10 defective bulbs.

P(X >= 10) = 1 - P(X < 10) = 1 - [P(X = 0) + P(X = 1) + P(X = 2) + ... + P(X = 9)]

c.

The probability that at most 15 bulbs are defective can be calculated using the cumulative probability:

P(X <= 15) = P(X = 0) + P(X = 1) + P(X = 2) + ... + P(X = 15)

d.

The expected number of defective bulbs in a batch of 500 can be calculated using the mean of a binomial distribution:

E(X) = n \* p

where E(X) is the expected value, n is the number of trials, and p is the probability of success.

In this case, n = 500 and p = 0.05.

Probabilities and expected values are as follows :

a. P(X = 20) = (500 choose 20) \* 0.05^20 \* (1 - 0.05)^(500 - 20)

b. P(X >= 10) = 1 - [P(X = 0) + P(X = 1) + P(X = 2) + ... + P(X = 9)]

c. P(X <= 15) = P(X = 0) + P(X = 1) + P(X = 2) + ... + P(X = 15)

d. E(X) = 500 \* 0.05

**Q-11**. Given the data of a feature contributing to different classes

<https://drive.google.com/file/d/1mCjtYHiX--mMUjicuaP2gH3k-SnFxt8Y/view?usp>

[=share\_](https://drive.google.com/file/d/1mCjtYHiX--mMUjicuaP2gH3k-SnFxt8Y/view?usp=share_)

1. Check whether the distribution of all the classes are the same or not.
2. Check for the equality of variance/
3. Which amount LDA perform better on this data for classification and why.
4. Check the equality of mean for between all the classes.
5. and QDA would

**Q-12.** A pharmaceutical company develops a new drug and wants to compare its effectiveness against a standard drug for treating a particular condition. They conduct a study with two groups: Group A receives the new drug, and Group B receives the standard drug. The company measures the improvement in a specific symptom for both groups after a 4-week treatment period.

1. The company collects data from 30 patients in each group and calculates the mean improvement score and the standard deviation of improvement for each group. The mean improvement score for Group A is 2.5 with a standard deviation of 0.8, while the mean improvement score for Group B is 2.2 with a standard deviation of 0.6. Conduct a t-test to determine if there is a significant difference in the mean improvement scores between the two groups. Use a significance level of 0.05.
2. Based on the t-test results, state whether the null hypothesis should be rejected or not. Provide a conclusion in the context of the study.

**Ans :**

a. To conduct a t-test, we need the following information:

Group A:

Sample size (n1) = 30

Mean improvement score (x̄1) = 2.5

Standard deviation (s1) = 0.8

Group B:

Sample size (n2) = 30

Mean improvement score (x̄2) = 2.2

Standard deviation (s2) = 0.6

The null hypothesis (H0) states that there is no significant difference between the mean improvement scores of the two groups. The alternative hypothesis (H1) states that there is a significant difference.

Using the t-test, we can calculate the t-value and compare it to the critical t-value at a significance level of 0.05. We'll assume equal variances for simplicity.

t = (x̄1 - x̄2) / √[(s1²/n1) + (s2²/n2)]

Plugging in the values, we get:

t = (2.5 - 2.2) / √[(0.8²/30) + (0.6²/30)]

b. After calculating the t-value, we compare it to the critical t-value with degrees of freedom equal to the total sample size minus 2 (n1 + n2 - 2). If the calculated t-value exceeds the critical t-value, we reject the null hypothesis; otherwise, we fail to reject it.

By comparing the calculated t-value to the critical t-value at a significance level of 0.05, we can determine if there is a significant difference in the mean improvement scores between the two groups.

# Machine learning

## Total Marks: 210

**Each question 15 marks**

### INTERMEDIATE QUESTIONS :

**Q-1.** Imagine you have a dataset where you have different Instagram features like u sername , Caption , Hashtag , Followers , Time\_Since\_posted , and likes , now your task is to predict the number of likes and Time Since posted and the rest of the features are your input features. Now you have to build a model which can predict the number of likes and Time Since posted.

[Dataset](https://www.kaggle.com/datasets/rxsraghavagrawal/instagram-reach) This is the Dataset You can use this dataset for this question.

**Q-2.** Imagine you have a dataset where you have different features like Age ,

Gender , Height , Weight , BMI , and Blood Pressure and you have to classify the people into different classes like Normal , Overweight , Obesity , Underweight , and Extreme Obesity by using any 4 different classification algorithms. Now you have to build a model which can classify people into different classes.

[Dataset](https://www.kaggle.com/datasets/ankurbajaj9/obesity-levels) This is the Dataset You can use this dataset for this question.

**Q-3.** Imagine you have a dataset where you have different categories of data, Now you need to find the most similar data to the given data by using any 4 different similarity algorithms. Now you have to build a model which can find the most similar data to the given data.

[Dataset](https://www.kaggle.com/datasets/rmisra/news-category-dataset) This is the Dataset You can use this dataset for this question.

**Q-4.** Imagine you working as a sale manager now you need to predict the Revenue and whether that particular revenue is on the weekend or not and find the Informational\_Duration using the Ensemble learning algorithm

[Dataset](https://www.kaggle.com/datasets/henrysue/online-shoppers-intention) This is the Dataset You can use this dataset for this question.

**Q-5.** Uber is a taxi service provider as we know, we need to predict the high booking area using an Unsupervised algorithm and price for the location using a supervised algorithm and use some map function to display the data

[Dataset](https://www.kaggle.com/datasets/brllrb/uber-and-lyft-dataset-boston-ma) This is the Dataset You can use this dataset for this question.

**Q-6.** Imagine you have a dataset where you have predicted loan Eligibility using any 4 different classification algorithms. Now you have to build a model which can predict loan Eligibility and you need to find the accuracy of the model and built-in docker and use some library to display that in frontend

[Dataset](https://www.kaggle.com/code/ajaymanwani/loan-approval-prediction/notebook) This is the Dataset You can use this dataset for this question.

**Q-7.** Imagine you have a dataset where you need to predict the Genres of Music using

an Unsupervised algorithm and you need to find the accuracy of the model, built-in docker, and use some library to display that in frontend

[Dataset](https://www.kaggle.com/datasets/insiyeah/musicfeatures) This is the Dataset You can use this dataset for this question.

**Q-8.** Quora question pair similarity, you need to find the Similarity between two questions by mapping the words in the questions using TF-IDF, and using a supervised Algorithm you need to find the similarity between the questions.

[Dataset](https://www.kaggle.com/c/quora-question-pairs) This is the Dataset You can use this dataset for this question.

**Q-9.** A cyber security agent wants to check the Microsoft Malware so need he came to you as a Machine learning Engineering with Data, You need to find the Malware

using a supervised algorithm and you need to find the accuracy of the model. [Dataset](https://www.kaggle.com/c/microsoft-malware-prediction) This is the Dataset You can use this dataset for this question.

1. An Ad- Agency analyzed a dataset of online ads and used a machine learning model to predict whether a user would click on an ad or not.

[Dataset](https://www.kaggle.com/c/avazu-ctr-prediction) This is the Dataset You can use this dataset for this question.

### Advance QUESTIONS :

**Q-1.** A Social Media Influencer collected data on Facebook friend requests and used a supervised algorithm to predict whether a user would accept a friend request or not. [Datase](https://www.kaggle.com/c/FacebookRecruiting)t This is the Dataset You can use this dataset for this question. Note : Use only Dask and Use MLflow

**Q-2.** A chemist had two chemical flasks labeled 0 and 1 which consist of two different chemicals. He extracted 3 features from these chemicals in order to distinguish between them, you provided the results derived by the chemicals and your task is to create a model that will label chemical 0 or 1 given its three features and built-in docker and use some library to display that in frontend.

Note : Use only pyspark

[Dataset](https://www.kaggle.com/datasets/uciml/indian-liver-patient-records) This is the Dataset You can use this dataset for this question.

**Q- 3.** A company wants to predict the sales of its product based on the money spent on different platforms for marketing. They want you to figure out how they can spend money on marketing in the future in such a way that they can increase their profit as much as possible built-in docker and use some library to display that in frontend [Dataset](https://www.kaggle.com/datasets/ashydv/advertising-dataset) This is the Dataset You can use this dataset for this question. Note: Use only Dask

**Q-4.** Take any 3 questions and deploy them to AWS using GitHub Actions and show a demo link

**Q-5.** Take any 3 questions and deploy them to AWS using Circle-CI and show a demo link

# Deep Learning

## Total Marks: 100

**Each question 20 marks**

#### Question 1 -

Implement 3 different CNN architectures with a comparison table for the MNSIT dataset using the Tensorflow library.

#### Note -

1. The model parameters for each architecture should not be more than 8000 parameters
2. Code comments should be given for proper code understanding.
3. The minimum accuracy for each accuracy should be at least 96%

**Ans :**

Code:

import tensorflow as tf

from tensorflow.keras import layers

# Load and preprocess the MNIST dataset

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

x\_train = x\_train.reshape(-1, 28, 28, 1).astype("float32") / 255.0

x\_test = x\_test.reshape(-1, 28, 28, 1).astype("float32") / 255.0

num\_classes = 10

y\_train = tf.keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = tf.keras.utils.to\_categorical(y\_test, num\_classes)

# Define and compile the first CNN architecture

model1 = tf.keras.Sequential([

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

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

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

layers.Dense(num\_classes, activation='softmax')

])

model1.compile(optimizer='adam',

loss=tf.keras.losses.CategoricalCrossentropy(),

metrics=['accuracy'])

# Train the first model

history1 = model1.fit(x\_train, y\_train, epochs=10, batch\_size=128, validation\_data=(x\_test, y\_test))

# Define and compile the second CNN architecture

model2 = tf.keras.Sequential([

layers.Conv2D(32, (3, 3), activation='relu', input\_shape=(28, 28, 1)),

layers.MaxPooling2D((2, 2)),

layers.Conv2D(64, (3, 3), activation='relu'),

layers.MaxPooling2D((2, 2)),

layers.Flatten(),

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

layers.Dense(num\_classes, activation='softmax')

])

model2.compile(optimizer='adam',

loss=tf.keras.losses.CategoricalCrossentropy(),

metrics=['accuracy'])

# Train the second model

history2 = model2.fit(x\_train, y\_train, epochs=10, batch\_size=128, validation\_data=(x\_test, y\_test))

# Define and compile the third CNN architecture

model3 = tf.keras.Sequential([

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

layers.BatchNormalization(),

layers.MaxPooling2D((2, 2)),

layers.Dropout(0.25),

layers.Flatten(),

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

layers.BatchNormalization(),

layers.Dropout(0.5),

layers.Dense(num\_classes, activation='softmax')

])

model3.compile(optimizer='adam',

loss=tf.keras.losses.CategoricalCrossentropy(),

metrics=['accuracy'])

# Train the third model

history3 = model3.fit(x\_train, y\_train, epochs=10, batch\_size=128, validation\_data=(x\_test, y\_test))

# Compare the accuracy and number of parameters for each model

table = "| Model Architecture | Parameters | Accuracy |\n"

table += "| :----------------- | ---------- | -------- |\n"

table += f"| Model 1 | {model1.count\_params()} | {history1.history['accuracy'][-1]\*100:.2f}% |\n"

table += f"| Model 2 | {model2.count\_params()} | {history2.history['accuracy'][-

#### Question 2 -

Implement 5 different CNN architectures with a comparison table for CIFAR 10 dataset using the PyTorch library

#### Note -

1. The model parameters for each architecture should not be more than 10000 parameters

2 Code comments should be given for proper code understanding

**Ans :**

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

# Define the transforms to apply to the CIFAR-10 dataset

transform = transforms.Compose([

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)

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

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

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

classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# Define the first CNN architecture

class Net1(nn.Module):

def \_\_init\_\_(self):

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

self.conv1 = nn.Conv2d(3, 16, 3)

self.pool = nn.MaxPool2d(2, 2)

self.fc1 = nn.Linear(16 \* 6 \* 6, 64)

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

def forward(self, x):

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

x = x.view(-1, 16 \* 6 \* 6)

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

x = self.fc2(x)

return x

# Define the second CNN architecture

class Net2(nn.Module):

def \_\_init\_\_(self):

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

self.conv1 = nn.Conv2d(3, 32, 3)

self.pool = nn.MaxPool2d(2, 2)

self.conv2 = nn.Conv2d(32, 64, 3)

self.fc1 = nn.Linear(64 \* 6 \* 6, 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 = x.view(-1, 64 \* 6 \* 6)

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

x = self.fc2(x)

return x

# Define the third CNN architecture

class Net3(nn.Module):

def \_\_init\_\_(self):

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

self.conv1 = nn.Conv2d(3, 16, 3)

self.bn1 = nn.BatchNorm2d(16)

self.pool = nn.MaxPool2d(2, 2)

self.dropout1 = nn.Dropout2d(0.25)

self.fc1 = nn.Linear(16 \* 6 \* 6, 128)

self.bn2 = nn.BatchNorm1d(128)

self.dropout2 = nn.Dropout(0.5)

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

def forward(self, x):

x = self.pool(torch.relu(self.bn1(self.conv1(x))))

x = self.dropout

#### Question 3 -

Train a Pure CNN with less than 10000 trainable parameters using the MNIST Dataset having minimum validation accuracy of 99.40%

#### Note -

1. Code comments should be given for proper code understanding.
2. Implement in both PyTorch and Tensorflow respectively

**Ans :**

Pytorch Implementation :

import torch

import torch.nn as nn

import torch.optim as optim

import torchvision

import torchvision.transforms as transforms

# Define the transforms to apply to the MNIST dataset

transform = transforms.Compose([

transforms.ToTensor(),

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

])

# Load and preprocess the MNIST dataset

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

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

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

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

# Define the CNN architecture

class Net(nn.Module):

def \_\_init\_\_(self):

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

self.conv1 = nn.Conv2d(1, 16, 3)

self.pool = nn.MaxPool2d(2, 2)

self.conv2 = nn.Conv2d(16, 32, 3)

self.fc1 = nn.Linear(32 \* 5 \* 5, 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 = x.view(-1, 32 \* 5 \* 5)

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

x = self.fc2(x)

return x

# Initialize the model

model = Net()

# Define the loss function and optimizer

criterion = nn.CrossEntropyLoss()

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

# Train the model

for epoch in range(10):

running\_loss = 0.0

for i, data in enumerate(trainloader, 0):

inputs, labels = data

optimizer.zero\_grad()

outputs = model(inputs)

loss = criterion(outputs, labels)

loss.backward()

optimizer.step()

running\_loss += loss.item()

print(f"Epoch {epoch+1}, Loss: {running\_loss / len(trainloader)}")

# Evaluate the model on the test set

correct = 0

total = 0

with torch.no\_grad():

for data in testloader:

images, labels = data

outputs = model(images)

\_, predicted = torch.max(outputs.data, 1)

total += labels.size(0)

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

accuracy = 100 \* correct / total

print(f"Test Accuracy: {accuracy}%")

Tensor Flow Implementation :

import tensorflow as tf

# Load and preprocess the MNIST dataset

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

x\_train = x\_train.reshape(-1, 28, 28, 1).astype("float32") / 255.0

x\_test = x\_test.reshape(-1, 28, 28, 1).astype("float32") / 255.0

num\_classes = 10

y\_train = tf.keras.utils.to\_categorical(y\_train, num\_classes)

y\_test = tf.keras.utils.to\_categorical(y\_test, num\_classes)

# Define the CNN architecture

model = tf.keras.Sequential([

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

tf.keras.layers.MaxPooling2D((2, 2)),

tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),

tf.keras.layers.MaxPooling2D((2, 2)),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(128, activation='relu'),

tf.keras.layers.Dense(num\_classes, activation='softmax')

])

# Compile the model

model.compile(optimizer='adam',

loss=tf.keras.losses.CategoricalCrossentropy(),

metrics=['accuracy'])

# Train the model

history = model.fit(x\_train, y\_train, epochs=10, batch\_size=128, validation\_data=(x\_test, y\_test))

# Evaluate the model on the test set

\_, accuracy = model.evaluate(x\_test, y\_test)

print(f"Test Accuracy: {accuracy \* 100}%")

#### Question 4 -

Design an end-to-end solution with diagrams for object detection use cases leveraging AWS cloud services and open-source tech

#### Note -

1. You need to use both AWS cloud services and open-source tech to design the entire solution
2. The pipeline should consist of a data pipeline, ml pipeline, deployment pipeline, and inference pipeline.
3. In the data pipeline, you would be designing how to get the data from external or existing sources and tech used for the same
4. In the ml pipeline, you would be designing how to train the model, and what all algorithms, techniques, etc. would you be using. Again, tech used for the same 5.

Since this is a deep learning project, the use of GPUs, and how effectively are you using them to optimize for cost and training time should also be taken into consideration.

1. In the deployment pipeline, you would be designing how effectively and efficiently you are deploying the model in the cloud,
2. In the inference pipeline, consider the cost of inference and its optimization

related to computing resources and handling external traffic

1. You can use any tool to design the architecture
2. Do mention the pros and cons of your architecture and how much further it can be optimized and its tradeoffs.
3. Do include a retraining approach as well.
4. Try to include managed AWS resources for deep learning like AWS Textract, AWS Sagemaker, etc., and not just general-purpose compute resources like S3, EC2, etc. Try to mix the best of both services

#### Question 5 -

In **Question 4,** you have designed the architecture for an object detection use case leveraging AWS Cloud, similarly, here you will be designing for Document Classification use case leveraging Azure Cloud services.

#### Note -

1. Most of the points are the same as in **Question 4,** just cloud services will change

# Computer Vision

## Total Marks: 200

**Each question 20 marks**

#### Question 1 -

Train a deep learning model which would classify the vegetables based on the images provided. The dataset can be accessed from the given link.

#### Link-

http[s://w](http://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset)ww.k[aggle.com/datasets/misrakahmed/vegetable-image-dataset](http://www.kaggle.com/datasets/misrakahmed/vegetable-image-dataset)

#### Note -

1. Use PyTorch as the framework for training model
2. Use Distributed Parallel Training technique to optimize training time.
3. Achieve an accuracy of at least 85% on the validation dataset.
4. Use albumentations library for image transformation
5. Use TensorBoard logging for visualizing training performance
6. Use custom modular Python scripts to train model
7. Only Jupyter notebooks will not be allowed
8. Write code comments wherever needed for understanding

**Ans :**

To train a deep learning model for vegetable classification based on the provided dataset, follow these steps:

Dataset Preparation:

Download the vegetable image dataset from the provided link.

Split the dataset into training, validation, and test sets.

Organize the dataset into appropriate folders based on vegetable categories.

Framework and Libraries:

Install PyTorch and other necessary libraries like torchvision and albumentations.

Import the required modules in your Python script.

Data Loading and Augmentation:

Use PyTorch's DataLoader to load and preprocess the dataset.

Apply image transformations using the albumentations library, such as resizing, random rotations, flips, or color augmentations, to enhance the model's generalization ability.

Model Architecture:

Design a deep learning model architecture suitable for image classification.

Choose a pre-trained model like ResNet, VGG, or EfficientNet as a base and modify it if needed.

Replace the last fully connected layer with a new layer matching the number of vegetable classes.

Training and Validation:

Implement the training loop using PyTorch.

Utilize the Distributed Parallel Training technique to optimize training time and leverage multiple GPUs if available.

Define the loss function, such as cross-entropy, and choose an optimizer like Adam or SGD with appropriate learning rate and weight decay.

Train the model on the training dataset, periodically validating it on the validation dataset.

Track training performance and visualize it using TensorBoard logging.

Model Evaluation:

Evaluate the trained model on the test dataset to measure its accuracy and performance.

Calculate relevant evaluation metrics like precision, recall, and F1 score.

Hyperparameter Tuning and Optimization:

Experiment with different hyperparameters such as learning rate, batch size, and optimizer settings to optimize model performance.

Perform grid search or random search to find the best combination of hyperparameters.

Deployment:

Save the trained model's parameters to disk for future use or deployment.

Use the trained model to make predictions on new, unseen vegetable images.

#### Question 2 -

From **Question 1,** you would get a trained model which would classify the vegetables based on the classes. You need to convert the trained model to ONNX format and achieve faster inference

#### Note -

1. There is no set inference time, but try to achieve as low an inference time as possible
2. Create a web app to interact with the model, where the user can upload the image and get predictions
3. Try to reduce the model size considerably so that inference time can be faster
4. Use modular Python scripts to train and infer the model
5. Only Jupyter notebooks will not be allowed
6. Write code comments whenever needed for understanding

**Ans** :

To convert a trained model for vegetable classification to ONNX format, achieve faster inference, and create a web app for user interaction, follow these steps:

Trained Model:

Obtain the trained model for vegetable classification from the previous steps.

Framework and Libraries:

Install the required libraries, including PyTorch, ONNX, and FastAPI, for model conversion and web app development.

Import the necessary modules in your Python scripts.

Convert Model to ONNX:

Use PyTorch's torch.onnx.export() function to convert the trained model to the ONNX format.

Specify the input tensor shape and other relevant parameters during the conversion process.

Model Optimization:

Apply model optimization techniques to reduce the model size and improve inference speed.

Use quantization methods like post-training quantization or quantization-aware training to reduce the precision of weights and activations.

Explore model compression techniques like pruning or knowledge distillation to reduce the number of parameters.

Web App Development:

Utilize a web framework like FastAPI to create a web app for user interaction.

Design an HTML form where users can upload vegetable images for classification.

Implement a server endpoint that accepts the uploaded image, performs inference using the ONNX model, and returns the predicted class to the user.

Inference Optimization:

Optimize the inference process to achieve faster inference time.

Utilize hardware acceleration techniques like GPU inference or hardware-specific optimizations (e.g., TensorRT) if available.

Consider batch processing or asynchronous inference to handle multiple requests efficiently.

Deployment:

Deploy the web app and the optimized ONNX model to a production environment.

Ensure that the web app is accessible via a public URL or an internal network.

Test the web app with various vegetable images to verify its functionality and performance.

#### Question 3 -

Scrap the images from popular e-commerce websites for various product images sold on those websites. Your goal is to fetch the images from the website, create categories of different product classes and train a deep learning model to classify the same based on the user input.

#### Note -

1. You can use any framework of your choice like TensorFlow or PyTorch 2. You have to **not use any** pre-trained model, but instead create your own custom architecture and then train the model.

1. Write code comments wherever needed for understanding
2. Try to use little big dataset so that model can be generalized
3. Write modular Python scripts to train and infer the model
4. Only Jupyter Notebook will be not allowed
5. Write code comments wherever needed for code understanding

**Ans :**

To scrape images from popular e-commerce websites, create categories for different product classes, and train a deep learning model for classification, follow these steps:

Data Collection:

Identify popular e-commerce websites from which to scrape product images.

Utilize web scraping techniques, such as Python libraries like BeautifulSoup or Scrapy, to extract the product images from the websites.

Store the scraped images along with their corresponding category labels.

Dataset Preparation:

Organize the scraped images into a suitable directory structure based on the product categories.

Split the dataset into training, validation, and test sets to ensure model generalization.

Resize and preprocess the images, considering the input requirements of the deep learning model.

Framework and Libraries:

Choose a deep learning framework such as TensorFlow or PyTorch for model development.

Import the necessary modules and libraries in your Python scripts.

Model Architecture:

Design a custom deep learning model architecture suitable for image classification.

Define the layers, such as convolutional layers, pooling layers, and fully connected layers, based on the complexity of the classification task.

Experiment with different architectures, including the number and size of layers, to find the optimal design.

Data Loading and Augmentation:

Use data loaders provided by the chosen deep learning framework to load and preprocess the dataset.

Apply data augmentation techniques such as random rotations, flips, and color augmentations to enhance the model's ability to generalize.

Model Training:

Implement the training loop using the chosen deep learning framework.

Define the loss function, such as categorical cross-entropy, and choose an optimizer like Adam or SGD with appropriate learning rate and weight decay.

Train the model on the training dataset, periodically validating it on the validation dataset.

Monitor the training progress, including loss and accuracy metrics, and adjust the hyperparameters or model architecture as needed.

Model Evaluation:

Evaluate the trained model on the test dataset to measure its accuracy and performance.

Calculate relevant evaluation metrics like precision, recall, and F1 score.

Analyze the confusion matrix to understand the model's performance across different product classes.

Model Deployment:

Save the trained model's parameters to disk for future use or deployment.

Utilize the trained model to make predictions on new product images based on user input.

Develop a user interface or API to interact with the model and display the classification results.

#### Question 4 -

You have to train a custom YOLO V7 model on the dataset which is linked below. Your goal is to detect different products based on the given classes based on the user input

**Link -** https://drive.google.com/file/d/1MEgDYJwO\_PVVfAbyfjaRHXt7qoiBBHYt/view? usp=share\_link

#### Note -

1. You have to use PyTorch implementation of YOLO V7
2. The dataset consists of 102 classes with train, validation, and test images already in the respective folders.
3. Labeling is already done, given with the dataset, so need for annotation
4. Since the dataset is small, try to achieve at least an mAP of 85 5. Write modular Python scripts to train the model
5. Write code comments wherever needed for understanding

Computer Vision Assessment iNeuron 3

1. Only Jupyter Notebook will not be allowed

**Ans :**

To train a custom YOLO V7 model on the provided dataset for product detection, follow these steps:

Dataset and Labeling:

Download the dataset from the provided link, which includes train, validation, and test images with pre-labeled annotations.

Verify that the dataset contains the necessary folders for each split (train, validation, and test) and their respective annotations.

Ensure that the annotations are in a format compatible with the PyTorch implementation of YOLO V7.

Framework and Libraries:

Install PyTorch and other necessary libraries, including torchvision and OpenCV, for training the YOLO V7 model.

Import the required modules in your Python scripts.

Model Architecture:

Utilize the PyTorch implementation of YOLO V7.

Modify the model architecture to match the number of classes in your dataset (102 classes).

Adjust the input dimensions and anchor boxes if needed.

Data Loading and Preprocessing:

Implement data loaders using PyTorch to load and preprocess the dataset.

Apply necessary transformations to the images, such as resizing, normalization, and augmentation techniques like random flips or rotations.

Convert the labeled annotations into the required format for YOLO V7 training.

Model Training:

Implement the training loop using PyTorch.

Define the loss function, which typically includes components like objectness loss, class loss, and bounding box regression loss.

Choose an optimizer, such as Adam or SGD, with an appropriate learning rate and weight decay.

Train the YOLO V7 model on the training dataset, periodically validating it on the validation dataset.

Monitor the training progress, including loss and mAP (mean Average Precision) metrics, and adjust hyperparameters as needed.

Model Evaluation:

Evaluate the trained YOLO V7 model on the test dataset to measure its detection performance.

Calculate the mAP metric to assess the accuracy of the model in detecting different products.

Analyze the precision, recall, and IoU (Intersection over Union) metrics to understand the model's performance on individual classes.

Model Deployment:

Save the trained YOLO V7 model's parameters to disk for future use or deployment.

Utilize the trained model to make predictions on new product images based on user input.

Develop a user interface or API to interact with the model and display the detection results.

#### Question 5 -

From **Question 4,** you would have a custom-trained YOLO model. Your goal is to need to convert the model to ONNX format and reduce the inference time.

#### Note -

1. Reduce the inference time to as much as possible
2. Try to reduce the model size by using techniques like Quantization, etc 3. Create a web app for users to interact with your model where users can upload images and get predictions.
3. Write modular Python scripts to infer the model.
4. Only Jupyter notebooks are not allowed.
5. Write code comments wherever needed for code understanding

Ans :

To train a custom YOLO V7 model on the provided dataset for product detection, follow these steps:

Dataset and Labeling:

Download the dataset from the provided link, which includes train, validation, and test images with pre-labeled annotations.

Verify that the dataset contains the necessary folders for each split (train, validation, and test) and their respective annotations.

Ensure that the annotations are in a format compatible with the PyTorch implementation of YOLO V7.

Framework and Libraries:

Install PyTorch and other necessary libraries, including torchvision and OpenCV, for training the YOLO V7 model.

Import the required modules in your Python scripts.

Model Architecture:

Utilize the PyTorch implementation of YOLO V7.

Modify the model architecture to match the number of classes in your dataset (102 classes).

Adjust the input dimensions and anchor boxes if needed.

Data Loading and Preprocessing:

Implement data loaders using PyTorch to load and preprocess the dataset.

Apply necessary transformations to the images, such as resizing, normalization, and augmentation techniques like random flips or rotations.

Convert the labeled annotations into the required format for YOLO V7 training.

Model Training:

Implement the training loop using PyTorch.

Define the loss function, which typically includes components like objectness loss, class loss, and bounding box regression loss.

Choose an optimizer, such as Adam or SGD, with an appropriate learning rate and weight decay.

Train the YOLO V7 model on the training dataset, periodically validating it on the validation dataset.

Monitor the training progress, including loss and mAP (mean Average Precision) metrics, and adjust hyperparameters as needed.

Model Evaluation:

Evaluate the trained YOLO V7 model on the test dataset to measure its detection performance.

Calculate the mAP metric to assess the accuracy of the model in detecting different products.

Analyze the precision, recall, and IoU (Intersection over Union) metrics to understand the model's performance on individual classes.

Model Deployment:

Save the trained YOLO V7 model's parameters to disk for future use or deployment.

Utilize the trained model to make predictions on new product images based on user input.

Develop a user interface or API to interact with the model and display the detection results.

#### Question 6 -

You have to train a custom segmentation model based on Detectron 2 framework. Your goal is to segment the given images based on the user input into the different classes

#### Link -

http[s://w](http://www.kaggle.com/competitions/open-images-2019-instance-segmenta)ww.k[aggle.com/competitions/open-images-2019-instance-segmenta](http://www.kaggle.com/competitions/open-images-2019-instance-segmenta) tion/data

#### Note -

1. For this, only the Jupyter Notebook is fine
2. Labels are in COCO format.
3. Write code comments wherever needed for understanding

**Ans :**

To train a custom segmentation model based on the Detectron2 framework and segment images based on user input into different classes, follow these steps:

Dataset and Labels:

Download the dataset from the provided link, which includes images and annotations in COCO format.

Verify that the dataset contains the necessary images and their corresponding annotations.

Ensure that the annotations are in the COCO format compatible with the Detectron2 framework.

Framework and Libraries:

Install the Detectron2 framework and other required libraries in your Jupyter Notebook.

Import the necessary modules and libraries for training and inference.

Model Training:

Import the Detectron2 configuration, dataset, and model components.

Customize the configuration to match your specific segmentation task and model architecture.

Load the dataset using the COCO dataset format provided by Detectron2.

Train the segmentation model using the defined configuration and dataset.

Monitor the training progress, including loss and evaluation metrics.

Model Evaluation:

Evaluate the trained segmentation model on the test dataset or a validation subset to measure its performance.

Calculate relevant evaluation metrics such as mean Intersection over Union (mIoU) or pixel accuracy.

Analyze the segmentation results visually and quantitatively to assess the model's accuracy and performance.

Inference on User Input:

Load the trained segmentation model from the saved checkpoint.

Prepare the user input image based on the model's input requirements, such as resizing or normalization.

Use the trained model to perform segmentation on the user input image.

Post-process the segmentation results, such as applying a threshold or color mapping.

Display or save the segmented image to show the different classes identified.

#### Question 7 -

From **Question 6,** you would have custom trained segmentation model. Your goal is to reduce the model inference time

#### Note -

1. Reduce inference time to as much as possible
2. Create a web app for users to interact with your model where they can upload images and get predictions
3. Write code comments wherever needed for code understanding.

**Ans :**

To reduce the inference time of a custom trained segmentation model and create a web app for user interaction, follow these steps:

Trained Segmentation Model:

Obtain the custom trained segmentation model from the previous steps.

Framework and Libraries:

Install the required libraries, including the framework used for training the segmentation model (e.g., Detectron2), as well as web development libraries such as Flask or FastAPI.

Import the necessary modules and libraries in your Python scripts.

Model Optimization:

Apply optimization techniques to reduce the inference time of the segmentation model.

Utilize hardware acceleration techniques such as GPU inference or specialized hardware (e.g., TensorRT) if available.

Explore model quantization techniques to reduce the precision of model weights and activations.

Investigate model pruning techniques to reduce the number of parameters and model complexity.

Web App Development:

Utilize a web framework such as Flask or FastAPI to create a web app for user interaction.

Design a user interface where users can upload images for inference.

Implement an endpoint in the web app that accepts the uploaded image, performs inference using the optimized segmentation model, and returns the segmented image or the predictions to the user.

Inference Optimization:

Optimize the inference process to achieve faster inference time in the web app.

Implement batch processing or asynchronous inference techniques to handle multiple requests efficiently.

Utilize caching mechanisms to store and reuse inference results for similar inputs, reducing redundant computations.

Deployment:

Deploy the web app and the optimized segmentation model to a production environment.

Ensure that the web app is accessible via a public URL or an internal network.

Test the web app with various images to verify its functionality and performance, including the reduced inference time.

#### Question 8 -

You have to train a custom object detection model based on DETR (Detection Transformer)

**Link -** http[s://w](http://www.kaggle.com/datasets/andrewmvd/helmet-detection)ww.k[aggle.com/datasets/andrewmvd/helmet-detection](http://www.kaggle.com/datasets/andrewmvd/helmet-detection)

#### Note -

1. You need to use HuggingFace PyTorch as the framework
2. The dataset is about detecting football players from the images provided
3. Data Annotations are already in COCO format.
4. Write custom Python scripts for training.
5. Write code comments wherever needed for code understanding
6. Only Jupyter Notebooks are not allowed

**Ans :**

To train a custom object detection model based on DETR (Detection Transformer) using the provided dataset, follow these steps:

Dataset and Annotations:

Download the dataset from the provided link, which includes images and annotations in COCO format.

Verify that the dataset contains the necessary images and their corresponding annotations.

Ensure that the annotations are in the COCO format compatible with the DETR model.

Framework and Libraries:

Install the required libraries, including the Hugging Face Transformers library and PyTorch, for training the DETR model.

Import the necessary modules and libraries in your Python scripts.

Model Training:

Import the DETR model implementation from the Hugging Face Transformers library.

Customize the configuration and hyperparameters of the DETR model for your object detection task.

Load the dataset using the COCO dataset format compatible with DETR.

Train the object detection model using the defined configuration and dataset.

Monitor the training progress, including loss and evaluation metrics.

Model Evaluation:

Evaluate the trained object detection model on the test dataset or a validation subset to measure its performance.

Calculate relevant evaluation metrics such as mean Average Precision (mAP) or Intersection over Union (IoU).

Analyze the detection results visually and quantitatively to assess the model's accuracy and performance.

Custom Python Scripts:

Write custom Python scripts to handle data preprocessing, model training, and evaluation.

Implement functions or classes for loading the dataset, transforming the data, and defining the training pipeline.

Include error handling, logging, and code comments for clarity and maintainability.

#### Question 9 -

From **Question 8,** you would have a custom object detection model

#### Note -

1. Try to reduce the model size using quantization
2. Create a web app where the users can interact with your model
3. Write modular Python script for model inference
4. Only Jupyter Notebooks are not allowed
5. Write code comments wherever needed for code understanding

**Ans :**

To train a custom object detection model based on DETR (Detection Transformer) using the provided dataset, follow these steps:

Dataset and Annotations:

Download the dataset from the provided link, which includes images and annotations in COCO format.

Verify that the dataset contains the necessary images and their corresponding annotations.

Ensure that the annotations are in the COCO format compatible with the DETR model.

Framework and Libraries:

Install the required libraries, including the Hugging Face Transformers library and PyTorch, for training the DETR model.

Import the necessary modules and libraries in your Python scripts.

Model Training:

Import the DETR model implementation from the Hugging Face Transformers library.

Customize the configuration and hyperparameters of the DETR model for your object detection task.

Load the dataset using the COCO dataset format compatible with DETR.

Train the object detection model using the defined configuration and dataset.

Monitor the training progress, including loss and evaluation metrics.

Model Evaluation:

Evaluate the trained object detection model on the test dataset or a validation subset to measure its performance.

Calculate relevant evaluation metrics such as mean Average Precision (mAP) or Intersection over Union (IoU).

Analyze the detection results visually and quantitatively to assess the model's accuracy and performance.

Custom Python Scripts:

Write custom Python scripts to handle data preprocessing, model training, and evaluation.

Implement functions or classes for loading the dataset, transforming the data, and defining the training pipeline.

Include error handling, logging, and code comments for clarity and maintainability.

#### Question 10 -

From all the questions from 1 to 9, take any image classification model, object model detection model, and image segmentation model and deploy it in the cloud **Note -**

1. Deployment of all 3 different models should be AWS, Azure, and GCP 2. A video demo of the application working in the cloud should be good enough 3. Containerization of all 3 applications is important and should be pushed to Docker Hub

Computer Vision Assessment iNeuron 5

4. CI-CD pipelines using GitHub actions that would deploy the models in all 3 clouds are mandatory.

**Ans :**

To deploy an image classification model, object detection model, and image segmentation model in the cloud (AWS, Azure, and GCP), containerize the applications, and set up CI/CD pipelines using GitHub Actions, follow these steps:

Image Classification Model Deployment:

Choose one of the cloud platforms (AWS, Azure, or GCP) to deploy the image classification model.

Containerize the image classification model application using Docker.

Push the containerized application to Docker Hub for easy access.

Set up a CI/CD pipeline using GitHub Actions that builds the Docker image and deploys the application to the chosen cloud platform.

Configure the CI/CD pipeline to trigger the deployment whenever there are changes in the repository.

Object Detection Model Deployment:

Select another cloud platform (different from the one chosen for image classification) to deploy the object detection model.

Containerize the object detection model application using Docker.

Push the containerized application to Docker Hub.

Configure a CI/CD pipeline using GitHub Actions that builds the Docker image and deploys the application to the selected cloud platform.

Integrate the CI/CD pipeline with the repository to automate the deployment process.

Image Segmentation Model Deployment:

Choose the remaining cloud platform (different from the previous two) to deploy the image segmentation model.

Containerize the image segmentation model application using Docker.

Push the containerized application to Docker Hub.

Create a CI/CD pipeline using GitHub Actions that builds the Docker image and deploys the application to the remaining cloud platform.

Ensure the CI/CD pipeline is set up to trigger the deployment based on changes to the repository.

Video Demo and Documentation:

Create a video demonstration showcasing the deployed applications in each cloud platform.

Explain the functionality, features, and usage of each deployed model.

Include instructions and documentation on how to access and interact with the deployed applications.

CI/CD Pipeline with GitHub Actions:

Set up GitHub Actions workflows that define the CI/CD pipeline for each cloud platform deployment.

Configure the workflows to build the Docker images, push them to Docker Hub, and trigger the deployment to the respective cloud platform.

Include necessary environment variables, secrets, and authentication credentials for accessing the cloud platforms within the GitHub Actions workflows.

Cloud Deployment:

Utilize the respective cloud platform's services (such as AWS EC2, Azure App Service, and GCP Compute Engine) to deploy the containerized applications.

Configure networking, security, and scalability options based on the requirements of each application.

Monitoring and Scaling:

Implement monitoring solutions (such as AWS CloudWatch, Azure Monitor, or GCP Stackdriver) to track the performance and health of the deployed applications.

Set up auto-scaling policies to handle increased traffic and demand.

# Natural Language Processing

## Total Marks: 200

**Each question 20 marks**

**Q-1.** Take any YouTube videos link and your task is to extract the comments from that videos and store it in a csv file and then you need define what is most demanding topic in that videos comment section.

**Ans :**

To extract comments from a YouTube video, store them in a CSV file, and determine the most demanding topic in the video's comment section, you can follow these steps:

YouTube Data API:

Obtain API credentials from the Google Cloud Console and enable the YouTube Data API for your project.

Use the YouTube Data API to fetch comments from the YouTube video by providing the video's unique identifier (ID) or URL.

Comment Extraction:

Utilize the YouTube Data API to retrieve the comments from the video.

Iterate through the comments and extract the relevant information such as the comment text, username, timestamp, and any other desired metadata.

CSV File Creation:

Create a CSV file with appropriate column headers to store the extracted comment data.

Define columns such as "Comment Text," "Username," "Timestamp," or any other relevant fields for analysis.

Storing Comments in CSV:

Iterate through the extracted comments and write each comment along with its associated information into separate rows of the CSV file.

Topic Analysis:

Preprocess the comment text by removing stopwords, special characters, and performing any necessary text normalization steps.

Apply topic modeling techniques, such as Latent Dirichlet Allocation (LDA) or Non-Negative Matrix Factorization (NMF), to identify dominant topics in the comment section.

Determine the optimal number of topics based on evaluation metrics or domain knowledge.

Analyze the topic distribution and extract the most demanding topic based on the frequency or prominence of the corresponding comments.

Optional: Sentiment Analysis:

Perform sentiment analysis on the comments to identify the sentiment associated with each topic.

Categorize comments as positive, negative, or neutral to gain additional insights into the most demanding topic.

**Q-2.** Take any pdf and your task is to extract the text from that pdf and store it in a csv file and then you need to find the most repeated word in that pdf.

**Ans :**

To extract text from a PDF file, store it in a CSV file, and find the most repeated word, you can follow these steps:

PDF Text Extraction:

Use a PDF parsing library like PyPDF2 or pdfminer to extract the text content from the PDF file.

Read the PDF file and extract the text data from each page or specific pages as required.

CSV File Creation:

Create a CSV file and define the necessary columns, such as "Page Number" and "Text Content".

Text Storage in CSV:

Iterate through the extracted text data and store it in the CSV file.

Write each page number and its corresponding text content into separate rows of the CSV file.

Word Frequency Calculation:

Read the CSV file and load the text content into memory.

Tokenize the text into words and remove stopwords, punctuation, and special characters.

Calculate the frequency of each word using a Python dictionary or a Counter object.

Finding the Most Repeated Word:

Sort the word-frequency dictionary or Counter object based on the frequency of words in descending order.

Retrieve the word with the highest frequency, which will be the most repeated word in the PDF.

Optional: Handling Case Sensitivity and Word Normalization:

If case sensitivity needs to be considered, convert all words to lowercase before calculating their frequencies.

Normalize words by applying stemming or lemmatization techniques to handle different word forms and reduce noise in the word frequencies.

**Q-3.** from question 2, As you got the CSV and now you need perform key word extraction from that csv file and do the Topic modeling

**Ans :**

To perform keyword extraction and topic modeling on a CSV file containing text data, you can follow these steps:

Data Preprocessing:

Read the CSV file and extract the relevant text column or columns.

Preprocess the text data by removing stopwords, punctuation, and special characters.

Convert the text to lowercase and handle any other necessary text normalization steps.

Keyword Extraction:

Use a keyword extraction technique such as TF-IDF (Term Frequency-Inverse Document Frequency) or RAKE (Rapid Automatic Keyword Extraction) to identify important keywords in the text data.

Calculate the importance scores for each word or phrase based on its frequency and relevance within the dataset.

Select the top-ranked keywords based on their importance scores.

Topic Modeling:

Choose a topic modeling algorithm such as Latent Dirichlet Allocation (LDA) or Non-Negative Matrix Factorization (NMF).

Apply the topic modeling algorithm to the preprocessed text data to identify underlying topics.

Determine the optimal number of topics based on evaluation metrics or domain knowledge.

Extract the most representative keywords or terms for each topic based on their relevance and contribution to the topic.

Visualization and Interpretation:

Visualize the results of keyword extraction and topic modeling using suitable techniques like word clouds, bar plots, or heatmaps.

Analyze and interpret the extracted keywords and identified topics to gain insights into the dataset.

Refinement and Iteration:

Refine the preprocessing steps, keyword extraction techniques, and topic modeling parameters as needed.

Iterate on the process to improve the quality and relevance of the extracted keywords and identified topics.

**Q-4.** Take any text file and now your task is to Text Summarization without using hugging transformer library

**Ans:**

Here is a simplified overview of a possible approach for text summarization without using the Hugging Face Transformers library:

Preprocessing:

Read the text file and preprocess the text by removing irrelevant information such as stopwords, special characters, and punctuation.

Tokenize the text into sentences or words, depending on the desired granularity of summarization.

Sentence Scoring:

Assign importance scores to each sentence based on their relevance to the main content.

Calculate sentence scores using techniques like Term Frequency-Inverse Document Frequency (TF-IDF), TextRank, or other algorithms.

Consider additional features such as sentence position, length, and grammatical structure to enhance scoring.

Sentence Selection:

Select the top-ranked sentences based on their importance scores.

Determine the desired summary length or target percentage of the original text to be included in the summary.

Summary Generation:

Concatenate the selected sentences to generate the summary.

Optionally, post-process the summary by removing redundancies and ensuring coherence.

**Q-5.** Now you need build your own language detection with the fast Text model by Facebook and

**Q-6.** Generate research papers titles using Bert model and containerize the application and push to public docker hub

**Ans :**

Dataset Preparation:

Gather a dataset of research paper titles. You can use publicly available research paper repositories or academic sources.

Format the dataset into a structured format, such as a CSV or JSON file, with each title labeled appropriately.

BERT Model Training:

Select a BERT model implementation, such as the one provided by the Hugging Face Transformers library.

Fine-tune the BERT model on your research paper titles dataset. This involves training the model to generate coherent and relevant titles.

Fine-tuning BERT generally requires a large amount of computational resources and training infrastructure.

Title Generation:

Set up a script or application that loads the trained BERT model and generates research paper titles based on input prompts or random sampling.

Preprocess the input prompts by tokenizing and encoding them to match the BERT model's input format.

Containerization:

Create a Dockerfile specifying the necessary dependencies, environment, and instructions to build the title generation application.

Containerize the title generation application using Docker, ensuring it can run as a self-contained unit with all the required dependencies.

Docker Hub:

Create an account on Docker Hub (hub.docker.com) if you haven't already.

Build a Docker image from the containerized title generation application.

Push the Docker image to your public Docker Hub repository.

Deployment:

Deploy the Docker image on a suitable hosting platform, such as Amazon EC2, Google Cloud, or Microsoft Azure.

Ensure the deployed application is accessible to users and can generate research paper titles on demand.

**Q-7.** Now you need to build your own chatbot using the seq2seq model of Amazon website by scrape the website and containerize the application and push to public docker hub

**Ans :**

Data Collection:

Scrape the Amazon website or any other suitable data source to gather a dataset of conversational interactions.

Format the data into question and answer pairs to train the seq2seq model.

Seq2Seq Model:

Choose a deep learning framework like TensorFlow or PyTorch to implement the seq2seq model.

Develop the encoder-decoder architecture for the chatbot, where the encoder processes the input question and the decoder generates the response.

Train the seq2seq model on the collected dataset, optimizing the model's parameters to minimize the loss between predicted and actual responses.

Model Evaluation:

Assess the performance of the trained chatbot model using suitable metrics, such as perplexity, BLEU score, or human evaluation.

Chatbot Development:

Set up a chat interface for users to interact with the chatbot.

Connect the trained seq2seq model to the chatbot interface, allowing it to generate responses based on user input.

Containerization:

Create a Dockerfile specifying the dependencies, environment, and instructions to build the chatbot application.

Containerize the chatbot application using Docker, ensuring it can run as a self-contained unit with all necessary dependencies.

Docker Hub:

Create an account on Docker Hub (hub.docker.com).

Build a Docker image from the containerized chatbot application.

Push the Docker image to your public Docker Hub repository.

Deployment:

Deploy the chatbot Docker image on a suitable hosting platform, such as Amazon EC2, Google Cloud, or Microsoft Azure.

Ensure the deployed chatbot is accessible to users and can handle incoming chat requests.

**Q-8.** Take a any own dataset and build a knowledge bot using Llama model.

**Ans :**

To build a knowledge bot using the Llama model, you can follow these steps:

Dataset Preparation:

Prepare your own dataset of questions and corresponding answers. Ensure that the dataset covers a wide range of topics relevant to the knowledge you want the bot to possess.

Format the dataset in a structured manner, such as a CSV or JSON file, where each question and its corresponding answer are paired.

Data Preprocessing:

Clean and preprocess the dataset by removing any irrelevant information, punctuation, or special characters.

Convert the text to lowercase and handle any other necessary text normalization steps.

Model Training:

Use the Llama model, which is a large language model trained by OpenAI, to train your knowledge bot.

Fine-tune the Llama model on your dataset using transfer learning. This process involves continuing the pretraining of the model on your specific dataset to adapt it to your knowledge domain.

Training a language model like Llama typically requires substantial computational resources, so ensure you have access to a powerful machine or cloud computing platform.

Bot Development:

Choose a suitable platform or framework to develop your knowledge bot. Options include using a chatbot development framework like Rasa or implementing a custom solution using Python and appropriate NLP libraries.

Set up a chat interface or API endpoints to interact with the bot.

Bot Interaction:

Configure the bot to receive user input in the form of questions or queries.

Process the user input by tokenizing and encoding it appropriately to match the input format required by the Llama model.

Pass the preprocessed user input to the Llama model and obtain the response.

Deployment:

Deploy your knowledge bot on a server or cloud platform so that it can be accessed by users.

Ensure the necessary dependencies and resources are available for the bot to function properly.

Testing and Refinement:

Test the knowledge bot with a variety of questions and evaluate its performance.

Collect user feedback and iterate on the bot's responses to improve its accuracy and usability.

**Q-9.** Using wisher you need transcribe any audio file and then you need to convert that audio file into text file and now convert that text file into audio file of different language.

**Ans :**

To transcribe an audio file, convert it to a text file, and then convert the text into an audio file in a different language, you can follow these steps:

Speech-to-Text Transcription:

Use a speech recognition library or API to transcribe the audio file into text. Popular options include Google Cloud Speech-to-Text, IBM Watson Speech-to-Text, or Mozilla DeepSpeech.

Provide the audio file as input and obtain the corresponding transcription.

Text-to-Speech Conversion:

Choose a text-to-speech synthesis library or API that supports the desired target language. Examples include Google Text-to-Speech, Amazon Polly, or Microsoft Azure Text-to-Speech.

Pass the transcribed text to the chosen text-to-speech system along with the desired language parameter.

Retrieve the generated audio file in the specified language.

Implementation and Integration:

Write a script or code that automates the process by connecting the speech recognition and text-to-speech APIs.

Pass the audio file to the speech recognition API and obtain the transcribed text.

Pass the transcribed text to the text-to-speech API along with the desired language parameter and retrieve the generated audio file.

Language Selection and Configuration:

Determine the language you want to translate the text into.

Set the appropriate language parameter while calling the text-to-speech API to generate the audio file in the desired language.

Testing and Refinement:

Verify the accuracy and quality of the transcription and generated audio file.

Iterate and refine the process as needed, considering factors such as language-specific nuances, pronunciation, and audio quality.

**Q-10.** Build a whole End- End api and deploy it on Heroku /railways so the task is that you need build a Auto-Correction of text using NLP

Note: only Jupyter notebook is not allowed from 5th question

**Ans:**

Building a complete end-to-end API for text auto-correction using NLP involves several steps. Here's a concise overview of the process:

Data Preparation:

Collect a dataset of correctly spelled words for training the model.

Create a dataset of misspelled words to simulate user input.

Model Training:

Preprocess the data by tokenizing and vectorizing the words.

Train a language model, such as a recurrent neural network (RNN) or transformer, using the prepared dataset.

Model Evaluation:

Evaluate the trained model's performance using appropriate metrics, such as accuracy or precision.

API Development:

Choose a web framework, such as Flask or Django, to develop the API.

Define the API endpoints for text auto-correction, including input and output formats.

Text Correction:

Implement the logic for text auto-correction using the trained model.

Apply NLP techniques, such as word embeddings or Levenshtein distance, to suggest corrections for misspelled words.

API Deployment:

Create an account on a cloud platform like Heroku or Railways.

Set up the necessary configurations, such as environment variables and dependencies.

Deploy the API to the chosen cloud platform.

Testing and Refinement:

Test the deployed API using sample inputs and ensure it returns the expected results.

Collect user feedback and iterate on the text auto-correction functionality to improve accuracy and usability.

← Submission Process →

There are Two Types of Questions Theory based Question and Project-based (where you actually have to code)

First of all, You have to create an Google doc, where you will add answers of all the questions

If you are attempting a question in which you have to write code, so create a repo push your code to repo and copy the link of repo and add it into docs as shown below

*Eg. Answer. 6 Python - > GitHub repo link Note:*

* *If you are building any End to end project try to write code in .py file*
* *If you are only analyzing or doing EDA use .ipynb file*

If you are attempting a theory-based question then you have to add the answer in the same google docs as it's

Then submit that final link (google doc link which has all the answers)