HCL Tech Chennai Internship Report

# Acknowledgment

I would like to express my sincere gratitude to HCL Technologies, Chennai, for providing me with the opportunity to undergo a one-month internship. Special thanks to my mentor who supported and helped me throughout this journey. His guidance helped me gain valuable insights into the technical world.

# Abstract

This report outlines my one-month internship experience at HCL Technologies, Chennai from 2nd June,2025 to 2nd July,2025. During the internship, I worked on Python fundamentals, Object-Oriented Programming (OOP), OpenCV basics, Machine Learning, Deep Learning models, and ONNX model comparison. Hands-on projects included Ames House Pricing prediction, handwriting classification,next word prediction using tokenization and word embeddings and building a medical dataset model which compares the various images of different people’s feet to classify if they have normal feet or ITW. Each day was structured to build on both conceptual understanding and practical implementation.

# Internship Objectives

- Strengthen foundational knowledge in Python programming.  
- Gain exposure to computer vision and OpenCV.  
- Learn machine learning and regression techniques.  
- Understand deep learning models and ONNX comparisons and work on quantization and its methods(uniform).  
- Work on real datasets and build predictive models.  
- Develop a medical classification system using ML techniques.

# Tools and Technologies Used

- Programming Language: Python  
- Libraries: OpenCV, scikit-learn, pandas, matplotlib, numpy  
- Tools: Jupyter Notebook,Google Colab , ONNX,Docker.  
- Dataset: Ames Housing, Handwriting Dataset, Medical Dataset,Student-Performance

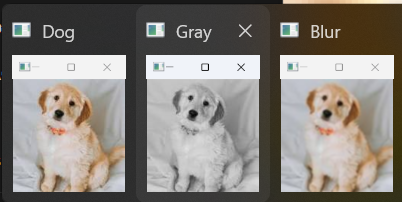
# Detailed Worklog and Learnings

## Day1

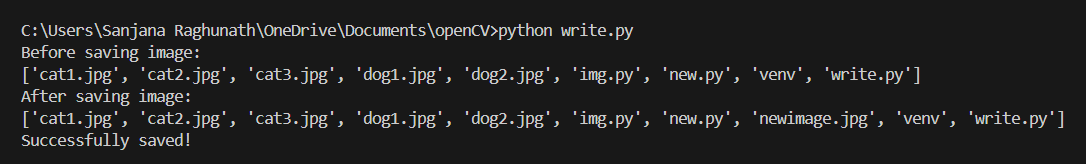
DAY1:BUILT IN PYTHON FUNCTIONS,OOPS,DECORATORS  
Basic python inbuilt functions:  
abs() function  
all()  
any()  
ascii()  
Python ascii() function returns a string containing a printable representation of an object and escapes the non-ASCII characters in the string using \x, \u or \U escapes.   
int()  
len()  
max()  
min()  
bool()  
chr()  
dict()  
divmod()  
enumerate()  
filter()  
codes:  
#absolute value  
var = float(input("Enter a number :"))  
print('Absolute value of floating point number is:', abs(var))  
#all function in lists  
l1=[]  
print(all(l1))  
l2=[1,9,0,-3]  
print(all(ele>0 for ele in l2))  
l3=[1,2,3,4]  
print(all(l3))  
#any function  
l4=[False,True,False]  
print(any(l4))  
l5=[1,0,-9,8]  
print(any(ele>0 for ele in l5))  
#ascii  
print(ascii("#"))  
s = "S a n j a n a"  
print(ascii(s))  
string='''Sanjana  
Raghunath'''  
print(ascii(string))  
#bool()  
x=bool(1)  
print(x)  
y=bool()  
print(y)  
bool(-1)  
bool(0.0)  
bool("string")  
#chr()  
num=32  
print("Character value of Number is:",chr(num))  
#dict()  
dictionary=dict(flower="rose",fruit="banana")  
print(dictionary)  
#divmod()  
print("5,10",divmod(5,10))  
print("10,3",divmod(10,3))  
#enumerate()  
list1=["my","name","is"]  
for i,name in list1:  
    print(f"index of {i}",{name})  
print(list(enumerate(list1)))  
#filter()  
def odd(n):  
    return n%2!=0  
list1=[1,2,3,4]  
list2=filter(odd,list1)  
print(list(list2))  
  
OUTPUT:  
OOPS CONCEPT BASIC CODES:  
class Animal:  
    species="Human"  
    def \_\_init\_\_(self,name,age):  
        self.name=name  
        self.age=age  
animal=Animal("Priya",30)  
animalnew=Animal("Ravi",32)  
print(animal.name)  
print(animal.age)  
print(animal.species)  
print(animalnew.age)  
print(animalnew.name)  
print(animalnew.species)  
animalnew.species="Male"  
print(animalnew.species)  
print(animal.species)  
OUTPUT:  
INHERITANCE:  
class Human:  
    species="Human"  
    def \_\_init\_\_(self,name,age,name1):  
        self.name=name  
        self.name1=name1  
        self.age=age  
    def display(self):  
        print("The name of the person is:",{self.name})  
class Men(Human):  
    def play(self):  
        print("Man's name:",self.name)  
        print("Men pay cricket.")  
class Women(Human):  
    def playtennis(self):  
        print("Woman's age and name:",self.age,self.name1)  
        print("Women play tennis")  
class children(Men,Women):  
    def playhop(self):  
        print("Child's name:",self.name)  
        print("Children play hopscotch")  
men=Men("antony","Jessi",29)  
men.display()  
men.play()  
women=Women("mark","jessica",30)  
women.display()  
women.playtennis()  
child=children("sam","san",30)  
child.display()  
child.play()  
child.playtennis()  
child.playhop()  
POLYMORPHISM:  
class Dog:  
 def speak(self):  
 return "Woof!"  
class Cat:  
 def speak(self):  
 return "Meow!"  
def animal\_sound(animal):  
 print(animal.speak())  
dog = Dog()  
cat = Cat()  
animal\_sound(dog) # Output: Woof!  
animal\_sound(cat) # Output: Meow!  
2)METHOD OVERRODING(USING INHERITANCE)  
class Animal:  
 def speak(self):  
 return "Some sound"  
class Dog(Animal):  
 def speak(self):  
 return "Bark"  
class Cat(Animal):  
 def speak(self):  
 return "Meow"  
animals = [Dog(), Cat(), Animal()]  
for animal in animals:  
 print(animal.speak())  
3)OPERATOR OVERLOADING  
class Vector:  
 def \_\_init\_\_(self, x, y):  
 self.x = x  
 self.y = y  
 def \_\_add\_\_(self, other):  
 return Vector(self.x + other.x, self.y + other.y)  
 def \_\_str\_\_(self):  
 return f"({self.x}, {self.y})"  
v1 = Vector(2, 4)  
v2 = Vector(3, 6)  
v3 = v1 + v2  
print(v3) # Output: (5, 10)  
ABSTRACTION:  
1)with one abstract method:  
from abc import ABC, abstractmethod  
class Animal(ABC):  
 @abstractmethod  
 def make\_sound(self):  
 pass  
class Dog(Animal):  
 def make\_sound(self):  
 print("Bark")  
class Cat(Animal):  
 def make\_sound(self):  
 print("Meow")  
dog = Dog()  
cat = Cat()  
dog.make\_sound() # Output: Bark  
cat.make\_sound() # Output: Meow  
with both abstract and concrete methods  
from abc import ABC, abstractmethod  
class Vehicle(ABC):  
 @abstractmethod  
 def start(self):  
 pass  
 def fuel\_type(self):  
 print("Petrol or Diesel")  
class Car(Vehicle):  
 def start(self):  
 print("Car engine started")  
car = Car()  
car.start() # Output: Car engine started  
car.fuel\_type() # Output: Petrol or Diesel  
DECORATORS:  
BASIC DECORATRS:  
def decorator\_function(original\_function):  
 def wrapper\_function():  
 print("Wrapper executed before", original\_function.\_\_name\_\_)  
 original\_function()  
 print("Wrapper executed after", original\_function.\_\_name\_\_)  
 return wrapper\_function  
@decorator\_function # This is the decorator  
def say\_hello():  
 print("Hello!")  
say\_hello()  
WITH ARGUMENTS:  
def decorator\_func(func):  
 def wrapper(\*args, \*\*kwargs):  
 print("Function is being called with:", args, kwargs)  
 return func(\*args, \*\*kwargs)  
 return wrapper  
@decorator\_func  
def add(a, b):  
 return a + b  
print(add(5, 10))  
Using functools.wraps()  
from functools import wraps  
def my\_decorator(func):  
 @wraps(func)  
 def wrapper(\*args, \*\*kwargs):  
 print("Calling", func.\_\_name\_\_)  
 return func(\*args, \*\*kwargs)  
 return wrapper  
@my\_decorator  
def greet():  
 print("Hi!")  
greet()

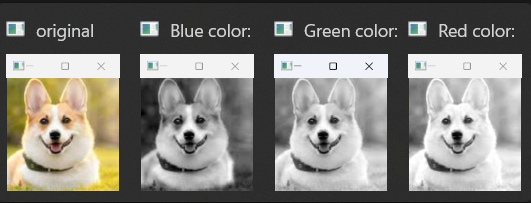
## Day 2 Opencv

DAY 2:OPENCV COMMANDS AND FUNCTIONS  
#READING AN IMAGE  
import cv2 as cv  
img=cv.imread('dog1.jpg')  
cv.imshow('Dog',img)  
#grayscale  
gray=cv.cvtColor(img,cv.COLOR\_BGR2GRAY)  
cv.imshow('Gray',gray)  
#blur  
blur=cv.GaussianBlur(img,(3,3),cv.BORDER\_DEFAULT)  
cv.imshow('Blur',blur)

  
#RESIZE  
def rescaleframe(frame,scale=5):  
    width=frame.shape[1]\*scale  
    height=frame.shape[0]\*scale  
    dimensions=(width,height)  
    return rescaleframe(frame,dimensions,interpolation=cv.INTER\_AREA)  
img=cv.imread('dog2.jpg')  
while True:  
    isTrue,frame=img.read()  
    frameresized=rescaleframe(frame)  
    cv.imshow('Resized image',frameresized)  
cv.waitKey(0)  
Resized image:

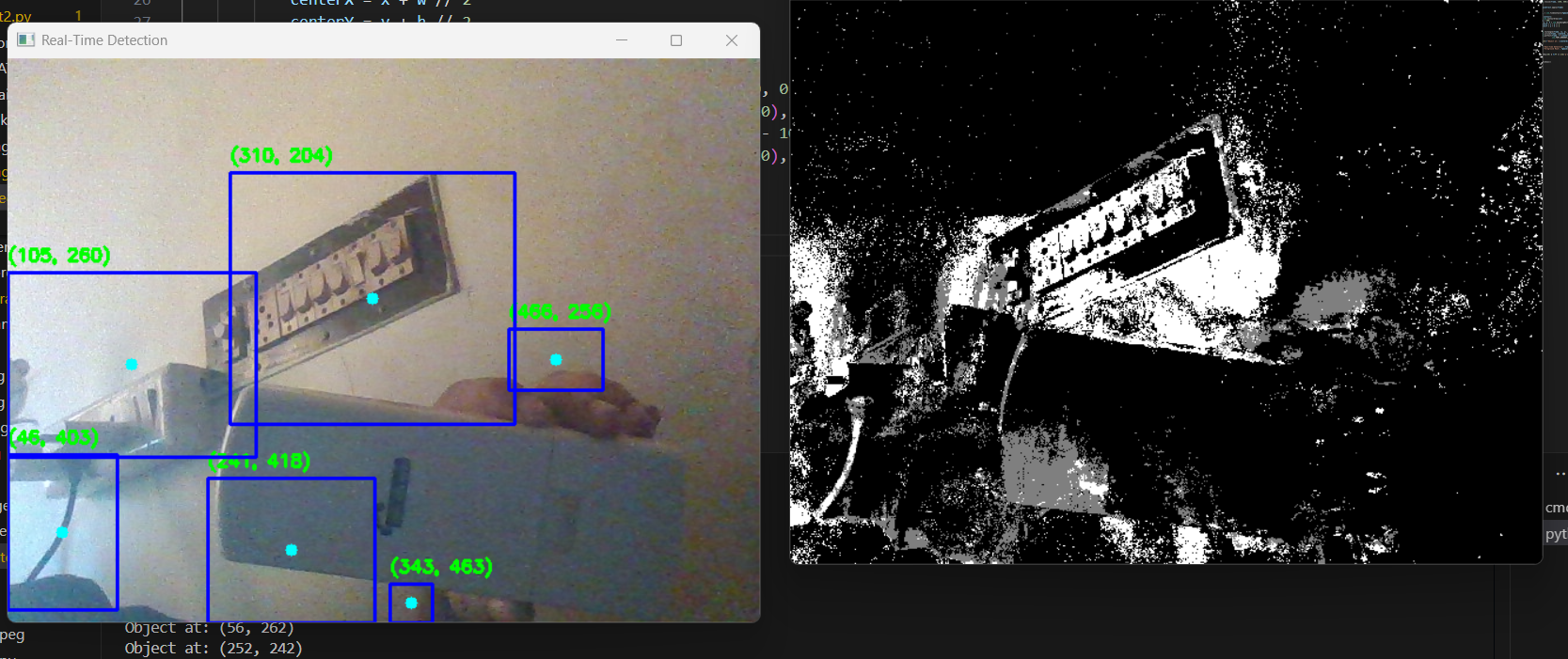
  
WRITING AN IMAGE INTO THE DIRECTORY:  
import cv2  
import os  
image\_path=r'C:\Users\Sanjana Raghunath\OneDrive\Documents\openCV\cat3.jpg'  
directory=r'C:\Users\Sanjana Raghunath\OneDrive\Documents\openCV'  
image=cv2.imread(image\_path)  
os.chdir(directory)  
print("Before saving image:")  
print(os.listdir(directory))  
filename='newimage.jpg'  
cv2.imwrite(filename,image)  
print("After saving image:")  
print(os.listdir(directory))  
print("Successfully saved!")  
OUTPUT:

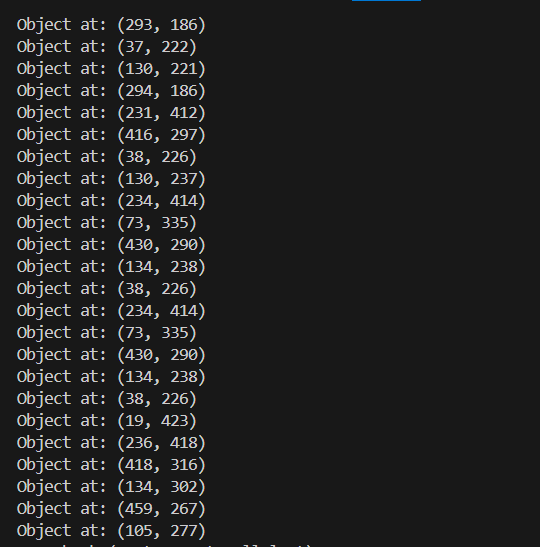
  
BGR SPLIT IN AN IMAGE:  
import cv2  
image=cv2.imread('dog2.jpg')  
B,G,R=cv2.split(image)  
cv2.imshow("original",image)  
cv2.imshow('Blue color:',B)  
cv2.imshow('Green color:',G)  
cv2.imshow('Red color:',R)  
cv2.waitKey(0)

  
ARITHMETIC OPERATIONS USING OPENCV:  
ADDITION:  
import cv2  
import numpy as np  
img1=cv2.imread('dog1.jpg')  
img2=cv2.imread('cat3.jpg')  
img2=cv2.resize(img2,(img1.shape[1],img1.shape[0]))  
weightedsum=cv2.addWeighted(img1,0.5,img2,0.5,0)  
cv2.imshow('Weighted image:',weightedsum)  
if cv2.waitKey(0) or 0xff==27:  
    cv2.destroyAllWindows()  
Subtraction:  
import cv2  
import numpy as np  
img1=cv2.imread('img1.png')  
img2=cv2.imread('img2.png')  
img2=cv2.resize(img2,(img1.shape[1],img1.shape[0]))  
subtractedimage=cv2.subtract(img1,img2)  
cv2.imshow('First image:',img1)  
cv2.imshow('Second image:',img2)  
cv2.imshow('Subtracted image:',subtractedimage)  
cv2.waitKey(0)  
BITWISE OPERATIONS:  
import cv2  
#AND  
import numpy as np  
and1=cv2.imread('and1.png')  
and2=cv2.imread('and2.png')  
and2=cv2.resize(and2,(and1.shape[1],and1.shape[0]))  
andimage=cv2.bitwise\_and(and1,and2,mask=None)  
cv2.imshow('First image:',and1)  
cv2.imshow('Second image:',and2)  
cv2.imshow('And image:',andimage)  
#OR operator image  
and1=cv2.imread('and1.png')  
and2=cv2.imread('and2.png')  
and2=cv2.resize(and2,(and1.shape[1],and1.shape[0]))  
orimage=cv2.bitwise\_or(and1,and2,mask=None)  
cv2.imshow('First image:',and1)  
cv2.imshow('Second image:',and2)  
cv2.imshow('or image:',orimage)  
#xor  
and1=cv2.imread('and1.png')  
and2=cv2.imread('and2.png')  
and2=cv2.resize(and2,(and1.shape[1],and1.shape[0]))  
xorimage=cv2.bitwise\_xor(and1,and2,mask=None)  
cv2.imshow('First image:',and1)  
cv2.imshow('Second image:',and2)  
cv2.imshow('xor image:',xorimage)  
# not  
and1=cv2.imread('and1.png')  
notimage=cv2.bitwise\_not(and1,mask=None)  
cv2.imshow('First image:',and1)  
cv2.imshow('not image:',notimage)  
OUTPUT:  
Erode()  
import cv2  
import numpy as np  
path = r'C:\Users\Sanjana Raghunath\OneDrive\Documents\openCV\cat2.jpg'  
image = cv2.imread(path)   
window\_name = 'Image'  
kernel = np.ones((5, 5), np.uint8)  
image = cv2.erode(image, kernel)   
cv2.imshow(window\_name, image)  
cv2.waitKey(0)  
Image border:  
import cv2   
path = r'C:\Users\Sanjana Raghunath\OneDrive\Documents\openCV\dog1.jpg'  
image = cv2.imread(path)  
window\_name = 'Image'  
image = cv2.copyMakeBorder(image, 10, 10, 10, 10, cv2.BORDER\_CONSTANT, None, value = 0)  
cv2.imshow(window\_name, image)  
cv2.waitKey(0)  
Grayscale :  
import cv2  
img = cv2.imread(r'C:\Users\Sanjana Raghunath\OneDrive\Documents\openCV\dog1.jpg')  
(row, col) = img.shape[0:2]  
for i in range(row):  
    for j in range(col):  
        img[i, j] = sum(img[i, j]) \* 0.33  
cv2.imshow('Grayscale Image', img)  
cv2.waitKey(0)  
cv2.destroyAllWindows()  
EROSION AND DILATION:  
import cv2  
import numpy as np  
img = cv2.imread('img1.jpg')  
gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
\_, thresh = cv2.threshold(gray, 120, 255, cv2.THRESH\_BINARY)  
kernel = np.ones((5, 5), np.uint8)  
eroded = cv2.erode(thresh, kernel, iterations=1)  
dilated = cv2.dilate(thresh, kernel, iterations=1)  
cv2.imshow('Original', img)  
cv2.imshow('Thresholded', thresh)  
cv2.imshow('Eroded', eroded)  
cv2.imshow('Dilated', dilated)  
cv2.waitKey(0)  
cv2.destroyAllWindows()  
histogram:  
import cv2  
from matplotlib import pyplot as plt  
img = cv2.imread('cat2.jpg', 0)  
histr = cv2.calcHist([img], [0], None, [256], [0, 256])  
plt.plot(histr)  
plt.show()  
histogram equalization:  
import cv2  
from matplotlib import pyplot as plt  
img = cv2.imread('cat2.jpg', 0)  
equalized = cv2.equalizeHist(img)  
cv2.imshow('Original', img)  
cv2.imshow('Equalized', equalized)  
cv2.waitKey(0)  
cv2.destroyAllWindows()  
plt.figure()  
plt.title("Histogram Comparison")  
plt.plot(cv2.calcHist([img], [0], None, [256], [0, 256]), label='Original')  
plt.plot(cv2.calcHist([equalized], [0], None, [256], [0, 256]), label='Equalized')  
plt.legend()  
plt.show()  
Spectral image map:  
import matplotlib.pyplot as plt  
import cv2  
img = cv2.imread('cat3.jpg')  
plt.imshow(img, cmap ='nipy\_spectral')  
cv2.waitKey(0)  
BILATERAL FILTERING:  
OpenCV has a function called bilateralFilter() with the following arguments:   
   
d: Diameter of each pixel neighborhood.  
sigmaColor: Value of   σ  in the color space. The greater the value, the colors farther to each other will start to get mixed.  
sigmaSpace: Value of σ  in the coordinate space. The greater its value, the more further pixels will mix together, given that their colors lie within the sigmaColor range.  
import cv2  
img = cv2.imread('scenery.jpg')  
bilateral = cv2.bilateralFilter(img, 15, 75, 75)  
cv2.imwrite('newscenery.jpg', bilateral)  
cv2.imshow('image',img)  
cv2.imshow('new clear image:',bilateral)  
OUTPUT:  
IMAGE INPAINTING:  
Image inpainting is the process of removing damage, such as noises, strokes or text, on images. It is particularly useful in the restoration of old photographs which might have scratched edges or ink spots on them.   
Code:  
import cv2  
import numpy as np  
img = cv2.imread(filename=r"cat4.png")  
height, width = img.shape[0], img.shape[1]  
for i in range(height):  
    for j in range(width):  
        if img[i, j].sum() > 0:  
            img[i, j] = 0  
        else:  
            img[i, j] = [255, 255, 255]  
mask = img  
cv2.imwrite('mask.jpg', mask)  
cv2.imshow("damaged image mask", mask)  
cv2.waitKey(0)  
cv2.destroyAllWindows()  
OUTPUT:  
Background masking:   
import cv2  
bg\_subtractor = cv2.createBackgroundSubtractorMOG2(history=100, varThreshold=40, detectShadows=True)  
cap = cv2.VideoCapture('cat3.jpg')  
while True:  
    ret, frame = cap.read()  
    if not ret:  
        break  
    frame = cv2.resize(frame, (640, 480))  
    fg\_mask = bg\_subtractor.apply(frame)  
    kernel = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE, (3, 3))  
    fg\_mask = cv2.morphologyEx(fg\_mask, cv2.MORPH\_OPEN, kernel)  
    cv2.imshow("Original", frame)  
    cv2.imshow("Foreground Mask", fg\_mask)  
   cap.release()  
cv2.waitKey(0)  
cv2.destroyAllWindows()  
IMAGE TRANSLATION:  
CODE:  
import cv2  
import numpy as np  
image = cv2.imread('dog2.jpg')  
height, width = image.shape[:2]  
quarter\_height, quarter\_width = height / 4, width / 4  
T = np.float32([[1, 0, quarter\_width], [0, 1, quarter\_height]])  
img\_translation = cv2.warpAffine(image, T, (width, height))  
cv2.imshow("Originalimage", image)  
cv2.imshow('Translation', img\_translation)  
cv2.waitKey()  
cv2.destroyAllWindows()  
IMAGE PYRAMID:  
Code:  
import cv2  
import matplotlib.pyplot as plt  
img = cv2.imread('cat1.jpg')  
layer = img.copy()  
for i in range(4):  
    plt.subplot(2, 2, i + 1)  
    layer = cv2.pyrDown(layer)  
    plt.imshow(layer)  
    cv2.imshow("str(i)", layer)  
    cv2.waitKey(0)  
cv2.destroyAllWindows()

## Day 3 Object Detection

DAY 3=OBJECT DETECTION USING OPEN CV AND CALCULATION OF X,Y COORDINATES OF THE RECTANGLE FORMED SURROUNDING THE OBJECT  
CODE:  
import cv2  
cap = cv2.VideoCapture(0) #here, the webcam takes the video live during the runtime and finds the realtime object detection  
bgsubtract = cv2.createBackgroundSubtractorMOG2()  
(for creating masking which can be used to give only the borders and no background)  
while True:  
    ret, frame = cap.read()  
    if not ret:  
        break  
    frame = cv2.resize(frame, (640, 480))  
    fgmask = bgsubtract.apply(frame)  
    contours, \_ = cv2.findContours(fgmask, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
    for cnt in contours:  
        area = cv2.contourArea(cnt)  
        if area > 500:    
            x, y, w, h = cv2.boundingRect(cnt)  
            centerX = x + w // 2  
            centerY = y + h // 2  
            cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)  
            cv2.circle(frame, (centerX, centerY), 5, (255, 255, 0), -1)  
            cv2.putText(frame, f"({centerX}, {centerY})", (x, y - 10),  
                        cv2.FONT\_HERSHEY\_SIMPLEX, 0.5, (0, 255, 0), 2)  
            print(f"Object at: ({centerX}, {centerY})")  
    cv2.imshow('Real-Time Detection', frame)  
    cv2.imshow('Foreground Mask', fgmask)  
    if cv2.waitKey(30) & 0xFF == ord('q'):  
        break  
cap.release()  
cv2.destroyAllWindows()  
OUTPUT:





## Day 4 Ml Basics

## DAY 4 :BASICS OF MACHINE LEARNING Why Machine Learning is Used in Industries & How It’s Applied

Machine Learning (ML) has become a cornerstone of innovation across various industries, enabling systems to learn from data, make decisions, and improve over time without explicit programming. Its versatility makes it highly applicable across sectors for automation, prediction, optimization, and pattern recognition.

### Key Industry Applications

**Healthcare:**  
ML enhances diagnostic accuracy by analyzing medical images such as X-rays and MRIs. It assists in early disease detection (like cancer or diabetes), drug discovery, and genetic analysis. It’s also useful for predicting patient outcomes and tailoring personalized treatment plans.

**Banking & Finance:**  
In finance, ML is widely used for credit scoring, fraud detection, algorithmic trading, and customer service automation. It supports predictive analytics for investment decisions and improves compliance through intelligent data analysis.

**Retail & E-Commerce:**  
Retailers use ML to offer personalized recommendations, predict buying patterns, optimize pricing, and manage inventory. Sentiment analysis of reviews also helps companies understand customer opinions and improve offerings.

**Automotive:**  
ML powers autonomous vehicles by processing sensor data and making real-time driving decisions. It supports predictive maintenance, improves navigation systems, and enables voice interaction through AI assistants in cars.

**Technology & IT:**  
Applications include language translation, image/speech recognition, spam filtering, and AI-driven virtual assistants like Alexa. ML also plays a major role in cybersecurity by identifying threats in real time.

**Manufacturing:**  
In this sector, ML predicts equipment failure, detects product defects using computer vision, and forecasts demand. It helps streamline operations and enables data-driven decisions on the assembly line.

**Education:**  
ML personalizes learning by adapting content to individual needs, predicts student performance, and enables auto-grading systems. It also supports AI tutors and plagiarism detection tools.

**Media & Entertainment:**  
Platforms like Netflix and YouTube use ML for content recommendations. It analyzes user reactions and preferences, supports targeted advertising, and assists in fake news detection and content tagging.

**Travel & Transportation:**  
ML is used for dynamic pricing of tickets, chatbot-based customer support, and travel recommendation engines. It also enables traffic forecasting and route optimization, especially in autonomous vehicles and drone systems.

**Agriculture:**  
ML helps forecast crop yields, detect diseases via drone imagery, monitor soil health, and automate irrigation. It also aids in market trend predictions to help farmers make better selling decisions.

**Telecommunications:**  
Telecom companies use ML for network optimization, churn prediction, automatic fault detection, and personalizing user plans. Voice recognition also enhances customer service experiences.

## 🧠 Types of Machine Learning Techniques

### 1. ****Supervised Learning****

This method uses labeled data to train models to predict outcomes. It includes:

**Classification**: For discrete outputs (e.g., spam detection)

**Regression**: For continuous outputs (e.g., stock prices)

**Workflow:**

Data is split into training (80%) and testing (20%) sets.

The model learns from labeled data to make accurate predictions.

Common algorithms: Linear/Logistic Regression, Decision Trees, Random Forests, SVM, and Gradient Boosting.

**Real-life examples**:

Fraud detection in banking

Disease prediction in healthcare

Email spam filtering

### 2. ****Unsupervised Learning****

This approach deals with **unlabeled data** and aims to discover hidden patterns or groupings.

**Techniques include:**

**Clustering**: Grouping similar data points (e.g., customer segmentation,K-Means, DBSCAN, Hierarchical clustering

**Association Rule Learning**: Finding relationships (e.g., market basket analysis)

Apriori, FP-Growth

**Dimensionality Reduction**: Reducing data complexity

PCA, LDA, Isomap

**Example**: Analyzing shopping patterns without predefined labels to create customer segments.

### 3. ****Reinforcement Learning (RL)****

In RL, an agent learns by interacting with an environment and receiving rewards or penalties for its actions.

**Core elements**:

**Agent**: Learner or decision-maker

**Environment**: Context or system it operates in

**State**: Current situation

**Action**: Decision option

**Reward**: Feedback from the environment

**How it works**:

The agent explores, acts, receives feedback, and adjusts its strategy to maximize cumulative rewards.

Common in robotics, game playing, and autonomous systems.

**Example**: A robot learning to navigate a maze by trial and error, avoiding hazards and reaching its goal.

## Day 5 Ml Regression Model

DAY 5: Training ML model for regression problems  
1.Choose best open source dataset works for regression problem  
2. ⁠Train the mL model and observe the result  
3. ⁠explanations needed for each parameter used in mL model why it’s been used  
1)Ames house price prediction model:  
Dataset from:Kaggle  
I am currently working on Google colab with python3 runtime type and CPU hardware accelerator for this problem  
CODE step by step parameter explanation and why it is used  
Step 1:IMPORTING NECESSARY MODULES FROM THE LIBRARIES  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.metrics import mean\_squared\_error, r2\_score  
import pandas as pd :  
this is used to import pandas library  
It helps you read, manipulate, and analyze structured data (like CSV files or tables).  
We use it to load the dataset, check for missing values, drop or fill columns, and create features.  
Import numpy as np  
 Provides efficient operations on arrays, matrices, and mathematical functions.  
 Helps in numerical operations like calculating RMSE (np.sqrt()), handling missing values, etc.  
Import matplotlib.pyplot as plt  
To create visualizations like bar charts, line plots, histograms, etc.  
Helps in understanding feature importance, data distributions, trends, etc.  
Import seaborn as sns  
  Makes it easier to create beautiful statistical visualizations (e.g., heatmaps, correlation matrices).  
  Great for EDA (Exploratory Data Analysis).  
from sklearn.model\_selection import train\_test\_split  
We train our model on one part of the data (training) and test its performance on unseen data (testing).  
from sklearn.preprocessing import StandardScaler  
To normalize or scale features so that they have zero mean and unit variance.  
 Why it's used:  
Some algorithms are sensitive to scale (e.g., linear models, distance-based models).  
Helps improve model performance and convergence.  
from sklearn.ensemble import RandomForestRegressor  
to use Random forest model in the problem  
It's a powerful and flexible ensemble model that combines multiple decision trees.  
 Works well on both linear and non-linear data.  
 Used here to predict house prices based on features.  
from sklearn.metrics import mean\_squared\_error, r2\_score  
to evaluate how good the model’s predictions are.   
 mean\_squared\_error: Measures the average squared difference between actual and predicted values.  
 r2\_score: Measures how well the predictions explain the variance in the target. R² = 1 is perfect, 0 is no better than average.  
STEP 2:INPORTING THE DATASET FRM KAGGLE USING PANDAS  
from google.colab import files  
uploaded = files.upload()  
Opens a file upload window so we can upload the AmesHousing.csv file from our computer.  
import pandas as pd  
data = pd.read\_csv("AmesHousing.csv")  
print(data.shape)  
data.head()  
pd.read\_csv(): Loads the CSV file into a DataFrame.  
data.shape: Shows the number of rows and columns.  
data.head(): Displays the first 5 rows of the dataset to get an idea of its structure.  
Step 3:Check for missing values and plot the distribution of target variable  
missing = data.isnull().sum().sort\_values(ascending=False)  
print(missing[missing > 0])  
data.dtypes.value\_counts()  
sns.histplot(data['SalePrice'], kde=True)  
plt.title("Sale Price Distribution")  
plt.show()  
isnull().sum(): Counts how many missing values exist in each column.  
missing > 0: Filters only those columns that have missing values.  
sort\_values: Sorts by the number of missing values in descending order.  
STEP4:DROP COLUMNS WITH MANY MISSING VALUES  
columns\_to\_drop = ['PoolQC', 'MiscFeature', 'Alley', 'Fence', 'FireplaceQu']  
existing\_cols = [col for col in columns\_to\_drop if col in data.columns]  
data = data.drop(columns=existing\_cols)  
These columns typically have 90%+ missing values and aren’t very useful for prediction.  
We check if the column exists before dropping to avoid errors  
STEP 5:Fill remaining missing values   
for column in data.columns:  
    if data[column].dtype == 'object':  
        data[column] = data[column].fillna(data[column].mode()[0])  
    else:  
        data[column] = data[column].fillna(data[column].median())  
 Categorical columns (object): Fill missing values with the most frequent value (mode).  
 Numerical columns: Fill missing values with the median, which is less sensitive to outliers than the mean.  
STEP 6:ENCODE CATEGORICAL VALUES  
data = pd.get\_dummies(data)  
print(data.shape)  
pd.get\_dummies converts categorical values to numerical dummy values(one hot encoding)  
We need this because ML models require numerical inputs  
STEP 7:SPLIT FEATURES AND TARGET  
X = data.drop("SalePrice", axis=1)  
y = data["SalePrice"]  
x:all features and parameters used for house price prediction except the target variable  
y:the target variable that is to be tested with the predicted house price value  
STEP 8:SPLIT INTO TRAINING AND TESTING SETS  
from sklearn.model\_selection import train\_test\_split  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
 Purpose: Split data into 80% training and 20% testing.  
 Training set: Used to train the model.  
 Testing set: Used to evaluate how well the model performs on new data.  
random\_state=42 ensures reproducibility (same split every time).  
STEP 9:FEATURE SCALING  
from sklearn.preprocessing import StandardScaler  
scaler = StandardScaler()  
X\_train\_scaled = scaler.fit\_transform(X\_train)  
X\_test\_scaled = scaler.transform(X\_test)  
Many ML models (like linear models or distance-based ones) perform better if features are scaled.  
 fit\_transform(): Learns the mean and std from training data and applies scaling.  
transform(): Applies the same scaling to the test data (important to prevent data leakage).  
STEP 10:TRAIN THE MODEL  
from sklearn.ensemble import RandomForestRegressor  
model = RandomForestRegressor(n\_estimators=100, random\_state=42)  
model.fit(X\_train\_scaled, y\_train)  
 RandomForestRegressor: A powerful ensemble model using multiple decision trees.  
 n\_estimators=100: Uses 100 trees.  
 fit(): Trains the model using scaled features and the sale prices.  
STEP 11:EVALUATING THE MODEL  
from sklearn.metrics import mean\_squared\_error, r2\_score  
import numpy as np  
y\_pred = model.predict(X\_test\_scaled)  
mse = mean\_squared\_error(y\_test, y\_pred)  
rmse = np.sqrt(mse)  
r2 = r2\_score(y\_test, y\_pred)  
print(f"RMSE: {rmse:.2f}")  
print(f"R² Score: {r2:.2f}")  
predict(): Predicts sale prices for the test set.  
mean\_squared\_error: Measures the average squared difference   
between predicted and actual prices.  
rmse: Square root of MSE; more interpretable (in dollars).  
r2\_score: Tells how much of the variance in SalePrice is explained by the model. 1.0 is perfect, 0 is bad.  
The Ames Housing Price Prediction model is widely regarded as one of the best ways to learn and practice regression in machine learning—and for good reason. It’s based on a real-world problem that’s easy to understand: predicting house prices. This makes it incredibly relatable and practical, whether you're a student, a data scientist, or just someone curious about how machine learning works in the real world. The dataset, created by Dean De Cock, is a modern, cleaner alternative to the older Boston Housing dataset, and it has become a favorite in courses, competitions, and tutorials.  
What really makes this dataset stand out is the variety and richness of its features. It contains over 80 different columns, covering everything from the size of the house and the year it was built, to the quality of the materials and the neighborhood it’s in. Because these features include numbers, ordered categories (like ratings from 1 to 10), and names of places or styles, you get to practice a wide range of preprocessing techniques—like handling missing values, encoding categories, and scaling features. These are all skills that are super important in any machine learning project.  
It’s also a great dataset for trying out different types of regression models. Whether you’re starting with something simple like Linear Regression or diving into more powerful techniques like Random Forests, XGBoost, or Neural Networks, the Ames data gives you a balanced, flexible playground to test and compare your models. It’s big enough to be meaningful, but small enough to run smoothly even on a basic laptop or in Google Colab.  
Beyond just building models, the Ames dataset lets you explore more advanced concepts too—like cross-validation, hyperparameter tuning, feature engineering, and model evaluation using metrics like RMSE and R² score. In other words, it supports the entire machine learning pipeline from start to finish.  
In short, the Ames Housing dataset is popular not just because it’s clean and manageable, but because it mirrors a real-life problem, gives you exposure to essential ML techniques, and helps you build confidence in developing and evaluating regression models. It’s an ideal learning tool for anyone who wants to move beyond theory and into practical, hands-on experience.

# Topic-Based Deep Dives

## Deep Learning

1. Introduction to Deep Learning  
Deep Learning is a subset of Machine Learning inspired by the structure and function of the human brain - called Artificial Neural Networks (ANNs). It is especially powerful for tasks involving large, unstructured data like images, speech, and text.  
2. Architecture of an Artificial Neural Network (ANN)  
An ANN consists of layers of neurons:  
- Input layer: Receives the raw data (e.g., pixels of an image)  
- Hidden layers: Perform computations using weights, biases, and activation functions  
- Output layer: Gives the final prediction (e.g., class label)  
Each connection has a weight and a bias, and passes through an activation function.  
These could be single layer perceptron or multi-layer perceptrons depending on the number of hidden layers.  
3. Weights and Biases  
Weights determine how strongly an input feature influences the output. Biases allow shifting the activation function left or right. These are learnable parameters optimized during training.  
Formula: z = 〖I=1wixi + b  
4. Dimensionality and Tensors  
Input data may be in the form of:  
- 1D (vectors)  
- 2D (matrices)  
- 3D/4D (tensors for images, batches)  
Tensors generalize matrices to higher dimensions and are the native format used in frameworks like TensorFlow and PyTorch.  
5. Activation Functions  
Activation functions introduce non-linearity to learn complex patterns.  
- Sigmoid: 1 / (1 + e^-z)  
- Tanh: (e^z- e^-z) / (e^z + e^-z) or 2/(1+e^-2z) +1  
- ReLU: max(0, z)  
- Softmax: Converts outputs to probabilities. In the range of 0 to 1  
6. How Neural Networks Learn  
The learning process involves:  
1. Forward Propagation  
2. Loss Calculation  
3. Backward Propagation (gradient calculation)  
4. Weight Update (using optimizers).  
7. Optimizer Algorithms  
- SGD: Stochastic Gradient Descent can work only on one sample or a small batch of samples.  
- Momentum: Adds velocity  
- RMSprop: Adapts learning rate  
- Adam: Combines momentum and RMSprop (widely used) and an extended version of SGD   
8. Hyperparameter Tuning  
Hyperparameters include:  
- Learning rate-controls how much the model's weights are updated in response to the estimated error (loss) each time the model learns.  
For the model handwriting detection, I just used 5 epochs but still managed to get an accuracy of 97percent whereas even after using 100 epochs in next word prediction model, I could get only 65 percent accuracy that could be due to low learning rate .  
- Batch size-Number of training samples processed before theweights are updated once  
- Epochs-number of times a model learns from the training set.  
- Number of layers and neurons  
Tuning methods: Grid Search, Random Search, Bayesian Optimization, or manual tuning using validation set.  
9. Training and Evaluation  
Split data into training, validation, and test sets. Use accuracy, precision, recall, and loss to monitor performance. Use dropout, early stopping, and regularization to prevent overfitting.  
10. Applications of Deep Learning  
Deep learning is used in:  
- Vision: Image classification, object detection  
- NLP: Chatbots, sentiment analysis  
- Healthcare: Disease detection  
- Finance: Fraud detection, stock forecasting.  
Conclusion  
Deep learning enables machines to learn from raw data through layers of abstraction using weights, biases, activation functions, and optimizers. It-s transforming fields like vision, language, and medicine.  
Hand writing detection model  
Next word Prediction;

## Ann Rnn Onnx Comparison

Comparative Report: ANN vs RNN vs ONNX with Use Cases  
1. Introduction  
This report provides a comparative overview of Artificial Neural Networks (ANN), Recurrent Neural Networks (RNN), and ONNX (Open Neural Network Exchange). It highlights their key differences, applications, and presents a sample implementation for each.  
2. Model Overview  
3. Use Case Implementation  
3.1 ANN Use Case: Health Risk Prediction (Tabular Data)  
Goal: Predict whether a person is at health risk based on age, weight, and blood pressure.  
- Type: Classification  
- Data: Tabular (Age, Weight, BP)  
- Tools: TensorFlow, Keras  
CODE;  
import numpy as np  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
X = np.array([  
    [25, 55, 120],  
    [40, 90, 145],  
    [30, 70, 130],  
    [60, 80, 160],  
    [22, 50, 110],  
    [55, 95, 170]  
])  
y = np.array([0, 1, 0, 1, 0, 1])   
PREPROCESSING  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.3)  
MODEL  
model\_ann = Sequential([  
    Dense(16, activation='relu', input\_shape=(3,)),  
    Dense(8, activation='relu'),  
    Dense(1, activation='sigmoid')  
])  
TRAINING AND COMPILATION  
model\_ann.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
model\_ann.fit(X\_train, y\_train, epochs=30, verbose=0)  
PREDICT  
print("ANN prediction (health risk):", model\_ann.predict(X\_test))  
Result:  
The model gave the outputs as probabilities as I have used sigmoid activation function at last.  
· 0.59988 → About 60% chance that the first sample is "high risk" (label 1)  
 · 0.42258 → About 42% chance that the second sample is high risk → Likely "low risk" (label 0)  
3.2 RNN Use Case: Symptom Text Classification  
Goal: Classify free-text symptom descriptions as emergency or non-emergency.  
- Type: Text classification  
- Data: Sequential (sentences)  
- Tools: TensorFlow, Keras, Tokenizer, Embedding, RNN layers  
MODEL TRAINED TO CLASSIY SYMPTOMS AS EMERGENCY OR NON-EMERGENCY BASED ON THE SEVERITY OF THE SYMPTOMS   
IMPORTING IMPORTANT MODULES   
import numpy as np  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Embedding,SimpleRNN,Dense  
from tensorflow.keras.preprocessing.text import Tokenizer  
from tensorflow.keras.preprocessing.sequence import pad\_sequences  
INPUT TEXTS AND OUTPUTS LABELS;  
texts = [  
    "chest pain and shortness of breath",  
    "mild headache and fatigue",  
    "severe bleeding after injury",  
    "dizziness after standing up",  
    "high fever and cough"  
]  
labels = [1, 0, 1, 0, 0]   
TOKENIZATION TO CONVERT TEXTS INTO TOKENS AND COMPARE THE SEVERITY BASED ON THE INPUT LABELS .1-EMERGENCY,0-NON-EMERGENCY  
tokenizer = Tokenizer()  
tokenizer.fit\_on\_texts(texts)  
seqs = tokenizer.texts\_to\_sequences(texts)  
padded = pad\_sequences(seqs)  
MODEL TRAINING AND EVALUATION;  
vocab\_size = len(tokenizer.word\_index) + 1  
model\_rnn = Sequential([  
    Embedding(input\_dim=vocab\_size, output\_dim=8, input\_length=padded.shape[1]),  
    SimpleRNN(16),  
    Dense(1, activation='sigmoid')  
])  
model\_rnn.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
model\_rnn.fit(padded, np.array(labels), epochs=50, verbose=0)  
CLASSIFICATION;  
sample = ["severe bleeding and chest pain"]  
sample\_seq = pad\_sequences(tokenizer.texts\_to\_sequences(sample), maxlen=padded.shape[1])  
print("RNN prediction (emergency):", model\_rnn.predict(sample\_seq))  
3.3 ONNX Use Case: Model Deployment  
Goal: Export a trained ANN or RNN model to ONNX format for cross-platform deployment (e.g., mobile or edge device).  
- Type: Model conversion  
- Tools: tf2onnx or PyTorch -> ONNX exporters  
- Output: `.onnx` file for inference in different environments  
#  Install tf2onnx if not already installed  
!pip install tf2onnx  
#  Import required libraries  
import numpy as np  
import tensorflow as tf  
import tf2onnx  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import StandardScaler  
# Prepare the dummy data  
# Features: [Age, Weight, Systolic BP]  
X = np.array([  
    [25, 55, 120],  
    [40, 90, 145],  
    [30, 70, 130],  
    [60, 80, 160],  
    [22, 50, 110],  
    [55, 95, 170]  
])  
y = np.array([0, 1, 0, 1, 0, 1])    
#  Preprocess the data  
scaler = StandardScaler()  
X\_scaled = scaler.fit\_transform(X)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_scaled, y, test\_size=0.3)  
# Define and train the ANN model  
model\_ann = Sequential([  
    Dense(16, activation='relu', input\_shape=(3,)),  
    Dense(8, activation='relu'),  
    Dense(1, activation='sigmoid')  
])  
model\_ann.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])  
model\_ann.fit(X\_train, y\_train, epochs=30, verbose=0)  
model\_ann.save("health\_model.h5")  
# Convert to ONNX directly from saved Keras model  
!python -m tf2onnx.convert --keras health\_model.h5 --output health\_risk\_ann\_model.onnx --opset 13  
print(" ANN model converted and saved as health\_risk\_ann\_model.onnx")  
4. Summary of Strengths  
5. Conclusion  
Each architecture serves a different purpose:  
- ANN is best suited for simple input-output mappings in structured data.  
- RNN is powerful for sequential data analysis and modeling temporal relationships.  
- ONNX enhances flexibility in deploying ML models across platforms and environments.  
  
By understanding these differences and strengths, developers can choose the appropriate model and deployment path based on their specific application.

## Medical Dataset Model

MEDICAL DATASET FOR COMPARISON BETWEEN HEALTHY AND IDEOPATHIC TOE WALKING TO IDENTIFY IF A PERSON IS HEALTHY OR SUFFERRING FROM ITW.  
IMPORTING THE ZIP FILES HEALTHY.ZIP AND ITW.ZIP  
from google.colab import files  
uploaded = files.upload()  
 STEP 2: Unzip the file to a known directory  
import zipfile  
import os  
zip\_path = "/content/Healthy.zip"  
extract\_dir = "/content/extracted\_c3d/Healthy"  
os.makedirs(extract\_dir, exist\_ok=True)  
with zipfile.ZipFile(zip\_path, 'r') as zip\_ref:  
    zip\_ref.extractall(extract\_dir)  
STEP3-UNZIP THE EXTRACTED C3D FILE  
import zipfile  
import os  
with zipfile.ZipFile("ITW.zip", 'r') as zip\_ref:  
    zip\_ref.extractall("extracted\_c3d/ITW")  
print("Unzipped files to extracted\_c3d/ITW")  
STEP 4: Define skeleton plotting and image conversion logic Because .c3d format is not identifiable by pytorch  
!pip install ezc3d  
import ezc3d  
import matplotlib.pyplot as plt  
from tqdm import tqdm  
def plot\_skeleton(c3d, frame\_idx, save\_path):  
    fig = plt.figure()  
    ax = fig.add\_subplot(111, projection='3d')  
    points = c3d['data']['points']  
    x, y, z = points[0, :, frame\_idx], points[1, :, frame\_idx], points[2, :, frame\_idx]  
    ax.scatter(x, y, z, c='blue')  
    ax.set\_axis\_off()  
    plt.savefig(save\_path)  
    plt.close(fig)  
def convert\_healthy\_c3d\_to\_images(healthy\_input\_path, output\_root="/content/healthy\_skeleton\_images", max\_frames=30):  
    os.makedirs(output\_root, exist\_ok=True)  
    files = [f for f in os.listdir(healthy\_input\_path) if f.endswith('.c3d')]  
    print(f"Found {len(files)} .c3d files.")  
    for filename in files:  
        file\_path = os.path.join(healthy\_input\_path, filename)  
        subject\_id = os.path.splitext(filename)[0]  
        subject\_out\_dir = os.path.join(output\_root, subject\_id)  
        os.makedirs(subject\_out\_dir, exist\_ok=True)  
        c3d = ezc3d.c3d(file\_path)  
        num\_frames = min(c3d['data']['points'].shape[2], max\_frames \* 5)  
        for i in tqdm(range(0, num\_frames, 5), desc=f"Healthy - {filename}"):  
            save\_path = os.path.join(subject\_out\_dir, f"frame\_{i:04d}.png")  
            plot\_skeleton(c3d, i, save\_path)  
import ezc3d  
import matplotlib.pyplot as plt  
from tqdm import tqdm  
def plot\_skeleton(c3d, frame\_idx, save\_path):  
    fig = plt.figure()  
    ax = fig.add\_subplot(111, projection='3d')  
    points = c3d['data']['points']  
    x, y, z = points[0, :, frame\_idx], points[1, :, frame\_idx], points[2, :, frame\_idx]  
    ax.scatter(x, y, z, c='blue')  
    ax.set\_axis\_off()  
    plt.savefig(save\_path)  
    plt.close(fig)  
def convert\_next\_100\_c3d\_to\_images(input\_dir, output\_dir, start=0, max\_frames=30):  
    import glob  
    import os  
    all\_c3d\_files = sorted(glob.glob(os.path.join(input\_dir, '\*\*', '\*.c3d'), recursive=True))  
    batch\_files = all\_c3d\_files[start:start + 100]  
    if not batch\_files:  
        print("No .c3d files found in this batch.")  
        return  
    os.makedirs(output\_dir, exist\_ok=True)  
    for file\_path in tqdm(batch\_files, desc=f"Processing {len(batch\_files)} files from index {start}"):  
        subject\_id = os.path.splitext(os.path.basename(file\_path))[0]  
        subject\_out\_dir = os.path.join(output\_dir, subject\_id)  
        os.makedirs(subject\_out\_dir, exist\_ok=True)  
        try:  
            c3d = ezc3d.c3d(file\_path)  
            num\_frames = min(c3d['data']['points'].shape[2], max\_frames \* 5)  
            for i in range(0, num\_frames, 5):  
                save\_path = os.path.join(subject\_out\_dir, f"frame\_{i:04d}.png")  
                plot\_skeleton(c3d, i, save\_path)  
        except Exception as e:  
            print(f"Error processing {file\_path}: {e}")  
put\_itw = "extracted\_c3d/ITW"  
output\_itw = "batched\_skeletons/ITW"  
convert\_next\_100\_c3d\_to\_images(input\_itw, output\_itw, start=0)  
convert\_next\_100\_c3d\_to\_images(input\_itw, output\_itw, start=100)  
similarly for remaining 300 images  
STEP 6-COUNT THE NUMBER OF IMAGES IN HEALTHY AND ITW FOLDERS TOCHECK IF ALL THE IMAGES HAVE BEEN CONVERTEDED IN SKELETON FORMAT(.png)  
STEP 7-COMBINING HEALTHY AND ITW DATASETS PROCESSED INTO A NEW FOLDER CLASSIFIED\_DATASET  
import os  
import shutil  
healthy\_src = "/content/healthy\_skeleton\_images"  
itw\_src = "/content/batched\_skeletons/ITW"  
base\_dest = "/content/classified\_dataset"  
healthy\_dest = os.path.join(base\_dest, "Healthy")  
itw\_dest = os.path.join(base\_dest, "ITW")  
os.makedirs(healthy\_dest, exist\_ok=True)  
os.makedirs(itw\_dest, exist\_ok=True)  
for folder in os.listdir(healthy\_src):  
    folder\_path = os.path.join(healthy\_src, folder)  
    if os.path.isdir(folder\_path):  
        for file in os.listdir(folder\_path):  
            if file.endswith(".png"):  
                src\_path = os.path.join(folder\_path, file)  
                dst\_path = os.path.join(healthy\_dest, f"{folder}\_{file}")  
                shutil.copy(src\_path, dst\_path)  
for folder in os.listdir(itw\_src):  
    folder\_path = os.path.join(itw\_src, folder)  
    if os.path.isdir(folder\_path):  
        for file in os.listdir(folder\_path):  
            if file.endswith(".png"):  
                src\_path = os.path.join(folder\_path, file)  
                dst\_path = os.path.join(itw\_dest, f"{folder}\_{file}")  
                shutil.copy(src\_path, dst\_path)  
print(" Dataset organized at:", base\_dest)  
SPLITTING THE DATA INTO TRAIN AND TEST DATA  
from torchvision import datasets, transforms  
from torch.utils.data import DataLoader  
transform = transforms.Compose([  
    transforms.Resize((224, 224)),  
    transforms.ToTensor()  
])  
dataset\_root = "/content/classified\_dataset"  # folder with 'Healthy' and 'ITW'  
train\_dataset = datasets.ImageFolder(root=dataset\_root, transform=transform)  
train\_loader = DataLoader(train\_dataset, batch\_size=32, shuffle=True)  
STEP 9-IMPORT LIBRARIES AND SET UP TRANSFORMS  
STEP 10-LOAD PRETRAINED MODEL(ResNET 18)  
STEP 11-DEFINE LOSS AND OPTIMIZER  
criterion = nn.CrossEntropyLoss()  
optimizer = optim.Adam(model.parameters(), lr=1e-4)  
TRAIN THE MODEL  
num\_epochs = 5  
for epoch in range(num\_epochs):  
 model.train()  
 total\_loss = 0  
 correct = 0  
 for images, labels in train\_loader:  
 images, labels = images.to(device), labels.to(device)  
 optimizer.zero\_grad()  
 outputs = model(images)  
 loss = criterion(outputs, labels)  
 loss.backward()  
 optimizer.step()  
 total\_loss += loss.item()  
 correct += (outputs.argmax(1) == labels).sum().item()  
 acc = correct / len(train\_loader.dataset)  
 print(f"Epoch {epoch+1}, Loss: {total\_loss:.4f}, Train Accuracy: {acc:.4f}")  
EVALUATE THE MODEL  
model.eval()  
all\_preds = []  
all\_labels = []  
with torch.no\_grad():  
 for images, labels in val\_loader:  
 images = images.to(device)  
 outputs = model(images)  
 preds = outputs.argmax(1).cpu().numpy()  
 all\_preds.extend(preds)  
 all\_labels.extend(labels.numpy())  
print(classification\_report(all\_labels, all\_preds, target\_names=class\_names))  
STEP 12-TRAINING THE MODEL  
 Trains the model using cross-entropy loss.  
 Updates model weights using Adam optimizer.  
STEP 13-EVALUATION OF THE MODEL  
 Precision for Healthy: 0.98, but recall is low (0.68), meaning some Healthy samples were misclassified.  
 Precision for ITW: 0.95, recall is perfect (1.00).  
 Overall accuracy: 96%

**QUANTIZATION METHODS AND PURPOSES**

### Quantization in Machine Learning and My Work on It

Quantization is a technique used in machine learning to reduce the precision of the numerical values (such as weights and activations) used in a model. Typically, models are trained using 32-bit floating-point numbers (FP32), which provide high accuracy but require significant memory and computation resources. Quantization reduces these values to lower-precision formats like 8-bit integers (INT8), resulting in smaller and faster models.

This process is particularly useful when deploying models on resource-constrained environments such as mobile devices, embedded systems, and Internet of Things (IoT) platforms. Quantized models consume less memory, perform faster inference, and require less energy, making them ideal for edge deployment.

#### Objective of the Task

During my internship, I implemented post-training quantization (PTQ) on a pre-trained deep learning model. The aim was to reduce the model size and inference time without compromising significantly on performance.

#### Steps Followed

**Model Training in Full Precision**  
Initially, I trained a neural network model using a standard dataset (such as MNIST or a medical dataset) with 32-bit floating-point precision. This model served as the baseline for comparison.

**Applying Quantization Techniques**  
After training, I applied post-training quantization to convert the model weights from float32 to int8. I experimented with:

Dynamic Quantization: Weights were quantized, while activations were left in float and quantized dynamically during inference.

Static Quantization: Both weights and activations were quantized using calibration data.

(Optional) Quantization Aware Training (QAT): A technique that simulates quantization effects during the training phase itself, to improve final performance.

**Evaluation and Comparison**  
I evaluated the performance of both the original and the quantized models. This involved comparing:

**APPLICATIONS**

· **Smartphones** (camera apps, AR, assistants)

· **IoT Devices** (sensors, voice recognition)

· **Embedded ML** (TinyML, edge AI)

· **On-device NLP models** (BERT, Whisper, etc.)

During my internship, I explored the concept of quantization in image processing, focusing on how it reduces complexity and prepares data for compression. My first task involved implementing uniform quantization for grayscale images. In this method, the full grayscale range (0 to 255) is divided into equally sized intervals, and each pixel in the image is mapped to the center value of its corresponding interval. This effectively reduces the number of distinct gray levels in the image, simplifying its representation. I used Python libraries such as OpenCV for image loading and manipulation, NumPy for efficient numerical operations, and Matplotlib for visualization. The original image was processed using a custom function that applied uniform quantization based on a specified number of levels. The output included the quantized image and a quantization error map, which highlighted differences between the original and quantized versions. This visualization helped me understand the trade-offs involved in reducing image detail for simplicity.

Building upon the understanding of quantization, I extended the project to implement a JPEG-style image compression system using the Discrete Cosine Transform (DCT). JPEG compression relies on frequency domain transformations followed by quantization, particularly on 8×8 image blocks. To mimic this process, I began by loading a grayscale image and padding it to ensure its dimensions were divisible by eight. I then applied a two-dimensional DCT on each 8×8 block of the image after centering its pixel values. A standard JPEG quantization matrix was used to compress the frequency coefficients. This matrix determined the level of compression by scaling the DCT output, where higher values produced greater compression at the cost of more image detail.

The compressed data was then decompressed by reversing the steps: multiplying the quantized coefficients with the same quantization matrix and applying the inverse DCT, followed by re-centering and cropping the result to match the original dimensions. The reconstructed image was compared visually against the original, demonstrating the effectiveness of the compression process. Although there was a visible reduction in fine detail, the overall image structure was well-preserved, confirming that quantization and DCT can achieve significant size reduction with acceptable loss in quality.

Through these experiments, I gained a deeper understanding of how uniform quantization simplifies image representation and how it plays a crucial role in real-world compression systems like JPEG. The hands-on experience also highlighted how frequency transformation and quantization work together to optimize data storage and transmission, especially in multimedia applications.

### Future Work and Scope

The work I did during this internship has given me a solid foundation to explore more advanced areas in the future. I’m especially interested in taking the machine learning models I worked on and making them suitable for real-world deployment, especially on mobile or edge devices where resources are limited. Exploring quantization-aware training is something I’d like to try, as it could help keep model accuracy high even after compression. On the image processing side, I’m curious about working with color image and video compression, and possibly combining machine learning with traditional methods like DCT to build smarter, adaptive systems. Overall, this internship has sparked several ideas, and I look forward to building on what I’ve learned.