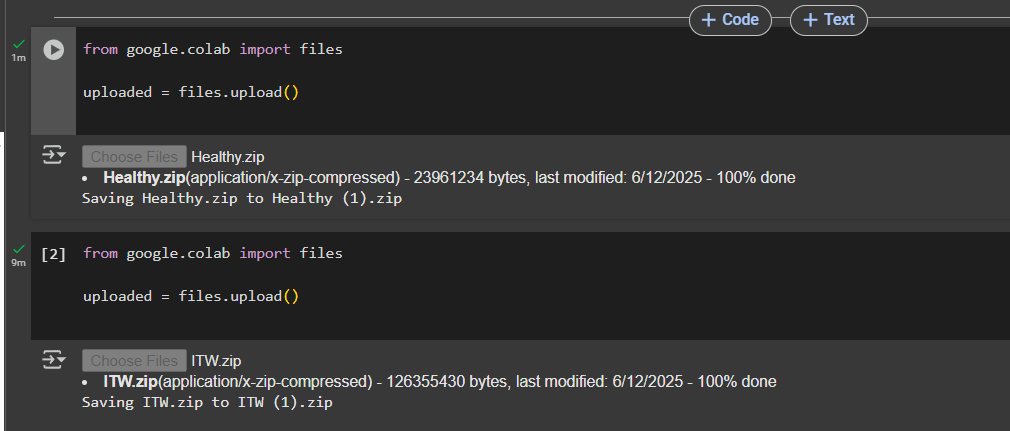
**MEDICAL DATASET FOR COMPARISON BETWEEN HEALTHY AND IDEOPATHIC TOE WALKING TO IDENTIFY IF A PERSON IS HEALTHY OR SUFFERRING FROM ITW.**

**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)

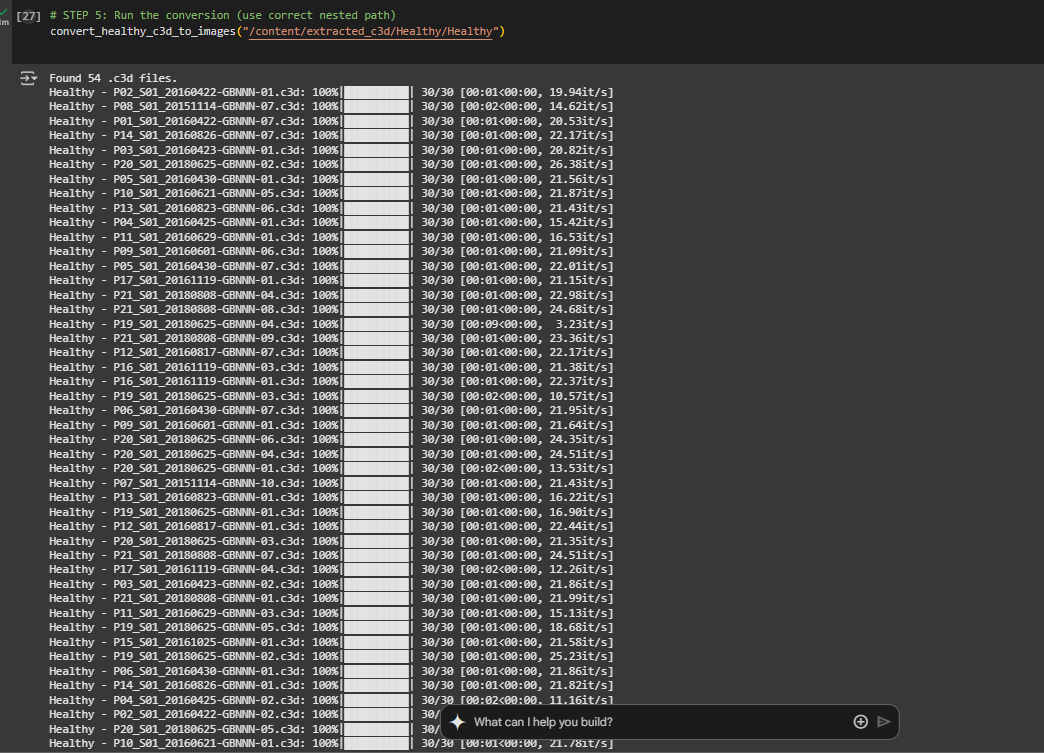
        c3d = ezc3d.c3d(file\_path)

        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"

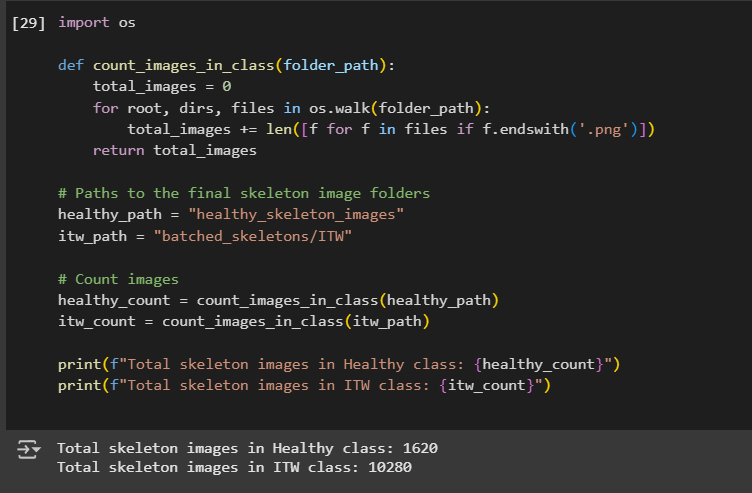
output\_itw = "batched\_skeletons/ITW"

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()

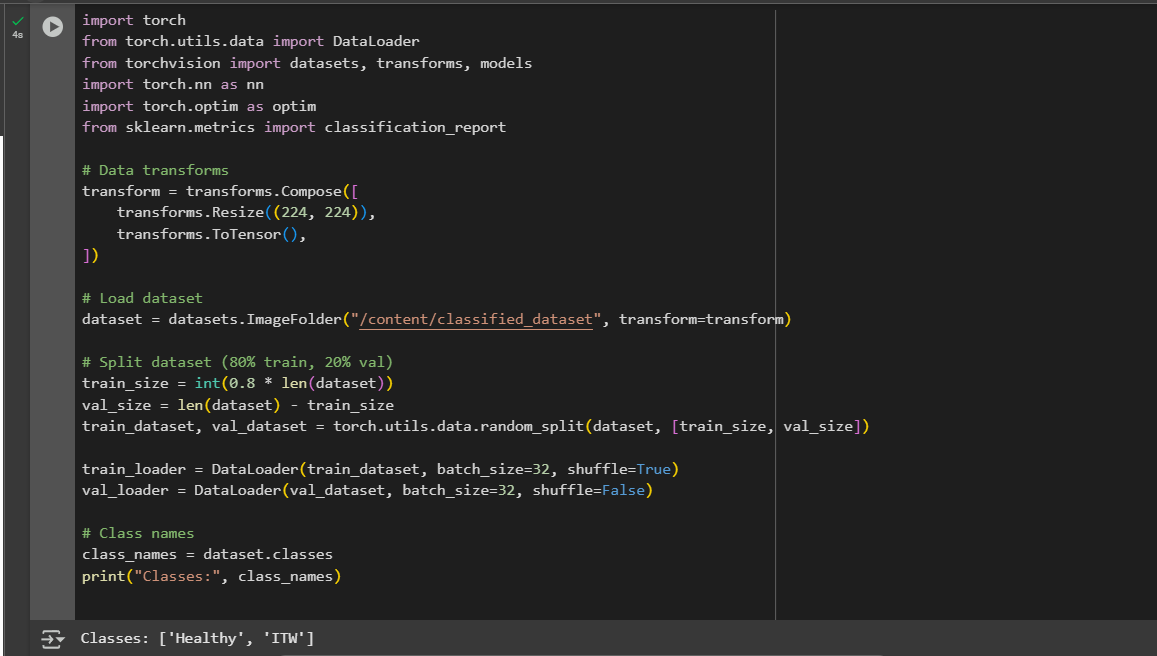
])

dataset\_root = "/content/classified\_dataset"  # folder with 'Healthy' and 'ITW'

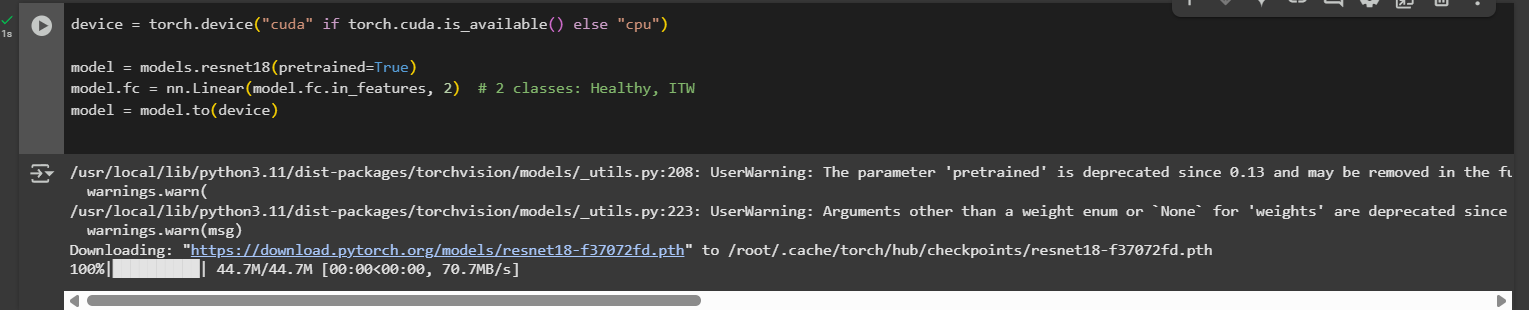
train\_dataset = datasets.ImageFolder(root=dataset\_root, transform=transform)

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)

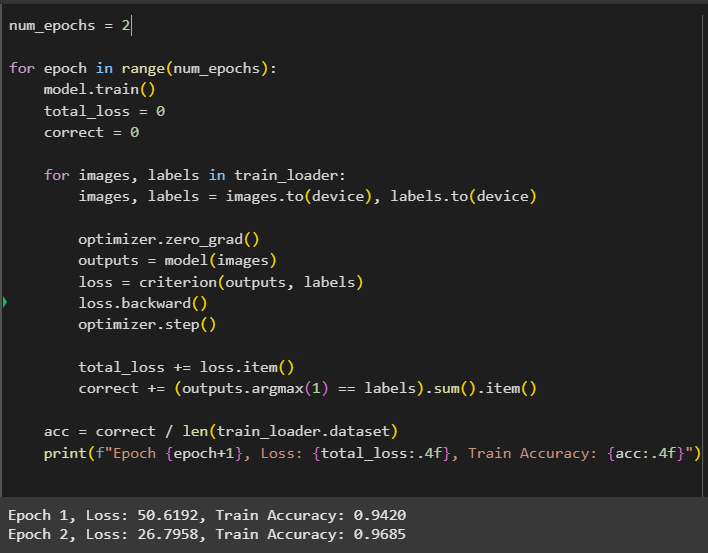
preds = outputs.argmax(1).cpu().numpy()

all\_preds.extend(preds)

all\_labels.extend(labels.numpy())

print(classification\_report(all\_labels, all\_preds, target\_names=class\_names))

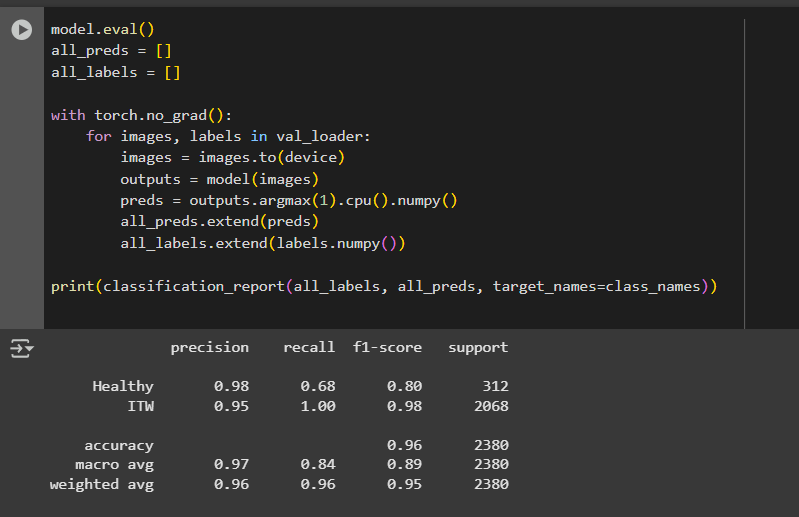
**STEP 12-TRAINING THE MODEL**



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%**