Python code for my AI Model -

Pulmonary Tuberculosis (TB) Detector

```
--coding: utf-8 --

# importing modules

import os

import random

import torch import torch.nn as nn

import torch.optim as optim from torchvision

import transforms, datasets from torch.utils.data

import DataLoader, Subset from PIL import Image, ImageTk, ImageDraw

import tkinter as tk from tkinter

 import filedialog, ttk from tkinter.font

import Font import threading import time from datetime

import datetime

 import matplotlib.pyplot as plt

#Ensure device is CPU

device = torch.device("cpu")

#Paths

DATASET_DIR = "C:/Users/grace/Downloads/dataset1" # Replace with your dataset directory
MODEL_PATH = "tb_model_cpu.pth"

#Define constants

IMAGE_SIZE = (224, 224)

 BATCH_SIZE = 32
 EPOCHS = 10

 LEARNING_RATE = 0.001

TB_THRESHOLD = 0.7

#Balance Dataset

def balance_dataset(dataset1):

        healthy_indices = [i for i, (, label) in enumerate(dataset1) if label == 0]

        tb_indices = [i for i, (, label) in enumerate(dataset1) if label == 1]

# Use all TB images and equal number of Healthy images
   balanced_indices = healthy_indices[:len(tb_indices)] + tb_indices
   random.shuffle(balanced_indices)
   return Subset(dataset, balanced_indices)


#Custom CNN Model

class CNNModel(nn.Module):

        def init(self, num_classes=2):

             super(CNNModel, self).init()

            self.features = nn.Sequential(

                      nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),

                       nn.ReLU(),

     nn.MaxPool2d(kernel_size=2, stride=2),

      nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),

     nn.ReLU(),

     nn.MaxPool2d(kernel_size=2, stride=2), )

self.classifier = nn.Sequential(

        nn.Flatten(),

        nn.Linear(64 * 56 * 56, 128),
        nn.ReLU(),

        nn.Dropout(0.5),

        nn.Linear(128, num_classes), )

def forward(self, x):
    x = self.features(x)
    x = self.classifier(x)
    return x
```

#Data Transformation

```
transform = transforms.Compose([

       transforms.Resize(IMAGE_SIZE),

       transforms.ToTensor(),

])
```

#Load Dataset and Balance

```
dataset = datasets.ImageFolder(DATASET_DIR, transform=transform)

 balanced_dataset = balance_dataset(dataset)

train_loader = DataLoader(balanced_dataset, batch_size=BATCH_SIZE, shuffle=True)
```

#Define Model, Loss, Optimizer

```
model = CNNModel(num_classes=len(dataset.classes)).to(device)

criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(model.parameters(), lr=LEARNING_RATE)
```

#Plot Class Distribution

```
def plot_class_distribution(dataset):

        class_counts = [0] * len(dataset.classes)

         for _, label in dataset:

                  class_counts[label] += 1
```

```
  plt.figure(figsize=(6, 4))
  plt.bar(dataset.classes, class_counts, color=['green', 'red'])
  plt.title("Class Distribution")
  plt.xlabel("Classes")
  plt.ylabel("Number of Images")
  plt.show()
```

#Train Model

```
def train_model():

       model.train()

      epoch_losses = []

     epoch_accuracies = []

for epoch in range(EPOCHS):
    running_loss = 0.0
    correct_predictions = 0
    total_predictions = 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()
        running_loss += loss.item()

        # Calculate accuracy
        _, predicted = torch.max(outputs, 1)
        correct_predictions += (predicted == labels).sum().item()
        total_predictions += labels.size(0)

    epoch_loss = running_loss / len(train_loader)
    epoch_losses.append(epoch_loss)

    epoch_accuracy = correct_predictions / total_predictions
    epoch_accuracies.append(epoch_accuracy)
    print(f"Epoch {epoch + 1}/{EPOCHS}, Loss: {epoch_loss:.4f}, Accuracy: {epoch_accuracy:.4f}")

# Save the model
torch.save(model.state_dict(), MODEL_PATH)
print("Model trained and saved as", MODEL_PATH)

# Plot Training Loss
plt.figure(figsize=(6, 4))
plt.plot(range(1, EPOCHS + 1), epoch_losses, marker='o', color='blue', label='Loss')
plt.title("Training Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.grid()
plt.legend()
plt.show()

# Plot Training Accuracy
plt.figure(figsize=(6, 4))
plt.plot(range(1, EPOCHS + 1), epoch_accuracies, marker='o', color='green', label='Accuracy')
plt.title("Training Accuracy")
plt.xlabel("Epochs")
plt.ylabel("Accuracy")
plt.grid()
plt.legend()
plt.show()


#Prediction Function

def predict_image(image_path):

     model.eval()

    img = Image.open(image_path).convert("RGB")

    img = transform(img).unsqueeze(0).to(device)

    with torch.no_grad():

        outputs = model(img)
```

```
               probs = torch.softmax(outputs, dim=1)

               tb_prob = probs[0][1].item()

              prediction = "Tuberculosis(TB)" if tb_prob >= TB_THRESHOLD else "Healthy"

    return prediction, tb_prob * 100

#Generate X-ray Findings

def generate_findings(prediction):

        if prediction == "Tuberculosis(TB)":

                return "Chest X-Ray Findings: Evidence of TB - Opacities and TB nodules detected."

         else:

return "Chest X-Ray Findings: Normal Lung fields with no signs of TB lesions."

#Highlight TB Area (Simulation)

def highlight_tb_area(image_path):

        img = Image.open(image_path)

       draw = ImageDraw.Draw(img)

       draw.rectangle([(50, 50), (200, 200)], outline="#00FF7F", width=3) # Fluorescent green color

         return img

#GUI

def open_file():

       file_paths = filedialog.askopenfilenames(filetypes=[("Image files", ".png;.jpg;*.jpeg")])

      if file_paths: for file_path in file_paths:

result_text.set("Analyzing Image...")

app.update() time.sleep(10) # Simulate processing delay
```

```
        prediction, prob = predict_image(file_path)
        result_text.set("")  # Clear "Analyzing Image" text

        img = highlight_tb_area(file_path) if prediction == "Tuberculosis(TB)" else
Image.open(file_path)
        img_tk = ImageTk.PhotoImage(img.resize((300, 300)))

        img_frame = tk.Frame(scrollable_frame, bg="#f0f8ff")
        img_frame.pack(pady=10, anchor="w")

        img_label = tk.Label(img_frame, image=img_tk, bg="#f0f8ff")
        img_label.image = img_tk  # Keep reference
        img_label.pack(side="left", padx=10)

        findings = generate_findings(prediction)
        report_datetime = datetime.now().strftime("%Y-%m-%d; %H:%M:%S")

        result_label = tk.Label(img_frame, text=f"Prediction: {prediction}",
                                font=label_font, fg="black", bg="#f0f8ff", anchor="w", justify="left")
        result_label.pack(side="top", anchor="w", pady=5)

        prob_label = tk.Label(img_frame, text=f"Probability of TB: {prob:.1f}%",
                              font=label_font, fg="black", bg="#f0f8ff", anchor="w", justify="left")
        prob_label.pack(side="top", anchor="w", pady=5)

        findings_label = tk.Label(img_frame, text=findings, font=label_font, fg="black",
bg="#f0f8ff", justify="left")
        findings_label.pack(side="top", anchor="w", pady=5)

        datetime_label = tk.Text(img_frame, font=label_font, bg="#f0f8ff", height=1, width=50,
bd=0)
        datetime_label.pack(side="top", anchor="w", pady=5)

        datetime_text = "Date and Time of Report: "
        datetime_number = report_datetime

        datetime_label.insert("1.0", datetime_text)
        datetime_label.insert("end", datetime_number, "darkblue")
        datetime_label.tag_configure("darkblue", foreground="darkblue")
```

```
    message_color = "red" if prediction == "Tuberculosis(TB)" else "green"
    message_text = "Please consult your Physician" if prediction == "Tuberculosis(TB)" else "Stay Healthy"
    advice_text = "Advice: " + message_text

    advice_label = tk.Label(img_frame, text=advice_text, font=label_font, fg=message_color, bg="#f0f8ff")
    advice_label.pack(side="top", anchor="w", pady=5)


#Start file processing

def start_processing():

    threading.Thread(target=open_file).start()

#GUI Application

app = tk.Tk()

app.title("Pulmonary Tuberculosis(TB) Detector")

 app.geometry("600x800")

 app.configure(bg="#f0f8ff")

#Fonts

title_font = Font(family="Arial", size=18, weight="bold", slant="italic")

label_font = Font(family="Arial", size=12)

 button_font = Font(family="Arial", size=10, weight="bold")

#Title Label

title_label = tk.Label(app, text="Pulmonary Tuberculosis(TB) Detector", font=title_font, fg="maroon", bg="#f0f8ff")

 title_label.pack(pady=20)

#Scrollable Frame

scroll_canvas = tk.Canvas(app, bg="#f0f8ff")

scroll_frame = tk.Frame(scroll_canvas, bg="#f0f8ff")
```

```
scrollbar = ttk.Scrollbar(app, orient="vertical", command=scroll_canvas.yview)
scroll_canvas.configure(yscrollcommand=scrollbar.set)

scrollbar.pack(side="right", fill="y")

scroll_canvas.pack(side="left", fill="both", expand=True)

scrollable_frame = tk.Frame(scroll_canvas, bg="#f0f8ff")

 scroll_canvas.create_window((0, 0), window=scrollable_frame, anchor="n")

def configure_scroll_region(event):

        scroll_canvas.configure(scrollregion=scroll_canvas.bbox("all"))

scrollable_frame.bind("", configure_scroll_region)

#Results Labels

result_text = tk.StringVar()

result_label = tk.Label(app, textvariable=result_text, font=label_font, bg="#f0f8ff")

result_label.pack(pady=5)

#Upload Button

upload_button = tk.Button(app, text="Upload Images", command=start_processing, font=button_font, bg="#32CD32", fg="white")

upload_button.pack(pady=20)

#Direction Label

direction_label = tk.Label(app, text="Please upload your recent CHEST X-RAY in PNG or JPG format", font=label_font, bg="#FFA500")

 direction_label.pack(pady=5)

#Exit Button

exit_button = ttk.Button(app, text="EXIT", command=app.destroy) exit_button.pack(pady=20)

#Model Loading

if os.path.exists(MODEL_PATH):
 model.load_state_dict(torch.load(MODEL_PATH, map_location=device), strict=True)
model.eval()

 print("Model loaded and ready for inference.")

 else:

 print("Model file not found. Training the model...")

train_model()

#Plot Class Distribution

plot_class_distribution(dataset)

app.mainloop()

        # BY

# Nitin Jonathan Lawrence Moses -   For ISEF Regeneron - 2025
```