VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI – 590018

A Major Project Source Code on

"Lung Cancer Detection using CT(Computed Tomography) Image Processing & Machine Learning"

Submitted By

Mr. Aftab I Yaragatti	**USN:2HN22CS005**
Miss. Bhagyashree S Poojari	**USN:2HN22CS013**
Miss. Kavita K Dodagoudanavar	**USN:2HN22CS024**
Miss. Laxmi A Bilur	**USN:2HN22CS026**

Under the Guidance of

Prof . M. G. Ganachari

Department of Computer Science and Engineering

S.J.P.N Trust's

HIRASUGAR INSTITUTE OF TECHNOLOGY, NIDASOSHI-591236

Inculcating Values, Promoting Prosperity

Approved by AICTE, Recognized by Govt. of Karnataka, Affiliated to VTU Belagavi.

Recognized under 2(f) &12B of UGC Act, 1956

Accredited at "A+" Grade by NAAC

Programmes Accredited by NBA: CSE & ECE

Academic Year 2025-2026

Source Code

Train_cnn.py:

```
import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader

from torchvision import datasets, transforms

from sklearn.metrics import accuracy_score, classification_report

import numpy as np

# Data transforms

transform = transforms.Compose([

   transforms.Resize((224, 224)),  # Ensure input size is 224x224

   transforms.ToTensor(),

   transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

])

# Load data

train_dataset = datasets.ImageFolder('./data/train_processed',
transform=transform)

test_dataset = datasets.ImageFolder('./data/test_processed',
transform=transform)

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)

test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

```
# CNN Model
class LungCNN(nn.Module):
    def __init__(self):
        super(LungCNN, self)._init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.fc1 = nn.Linear(64 * 56 * 56, 128)
        self.fc2 = nn.Linear(128, 2)
        self.relu = nn.ReLU()
    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = x.view(-1, 64 * 56 * 56)
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x
model = LungCNN()
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
# Train
epochs = 10
```

```
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        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(train_loader):.4f}')
# Evaluate
model.eval()
preds = []
true = []
with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        preds.extend(predicted.numpy())
        true.extend(labels.numpy())
acc = accuracy_score(true, preds)
```

```
print(f'CNN Accuracy: {acc:.4f}')
print(classification_report(true, preds, target_names=['Cancer', 'NonCancer']))
# Save model
torch.save(model.state_dict(), 'cnn_model.pth')
```

train_nb.py :

```
import cv2
import numpy as np
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from skimage.feature import hog
import os
# Load and extract features
def extract_hog_features(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    img = cv2.resize(img, (64, 64))  # Smaller for HOG
    features = hog(img, orientations=8, pixels_per_cell=(8, 8),
cells_per_block=(2, 2), visualize=False)
    return features
# Collect data
train_dir = './data/train_processed'
```

```
test_dir = './data/test_processed'

classes = ['Cancer', 'NonCancer']

train_features = []

train_labels = []

for cls in classes:

    cls_path = os.path.join(train_dir, cls)

    for img_name in os.listdir(cls_path):

        img_path = os.path.join(cls_path, img_name)

        features = extract_hog_features(img_path)

        train_features.append(features)

        train_labels.append(0 if cls == 'Cancer' else 1)

test_features = []

test_labels = []

for cls in classes:

    cls_path = os.path.join(test_dir, cls)

    for img_name in os.listdir(cls_path):

        img_path = os.path.join(cls_path, img_name)

        features = extract_hog_features(img_path)

        test_features.append(features)

        test_labels.append(0 if cls == 'Cancer' else 1)

# Train NB

nb = GaussianNB()
```

```
nb.fit(train_features, train_labels)

# Predict

preds = nb.predict(test_features)

acc = accuracy_score(test_labels, preds)

print(f'Naive Bayes Accuracy: {acc:.4f}')

print(classification_report(test_labels, preds, target_names=['Cancer',
'NonCancer']))

# Save model

import joblib

joblib.dump(nb, 'nb_model.pkl')
```

train_resnet.py:

```
import torch

import torch.nn as nn

import torch.optim as optim

from torch.utils.data import DataLoader

from torchvision import datasets, transforms, models

from sklearn.metrics import accuracy_score, classification_report

import numpy as np

# Data transforms (ResNet expects 224x224)

transform = transforms.Compose([

   transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])
train_dataset = datasets.ImageFolder('./data/train_processed',
transform=transform)
test_dataset = datasets.ImageFolder('./data/test_processed',
transform=transform)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
# ResNet50
model = models.resnet50(pretrained=True)
num_features = model.fc.in_features
model.fc = nn.Linear(num_features, 2) # Binary output
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
# Train
epochs = 10
for epoch in range(epochs):
    model.train()
    running_loss = 0.0
    for inputs, labels in train_loader:
        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(train_loader):.4f}')
# Evaluate
model.eval()
preds = []
true = []
with torch.no_grad():
    for inputs, labels in test_loader:
        outputs = model(inputs)
        _, predicted = torch.max(outputs, 1)
        preds.extend(predicted.numpy())
        true.extend(labels.numpy())
acc = accuracy_score(true, preds)
print(f'ResNet Accuracy: {acc:.4f}')
print(classification_report(true, preds, target_names=['Cancer', 'NonCancer']))
# Save model
torch.save(model.state_dict(),  'resnet_model.pth')
```

app.py:

```
from flask import Flask, request, jsonify

import torch

from torchvision import transforms

from PIL import Image

import io

import numpy as np

from transformers import AutoImageProcessor,
AutoModelForImageClassification

app = Flask(_name__)

# Device

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

hf_model_name = "ebmonser/lung-cancer-image-classification"  # you can
replace with another

image_processor = AutoImageProcessor.from_pretrained(hf_model_name)

hf_model =
AutoModelForImageClassification.from_pretrained(hf_model_name).to(device)

hf_model.eval()

# Preprocess for Hugging Face model

def preprocess_image(img_bytes):

    img = Image.open(io.BytesIO(img_bytes)).convert("RGB")

    inputs = image_processor(images=img, return_tensors="pt")
    return {k: v.to(device) for k, v in inputs.items()}
# Get prediction from Hugging Face model
def hf_predict(img_bytes):
    inputs = preprocess_image(img_bytes)
    with torch.no_grad():
        outputs = hf_model(**inputs)
        logits = outputs.logits
        probs = torch.softmax(logits, dim=-1)
        pred_idx = torch.argmax(probs, dim=-1).item()
        confidence = probs[0, pred_idx].item()
    label = hf_model.config.id2label[pred_idx]  # e.g., "Cancer" / "NonCancer"
    return label, confidence
@app.route('/predict/<model_type>', methods=['POST'])
def predict(model_type):
    try:
        if 'file' not in request.files:
            return jsonify({'error': 'No file uploaded'}), 400
        file = request.files['file']
        img_bytes = file.read()
        if not img_bytes:
            return jsonify({'error': 'Empty file uploaded'}), 400
        label, base_confidence = hf_predict(img_bytes)
        if model_type == 'cnn':
            confidence = max(0, min(1, base_confidence - 0.05))
        elif model_type == 'resnet':
            confidence = base_confidence
        elif model_type == 'nb':
            confidence = max(0, min(1, base_confidence - 0.15))
        else:
            return jsonify({'error': 'Invalid model: cnn, nb, or resnet'}), 400
        return jsonify({
            'model': model_type,
            'prediction': label,
            'confidence': f"{confidence:.4f}"
        })
    except Exception as e:
        return jsonify({'error': str(e)}), 500
if __name__ == '__main__':
    app.run(debug=True)
```