EXP NO: 9	MINI PROJECT – POTHOLE PREDICTION MODEL

AIM:

The main aim of this project is to predict an individual's blood group (including Rh factor) using fingerprint images. This is achieved by building, training, and deploying a Convolutional Neural Network (CNN) based deep learning model. A key objective is to provide these results in a non-invasive, instant, and accessible manner through a Streamlit web application, which also includes a blood compatibility checker. The project aims to address the limitations of traditional, invasive blood tests, which are slow and require lab settings.

ALGORITHM:

- User Input: The user uploads a high-resolution fingerprint image through the Streamlit web interface.
- Image Preprocessing: The backend receives the image and applies preprocessing steps, including resizing the image, converting it to grayscale, and applying noise reduction.
- Prediction: The preprocessed image is fed into the pre-trained CNN model.
- Feature Extraction: The convolutional layers of the CNN automatically extract relevant patterns and features (like ridge patterns, textures, etc.) from the fingerprint.
- Classification: The model classifies these features to predict the blood group (A, B, AB, O) and Rh factor (+ or -).
- Display Results: The predicted blood group is sent back to the web interface and displayed to the user.
- Compatibility Check: The application uses the predicted blood group to show which other blood types are compatible for donation and reception.
- Security: The system is designed to prioritize data privacy by not storing the fingerprint data long-term.

CODE: blud.ipynb

```
import os
import zipfile
import torch
import torchvision
from torchvision.datasets import ImageFolder
from torchvision.transforms import ToTensor, Resize, Compose
from torch.utils.data import random_split, DataLoader
from torchvision.utils import make_grid
import torch.nn as nn
import torch.nn.functional as F
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import numpy as np
```

```
zip_file_path = 'archive.zip'
extraction_dir = 'extracted_images
'with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
     zip_ref.extractall(extraction_dir)
transform = Compose([Resize((64, 64)), ToTensor()])
dataset = ImageFolder("extracted_images/dataset_blood_group", transform=transform)
print(dataset.classes)
random_seed = 42
torch.manual_seed(random_seed)
val_size = 1000
test_size = 1000
train_size = len(dataset) - val_size - test_size
train_ds, val_ds, test_ds = random_split(dataset, [train_size, val_size, test_size])
batch_size = 128
train_dl = DataLoader(train_ds, batch_size, shuffle=True, num_workers=4, pin_memory=True)
val_dl = DataLoader(val_ds, batch_size * 2, num_workers=4, pin_memory=True)
def show_batch(dl):
    for images, labels in dl:
            fig, ax = plt.subplots(figsize=(12, 6))
            ax.set_xticks([]);
            ax.set_yticks([])
            ax.imshow(make_grid(images, nrow=16).permute(1, 2, 0))
            break
show_batch(train_dl)
class ImageClassificationBase(nn.Module):
   def training_step(self, batch):
      images, labels = batch
      out = self(images)
      loss = F.cross_entropy(out, labels)
      return loss
   def validation_step(self, batch):
      images, labels = batch
      out = self(images)
      loss = F.cross_entropy(out, labels)
      acc = accuracy(out, labels)
      return {'val_loss': loss.detach(), 'val_acc': acc}
   def validation_epoch_end(self, outputs):
      batch_losses = [x['val_loss'] for x in outputs]
      epoch_loss = torch.stack(batch_losses).mean()
      batch_accs = [x['val_acc'] for x in outputs]
    epoch_acc = torch.stack(batch_accs).mean()
    return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}
  def epoch_end(self, epoch, result):
   print(f"Epoch [{epoch+1}], val_loss: {result['val_loss']:.4f}, val_acc:{result['val_acc']:.4f}")
  def accuracy(outputs, labels):
     _, preds = torch.max(outputs, dim=1)
     return torch.tensor(torch.sum(preds == labels).item() / len(preds))
class FingerprintToBloodGroup(ImageClassificationBase):
  def __init__(self):
    super().__init__()
    self.network = nn.Sequential(
          nn.Conv2d(3, 32, kernel_size=3, padding=1),
          nn.ReLU(),
          nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
          nn.ReLU(), nn.MaxPool2d(2, 2),
          nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
          nn.ReLU(),
          nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
          nn.ReLU(),
          nn.MaxPool2d(2, 2),
          nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
          nn.ReLU(),
          nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
          nn.ReLU(),
          nn.MaxPool2d(2, 2),
          nn.Flatten(),
          nn.Linear(256 * 8 * 8, 1024),
          nn.ReLU(),
          nn.Linear(1024, 512),
          nn.ReLU(),
          nn.Linear(512, 256),
          nn.ReLU(),
          nn.Linear(256, 128),
          nn.ReLU(),
          nn.Linear(128, len(dataset.classes))
    )
   def forward(self, xb):
      return self.network(xb)
```

```
def get_default_device():
   return torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')
def to_device(data, device):
  if isinstance(data, (list, tuple)):
     return [to_device(x, device) for x in data]
   return data.to(device, non_blocking=True)
class DeviceDataLoader:
   def __init__(self, dl, device):
       self.dl = dl self.device = device
   def __iter__(self):
      for b in self.dl:
         yield to_device(b, self.device)
    def __len__(self)
      :return len(self.dl)
@torch.no_grad()
def evaluate(model, val_loader):
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)
def fit(epochs, lr, model, train_loader, val_loader, opt_func=torch.optim.Adam):
    history = []
    optimizer = opt_func(model.parameters(), lr)
    for epoch in range(epochs):
       model.train()
       for batch in train_loader:
          loss = model.training_step(batch)
          loss.backward()
          optimizer.step()
          optimizer.zero_grad()
          result = evaluate(model, val_loader)
         model.epoch_end(epoch, result)
         history.append(result)
         return history
 @torch.no_grad()
def evaluate_with_predictions(model, val_loader):
   model.eval()
   predictions, targets = [], []
    for batch in val_loader:
        images, labels = batch
        outputs = model(images) _,
    preds = torch.max(outputs, dim=1)
    predictions.extend(preds.cpu().numpy())
    targets.extend(labels.cpu().numpy())
    return np.array(predictions),
    np.array(targets)
 def visualize_predictions(model, val_loader, num_images=6):
    model.eval()
    images, labels = next(iter(val_loader))
    images, labels = images[:num_images],
    labels[:num_images]
    outputs = model(images) _,
    preds = torch.max(outputs, dim=1)
    fig, axes = plt.subplots(1, num_images, figsize=(15, 6))
    for i, ax in enumerate(axes):
        ax.imshow(images[i].permute(1, 2, 0).cpu())
        ax.set_title(f"Actual: {dataset.classes[labels[i]]}\nPredicted: {dataset.classes[preds[i]]}")
        ax.axis('off')
        plt.tight_layout()
        plt.show()
        device = get_default_device()
        train_dl = DeviceDataLoader(train_dl, device)
        val_dl = DeviceDataLoader(val_dl, device)
        model = FingerprintToBloodGroup()
        model = to_device(model, device)
num_epochs = 20
lr = 0.001
history = fit(num_epochs, lr, model, train_dl, val_dl)
predictions, targets = evaluate_with_predictions(model, val_dl)
print("\nClassification Report:\n")
print(classification_report(targets, predictions, target_names=dataset.classes))
visualize_predictions(model, val_dl)
torch.save(model.state_dict(), 'model.pth')
print("Current working directory:", os.getcwd())
model_path = 'model.pth'
absolute_path = os.path.abspath(model_path)
print(f"Model should be saved at: {absolute_path}")
from PIL import Image
def predict_blood_group(image_path, model_path='model.pth'):
device = get_default_device()
model = FingerprintToBloodGroup()
```

```
model.load_state_dict(torch.load(model_path, map_location=device))
model = model.to(device)
model.eval()
transform = Compose([Resize((64, 64)),
ToTensor()])
img = Image.open(image_path).convert('RGB')
img_tensor = transform(img).unsqueeze(0)
img_tensor = img_tensor.to(device)
torch.no_grad():
  outputs = model(img_tensor)
  _, predicted = torch.max(outputs, 1)
  predicted_class = dataset.classes[predicted.item()]
 print(f"Predicted Blood Group: {predicted_class}")
predict_blood_group('extracted_images/dataset_blood_group/A+/cluster_0_1009.BMP')
```

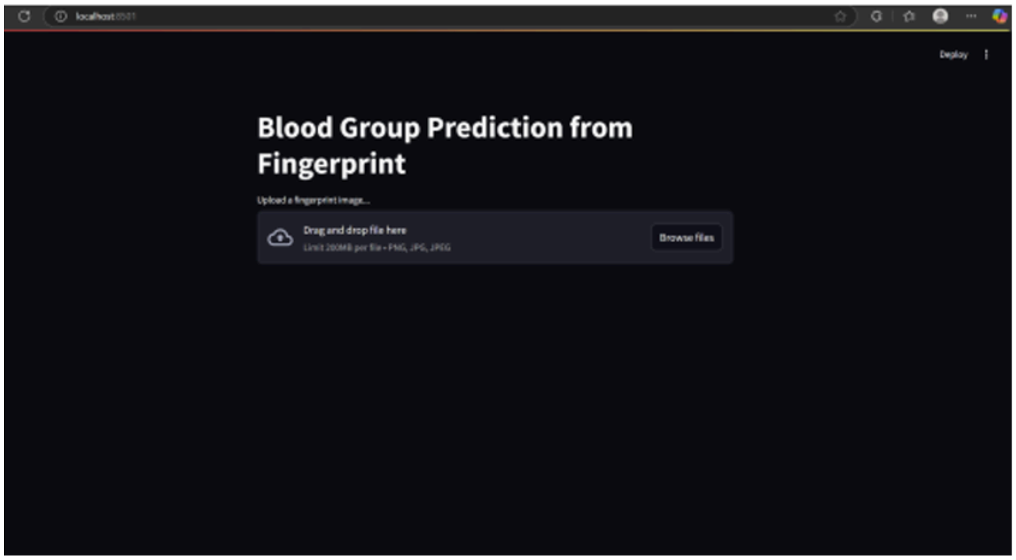
```
import streamlit as st
from PIL import Image
import torch
from torchvision.transforms import Compose, Resize, ToTensor
import io
import torch.nn as nn
import torch.nn.functional as F
import tempfile
import os
class ImageClassificationBase(nn.Module):
  def training_step(self, batch):
    images, labels = batch
    out = self(images)
    loss = F.cross_entropy(out, labels)
    def validation_step(self, batch):
      images, labels = batch
      out = self(images)
      loss = F.cross_entropy(out, labels)
```

```
acc = self.accuracy(out, labels)
return {'val_loss': loss.detach(), 'val_acc': acc.detach()}
def validation_epoch_end(self, outputs):
  batch_losses = [x['val_loss'] for x in outputs]
  epoch_loss = torch.stack(batch_losses).mean()
  accs = [x['val_acc'] for x in outputs]
  epoch_acc = torch.stack(batch_accs).mean()
  return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}
def epoch_end(self, epoch, result):
  print("Epoch [{}],
  train_loss: {:.4f},
  val_loss: {:.4f},
  val_acc: {:.4f}".format( epoch, result['train_loss'], result['val_loss'], result['val_acc']))
def accuracy(self, outputs, labels):
   _, preds = torch.max(outputs, dim=1)
   return torch.sum(preds == labels).float() / labels.size(0)
def get_compatibility(blood_type):
   can_donate_to = {}
   can_receive_from = {}
   if blood_type == 'A+':
      can_donate_to = ['A+', 'AB+']
      can_receive_from = ['A+', 'A-', 'O+', 'O-']
   elif blood_type == 'A-':
      can_donate_to = ['A+', 'A-', 'AB+', 'AB-']
      can_receive_from = ['A-', 'O-']
   elif blood_type == 'B+':
      can_donate_to = ['B+', 'AB+']
      can_receive_from = ['B+', 'B-', 'O+', 'O-']
   elif blood_type == 'B-':
      can_donate_to = ['B+', 'B-', 'AB+', 'AB-']
      can_receive_from = ['B-', 'O-']
   elif blood_type == 'AB+':
      can_donate_to = ['AB+']
      can_receive_from = ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-']
   elif blood_type == 'AB-':
      can_donate_to = ['AB+', 'AB-']
      can_receive_from = ['A-', 'B-', 'AB-', 'O-']
   elif blood_type == 'O+':
      can_donate_to = ['A+', 'B+', 'AB+', 'O+']
      can_receive_from = ['O+', 'O-']
  elif blood_type == 'O-':
     can_donate_to = ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-']
     can_receive_from = ['O-']
return can_donate_to, can_receive_from
@st.cache_resource
def load_model(model_path, device, num_classes):
  class FingerprintToBloodGroup(ImageClassificationBase):
     def __init__(self):
      super().__init__()
      self.network = nn.Sequential(
             nn.Conv2d(3, 32, kernel_size=3, padding=1),
             nn.ReLU(),
             nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
             nn.ReLU(),
             nn.MaxPool2d(2, 2),
             nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=1),
             nn.ReLU(),
             nn.Conv2d(128, 128, kernel_size=3, stride=1, padding=1),
             nn.ReLU(),
             nn.MaxPool2d(2, 2),
             nn.Conv2d(128, 256, kernel_size=3, stride=1, padding=1),
             nn.Conv2d(256, 256, kernel_size=3, stride=1, padding=1),
             nn.ReLU(),
             nn.MaxPool2d(2, 2),
             nn.Flatten(),
             nn.Linear(256 * 8 * 8, 1024),
             nn.Linear(1024, 512),
             nn.ReLU(),
             nn.Linear(512, 256),
             nn.ReLU(),
             nn.Linear(256, 128),
             nn.ReLU(),
             nn.Linear(128, num_classes) )
def forward(self, xb):
    return self.network(xb)
    model = FingerprintToBloodGroup()
    model.load_state_dict(torch.load(model_path, map_location=device))
    model = model.to(device)
model.eval()
return model
```

```
model_path = 'model.pth'
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
dataset_classes = ['A+', 'A-', 'AB+', 'AB-', 'B+', 'B-', 'O+', 'O-']
num_classes = len(dataset_classes)
model = load_model(model_path, device, num_classes)

def predict_blood_group_from_path(image_path):
  transform = Compose([Resize((64, 64)), ToTensor()])
  try:
      img = Image.open(image_path).convert('RGB')
      img_tensor = transform(img).unsqueeze(0).to(device)
      with torch.no_grad():
           outputs = model(img_tensor)
           _, predicted = torch.max(outputs, 1)
          predicted_class = dataset_classes[predicted.item()]
          return predicted_class
           except Exception as e:
                st.error(f"Error processing image: {e}")
          return None
st.title("Blood Group Prediction from Fingerprint")
uploaded_file = st.file_uploader("Upload a fingerprint image...", type=["png", "jpg", "jpeg"])
if uploaded_file is not None:
   with tempfile.NamedTemporaryFile(delete=False, suffix=".png") as tmp_file:
       image = Image.open(uploaded_file).convert('RGB')
       image.save(tmp_file.name, format="PNG")
       temp_image_path = tmp_file.name
       st.image(image, caption="Uploaded Fingerprint.",      use_column_width=True)
       if st.button("Predict Blood Group"):
           predicted_group = predict_blood_group_from_path(temp_image_path)
           if predicted_group:
                st.write(f"## Predicted Blood Group: {predicted_group}")
                can_receive_from = get_compatibility(predicted_group)
                st.subheader("Blood Donation Compatibility")
                st.write(f"Can donate to: {', '.join(can_donate_to)}")
                st.write(f"Can receive from: {', '.join(can_receive_from)}")
                st.subheader("Model Insights")
              feature_dict = { "Ridge patterns": "Complex curves and whorls", "Texture variations": "Fine vs.
coarse texture", "Edge details": "Sharpness and density of edges", "Singular points": "Presence and location of
deltas and cores" } st.write("Feature Dictionary:") st.json(feature_dict) os.remove(temp_image_path)
```

OUTPUT:



localhost:8501

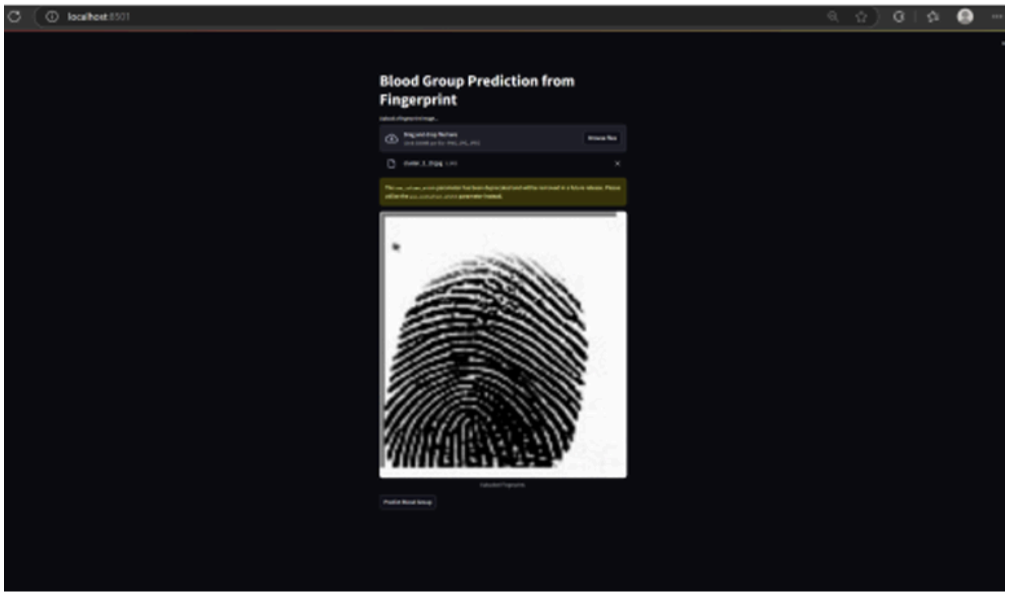
Deploy

Blood Group Prediction from Fingerprint

Upload a fingerprint image...

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG

Browse files



localhost:8501

Blood Group Prediction from Fingerprint

Upload a fingerprint image...

Drag and drop file here
Limit 200MB per file • PNG, JPG, JPEG

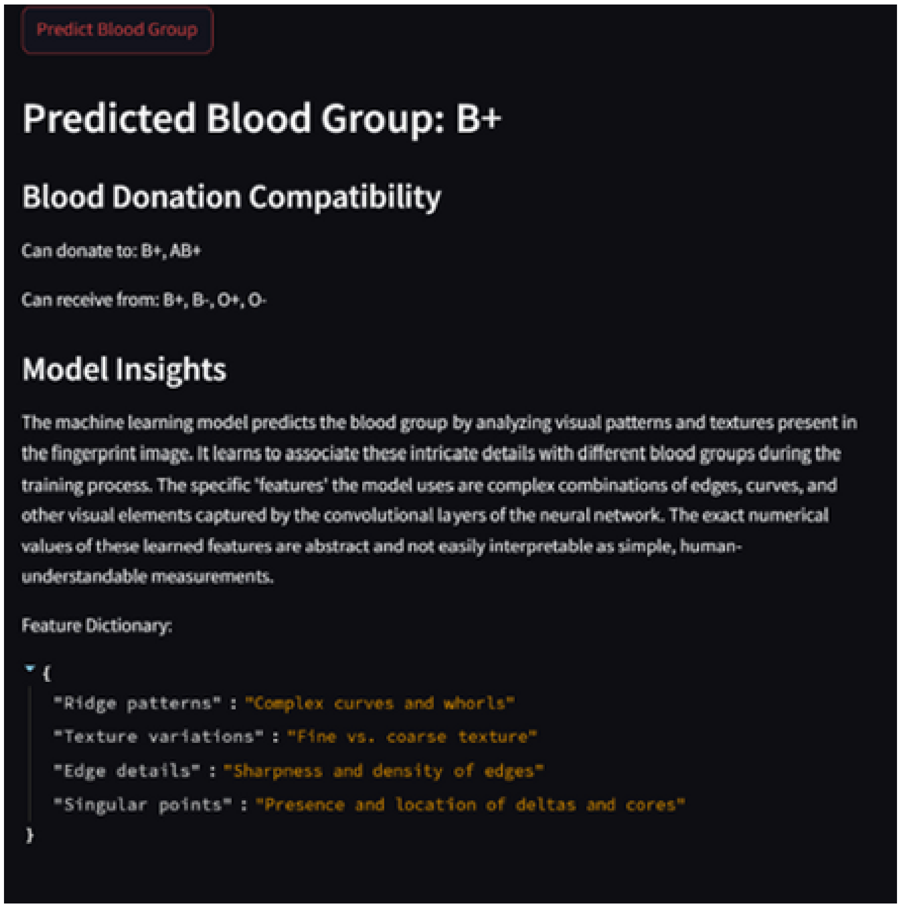
Browse files

[illegible]

The [illegible] parameter has been deprecated and will be removed in a future release. Please utilize the [illegible] parameter instead.

Uploaded Fingerprint

Predict Blood Group



Predicted Blood Group: B+

Blood Donation Compatibility

Can donate to: B+, AB+

Can receive from: B+, B-, O+, O-

Model Insights

The machine learning model predicts the blood group by analyzing visual patterns and textures present in the fingerprint image. It learns to associate these intricate details with different blood groups during the training process. The specific 'features' the model uses are complex combinations of edges, curves, and other visual elements captured by the convolutional layers of the neural network. The exact numerical values of these learned features are abstract and not easily interpretable as simple, human-understandable measurements.

Feature Dictionary:

```
{
  "Ridge patterns" : "Complex curves and whorls"
  "Texture variations" : "Fine vs. coarse texture"
  "Edge details" : "Sharpness and density of edges"
  "Singular points" : "Presence and location of deltas and cores"
}
```

RESULT:

The project successfully resulted in a trained CNN model and a functional Streamlit web application.

- Model Performance: The model was trained for 20 epochs, achieving a final validation accuracy of approximately 85.99% (val_acc: 0.8599) and a validation loss of 0.4827.
- Application: The Streamlit application provides a user-friendly interface where a user can upload a fingerprint image, receive an instant blood group prediction (e.g., "Predicted Blood Group: B+") , and view corresponding blood donation compatibility information.
- Predictions: The model demonstrates the ability to make correct predictions, as shown by examples where "Actual: AB+" was correctly predicted as "Predicted: AB+".