# Final Year Blood Group Detection Using Fingerprint Project:

## Dataset Training and Testing Code (Python):

```
 1. !pip install tensorflow
```

```
2. import tensorflow as tf
print("Num GPUs Available:",
len(tf.config.list_physical_devices('GPU')))

3. from google.colab import drive
drive.mount('/content/drive')

4. import os
dataset_path = '/content/drive/MyDrive/dataset_blood_group'
print(os.listdir(dataset_path))


5. # Step 2: Import Libraries
import os
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Step 3: Define Paths (Update with your actual dataset path)
dataset_path = "/content/drive/MyDrive/dataset_blood_group"  # Change
this path accordingly

# Step 4: Image Preprocessing
img_size = (128, 128)
batch_size = 32

datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,  # 80% Train, 20% Validation
    rotation_range=20,
    zoom_range=0.2,
    horizontal_flip=True
)

train_data = datagen.flow_from_directory(
    dataset_path,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='training'
)

val_data = datagen.flow_from_directory(
    dataset_path,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    subset='validation'
)

# Step 5: Define CNN Model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3,3), activation='relu',
input_shape=(128, 128, 3)),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Conv2D(128, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(train_data.num_classes, activation='softmax')
])

# Step 6: Compile the Model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Step 7: Train the Model
epochs = 20  # Adjust based on performance
history = model.fit(train_data, validation_data=val_data,
epochs=epochs)

# Step 8: Save the Model
```

```
model.save("/content/drive/My
Drive/dataset_blood_group/blood_group_model.h5")

# Step 9: Evaluate Model
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
plt.plot(acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.legend()
plt.show()

# Step 10: Load and Test Model
from tensorflow.keras.preprocessing import image

def predict_blood_group(img_path):
    img = image.load_img(img_path, target_size=img_size)
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)
    class_names = list(train_data.class_indices.keys())
    return class_names[np.argmax(prediction)]

# Example: Predict a single image (Change the path to an actual image)
test_image_path =
"/content/drive/MyDrive/dataset_blood_group/A+/test1.jpg"
print("Predicted Blood Group:", predict_blood_group(test_image_path))


6. from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_dir = '/content/drive/MyDrive/dataset_blood_group'

datagen = ImageDataGenerator(rescale=1./255)

train_data = datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'  # or 'binary' if only 2 classes
)

print("Detected Classes:", train_data.class_indices)
```

```
7. from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten,
Dense, Dropout

model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(4, activation='softmax')  # 4 classes: A, B, AB, O
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])



8. val_dir = '/content/drive/MyDrive/dataset_blood_group'

val_data = datagen.flow_from_directory(
    val_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='categorical'
)




9. model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150, 3)),
    MaxPooling2D(pool_size=(2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D(pool_size=(2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(8, activation='softmax')  # ✅ 8 classes, not 4
])
```

```
10. model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

history = model.fit(train_data, validation_data=val_data, epochs=20,
callbacks=[early_stop])
```

```
11. !pip install flask flask-cors tensorflow
```

```
12. import os
saved_model_path = "/content/drive/My
Drive/dataset_blood_group/blood_group_model.h5"
print(os.path.exists(saved_model_path))  # Should print True if model
is saved
```

```
13. from tensorflow.keras.models import load_model

model = load_model("/content/drive/My
Drive/dataset_blood_group/blood_group_model.h5")
```

```
14. model = tf.keras.models.load_model(
    "/content/drive/My Drive/dataset_blood_group/blood_group_model.h5",
compile=False
)
```

```
15. def predict_blood_group(img_path):
    img = image.load_img(img_path, target_size=(128, 128))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)
    class_names = list(train_data.class_indices.keys())
    return class_names[np.argmax(prediction)]

# Example test
test_image_path =
"/content/drive/MyDrive/dataset_blood_group/A+/cluster_0_1026.BMP"
print("Predicted Blood Group:", predict_blood_group(test_image_path))


16. import os
print(os.listdir())
```

```
17. from flask import Flask, request, jsonify
import tensorflow as tf
from tensorflow.keras.preprocessing import image
import numpy as np
import os
from flask_cors import CORS

app = Flask(__name__)  # <-- fix _name_ to __name__
CORS(app)

# Updated path to model
model = tf.keras.models.load_model("/content/drive/My
Drive/dataset_blood_group/blood_group_model.h5")
class_names = ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-']

@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No image uploaded'}), 400

    file = request.files['file']
    img_path = os.path.join('temp.jpg')
    file.save(img_path)

    img = image.load_img(img_path, target_size=(150, 150))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)
    predicted_class = class_names[np.argmax(prediction)]

    return jsonify({'blood_group': predicted_class})

# Fix __name__ typo
if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

**Python Flask ((Backend):**

```
!pip install flask flask-cors pyngrok




from flask import Flask, request, jsonify
from flask_cors import CORS
import tensorflow as tf
import numpy as np
from tensorflow.keras.preprocessing import image
from pyngrok import ngrok
import os

# Load the model
model =
tf.keras.models.load_model("/content/drive/MyDrive/dataset_blood_group/
blood_group_model.h5")
class_names = ['A+', 'A-', 'B+', 'B-', 'AB+', 'AB-', 'O+', 'O-']

app = Flask(__name__)
CORS(app)

@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No image uploaded'}), 400

    file = request.files['file']
    file.save("temp.jpg")

    img = image.load_img("temp.jpg", target_size=(128, 128))
    img_array = image.img_to_array(img) / 255.0
    img_array = np.expand_dims(img_array, axis=0)

    prediction = model.predict(img_array)
    predicted_class = class_names[np.argmax(prediction)]

    return jsonify({'blood_group': predicted_class})

# Open the Flask app via ngrok
```

```
public_url = ngrok.connect(5000)
print(f"Flask app is live at: {public_url}")

app.run(port=5000)
```

**React.js (Front end):**

**app.js**

```
// App.js
import React from 'react';
import BloodGroupDetector from './BloodGroupDetector';
import './App.css';

function App() {
  return (
    <div className="App">
      <h1>Blood Group Detection using Fingerprint</h1>
      <BloodGroupDetector />
    </div>
  );
}

export default App;
```

**bloodgroupdetector.js (src—>file(add by yourself)):**

```
// BloodGroupDetector.js
import React, { useState } from 'react';
import axios from 'axios';
import './App.css';

function BloodGroupDetector() {
  const [file, setFile] = useState(null);
  const [prediction, setPrediction] = useState('');
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');
  const handleUpload = async () => {
    if (!file) {
      setError("Please select an image.");
      return;
    }

    setLoading(true);
    setError('');
    setPrediction('');

    const formData = new FormData();
    formData.append('file', file);

    try {
      // Send to Python backend for prediction
      const response = await
axios.post('https://339b-34-19-106-48.ngrok-free.app/predict',
formData);
      const predictedGroup = response.data.blood_group;
      setPrediction(predictedGroup);

      // Send to Node backend for storage
      await axios.post('http://localhost:3001/store', { blood_group:
predictedGroup });

    } catch (err) {
      setError('Error while uploading or processing. Check server.');
    } finally {
      setLoading(false);
    }
  };

  return (
    <div className="detector-container">
      <input
        type="file"
        accept="image/*"
        onChange={e => setFile(e.target.files[0])}
        className="file-input"
      />
      <button onClick={handleUpload} className="upload-button">
        {loading ? 'Predicting...' : 'Predict'}
      </button>

      {prediction && <h2 className="result">Predicted Blood Group:
{prediction}</h2>}
      {error && <p className="error">{error}</p>}
    </div>
  );
}

export default BloodGroupDetector;
```

**Package.json(add at last curl braces )**

```
"proxy": "http://localhost:5000"
```

**App.css**

```
/* App.css */
.App {
  text-align: center;
  padding: 40px;
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  background-color: #f2f2f2;
  min-height: 100vh;
}

h1 {
  color: #333;
  margin-bottom: 30px;
}

.detector-container {
  display: flex;
  flex-direction: column;
  align-items: center;
}

.file-input {
  padding: 10px;
  margin-bottom: 20px;
  font-size: 16px;
}

.upload-button {
  padding: 10px 20px;
  background-color: #007bff;
  border: none;
  color: white;
  font-size: 16px;
  cursor: pointer;
  border-radius: 5px;
}

.upload-button:hover {
  background-color: #0056b3;
}

.result {
  color: green;
  margin-top: 20px;
  font-size: 20px;
}

.error {
  color: red;
  margin-top: 15px;
}
```

**Node.js(Backend data to store):**

**index.js**

```
// index.js
const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');
const mysql = require('mysql2');

const app = express();
const port = 3001;

app.use(cors());
app.use(bodyParser.json());

// MySQL connection setup
const db = mysql.createConnection({
  host: 'localhost',
  user: 'root',
  password: 'root', // change this
  database: 'blood_group_db'       // make sure this DB exists
});

db.connect(err => {
  if (err) {
    console.error('MySQL connection error:', err);
  } else {
    console.log('Connected to MySQL');
  }
});

// Create table if not exists
db.query(`
  CREATE TABLE IF NOT EXISTS predictions (
    id INT AUTO_INCREMENT PRIMARY KEY,
    blood_group VARCHAR(10),
    predicted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
  )
`);

// API endpoint to store prediction
app.post('/store', (req, res) => {
  const { blood_group } = req.body;

  if (!blood_group) {
    return res.status(400).json({ error: 'Blood group is required' });
  }

  const query = 'INSERT INTO predictions (blood_group) VALUES (?)';
  db.query(query, [blood_group], (err, result) => {
    if (err) {
      console.error('Error storing data:', err);
      res.status(500).json({ error: 'Failed to store blood group' });
    } else {
      res.status(200).json({ message: 'Blood group stored successfully'
});
```

```
    }
  });
});

app.listen(port, () => {
  console.log(`Node server running on http://localhost:${port}`);
});
```

**Mysql:**

CREATE DATABASE IF NOT EXISTS blood_group_db;
USE blood_group_db;

CREATE TABLE IF NOT EXISTS predictions (
  id INT AUTO_INCREMENT PRIMARY KEY,
  blood_group VARCHAR(10),
  predicted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
Select * from predictions;

**To run 👍**

**React:**
**cd filename**
**Npm start**

**Node:**
**node filename.js—--------->output: connected to mysql**

**For using Flask in google colab need to install (ngrok.com)**
**https://dashboard.ngrok.com/agentsa**
**And get token** 2wlyVq8at1fySO6ChYjxSfvMY6K_3KagheQZcmP3nnvZr1DE7 (my account token)

**After this only flask for backend would run otherwise you need to install flask app**