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

```
Mounted at /content/drive
```

```
from tensorflow.keras.models import load_model

# Load the pre-trained model
model = load_model('/content/drive/MyDrive/Docs/blood_group_classifier.keras')
```

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

# Define the ImageDataGenerator for rescaling test data
test_gen = ImageDataGenerator(rescale=1./255)  # Rescale pixel values to [0, 1]

# Load the test data from the directory
test_data = test_gen.flow_from_directory(
    '/content/drive/MyDrive/split_data/test',  # Replace with the path to your test dataset
    target_size=(224, 224),   # Resize all images to 224x224 (adjust based on your model's input size)
    batch_size=32,            # Number of samples per batch
    class_mode='categorical',  # 'categorical' for multi-class classification
    shuffle=False             # Do not shuffle test data (important for accurate evaluation)
)
```

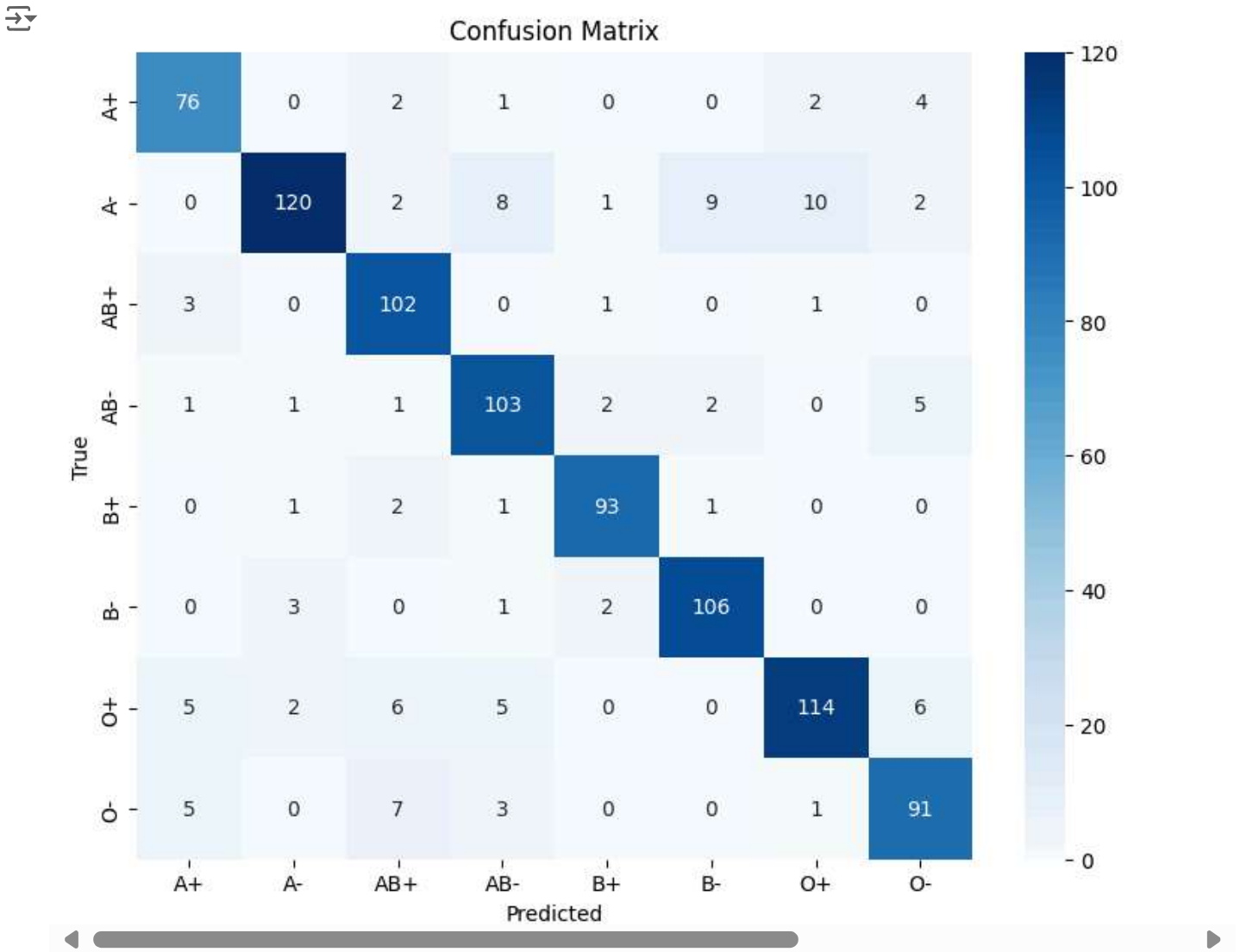
```
Found 914 images belonging to 8 classes.
```

```
import numpy as np
from sklearn.metrics import classification_report, accuracy_score

# Get predictions (probabilities) on the test dataset
y_pred_prob = model.predict(test_data)

# Convert probabilities to class labels (i.e., the class with the highest probability)
y_pred = np.argmax(y_pred_prob, axis=1)

# Get the true labels (class indices) from the test dataset
y_true = test_data.classes

# Print classification report (precision, recall, F1-score, etc.)
print(classification_report(y_true, y_pred))

# Calculate overall accuracy
accuracy = accuracy_score(y_true, y_pred)
print(f'Accuracy: {accuracy}')
```

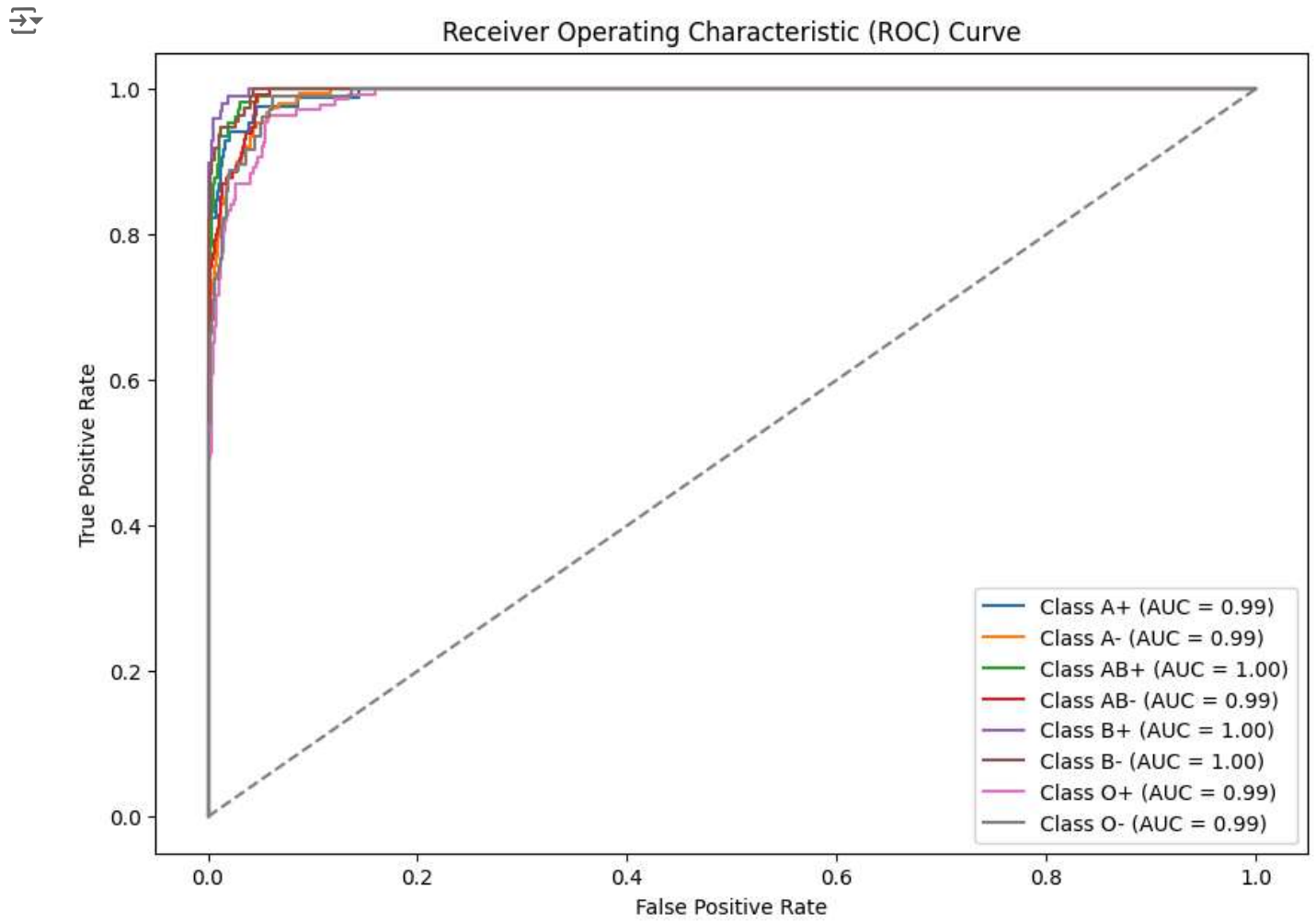
```
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: Use
  self._warn_if_super_not_called()
29/29 ━━━━━━━━━━━━━━━━━━━━ 271s 9s/step
              precision    recall  f1-score   support

           0       0.84      0.89      0.87        85
           1       0.94      0.79      0.86       152
           2       0.84      0.95      0.89       107
           3       0.84      0.90      0.87       115
           4       0.94      0.95      0.94        98
           5       0.90      0.95      0.92       112
```

```
              6       0.89      0.83      0.86       138
              7       0.84      0.85      0.85       107

       accuracy                           0.88       914
      macro avg       0.88      0.89      0.88       914
   weighted avg       0.88      0.88      0.88       914

   Accuracy: 0.8807439824945296
```

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Generate the confusion matrix
cm = confusion_matrix(y_true, y_pred)

# Plot the confusion matrix as a heatmap
plt.figure(figsize=(9, 7))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=test_data.class_indices.keys(), yticklabels=
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```



```
from sklearn.metrics import roc_curve, auc
from sklearn.preprocessing import label_binarize

# Binarize the true labels for multi-class classification
y_true_bin = label_binarize(y_true, classes=np.arange(len(test_data.class_indices)))

# Calculate ROC curve and AUC for each class
fpr, tpr, roc_auc = {}, {}, {}
for i in range(len(test_data.class_indices)):
    fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], y_pred_prob[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve for each class
plt.figure(figsize=(10, 7))
for i in range(len(test_data.class_indices)):
    plt.plot(fpr[i], tpr[i], label=f'Class {list(test_data.class_indices.keys())[i]} (AUC = {roc_auc[i]:.2f

plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```



```
import io
import tensorflow as tf
from google.colab import files
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Load the pre-trained model
model = tf.keras.models.load_model('/content/drive/MyDrive/Docs/blood_group_classifier.keras')

# Define a function to prepare the image (resize and rescale)
def prepare_image(image_bytes):
    img = Image.open(io.BytesIO(image_bytes))
    img = img.convert("RGB")  # Convert image to RGB format (removes alpha channel if present)
    img = img.resize((224, 224))  # Resize to match model input
    img_array = np.array(img) / 255.0  # Rescale the image to [0, 1]
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array

# Upload the image
uploaded = files.upload()

# Assuming only one image is uploaded
for fn in uploaded.keys():
    img_path = fn
    img_bytes = uploaded[fn]
    img_array = prepare_image(img_bytes)

    # Make prediction
    y_pred_prob = model.predict(img_array)
    y_pred = np.argmax(y_pred_prob, axis=1)
    blood_groups = ['A+', 'A-', 'AB+', 'AB-', 'B+', 'B-', 'O+', 'O-']
    predicted_blood_group = blood_groups[y_pred[0]]

    # Display the uploaded image and predicted result
    img = Image.open(io.BytesIO(img_bytes))
    plt.figure(figsize=(6, 6))
    plt.imshow(img)
    plt.title(f'Predicted: {predicted_blood_group}')
    plt.axis('off')
    plt.show()

    # If you have the actual label (for comparison), print it here
    # Example: Let's assume the actual label is "A+" for demonstration
    actual_blood_group = 'O-'  # Replace this with actual if available
    print(f"Actual Blood Group: {actual_blood_group}")
    print(f"Predicted Blood Group: {predicted_blood_group}")
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving cluster_7_2732.BMP to cluster_7_2732.BMP
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 373ms/step
```


```
Actual Blood Group: O-
```

```
!pip install gradio
```

```
Collecting gradio
  Downloading gradio-5.7.1-py3-none-any.whl.metadata (16 kB)
Collecting aiofiles<24.0,>=22.0 (from gradio)
  Downloading aiofiles-23.2.1-py3-none-any.whl.metadata (9.7 kB)
Requirement already satisfied: anyio<5.0,>=3.0 in /usr/local/lib/python3.10/dist-packages (from grad
Collecting fastapi<1.0,>=0.115.2 (from gradio)
  Downloading fastapi-0.115.5-py3-none-any.whl.metadata (27 kB)
Collecting ffmpy (from gradio)
  Downloading ffmpy-0.4.0-py3-none-any.whl.metadata (2.9 kB)
Collecting gradio-client==1.5.0 (from gradio)
  Downloading gradio_client-1.5.0-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: httpx>=0.24.1 in /usr/local/lib/python3.10/dist-packages (from gradio
Requirement already satisfied: huggingface-hub>=0.25.1 in /usr/local/lib/python3.10/dist-packages (f
Requirement already satisfied: jinja2<4.0 in /usr/local/lib/python3.10/dist-packages (from gradio) (
Collecting markupsafe~=2.0 (from gradio)
  Downloading MarkupSafe-2.1.5-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (
Requirement already satisfied: numpy<3.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from grad
Requirement already satisfied: orjson~=3.0 in /usr/local/lib/python3.10/dist-packages (from gradio)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from gradio) (2
Requirement already satisfied: pandas<3.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from gra
Requirement already satisfied: pillow<12.0,>=8.0 in /usr/local/lib/python3.10/dist-packages (from gr
Requirement already satisfied: pydantic>=2.0 in /usr/local/lib/python3.10/dist-packages (from gradio
Collecting pydub (from gradio)
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting python-multipart==0.0.12 (from gradio)
  Downloading python_multipart-0.0.12-py3-none-any.whl.metadata (1.9 kB)
```

```
Requirement already satisfied: pyyaml<7.0,>=5.0 in /usr/local/lib/python3.10/dist-packages (from gra
Collecting ruff>=0.2.2 (from gradio)
  Downloading ruff-0.8.1-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (25 kB)
Collecting safehttpx<1.0,>=0.1.1 (from gradio)
  Downloading safehttpx-0.1.1-py3-none-any.whl.metadata (4.1 kB)
Collecting semantic-version~=2.0 (from gradio)
  Downloading semantic_version-2.10.0-py2.py3-none-any.whl.metadata (9.7 kB)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.41.3-py3-none-any.whl.metadata (6.0 kB)
Collecting tomlkit==0.12.0 (from gradio)
  Downloading tomlkit-0.12.0-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.10/dist-packages (from gra
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.10/dist-packages (fr
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.32.1-py3-none-any.whl.metadata (6.6 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from gradio-client
Collecting websockets<13.0,>=10.0 (from gradio-client==1.5.0->gradio)
  Downloading websockets-12.0-cp310-cp310-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_17_x86_
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-packages (from anyio<5.0,
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.10/dist-packages (from anyio<5
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-packages (from anyio
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from httpx>=0.24.
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.10/dist-packages (from httpx>
Requirement already satisfied: h11<0.15,>=0.13 in /usr/local/lib/python3.10/dist-packages (from http
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from huggingface
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from huggingface
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.10/dist-packages (from hugging
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas<
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from panda
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.10/dist-packages (fr
```

```
import gradio as gr
import tensorflow as tf
import numpy as np
from PIL import Image

# Load the pre-trained model
model = tf.keras.models.load_model('/content/drive/MyDrive/Docs/blood_group_classifier.keras')

# Define a function to prepare the image (resize and rescale)
def prepare_image(image):
    img = image.convert("RGB")  # Convert image to RGB format (removes alpha channel if present)
    img = img.resize((224, 224))  # Resize to match model input
    img_array = np.array(img) / 255.0  # Rescale the image to [0, 1]
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array

# Define prediction function
def predict_blood_group(image):
    img_array = prepare_image(image)
    # Make prediction
    y_pred_prob = model.predict(img_array)
    y_pred = np.argmax(y_pred_prob, axis=1)
    blood_groups = ['A+', 'A-', 'AB+', 'AB-', 'B+', 'B-', 'O+', 'O-']
    predicted_blood_group = blood_groups[y_pred[0]]
    return predicted_blood_group

# Create Gradio Interface
iface = gr.Interface(
    fn=predict_blood_group,               # Prediction function
```

```
    inputs=gr.Image(type="pil"),          # Image input (upload image)
    outputs="text",                       # Output is text (predicted blood group)
    live=True,                           # Make predictions live as the user uploads
    title="Blood Group Prediction Using Fingerprint",  # Title (Heading)
    description="Upload a fingerprint image, and the model will predict the blood group.",  # Description
)

# Launch the interface
iface.launch()
```

```
Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you ca

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://cab9f059f8f198b404.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` f
```

No interface is running right now

```
import gradio as gr
import tensorflow as tf
import numpy as np
from PIL import Image

# Load the pre-trained model
model = tf.keras.models.load_model('/content/drive/MyDrive/Docs/blood_group_classifier.keras')

# Define a function to prepare the image (resize and rescale)
def prepare_image(image):
    img = image.convert("RGB")  # Convert image to RGB format (removes alpha channel if present)
    img = img.resize((224, 224))  # Resize to match model input
    img_array = np.array(img) / 255.0  # Rescale the image to [0, 1]
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array
```

```
    return img_array

# Define prediction function
def predict_blood_group(image):
    img_array = prepare_image(image)
    # Make prediction
    y_pred_prob = model.predict(img_array)
    y_pred = np.argmax(y_pred_prob, axis=1)
    blood_groups = ['A+', 'A-', 'AB+', 'AB-', 'B+', 'B-', 'O+', 'O-']
    predicted_blood_group = blood_groups[y_pred[0]]

    # Define actual output for comparison (this can be replaced by actual labels in real scenarios)
    actual_blood_group = 'A+'  # Replace this with the actual blood group if available

    # Return both predicted and actual results
    return f"Predicted Output: {predicted_blood_group}", f"Actual Output: {actual_blood_group}"

# Create Gradio Interface
iface = gr.Interface(
    fn=predict_blood_group,               # Prediction function
    inputs=gr.Image(type="pil"),          # Image input (upload image)
    outputs=[gr.Textbox(), gr.Textbox()], # Two text outputs (predicted and actual outputs)
    live=True,                            # Make predictions live as the user uploads
    title="Blood Group Prediction Using Fingerprint",  # Title of the interface
    description="Upload a fingerprint image to predict the blood group."  # Description
)

# Launch the interface
iface.launch()
```

```
Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you ca

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://824f82a89098c15e4b.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` f
```

No interface is running right now

```
import gradio as gr
import tensorflow as tf
import numpy as np
from PIL import Image

# Load the pre-trained model
model = tf.keras.models.load_model('/content/drive/MyDrive/Docs/blood_group_classifier.keras')

# Define a function to prepare the image (resize and rescale)
def prepare_image(image):
    img = image.convert("RGB")  # Convert image to RGB format (removes alpha channel if present)
    img = img.resize((224, 224))  # Resize to match model input
    img_array = np.array(img) / 255.0  # Rescale the image to [0, 1]
    img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
    return img_array

# Define prediction function
def predict_blood_group(image, actual_blood_group):
    img_array = prepare_image(image)
    # Make prediction
    y_pred_prob = model.predict(img_array)
    y_pred = np.argmax(y_pred_prob, axis=1)
    blood_groups = ['A+', 'A-', 'AB+', 'AB-', 'B+', 'B-', 'O+', 'O-']
    predicted_blood_group = blood_groups[y_pred[0]]
```

```
    # Return both predicted and actual results in one output
    return f"Predicted Blood Group: {predicted_blood_group}\nActual Blood Group: {actual_blood_group}"

# Create Gradio Interface
iface = gr.Interface(
    fn=predict_blood_group,               # Prediction function
    inputs=[gr.Image(type="pil"), gr.Textbox(label="Enter Actual Blood Group")],  # Image input and text ir
    outputs=gr.Textbox(),                  # Single output (both predicted and actual blood groups)
    live=True,                            # Make predictions live as the user uploads
    title="Blood Group Prediction Using Fingerprint",  # Title of the interface
    description="Upload a fingerprint imageto predict the blood group based on the fingerprint image."  # D
)

# Launch the interface
iface.launch()
```

```
Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you ca

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://03b2d164779733c5dc.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` f
```

No interface is running right now