```
# Mount Google Drive to access dataset
from google.colab import drive
drive.mount('/content/drive')
```

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

```
# Define dataset paths
train_data_dir = '/content/drive/MyDrive/Bone_data/train'
test_data_dir = '/content/drive/MyDrive/Bone_data/test'
```

```
from tensorflow.keras.applications import DenseNet121
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam

# Define the DenseNet121 model
base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the base model
base_model.trainable = False

# Create a new model on top of it
model = Sequential([
    base_model,
    Flatten(),
    Dense(256, activation='relu'),
    Dense(128, activation='relu'),
    Dense(6, activation='softmax')  # Assuming 6 classes of bone fractures
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Fit the model
history = model.fit(
    train_generator,
    epochs=100,
    validation_data=test_generator
)
```

```
Epoch 1/100
 2/11 ━━━━━━━━━━━━━━━━━━━━ 58s 6s/step - accuracy: 0.1719 - loss: 8.8707  /usr/local/lib/python3.10/dist-packages/PIL/Image.py:1056: UserWarning: Palette images with Transpa
  warnings.warn(
11/11 ━━━━━━━━━━━━━━━━━━━━ 201s 16s/step - accuracy: 0.1677 - loss: 17.1094 - val_accuracy: 0.3935 - val_loss: 5.1284
Epoch 2/100
```

```
11/11 ━━━━━━━━━━━━━━━━━━━━ 136s 10s/step - accuracy: 0.3399 - loss: 3.6424 - val_accuracy: 0.3742 - val_loss: 3.1201
Epoch 3/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 104s 9s/step - accuracy: 0.4882 - loss: 2.2086 - val_accuracy: 0.5290 - val_loss: 1.7122
Epoch 4/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 119s 11s/step - accuracy: 0.7361 - loss: 0.9336 - val_accuracy: 0.6968 - val_loss: 0.9776
Epoch 5/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 131s 10s/step - accuracy: 0.7937 - loss: 0.5055 - val_accuracy: 0.7677 - val_loss: 0.9064
Epoch 6/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 106s 9s/step - accuracy: 0.8836 - loss: 0.4348 - val_accuracy: 0.7677 - val_loss: 1.0674
Epoch 7/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 141s 9s/step - accuracy: 0.8618 - loss: 0.3832 - val_accuracy: 0.8000 - val_loss: 0.9373
Epoch 8/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 142s 9s/step - accuracy: 0.8915 - loss: 0.3174 - val_accuracy: 0.8323 - val_loss: 0.7162
Epoch 9/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 118s 11s/step - accuracy: 0.9384 - loss: 0.1725 - val_accuracy: 0.8194 - val_loss: 0.7417
Epoch 10/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 128s 9s/step - accuracy: 0.9435 - loss: 0.1887 - val_accuracy: 0.8065 - val_loss: 0.7604
Epoch 11/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 106s 9s/step - accuracy: 0.9681 - loss: 0.1009 - val_accuracy: 0.8387 - val_loss: 0.8127
Epoch 12/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 107s 9s/step - accuracy: 0.9605 - loss: 0.1329 - val_accuracy: 0.8387 - val_loss: 0.7666
Epoch 13/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 140s 9s/step - accuracy: 0.9708 - loss: 0.1426 - val_accuracy: 0.8516 - val_loss: 0.7364
Epoch 14/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 114s 10s/step - accuracy: 0.9650 - loss: 0.1163 - val_accuracy: 0.8387 - val_loss: 0.7893
Epoch 15/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 132s 9s/step - accuracy: 0.9836 - loss: 0.1024 - val_accuracy: 0.8581 - val_loss: 0.8087
Epoch 16/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 104s 9s/step - accuracy: 0.9846 - loss: 0.0635 - val_accuracy: 0.8516 - val_loss: 0.8057
Epoch 17/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 106s 9s/step - accuracy: 0.9588 - loss: 0.1605 - val_accuracy: 0.8516 - val_loss: 0.6986
Epoch 18/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 140s 9s/step - accuracy: 0.9625 - loss: 0.1142 - val_accuracy: 0.8645 - val_loss: 0.8220
Epoch 19/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 151s 10s/step - accuracy: 0.9857 - loss: 0.0337 - val_accuracy: 0.8710 - val_loss: 0.7806
Epoch 20/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 105s 9s/step - accuracy: 0.9885 - loss: 0.0691 - val_accuracy: 0.8645 - val_loss: 0.8739
Epoch 21/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 104s 9s/step - accuracy: 0.9662 - loss: 0.1043 - val_accuracy: 0.8452 - val_loss: 0.9489
Epoch 22/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 108s 9s/step - accuracy: 0.9772 - loss: 0.0753 - val_accuracy: 0.8258 - val_loss: 0.9028
Epoch 23/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 106s 9s/step - accuracy: 0.9703 - loss: 0.1192 - val_accuracy: 0.8581 - val_loss: 0.8490
Epoch 24/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 146s 10s/step - accuracy: 0.9853 - loss: 0.0714 - val_accuracy: 0.8387 - val_loss: 0.8900
Epoch 25/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 107s 9s/step - accuracy: 0.9771 - loss: 0.0751 - val_accuracy: 0.8581 - val_loss: 0.8549
Epoch 26/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 105s 9s/step - accuracy: 0.9806 - loss: 0.0970 - val_accuracy: 0.8516 - val_loss: 0.9147
Epoch 27/100
11/11 ━━━━━━━━━━━━━━━━━━━━ 153s 10s/step - accuracy: 0.9842 - loss: 0.0501 - val_accuracy: 0.8516 - val_loss: 0.9187
Epoch 28/100
```

```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_generator)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 31s 6s/step - accuracy: 0.8962 - loss: 0.8905
Test Accuracy: 86.45%
```

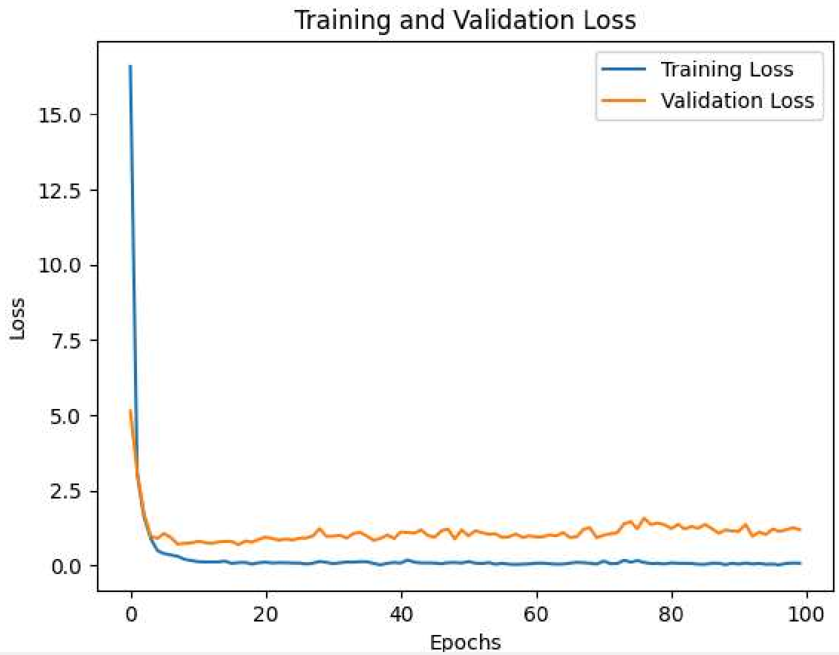
```
# Extracting True Labels and Predictions
import numpy as np

y_pred = model.predict(test_generator)
y_pred_classes = np.argmax(y_pred, axis=1)  # Convert one-hot to class indices
y_true = test_generator.classes              # True class labels

# Confusion Matrix and Classification Report
from sklearn.metrics import classification_report, confusion_matrix

# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred_classes)
print("Confusion Matrix:")
print(conf_matrix)

# Classification Report (Precision, Recall, F1-score)
class_report = classification_report(y_true, y_pred_classes, target_names=test_generator.class_indices.keys())
print("Classification Report:")
print(class_report)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 48s 7s/step
Confusion Matrix:
[[ 5  5  6  6  0  5]
 [ 6  4  3  7  0  5]
 [ 9  6 12  7  2  6]
 [ 8  5  6  4  4  3]
 [ 1  1  5  4  0  1]
 [ 2  4  6  5  0  2]]
Classification Report:
                         precision    recall  f1-score   support

       Avulsion fracture       0.16      0.19      0.17        27
     Greenstick fracture       0.16      0.16      0.16        25
       Hairline Fracture       0.32      0.29      0.30        42
Intra-articular fracture       0.12      0.13      0.13        30
        Oblique fracture       0.00      0.00      0.00        12
         Spiral Fracture       0.09      0.11      0.10        19

                accuracy                           0.17       155
               macro avg       0.14      0.14      0.14       155
            weighted avg       0.17      0.17      0.17       155
```

```
import matplotlib.pyplot as plt

# Retrieve accuracy data from history
train_acc = history.history['accuracy']
val_acc = history.history['val_accuracy']

# Plot accuracy
plt.plot(train_acc, label='Training Accuracy')
plt.plot(val_acc, label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```


```
train_loss = history.history['loss']
val_loss = history.history['val_loss']

# Plot loss
plt.plot(train_loss, label='Training Loss')
plt.plot(val_loss, label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
```

```
plt.legend()
plt.show()
```




```
from google.colab import files
import cv2
import numpy as np
from tensorflow.keras.preprocessing.image import img_to_array

# Step 1: Upload the image
uploaded = files.upload()

# Step 2: Load and preprocess the image
test_image_path = list(uploaded.keys())[0]
image = cv2.imread(test_image_path)
image_resized = cv2.resize(image, (224, 224))
image_array = img_to_array(image_resized) / 255.0
image_array = np.expand_dims(image_array, axis=0)

# Step 3: Make predictions using the model
predictions = model.predict(image_array)
predicted_class_index = np.argmax(predictions, axis=1)
class_labels = list(train_generator.class_indices.keys())
predicted_class_label = class_labels[predicted_class_index[0]]
```

```
print(f"The predicted class for the uploaded image is: {predicted_class_label}")
```

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 C7-FF2-1.jpg to C7-FF2-1.jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 5s 5s/step
```

```
from google.colab import files
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import img_to_array

# Step 1: Upload the image
uploaded = files.upload()

# Step 2: Load and preprocess the image
test_image_path = list(uploaded.keys())[0]
image = cv2.imread(test_image_path)
image_resized = cv2.resize(image, (224, 224))
image_array = img_to_array(image_resized) / 255.0
image_array = np.expand_dims(image_array, axis=0)

# Step 3: Make predictions using the model
predictions = model.predict(image_array)
predicted_class_index = np.argmax(predictions, axis=1)
class_labels = list(train_generator.class_indices.keys())
predicted_class_label = class_labels[predicted_class_index[0]]

# Step 4: Display the image and prediction
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB for correct color display
plt.title(f"Predicted Class: {predicted_class_label}")
plt.axis('off')  # Hide axes
plt.show()
```

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 7590tn.jpg to 7590tn.jpg
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step
```

Predicted Class: Spiral Fracture

```
from google.colab import files
import cv2
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing.image import img_to_array

# Step 1: Upload the image
uploaded = files.upload()

# Step 2: Load and preprocess the image
test_image_path = list(uploaded.keys())[0]
image = cv2.imread(test_image_path)
image_resized = cv2.resize(image, (224, 224))
image_array = img_to_array(image_resized) / 255.0
image_array = np.expand_dims(image_array, axis=0)

# Step 3: Make predictions using the model
predictions = model.predict(image_array)
predicted_class_index = np.argmax(predictions, axis=1)
class_labels = list(train_generator.class_indices.keys())
predicted_class_label = class_labels[predicted_class_index[0]]

# Step 4: Display the image and prediction
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB for correct color display
plt.title(f"Predicted Class: {predicted_class_label}")
```

```
    plt.title(f'Predicted Class: {predicted_class_label}')
    plt.axis('off')  # Hide axes
    plt.show()
```

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 94f6a-pclavulsion1a.png to 94f6a-pclavulsion1a.png
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 192ms/step
```

Predicted Class: Avulsion fracture