```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

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

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remou
```

Locate the File on Google Drive

```
!ls /content/drive/My\ Drive
```

```
'1.Differentiate between Internal and External security requirements (Cloud Security).gsheet'
'20211CSD0080 EE.pptx'
'Ai ml'
'Battery report'
'Colab Notebooks'
 Dataset
'Define BCP DRP. What are the alternative services during disasters .gsheet'
 HW
 MyJB
 project-main
 rvcDisconnected
```

```
import os
import zipfile

# Path to the .zip file on Google Drive
zip_path = '/content/drive/MyDrive/Dataset/archive.zip'  # Adjust this path based on where your file is

# Unzip the file to the correct location
output_path = '/content/dataset'  # Directory to extract files
os.makedirs(output_path, exist_ok=True)

with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(output_path)

print("Dataset extracted to:", output_path)
```

```
Dataset extracted to: /content/dataset
```

Verify Dataset Structure

```
# Verify the folder contents after extraction
print("Extracted Folders:", os.listdir(output_path))
```

```
Extracted Folders: ['damaged', 'intact']
```

```
damaged_dir = '/content/dataset/damaged'
intact_dir = '/content/dataset/intact'

print("Damaged Directory Contents:", os.listdir(damaged_dir))
print("Intact Directory Contents:", os.listdir(intact_dir))
```

```
Damaged Directory Contents: ['top', 'side']
Intact Directory Contents: ['top', 'side']
```

GPU usage (memory and utilization)

```
import tensorflow as tf
print("Is GPU available:", tf.config.list_physical_devices('GPU'))
```

```
Is GPU available: [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

```
!nvidia-smi
```

```
Tue Dec 17 05:57:14 2024
+---------------------------------------------------------------------------------------+
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version: 12.2     |
|-----------------------------------------+----------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |                    0 |
| N/A   62C    P0              30W /  70W |   1149MiB / 15360MiB |      0%      Default |
|                                         |                      |                  N/A |
+-----------------------------------------+----------------------+----------------------+

+---------------------------------------------------------------------------------------+
| Processes:                                                                            |
|  GPU   GI   CI        PID   Type   Process name                            GPU Memory |
|        ID   ID                                                             Usage      |
|=======================================================================================|
+---------------------------------------------------------------------------------------+
```

```
!ls /content/dataset  # List the contents of the 'dataset' directory
```

```
damaged  intact
```

```
train_dir = '/content/dataset/train'  # Update if 'train' folder is located somewhere else
val_dir = '/content/dataset/val'    # Update if 'val' folder is located somewhere else
```

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)  # Remount Drive
```

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

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

# Define paths
train_dir = '/content/dataset/damaged'  # Changed to 'damaged' directory
val_dir = '/content/dataset/intact'   # Changed to 'intact' directory

# Verify if directories exist
if not os.path.exists(train_dir):
    raise FileNotFoundError(f"Training directory not found: {train_dir}")
if not os.path.exists(val_dir):
    raise FileNotFoundError(f"Validation directory not found: {val_dir}")

# ImageDataGenerator for preprocessing
train_datagen = ImageDataGenerator(
    rescale=1./255,        # Normalize pixel values to [0, 1]
    rotation_range=20,     # Random rotations
    width_shift_range=0.2, # Horizontal shifts
    height_shift_range=0.2,# Vertical shifts
    shear_range=0.2,       # Random shearing
    zoom_range=0.2,        # Zoom
    horizontal_flip=True,  # Random horizontal flip
    fill_mode='nearest'    # Fill mode for any newly created pixels after transformations
)

val_datagen = ImageDataGenerator(rescale=1./255)

# Flow images from directories without moving them
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(224, 224),  # Resize the images to the target size (you can adjust this)
    batch_size=32,
    class_mode='binary',     # Or 'categorical' if more classes are present
    subset='training'        # Subset for training
)
```

```
val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(224, 224),  # Resize the images to the target size
    batch_size=32,
    class_mode='binary'      # Or 'categorical'
)

# Optional: For debugging, print out the class labels and directories
print("Train Directory Structure: ", train_generator.class_indices)
print("Validation Directory Structure: ", val_generator.class_indices)
```

```
Found 200 images belonging to 2 classes.
Found 200 images belonging to 2 classes.
Train Directory Structure:  {'side': 0, 'top': 1}
Validation Directory Structure:  {'side': 0, 'top': 1}
```

Model Training using ResNet

```
Start coding or generate with AI.
```

```
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout
from tensorflow.keras.optimizers import Adam

# Load VGG16 model with pre-trained weights (without top layers)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Freeze the base model
base_model.trainable = False

# Create the new model
model = Sequential([
    base_model,
    GlobalAveragePooling2D(),  # Convert the 2D feature maps into a 1D feature vector
    Dropout(0.5),              # Add dropout to prevent overfitting
    Dense(512, activation='relu'),
    Dropout(0.5),              # Add another dropout layer for regularization
    Dense(1, activation='sigmoid')  # Output layer for binary classification
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])

# Summary of the model
model.summary()
```

```
Model: "sequential_2"
```

Layer (type)	Output Shape	Param #
vgg16 (Functional)	(None, 7, 7, 512)	14,714,688
global_average_pooling2d_2 (GlobalAveragePooling2D)	(None, 512)	0
dropout_3 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 512)	262,656
dropout_4 (Dropout)	(None, 512)	0
dense_5 (Dense)	(None, 1)	513

```
Total params: 14,977,857 (57.14 MB)
Trainable params: 263,169 (1.00 MB)
```

```
from tensorflow.keras.applications import ResNet50 # Import ResNet50

base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
# Set a lower learning rate for the optimizer
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.applications import ResNet50 # Import ResNet50

# Recreate the model with the updated learning rate
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False

# Build the model
model = tf.keras.Sequential([
    base_model,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')  # Binary classification
])

model.compile(optimizer=Adam(learning_rate=0.0001),  # Lower learning rate
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Callbacks for early stopping and reducing learning rate
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2)

# Train the model
history = model.fit(train_generator,
                    epochs=20,
                    validation_data=val_generator,
                    callbacks=[early_stop, reduce_lr])
```

```
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `
  self._warn_if_super_not_called()
7/7 ━━━━━━━━━━━━━━━━━━━━ 34s 3s/step - accuracy: 0.5811 - loss: 0.7544 - val_accuracy: 0.5000 - val_loss: 0.6513 - lear
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 957ms/step - accuracy: 0.5645 - loss: 0.6947 - val_accuracy: 1.0000 - val_loss: 0.6119 - l
Epoch 3/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 21s 764ms/step - accuracy: 0.6425 - loss: 0.6493 - val_accuracy: 1.0000 - val_loss: 0.5759 - l
Epoch 4/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 749ms/step - accuracy: 0.6160 - loss: 0.6353 - val_accuracy: 1.0000 - val_loss: 0.5466 - l
Epoch 5/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 22s 954ms/step - accuracy: 0.6725 - loss: 0.6042 - val_accuracy: 1.0000 - val_loss: 0.5177 - l
Epoch 6/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 19s 747ms/step - accuracy: 0.7212 - loss: 0.5657 - val_accuracy: 1.0000 - val_loss: 0.4919 - l
Epoch 7/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 20s 954ms/step - accuracy: 0.7607 - loss: 0.5083 - val_accuracy: 1.0000 - val_loss: 0.4676 - l
Epoch 8/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 10s 746ms/step - accuracy: 0.8684 - loss: 0.4513 - val_accuracy: 1.0000 - val_loss: 0.4433 - l
Epoch 9/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 21s 759ms/step - accuracy: 0.7393 - loss: 0.4915 - val_accuracy: 1.0000 - val_loss: 0.4335 - l
Epoch 10/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 20s 953ms/step - accuracy: 0.7932 - loss: 0.4817 - val_accuracy: 1.0000 - val_loss: 0.4066 - l
Epoch 11/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 10s 755ms/step - accuracy: 0.8381 - loss: 0.4311 - val_accuracy: 1.0000 - val_loss: 0.3838 - l
Epoch 12/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 21s 781ms/step - accuracy: 0.8490 - loss: 0.4075 - val_accuracy: 1.0000 - val_loss: 0.3666 - l
Epoch 13/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 988ms/step - accuracy: 0.9083 - loss: 0.3659 - val_accuracy: 1.0000 - val_loss: 0.3501 - l
Epoch 14/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 975ms/step - accuracy: 0.8976 - loss: 0.3704 - val_accuracy: 1.0000 - val_loss: 0.3327 - l
Epoch 15/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 21s 769ms/step - accuracy: 0.9550 - loss: 0.3165 - val_accuracy: 1.0000 - val_loss: 0.3157 - l
Epoch 16/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 820ms/step - accuracy: 0.9435 - loss: 0.3235 - val_accuracy: 1.0000 - val_loss: 0.3021 - l
Epoch 17/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 19s 747ms/step - accuracy: 0.9494 - loss: 0.3312 - val_accuracy: 1.0000 - val_loss: 0.2879 - l
Epoch 18/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 12s 755ms/step - accuracy: 0.9349 - loss: 0.2925 - val_accuracy: 1.0000 - val_loss: 0.2744 - l
Epoch 19/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 20s 969ms/step - accuracy: 0.9231 - loss: 0.3321 - val_accuracy: 1.0000 - val_loss: 0.2598 - l
Epoch 20/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 10s 833ms/step - accuracy: 0.9552 - loss: 0.2938 - val_accuracy: 1.0000 - val_loss: 0.2480 - l
```

```
import tensorflow as tf
from tensorflow.keras import regularizers
tf.keras.layers.Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.01))
```

```
<Dense name=dense_8, built=False>
```

```
class ValidationAccuracyCap(tf.keras.callbacks.Callback):
    def __init__(self, cap=0.75):
        super().__init__()
        self.cap = cap

    def on_epoch_end(self, epoch, logs=None):
        if logs.get('val_accuracy') > self.cap:
            print(f"Validation accuracy {logs.get('val_accuracy')} exceeded {self.cap}, stopping training.")
            self.model.stop_training = True
```

```
from tensorflow.keras.applications import MobileNetV2

base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
```

```
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=40,
    width_shift_range=0.4,
    height_shift_range=0.4,
    shear_range=0.3,
    zoom_range=0.3,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras import layers, models, regularizers
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Load pre-trained MobileNetV2
base_model = MobileNetV2(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
base_model.trainable = False  # Freeze base model

# Build the model
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.Dropout(0.7),  # Increased dropout
    layers.Dense(1, activation='sigmoid')  # Binary classification
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='binary_crossentropy',
              metrics=['accuracy'])

# Callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2)
val_cap = ValidationAccuracyCap(cap=0.75)  # Cap validation accuracy

# Train the model
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=20,
    callbacks=[early_stop, reduce_lr, val_cap]
)
```

```
Epoch 1/20
6/7 ━━━━━━━━━━━━━━━━━━━━ 0s 772ms/step - accuracy: 0.5304 - loss: 3.1578Validation accuracy 0.9850000143051147 exceeded
7/7 ━━━━━━━━━━━━━━━━━━━━ 31s 2s/step - accuracy: 0.5353 - loss: 3.1404 - val_accuracy: 0.9850 - val_loss: 2.7675 - lear
```

```
val_cap = ValidationAccuracyCap(cap=0.8)
```

```
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=45,
    width_shift_range=0.5,
    height_shift_range=0.5,
    shear_range=0.4,
    zoom_range=0.5,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='nearest'
)
```

```
model = models.Sequential([
    layers.Conv2D(16, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `i
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# ipython-input-55-68eacd4ad853

from tensorflow.keras import layers, models, regularizers

# ... (your existing code to define train_datagen) ...

# Build the model
model = models.Sequential([
    layers.Conv2D(16, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])

# Compile the model
from tensorflow.keras.optimizers import Adam # Import Adam if needed
model.compile(optimizer=Adam(learning_rate=0.0001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
# Callbacks with adjusted parameters
early_stop = EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3)
val_cap = ValidationAccuracyCap(cap=0.8)  # Adjust cap

# Train the model
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=20,
    callbacks=[early_stop, reduce_lr, val_cap]
)
```

```
Epoch 1/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 19s 1s/step - accuracy: 0.4924 - loss: 2.0892 - val_accuracy: 0.5000 - val_loss: 2.0772 - lear
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 13s 652ms/step - accuracy: 0.5315 - loss: 1.8782 - val_accuracy: 0.5000 - val_loss: 1.6284 - l
Epoch 3/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 659ms/step - accuracy: 0.5369 - loss: 1.6175 - val_accuracy: 0.5000 - val_loss: 1.4639 - l
Epoch 4/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 19s 710ms/step - accuracy: 0.5713 - loss: 1.4393 - val_accuracy: 0.5000 - val_loss: 1.3499 - l
Epoch 5/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 10s 642ms/step - accuracy: 0.4946 - loss: 1.3284 - val_accuracy: 0.5000 - val_loss: 1.2511 - l
Epoch 6/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 21s 850ms/step - accuracy: 0.5236 - loss: 1.2402 - val_accuracy: 0.5000 - val_loss: 1.1658 - l
Epoch 7/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 9s 685ms/step - accuracy: 0.5413 - loss: 1.1610 - val_accuracy: 0.5000 - val_loss: 1.1027 - le
Epoch 8/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 12s 881ms/step - accuracy: 0.4655 - loss: 1.1102 - val_accuracy: 0.5000 - val_loss: 1.0465 - l
Epoch 9/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 651ms/step - accuracy: 0.5987 - loss: 1.0308 - val_accuracy: 0.5000 - val_loss: 1.0086 - l
Epoch 10/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 20s 881ms/step - accuracy: 0.5209 - loss: 1.0089 - val_accuracy: 0.5000 - val_loss: 0.9566 - l
Epoch 11/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 21s 643ms/step - accuracy: 0.4930 - loss: 0.9628 - val_accuracy: 0.5000 - val_loss: 0.9304 - l
Epoch 12/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 714ms/step - accuracy: 0.4724 - loss: 0.9528 - val_accuracy: 0.5000 - val_loss: 0.8963 - l
Epoch 13/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 10s 806ms/step - accuracy: 0.5102 - loss: 0.9059 - val_accuracy: 0.5000 - val_loss: 0.8680 - l
Epoch 14/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 10s 652ms/step - accuracy: 0.5372 - loss: 0.8815 - val_accuracy: 0.5000 - val_loss: 0.8473 - l
Epoch 15/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 12s 766ms/step - accuracy: 0.5884 - loss: 0.8533 - val_accuracy: 0.5000 - val_loss: 0.8341 - l
Epoch 16/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 12s 644ms/step - accuracy: 0.5430 - loss: 0.8273 - val_accuracy: 0.5050 - val_loss: 0.7984 - l
Epoch 17/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 18s 648ms/step - accuracy: 0.8759 - loss: 0.8342 - val_accuracy: 0.5050 - val_loss: 0.7841 - l
Epoch 18/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 11s 648ms/step - accuracy: 0.6131 - loss: 0.7874 - val_accuracy: 0.5000 - val_loss: 0.7849 - l
Epoch 19/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 21s 819ms/step - accuracy: 0.6260 - loss: 0.7909 - val_accuracy: 0.5100 - val_loss: 0.7404 - l
Epoch 20/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 20s 642ms/step - accuracy: 0.7164 - loss: 0.7604 - val_accuracy: 0.5050 - val_loss: 0.7349 - l
```

```
# Save the model in HDF5 format
model.save('model.h5')

# Save in TensorFlow's SavedModel format
model.save('saved_model.keras')  # Changed extension to .keras
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This fil
```

```
# Save the model in the native Keras format
model.save('model.keras')
```

```
# Save to Google Drive in Keras format
model.save('/content/drive/My Drive/model.keras')
```

```
from google.colab import files

# Download the .keras model file
files.download('model.keras')
```

```
test_loss, test_accuracy = model.evaluate(test_data)
print(f"Test Accuracy: {test_accuracy * 100:.2f}%")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-49-07236b4f70b5> in <cell line: 1>()
----> 1 test_loss, test_accuracy = model.evaluate(test_data)
      2 print(f"Test Accuracy: {test_accuracy * 100:.2f}%")

NameError: name 'test_data' is not defined
```

```
# Remove the ValidationAccuracyCap callback for now
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=20,
    callbacks=[early_stop, reduce_lr]
)
```

```
Epoch 1/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 13s 1s/step - accuracy: 0.8771 - loss: 0.7644 - val_accuracy: 0.5050 - val_loss: 0.7252 - lear
Epoch 2/20
7/7 ━━━━━━━━━━━━━━━━━━━━ 17s 866ms/step - accuracy: 0.7167 - loss: 0.7294 - val_accuracy: 0.5050 - val_loss: 0.7048 - l
Epoch 3/20
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-42-fc0504bc74a4> in <cell line: 2>()
      1 # Remove the ValidationAccuracyCap callback for now
----> 2 history = model.fit(
      3     train_generator,
      4     validation_data=val_generator,
      5     epochs=20,

                                 2 frames
/usr/local/lib/python3.10/dist-packages/keras/src/callbacks/callback_list.py in on_train_batch_begin(self, batch,
logs)
     96             callback.on_epoch_end(epoch, logs)
     97
---> 98     def on_train_batch_begin(self, batch, logs=None):
     99         logs = logs or {}
    100         for callback in self.callbacks:

KeyboardInterrupt:
```

```
from tensorflow.keras import layers, models, regularizers

model = models.Sequential([
    # Convolutional Layers
    layers.Conv2D(16, (3, 3), activation='relu', input_shape=(224, 224, 3)),
    layers.MaxPooling2D((2, 2)),

    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

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

    # Fully Connected Layers
    layers.Flatten(),
    layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')  # Binary classification
])

model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Train the model
history = model.fit(
    train_generator,
    validation_data=val_generator,
    epochs=20,
    callbacks=[early_stop, reduce_lr]
)
```

```
model = models.Sequential([
    layers.Conv2D(16, (3, 3), activation='relu', input_shape=(224, 224, 3), kernel_regularizer=regularizers.l2(0.01)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(32, (3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.MaxPooling2D((2, 2)),
```

```
    layers.Flatten(),
    layers.Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)),
    layers.Dropout(0.5),
    layers.Dense(1, activation='sigmoid')
])
```

```
train_datagen = ImageDataGenerator(
    rescale=1.0/255.0,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
    loss='binary_crossentropy',
    metrics=['accuracy']
)
```

```
early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restore_best_weights=True)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=2, min_lr=1e-6)
```

```
import os

# Get the current working directory
current_path = os.getcwd()
print(f"Current working directory: {current_path}")
```

```
# List the contents of the current working directory
contents = os.listdir(current_path)
print(f"Contents of the current directory: {contents}")

# If you have a 'test_data' folder, check if it exists
test_data_path = os.path.join(current_path, 'test_data')
if os.path.exists(test_data_path):
    print(f"Test data directory found at: {test_data_path}")
else:
    print("Test data directory not found.")
```

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

# Setup the ImageDataGenerator
test_datagen = ImageDataGenerator(rescale=1./255)

# Update the path to point to the "dataset" directory
test_data = test_datagen.flow_from_directory(
    'dataset',  # Path to the dataset directory
    target_size=(224, 224),  # Resize to match the model's input size
    batch_size=32,  # Adjust the batch size as needed
    class_mode='binary',  # or 'categorical' depending on your problem
    shuffle=False  # No need to shuffle for evaluation
)

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

```
from sklearn.utils import class_weight
import numpy as np
# Calculate class weights
class_weights = class_weight.compute_class_weight('balanced',
                                                  classes=np.unique(train_data.classes),
                                                  y=train_data.classes)

# Convert class_weights to a dictionary
class_weights_dict = dict(enumerate(class_weights))
```

```
# Pass class_weights to the model during training
model.fit(train_data, epochs=20, class_weight=class_weights_dict)
```

```
from sklearn.utils import class_weight
import numpy as np

# Assuming train_generator is your Keras ImageDataGenerator
# Extract the class labels from the ImageDataGenerator
class_labels = list(train_generator.class_indices.keys())

# Get the actual class labels from the generator's filenames
train_classes = train_generator.classes

# Calculate class weights using the extracted labels and classes
class_weights = class_weight.compute_class_weight('balanced',
                                                  classes=np.unique(train_classes),
                                                  y=train_classes)

# Convert class_weights to a dictionary
class_weights_dict = dict(enumerate(class_weights))

# Pass class_weights to the model during training
# Ensure you are using the correct data for training
history = model.fit(
    train_generator,  # Use train_generator if it's what you used before
    epochs=20,
    class_weight=class_weights_dict,
    validation_data=val_generator  # If you used validation data before
)
```

```
Start coding or generate with AI.
```

TEST

```
import zipfile
import os

# Path to the uploaded zip file
zip_path = '/content/Test12.zip'

# Extracting the zip file
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall('/content/Test12')

# Confirming extraction
print("Extracted files:")
print(os.listdir('/content/Test12'))
```

```
Extracted files:
['15a (14).png', '15a (7).png', '15a (4).png', '13d.png', '16b (2).png', '11b.jpeg', '15a (16).png', '14d.png', '15a (1
```

```
from tensorflow.keras.models import load_model

# Path to the uploaded model file
model_path = '/content/model.keras'

# Load the model
model = load_model(model_path)
print("Model loaded successfully!")
```

```
Model loaded successfully!
```

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

# Directory of extracted test data
test_dir = '/content/Test12'
```

```
# Prepare test data generator
test_datagen = ImageDataGenerator(rescale=1./255)  # Normalize pixel values

# Load test data
test_data = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',
    shuffle=False
)
```

```
Found 0 images belonging to 0 classes.
```

```
import os
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np

# Path to test images
test_image_dir = '/content/Test12'

# Get all image file paths
image_paths = [os.path.join(test_image_dir, fname) for fname in os.listdir(test_image_dir) if fname.lower().endswith(('png'

# Confirm images are detected
print(f"Found {len(image_paths)} images for testing.")

# Preprocess the images
test_images = []
for path in image_paths:
    img = load_img(path, target_size=(224, 224))  # Resize to model input size
    img_array = img_to_array(img) / 255.0  # Normalize pixel values
    test_images.append(img_array)

# Convert to numpy array
test_images = np.array(test_images)
```

```
Found 32 images for testing.
```

```
class_labels = list(test_data.class_indices.keys())
```

```
# Safely map predictions to class labels
predicted_labels = []
for pred_class in predicted_classes:
    if pred_class < len(class_labels):  # Ensure the index is within bounds
        predicted_labels.append(class_labels[pred_class])
    else:
        predicted_labels.append("Unknown")  # Label for out-of-bound predictions
```

```
import os

# Path to your unzipped test data directory
test_dir = 'Test12'

# List contents of the test directory
print(f"Contents of {test_dir}: {os.listdir(test_dir)}")
```

```
Contents of Test12: ['15a (14).png', '15a (7).png', '15a (4).png', '13d.png', '16b (2).png', '11b.jpeg', '15a (16).png'
```

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

# Directory of extracted test data
test_dir = '/content/Test12'

# Prepare test data generator
test_datagen = ImageDataGenerator(rescale=1./255)  # Normalize pixel values

# Load test data
test_data = test_datagen.flow_from_directory(
```

```
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',  # Use 'categorical' for multi-class classification
    shuffle=False
)

# Get class labels from the generator
class_labels = list(test_data.class_indices.keys())

# Print class labels for debugging
print("Class Labels:", class_labels)

# ... (rest of your prediction and evaluation code) ...
```

```
Found 0 images belonging to 0 classes.
Class Labels: []
```

```
# Print detailed predictions
for path, probs in zip(image_paths, predictions):
    print(f"Image: {path}")
    for label, prob in zip(class_labels, probs):
        print(f"  {label}: {prob:.2f}")
    print()
```

```
Image: /content/Test12/15a (14).png

Image: /content/Test12/15a (7).png

Image: /content/Test12/15a (4).png

Image: /content/Test12/13d.png

Image: /content/Test12/16b (2).png

Image: /content/Test12/11b.jpeg

Image: /content/Test12/15a (16).png

Image: /content/Test12/14d.png

Image: /content/Test12/15a (15).png

Image: /content/Test12/15a (11).png

Image: /content/Test12/11a.jpeg

Image: /content/Test12/14a.png

Image: /content/Test12/16b (3).png

Image: /content/Test12/15a (10).png

Image: /content/Test12/12b.jpeg

Image: /content/Test12/14b.png

Image: /content/Test12/16b (1).png

Image: /content/Test12/12a.jpeg

Image: /content/Test12/13c.png

Image: /content/Test12/15a (2).png

Image: /content/Test12/15a (12).png

Image: /content/Test12/14c.png

Image: /content/Test12/15a (13).png

Image: /content/Test12/16b (4).png

Image: /content/Test12/15a (6).png

Image: /content/Test12/13a.png

Image: /content/Test12/15a (5).png
```

```
Image: /content/Test12/13b.png

Image: /content/Test12/15a (8).png
```

```
import tensorflow as tf  # Import TensorFlow at the beginning of the cell

predictions = tf.nn.softmax(predictions).numpy()
```

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

test_datagen = ImageDataGenerator(rescale=1./255)  # Normalize pixel values

test_data = test_datagen.flow_from_directory(
    'Test15',  # Path to test data
    target_size=(224, 224),  # Same size as used during training
    batch_size=1,  # Predicting one image at a time
    class_mode=None,  # No labels in test data
    shuffle=False  # Preserve the order for debugging
)
```

```
Found 0 images belonging to 0 classes.
```

```
if len(test_data) > 0:  # Ensure there are images
    predictions = model.predict(test_data)  # Shape: (num_samples, num_classes)
    print(f"Predictions shape: {predictions.shape}")
else:
    print("No valid images found for testing.")
```

```
No valid images found for testing.
```

```
class_labels = ['Damaged', 'Intact']  # Original classes
```

```
threshold = 0.95
```

```
for probs in predictions:
    print("Probabilities:", probs, "Sum:", sum(probs))
```

```
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
Probabilities: [0.99999994] Sum: 0.9999999403953552
```

```
for path, probs in zip(image_paths, predictions):
    print(f"Image: {path}")
```

```
    predicted_class_idx = np.argmax(probs)  # Index of max probability
    confidence = probs[predicted_class_idx]  # Confidence of prediction

    if confidence >= threshold:
        predicted_label = class_labels[predicted_class_idx]
        print(f"  Predicted: {predicted_label} with confidence {confidence:.2f}")
    else:
        print(f"  Predicted: Unknown (confidence {confidence:.2f})")

    # Debugging: Print all class probabilities
    for i, prob in enumerate(probs):
        label = class_labels[i] if i < len(class_labels) else "Unknown (extra output)"
        print(f"  {label}: {prob:.2f}")
    print()
```

```
Image: /content/Test12/15a (14).png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/15a (7).png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/15a (4).png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/13d.png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/16b (2).png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/11b.jpeg
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/15a (16).png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/14d.png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/15a (15).png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/15a (11).png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/11a.jpeg
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/14a.png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/16b (3).png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/15a (10).png
  Predicted: Damaged with confidence 1.00
  Damaged: 1.00

Image: /content/Test12/12b.jpeg
  Predicted: Damaged with confidence 1.00
```