[4]
```
!pip install -q tensorflow opencv-python matplotlib
```

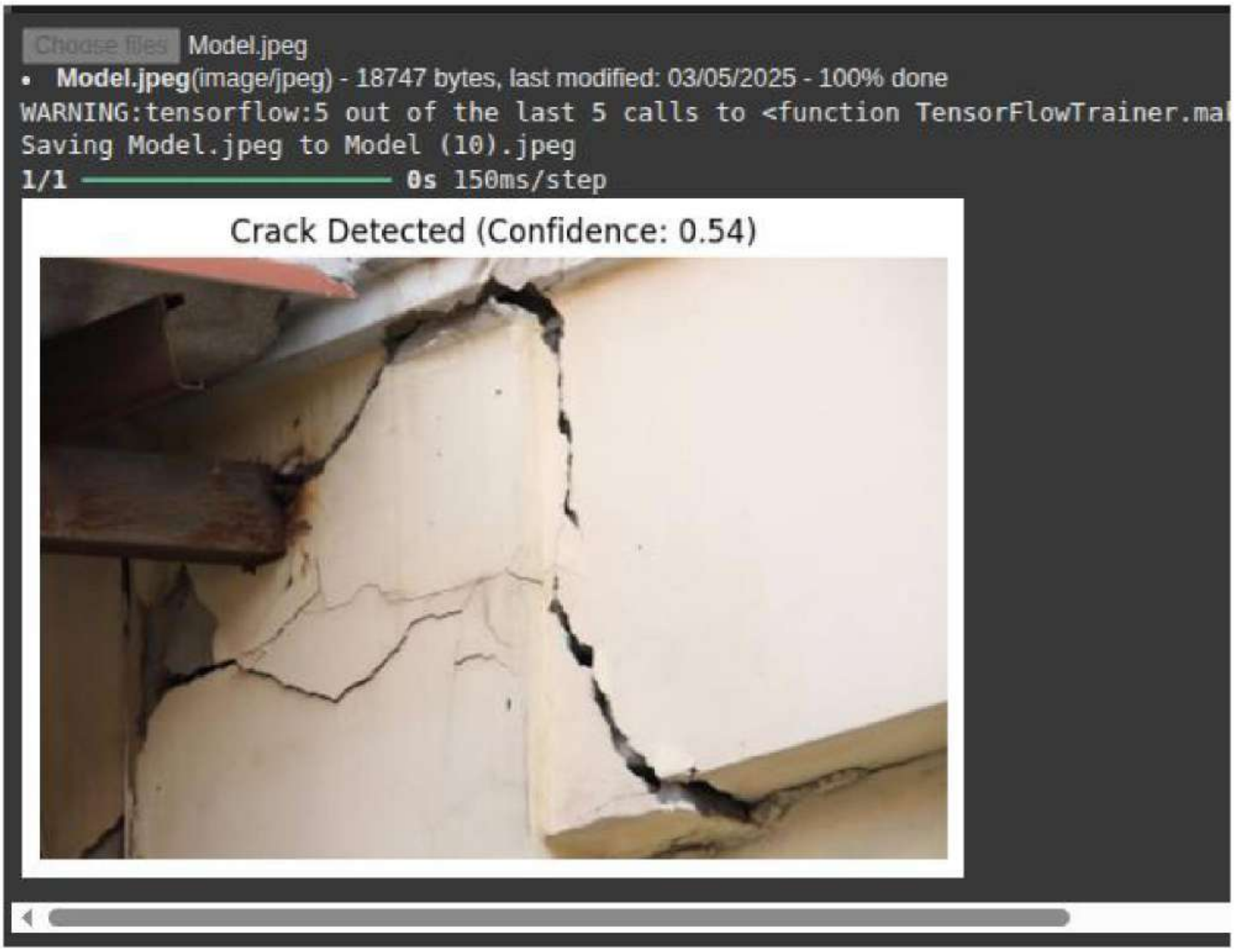
[6]
```
import cv2
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import img_to_array
from google.colab import files
```

Double-click (or enter) to edit

```
import numpy as np
import cv2
import io
import tensorflow as tf
from PIL import Image
from google.colab import files
import matplotlib.pyplot as plt

# Define the function to create the model
def create_simple_crack_model(input_shape=(224, 224, 3)):
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
        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(1, activation='sigmoid')  # Binary classification
    ])

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

# Initialize the model
model = create_simple_crack_model()
```

```python
    model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

# Initialize the model
model = create_simple_crack_model()

# Assuming the model is already trained, or you can train it here.
# If you have a pre-trained model saved, you can load it using:
# model = tf.keras.models.load_model('path_to_model')

# Upload images
uploaded = files.upload()

# Define the detection function
def detect_crack(img_array, model):
    img = cv2.resize(img_array, (224, 224))
    img = img.astype('float32') / 255.0
    img = np.expand_dims(img, axis=0)
    prediction = model.predict(img)[0][0]
    label = "Crack Detected" if prediction > 0.1 else "No Crack Detected"
    return label, prediction

# Process uploaded images
for fname in uploaded.keys():
    # Read using PIL to support all formats, then convert to OpenCV format
    image_data = uploaded[fname]
    pil_img = Image.open(io.BytesIO(image_data)).convert('RGB')
    img_array = np.array(pil_img)
    img_bgr = cv2.cvtColor(img_array, cv2.COLOR_RGB2BGR)

    # Get prediction and confidence
    label, conf = detect_crack(img_bgr, model)

    # Display result
    plt.imshow(pil_img)
    plt.title(f'{label} (Confidence: {conf:.2f})')
    plt.axis('off')
    plt.show()
```

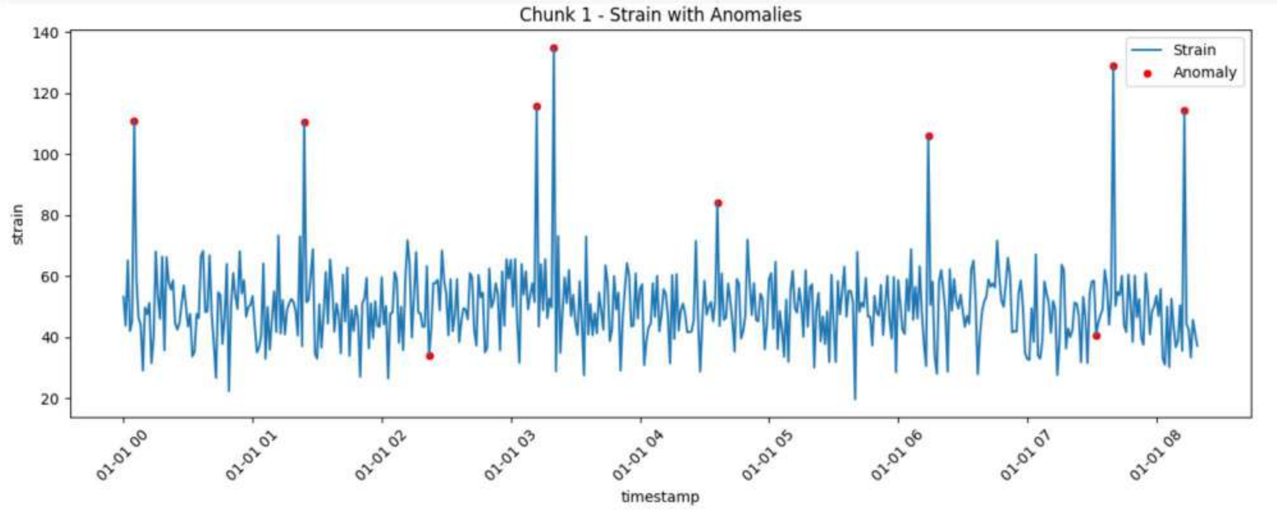
output:


Choose files Model.jpeg
• Model.jpeg(image/jpeg) - 18747 bytes, last modified: 03/05/2025 - 100% done
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.ma
Saving Model.jpeg to Model (10).jpeg
1/1 0s 150ms/step
Crack Detected (Confidence: 0.54)

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
import os
from datetime import datetime

# --------- Setup Paths ----------
BASE_DIR = "/content"
RAW_DIR = os.path.join(BASE_DIR, "raw")
os.makedirs(RAW_DIR, exist_ok=True)
CSV_PATH = os.path.join(RAW_DIR, "shm_data.csv")
CHUNK_SIZE = 500
MAX_CHUNKS = 2  # Limit for concise output

# --------- Generate Synthetic Data ----------
def generate_data(filepath, rows=1000):
    timestamps = pd.date_range(start="2023-01-01", periods=rows, freq="1min")
    df = pd.DataFrame({
        "timestamp": timestamps,
        "strain": np.random.normal(50, 10, rows),
        "vibration": np.random.normal(0.3, 0.1, rows),
        "displacement": np.random.normal(5, 2, rows),
        "temperature": np.random.normal(30, 3, rows),
    })

    anomalies = np.random.choice(rows, size=20, replace=False)
    df.loc[anomalies, 'strain'] += np.random.normal(80, 15, len(anomalies))
    df.loc[anomalies, 'vibration'] += np.random.normal(1.5, 0.3, len(anomalies))

    df.to_csv(filepath, index=False)
    print(f"Data saved at {filepath}")
```

```python
# --------- Read Data in Chunks ----------
def read_chunks(path, chunk_size):
    return pd.read_csv(path, chunksize=chunk_size, parse_dates=["timestamp"])

# --------- Detect Anomalies ----------
def detect_anomalies(df):
    features = ["strain", "vibration", "displacement", "temperature"]
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(df[features])
    model = IsolationForest(contamination=0.02, random_state=42)
    df['anomaly'] = model.fit_predict(X_scaled)
    return df[df['anomaly'] == -1]

# --------- Plot One Graph Only ----------
def plot_anomalies(df, anomalies, chunk_num):
    plt.figure(figsize=(12, 5))
    sns.lineplot(x='timestamp', y='strain', data=df, label='Strain')
    if not anomalies.empty:
        sns.scatterplot(x='timestamp', y='strain', data=anomalies, color='red', label='Anomaly')
    plt.title(f"Chunk {chunk_num} - Strain with Anomalies")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()

# --------- Main SHM Function ----------
def monitor_shm():
    if not os.path.exists(CSV_PATH):
        generate_data(CSV_PATH)

    for i, chunk in enumerate(read_chunks(CSV_PATH, CHUNK_SIZE)):
        if i >= MAX_CHUNKS:
            break
        anomalies = detect_anomalies(chunk)
        print(f"Chunk {i+1}: {len(anomalies)} anomalies detected")
        plot_anomalies(chunk, anomalies, i+1)

monitor_shm()
```

Output:




```
Chunk 2: 10 anomalies detected
```