Frontend development deploying Surface Crack Detection Deep Learning model

Summer Intern project report submitted to
Indian Institute of Technology Kharagpur
in partial fulfilment for the award of the dual degree
in
Structural Engineering of the
Civil Engineering Department
by

Vadde Manjula Murari
(19CE31001)

Under the supervision of
Professor Damodar Maity

DEPARTMENT OF CIVIL ENGINEERING

Indian Institute of Technology Kharagpur

Autumn Semester 2023-2024

DEPARTMENT OF CIVIL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

CERTIFICATE

TO WHOM IT MAY CONCERN

This is to certify that Mr. Vadde Manjula Murari (19CE31001) is a student of the Department of Civil Engineering, Indian Institute of Technology Kharagpur and is pursuing fifth year dual degree in Structural engineering specialization at present. He has carried out his summer internship under my guidance during 1st June to 31st July 2023. The title of his summer project was "**Front- end development deploying surface crack detection using deep learning**". He has the ability and confidence to deliver his knowledge and idea. He is diligent and has a strong motivation for work. He has a sound moral character and a pleasant personality.

I wish him all success in his endeavours.

Prof. Damodar Maity

Professor Department of Civil Engineering

Indian Institute of Technology Kharagpur

Date: August 25th ,2023

Place: Kharagpur

Declaration

I certify that

(a) The work contained in this report has been done by me under the guidance of my supervisor.

(b) The work has not been submitted to any other Institute for any degree or diploma.

(c) I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

(d) Whenever I have used materials (data, theoretical analysis, figures, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references. Further, I have taken permission from the copyright owners of the sources, whenever necessary.

Date: 25th August, 2023 (Vadde Manjula Murari)

Place: Kharagpur (19CE31001)

Acknowledgements

First of all, I would like to thank my project advisor Prof. Damodar Maity for all the help and guidance throughout the project in the right direction and solving my doubts and queries whenever I got stuck.

I would also like to thank all others whose direct or indirect help has benefited me in doing this project.

Lastly, I would like to thank my friends and my family members for the encouragement and support they have provided me during this whole project. This accomplishment would not have been possible without them. Thank you.

Contents

Abstract

This report outlines the development and deployment of the EchoAI website, a groundbreaking project by EchoAI, a startup specializing in artificial intelligence and deep learning solutions. The website integrates deep learning technology, offering live camera access, file management, and real-time image analysis. This report explores the methodology, features, outcomes, and future prospects of the project.

Introduction

The project aimed to create a user-friendly web application that integrates live camera access, file management, and deep learning-based image analysis of concrete surfaces. The motivation behind this project was to provide a platform for real-time image processing and analysis of concrete structures and monitor their health.

Technologies Used:

Front-End: HTML, CSS, JavaScript

Back-End: Flask

Deep Learning: Custom model for image analysis built in Bachelor Technology Thesis.

Crack detection in a structure is one of the most important tools to judge the safety, durability of the structure. After detecting the crack, it is also important to study the characteristics of the crack such as length of the crack, width of the crack and on which portion of the structure the crack is present etc. in order to find the severity of the crack on the structure. Although human based detection of a crack and its characteristics is the best way but it is very time consuming as well as risky according to the location of the crack. Inspectors cannot analysis all the cracks of the building, as they evaluate according to their experience and existing guidelines but it is necessary to study about all the cracks in the structure to judge safety of the structure with precision.

To overcome the drawbacks of human way of detection and evaluating crack many image processing techniques have been developed. Among all of them one of the best and accurate way is with Deep Learning. Artificial Neural Networks and few machine learning techniques can be used to build the model which detects the crack and returns the characteristics of the crack. Though the performance of the model depends on various factors such as diversity of the data, huge data, architecture od neural network etc.

Among all the **Artificial Neural Network** techniques the best one for image classification is *Convolutional Neural Network* (CNN). In this paper a CNN model is proposed to classify the images if it is cracked or not. The one very good advantage of CNN is, it is not necessary to change the format of input and the model tries to learn the patterns and features in the data automatically without any traditional calculations. This actually reduces the work load of mathematical calculation. CNN also fundamentally alters with the way we approach any computer vision problem.

Objective

The primary objective of this project is to develop a web-based system that enables real-time monitoring and analysis of video streams from a device's camera, as well as the retrospective analysis of uploaded video files. The system aims to accurately detect cracks or defects in the visual content and provide users with actionable insights, including real-time alerts and downloadable analysis results.

Literature review

- Image-Based Concrete Crack Detection Using Convolutional Neural Network and Exhaustive Search Technique by Shengyuan Li and Xuefeng Zhao, *Advances in Civil Engineering*, vol. 2019, in this paper, a deep CNN (AlexNet) is proposed to establish an image classifier for crack detection. The outstanding advantage of the proposed CNN-based crack detection is that it spares multifarious work from features pre-extraction and calculation compared to traditional methods.

- Performance Evaluation of Deep CNN-Based Crack Detection and Localization Techniques for Concrete Structures by Luqman Ali, Fady Alnajjar, Hamad Al Jassmi, Munkhjargal Gocho, Wasif Khan, M. Adel Serhani, 2021. In this paper compares the performance of Customized CNN, VGG-16, VGG-19, ResNet-50, Inception V3 convolutional neural networks architectures/models for image-based crack detection. This paper says the VGG-19 model benefited from larger datasets. The ResNet-50 and Inception-V3 models demonstrated a slight change in performance after increasing the size of the training data.

- Cha, Y.J.; Choi, W.; Büyüköztürk, O. Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks. *Comput. Civ. Infrastruct. Eng.* 2017, *32*, 361–378. In this article there is clear description of the operations that a convolutional neural network performs and show a very simple model that gives high accuracy. There is also mentioned that the accuracy increases with the DB size of the image.

- Yann LeCun, Bernhard Boser, John Denker, Donnie Herderson, R. Howard, Wayne Hubbard, Lawrence Jackel handwritten digit recognition with a back-propagation network Adv. Neural Inf. Proces. Syst., 2 (1989), pp. 396-404. In this article there is clear description of how back propagation performs to update the weights of the model.

- Crack detection using image processing: A critical review and analysis Arun Mohan, Sumathi Poobal, Alexandria Engineering Journal Volume 57, Issue 2, June 2018, Pages 787-798. In this paper there is clear description how cracks can be detected using Video image and camera based, image processing techniques.

- Xiuying Meng, "Concrete Crack Detection Algorithm Based on Deep Residual Neural Networks", Scientific Programming, vol. 2021, Article ID 3137083, 7 pages, 2021. In this paper author discussed about an effective architecture which works as encoder and decoder. It also discussed about data labelling image processing technique to highlight the crack and data augmentation is performed to increase the dataset and diverse.

- X. Yang, H. Li, Y. Yu, X. Luo, T. Huang, and X. Yang, “Automatic pixel-level crack detection and measurement using fully convolutional network,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 33, no. 12, pp. 1090–1109, 2018. In this article a model is built from basic convolution, soft-max, max pooling layers and a very good accuracy is obtained. There clear description of how a model is trained and weights are assigned and how loss function is minimized.

Methodology

Project Architecture:

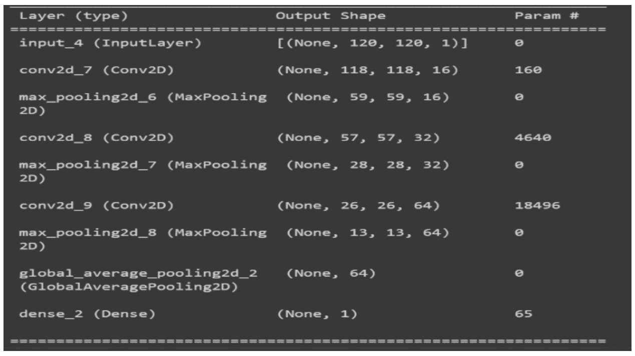
The EchoAI website consists of a robust front-end and a dynamic back-end. The front-end caters to user interactions, while the back-end handles data processing and analysis.
The web application should take the dynamic images as input analysis them using the deep learning model than works in backend of the application, if there is a crack then the backend should store the frame of the for further analysis of deriving characteristics of the crack using such as length of the crack, width of the crack.

Deep Learning Model

We've developed a custom neural network model to perform image analysis. The model is trained to detect objects and patterns in real-time camera frames, making it a vital component of the website's functionality.

Model summary:

```
inputs = tf.keras.Input(shape=(120, 120,1))
x = tf.keras.layers.Conv2D(filters=16, kernel_size=(3, 3), activation='relu')(inputs)
x = tf.keras.layers.MaxPool2D(pool_size=(2, 2))(x)
x = tf.keras.layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu')(x)
x = tf.keras.layers.MaxPool2D(pool_size=(2, 2))(x)
x = tf.keras.layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu')(x)
x = tf.keras.layers.MaxPool2D(pool_size=(2, 2))(x)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)
```



```
 Layer (type)                Output Shape              Param #
=================================================================
 input_4 (InputLayer)        [(None, 120, 120, 1)]     0

 conv2d_7 (Conv2D)           (None, 118, 118, 16)      160

 max_pooling2d_6 (MaxPooling  (None, 59, 59, 16)       0
 2D)

 conv2d_8 (Conv2D)           (None, 57, 57, 32)        4640

 max_pooling2d_7 (MaxPooling  (None, 28, 28, 32)       0
 2D)

 conv2d_9 (Conv2D)           (None, 26, 26, 64)        18496

 max_pooling2d_8 (MaxPooling  (None, 13, 13, 64)       0
 2D)

 global_average_pooling2d_2   (None, 64)               0
 (GlobalAveragePooling2D)

 dense_2 (Dense)             (None, 1)                 65

=================================================================
```

This the model built for the detecting the surface cracks using images of two set cracked image dataset and uncracked image set. We have attained an approximately 98% of both testing and training accuracy, this shows the model is not overfitting.

Key Features:

Live Camera Access: Users can access their device's camera in real-time through the website.
File Upload/Download: File management features enable users to upload and download files for analysis.
Real-Time Image Analysis: Captured frames from the camera or uploaded files are processed using our deep learning model for instant insights.

Failed Approach:

Image Segmentation: The algorithm starts by dividing each video frame into smaller segments. For example, if the original frame size is (1920, 1920, 3), it is divided into 256 equal-sized segments (assuming 16x16 grid).
Segment Analysis: Each of these smaller segments is then independently fed into a deep learning model for analysis. The purpose is to determine whether there is a crack present in each segment. This analysis is performed for every frame in the video.
Crack Detection: If a crack is detected in a segment, the algorithm marks or highlights that specific segment within the frame to indicate the presence of a crack. This is typically done using visual cues, such as drawing a bounding box around the cracked area.
Time Complexity: The time complexity of this algorithm is a critical consideration, especially for high-resolution videos. In this case, the time complexity is O(n^2), where 'n' represents the number of segments into which each frame is divided.

In our example, where each frame is divided into 256 segments (16x16 grid), the algorithm performs analysis on 256 segments.
For each frame in the video, this analysis is repeated. Therefore, for 'm' frames in the video, the total time complexity becomes O(m * n^2).

Implications of High Time Complexity:

Computational Cost: As the video resolution increases or the number of segments per frame increases, the computational cost grows quadratically. This can lead to significantly longer processing times, making it impractical for real-time analysis of high-resolution videos.
Resource Intensive: The high time complexity also requires substantial computational resources, which may limit the algorithm's usability on standard hardware.
Optimization Challenges: Optimizing the algorithm to reduce its time complexity is essential for real-world applications, especially when dealing with large video datasets.

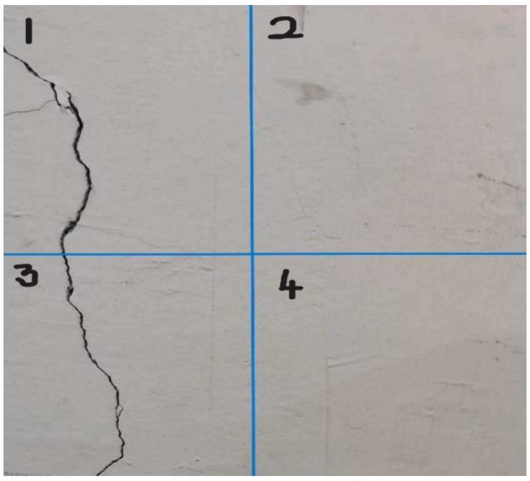
If img is the pointer of a image then here the algorithm:

```
for k in range(0,1920,120):
    for j in range(0, 1920, 120):
        a = np_img[k:k+120,j:j+120]
        b = np.expand_dims(a, axis=0)
        if model.predict(b) >= 0.5:
```

```
                    np_img[k:k+120,j:j+120]=np_img[k:k+120,j:j+120]*255
                    cv2.rectangle(np_img1, (k, j), (120, 120), (0, 0, 255), 2)
                    np_img[k:k+120,j:j+120, 1] = np_img[k:k+120,j:j+120, 1] / 2
          np_img[k:k+120,j:j+120, 2] = np_img[k:k+120,j:j+120, 2] / 2
```

Approach:

The approach that is used to optimize this is using the binary approach. Firstly, the image is divided into 4 equal parts and fed to the model. If there is a crack in the crack then the segment is further divided into 4 parts and fed to the model recursively until if the size of the image is 120x120x3. If there is no crack in the image then the image is ignored and no segmentation of the image is done.
So in the previous approach we have divided 1920x1920 image into 256 parts of 120X120 and tested each of the segment, time cost of this algorithm is 100 units for a frame/image. This approach can optimize the time cost to much smaller.



- Here image is divided the image to 4 equal square as shown in the image above.
- The image segment 1 is fed to model first, if there is a crack in the image then image 1 is further segmented to 4 parts and fed to the model recursively.
- After completing the recursion of the image segment 1, image segment 2 is fed to the model, as there is no crack in segment 2, the algorithm ignores segment 2, no further segmentation is done.
- As segment 1, segment 3 is tested recursively and segment 4 is ignored for further testing as there is no crack, same as segment 2.
- Here if observed segment 2, segment 4 are ignored, 25 parts from segment 2 and 25 parts from segment 4 are ignored. 50 operations of computation is done in just 2 operations, optimized.

Here is the code for the algorithm:

```
def rec(img,l,b,i,j):
    img1=img
    if(l!=120 and b!=120):
        img = np.resize(img, (120, 120,3))
    if(model.predict(img)<0.5):
        return img
    if(l==120 and b==120):
        if(model.predict(img)>=0.5):
            img[:,:,1]=img[:,:,1]/2
            return img
    else:
        img1[i:l/2,j:b/2]=rec(img1[i:l/2,j:b/2],l/2,b/2,i,j)
        img1[l/2:l,j:b/2]=rec(img1[l/2:l,j:b/2],l,b/2,l/2,j)
        img1[i:l/2,b/2:b]=rec(img1[i:l/2,b/2:b],l/2,b,i,b/2)
        img1[l/2:l,b/2:b]=rec(img1[l/2:l,b/2:b],l,b,l/2,b/2)
    return img1
```

The above code tests for every frame and modifies the frames highlighting the cracked portion of the crack and every original frame is stored for further computation of deriving other characteristics of the crack. For all of these the above algorithm optimizes the time.

Reasons why the above algorithm failed:

- Slicing of 3D Numpy Array: In the first step, a 3D Numpy array with dimensions n x n is sliced into four equal-sized 3D Numpy arrays. This slicing divides the original array into four smaller quadrants.
- Data Division: Each of these smaller quadrants represents a portion of the original data. The division is performed to enable further processing on smaller, manageable chunks of data.
- Resizing for Model Compatibility: To feed these smaller 3D arrays into a deep learning model, they are resized to a consistent size of 120x120x3 pixels. This resizing is necessary to ensure compatibility between the input data and the model's expectations.
- Data Loss: A significant issue arises during the resizing process. To obtain the 120x120x3 size, only the first n/2 x n/2 elements of each smaller 3D array are considered. This selection results in a considerable loss of data, as the remaining elements are ignored.
- Recursive Process: This entire process is typically performed recursively. Each of the four smaller quadrants is further divided into four even smaller quadrants, and the resizing and data loss process is repeated at each level of recursion.
- Cumulative Data Loss: With each recursive step, more data is lost due to the resizing operation. This cumulative data loss can have a substantial impact on the quality and accuracy of the model's output.
- Model Output: As a result of the data loss, the model's output may not meet the desired expectations. The reduced amount of information provided to the model can lead to a decrease in its ability to make accurate predictions or analyses.

- Trade-off: The described process represents a trade-off between reducing the computational complexity by working with smaller chunks of data and the loss of information that occurs during resizing. It's essential to carefully consider the balance between these factors to achieve the desired results in a deep learning application.

How to deal this approach

Multi-Resolution Models: The key idea is to build multiple deep learning models, each optimized for a specific image size. You start with a base model designed to handle large images with dimensions of 1920x1920x3.
Crack Detection at Full Resolution: When an image is fed into the initial model, it first checks for the presence of cracks at the full resolution. If a crack is detected, further analysis is needed.
Image Segmentation: If a crack is identified, the image is divided into four equal-sized segments, each with dimensions of 960x960x3. This segmentation process decomposes the image into smaller parts to focus on areas of interest.
Segment-Specific Models: At this point, a different deep learning model, specifically tailored for 960x960x3 images, is used to analyze each of the four segments. These segment-specific models are designed to be more efficient for processing smaller areas.
Crack Detection at Intermediate Scale: The segment-specific models examine their respective segments and determine whether any contain cracks. If one or more segments are found to have cracks, you proceed to analyze those segments further.
Recursive Segmentation: If a segment is identified as having a crack, it can be further divided into four smaller segments, each with dimensions of 480x480x3. This recursive process continues as long as cracks are detected within the segments.
Final Crack Detection: The recursion stops when the segments reach a predetermined minimum size, e.g., 120x120x3. At this smallest scale, if a segment contains a crack, it is marked or highlighted to indicate the presence of a defect.
Advantages:
Efficiency: This approach optimizes the use of computational resources by employing models that match the size of the input data. Smaller models are used for smaller segments, reducing the computational cost.
Multi-Scale Analysis: By considering images at multiple scales, the approach ensures that cracks of various sizes can be detected efficiently.
Localized Analysis: It allows for localized analysis, focusing computational efforts on regions of interest where cracks are suspected.
Challenges:
Model Coordination: Coordinating the activities of multiple models, especially in a recursive setup, can be complex and may require careful synchronization.
Training: Training and fine-tuning multiple models for different image sizes can be time-consuming and require substantial labeled data.

Segmentation Logic: Implementing the logic for recursive segmentation and crack marking can be intricate. Which I have mentioned the code above.
This hierarchical deep learning approach is suitable for scenarios where crack detection needs to be efficient across images of varying sizes, making it adaptable to different use cases and types of images.

Key features of the website

Real-Time Camera Access: Users visiting the website have the capability to access their device's camera in real-time.
Frame Capture: The live camera feed continuously captures frames at a certain frame rate (e.g., 30 frames per second). These frames represent individual images that make up the video stream.
Flask Backend: On the backend, a Flask server is set up to handle incoming requests from the website. This server includes a defined video route that is responsible for processing video frames.
Frame Transmission: As each frame is captured by the camera, it is sent as a request to the Flask server via the defined video route. The data for each frame is transmitted as part of the HTTP request.
Model Processing: Upon receiving a frame, the Flask server passes the frame data to a deep learning model for analysis. The model is specifically designed for crack detection.
Crack Detection: The deep learning model analyzes each frame to determine whether there is a crack present. This analysis can involve complex computer vision algorithms and neural networks trained to identify cracks within images.
Frame Storage: If the model detects a crack in a frame, that particular frame is stored in a predefined folder on the server's filesystem. This allows for later inspection or further action, such as notifying users or generating reports.
Real-Time Feedback: The website's user interface can provide real-time feedback to users based on the model's analysis. For example, it can display notifications or warnings when a crack is detected in the live camera feed.
Continuous Processing: This process continues as long as the camera feed is active, with each frame being transmitted to the Flask server, analyzed by the model, and stored if a crack is found.
User Interaction: Users can interact with the website to control the camera feed, access historical data, or perform other actions related to crack detection.
This setup allows for real-time crack detection in a live camera feed through a web application. It leverages Flask on the backend to receive and process video frames, and a deep learning model for crack detection. The frames with detected cracks are stored for further analysis or action, providing a valuable tool for monitoring and inspection purposes.
File Upload Option: Within your web application's user interface, you've provided users

with the option to upload a file. This can typically be done through an HTML form that allows users to select and submit a file from their local device.

Flask Backend Routing: On the Flask backend, you've defined a specific route that handles incoming file uploads. This route is responsible for processing and storing the uploaded files. You may have used the Flask request object to access the uploaded file data.

File Storage Location: The Flask backend specifies a folder where uploaded files should be stored. This folder is typically designated as an "uploads" folder, and it's where all user-submitted files are saved.

Analysis of Uploaded Video: When a user uploads a video file, your backend begins the analysis process. It may involve parsing the video file frame by frame, similar to how live camera frames are processed. Your deep learning model or other analysis algorithms are applied to each frame to detect cracks or perform other actions.

Frame-Level Analysis: Frame-level analysis continues frame by frame until the entire video is processed. The results of the analysis, such as frame modifications or any detected cracks, are stored or recorded as needed.

Download Route: After the video has been analyzed and potentially modified frame by frame, you've set up a Flask route that allows users to download the processed video. This route sends the modified video to the user's browser in a downloadable format.

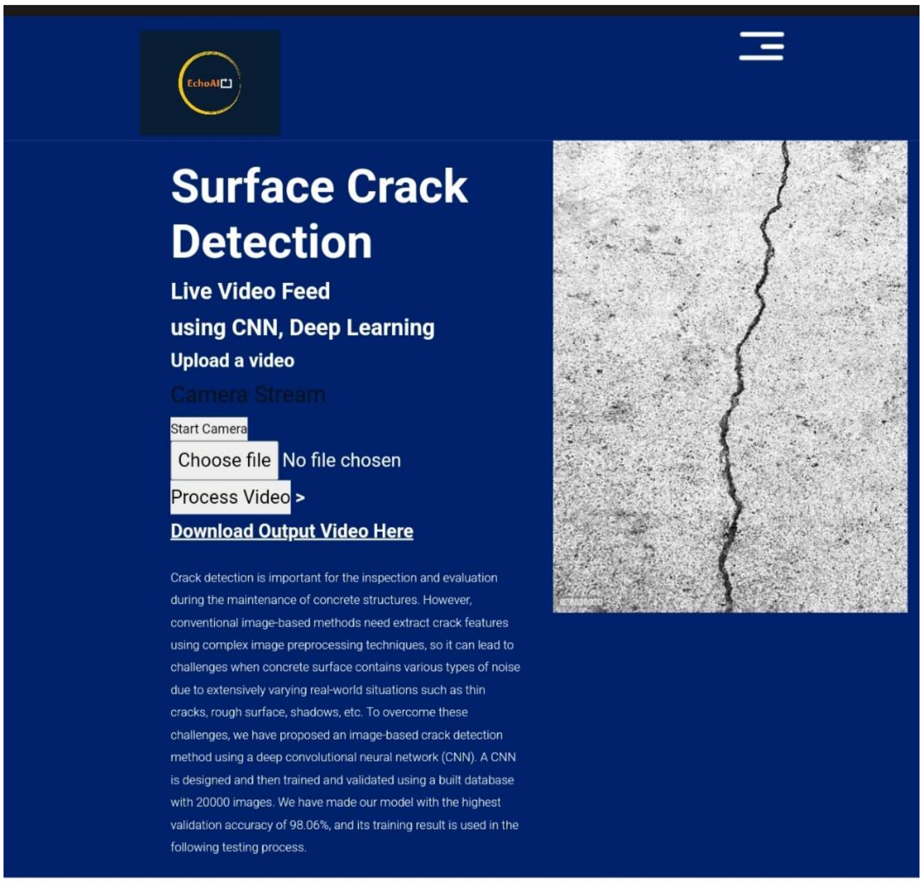
Overall, this setup allows users to upload video files for analysis through your web application. The uploaded files are processed frame by frame, and the resulting modified video can be downloaded by users. This feature is useful for users who want to analyze pre-recorded videos for cracks or other purposes.

Real-Time Image Analysis: Captured frames from the camera or uploaded files are processed using our deep learning model for instant insights.

```
import flask
from flask import Flask, render_template, request, redirect,
send_from_directory, send_file
import os
import threading
from flask import Response
from tensorflow import keras
import numpy as np
import plotly.express
import tensorflow as tf
import numpy as np
from PIL import Image
import pandas as pd
import matplotlib.pyplot as plt
import cv2

app = Flask(__name__)
```

```
app.config['SEND_FILE_MAX_AGE_DEFAULT'] = 0

model = keras.models.load_model(r"model.h5")

UPLOAD_FOLDER = 'uploads'
OUTPUT_FOLDER = 'outputs'
P_F='positive_frames'

@app.route('/')
def index():
    return render_template('index.html')
camera = cv2.VideoCapture(0)
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)
if not os.path.exists(OUTPUT_FOLDER):
    os.makedirs(OUTPUT_FOLDER)
if not os.path.exists(P_F):
    os.makedirs(P_F)
def process_frame(frame):
    # Preprocess the frame for input to the model
    preprocessed_frame = cv2.resize(frame, (1200, 1200))
    preprocessed_frame1 = cv2.resize(frame, (120, 120))
    # Apply the model on the preprocessed frame
    np_img1 = np.array(preprocessed_frame1)
    np_img2 = np.expand_dims(np_img1, axis=0)
    prediction = model.predict(np_img2)
    np_img = np.array(preprocessed_frame)
    if prediction > 0.49999:
        cv2.rectangle(np_img, (100, 100), (1000, 1000), (0, 0, 255), 2)
                data = Image.fromarray(np_img.astype(np.uint8))
        return np_img, prediction

def StartCamera():
    while True:
        # Capture frame-by-frame
        ret, frame = camera.read()
        # Send the frame to the Flask app
        ## read the camera frame
        success, frame = camera.read()
        frame, pred = process_frame(frame)
        if not success:
            break
        else:
            ret, buffer = cv2.imencode('.jpg', frame)
            frame = buffer.tobytes()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')




@app.route('/video')
def video():
    return Response(StartCamera(),mimetype='multipart/x-mixed-replace;
boundary=frame')
@app.route('/upload', methods=['POST'])
def upload():
    videos = request.files['video']
    video_path = os.path.join('uploads', videos.filename)
    videos.save(video_path)
    cap = cv2.VideoCapture(video_path)
    from tensorflow import keras
    model = keras.models.load_model(r"model.h5")
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    output_path = 'outputs/output3.mp4'
    out = cv2.VideoWriter(output_path, fourcc, 30, (1200, 1200))
    while True:
        # Capture frame-by-frame
        ret, frame = cap.read()
        if not ret:
            break
        # Preprocess the frame for input to the model
        preprocessed_frame = cv2.resize(frame, (1200, 1200))
        preprocessed_frame1 = cv2.resize(frame, (120, 120))
        # Apply the model on the preprocessed frame
        np_img1 = np.array(preprocessed_frame1)
        np_img2 = np.expand_dims(np_img1, axis=0)
        prediction = model.predict(np_img2)

        np_img = np.array(preprocessed_frame)
        if prediction > 0.49999:
            cv2.rectangle(np_img, (100, 100), (1000, 1000), (0, 0, 255), 2)

        data = Image.fromarray(np_img.astype(np.uint8))

        out.write(np_img)
    # Call your deep learning model to process the video and generate the
output

    # Save the output video
    return redirect('/download')
@app.route('/download')
def download():
    output_path = 'outputs/output3.mp4'
    return send_file(output_path, as_attachment=True)
if __name__ == '__main__':
    app.run( debug=True)
    # Provide the output video for download
```

Website Overview



Surface Crack Detection

Live Video Feed

using CNN, Deep Learning

Upload a video

Camera Stream

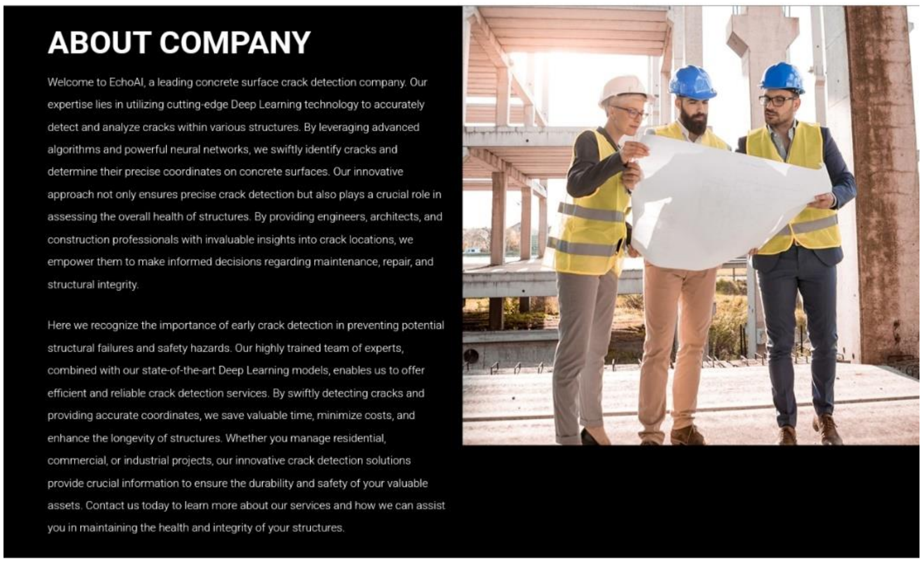
Start Camera

Choose file No file chosen

Process Video >

Download Output Video Here

Crack detection is important for the inspection and evaluation during the maintenance of concrete structures. However, conventional image-based methods need extract crack features using complex image preprocessing techniques, so it can lead to challenges when concrete surface contains various types of noise due to extensively varying real-world situations such as thin cracks, rough surface, shadows, etc. To overcome these challenges, we have proposed an image-based crack detection method using a deep convolutional neural network (CNN). A CNN is designed and then trained and validated using a built database with 20000 images. We have made our model with the highest validation accuracy of 98.06%, and its training result is used in the following testing process.



ABOUT COMPANY

Welcome to EchoAI, a leading concrete surface crack detection company. Our expertise lies in utilizing cutting-edge Deep Learning technology to accurately detect and analyze cracks within various structures. By leveraging advanced algorithms and powerful neural networks, we swiftly identify cracks and determine their precise coordinates on concrete surfaces. Our innovative approach not only ensures precise crack detection but also plays a crucial role in assessing the overall health of structures. By providing engineers, architects, and construction professionals with invaluable insights into crack locations, we empower them to make informed decisions regarding maintenance, repair, and structural integrity.

Here we recognize the importance of early crack detection in preventing potential structural failures and safety hazards. Our highly trained team of experts, combined with our state-of-the-art Deep Learning models, enables us to offer efficient and reliable crack detection services. By swiftly detecting cracks and providing accurate coordinates, we save valuable time, minimize costs, and enhance the longevity of structures. Whether you manage residential, commercial, or industrial projects, our innovative crack detection solutions provide crucial information to ensure the durability and safety of your valuable assets. Contact us today to learn more about our services and how we can assist you in maintaining the health and integrity of your structures.

OUR SERVICES

Structural health monitoring is the process of establishing a damage detection for engineering structures such as buildings, bridges etc. It enables us to know the current condition of the structure.

Only frontend: https://web-application-deploying-deeplearning-github-io.vercel.app/

Future Scope:

Model Optimization: Continue to explore and develop more efficient and accurate deep learning models for crack detection. This could involve advancements in model architectures, such as convolutional neural networks (CNNs) or newer techniques like transformer-based models, to achieve better results.
Real-time Optimization: Enhance the processing speed for real-time analysis. Explore hardware acceleration options like GPU or FPGA to speed up model inference, especially for larger images.
Feedback Mechanisms: Incorporate feedback mechanisms that allow users to provide input on detected cracks or false positives/negatives. This can help improve model performance over time.
Continuous Training: Establish a system for continuous model training and updates to keep the crack detection capabilities up to date with evolving scenarios.
Collaboration with Domain Experts: Collaborate with domain experts, such as structural engineers or quality control specialists, to tailor the system to specific industry needs and challenges.

Discussion:

Real-time Camera Access: The project successfully enables users to access their device's camera in real-time through the web application. This feature finds applications in live monitoring, security, and inspections.
Frame Analysis: The implementation of frame-by-frame analysis, powered by a deep learning model, allows for efficient and accurate detection of cracks in both live camera feeds and uploaded video files.
File Management: The addition of file management features enhances the project's versatility. Users can upload video files for retrospective analysis, making it a valuable tool for inspections and historical data review.
User Feedback: The project provides real-time feedback to users, such as notifications for crack detection, progress indicators, and downloadable results. User interaction is intuitive and informative.

Conclusion:

In conclusion, the project has successfully achieved its primary objectives of real-time camera access, video analysis for crack detection, and file management. It serves as a powerful tool for real-time monitoring, inspections, and retrospective analysis of videos. The utilization of deep learning models for crack detection ensures high accuracy in identifying defects, enhancing safety and quality assurance in various industries. The seamless integration of real-time camera access and file management makes the project user-friendly and adaptable to a wide range of scenarios.