DETECTING BUILDING DEFECTS USING VGG16 & IBM WATSON

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE

IBM ARTIFICIAL INTELLIGENCE EXTERNSHIP PROGRAM

BY

G O NARENDRA

DIVYASHREE S

ARAVINDA B

JUNE 2022

TABLE OF CONTENTS

1.	INTRODUCTION	3
	1. OVERVIEW	
	2. PURPOSE	
2.	LITERATURE SURVEY	4
	1. EXISTING SYSTEM	
	2. PROPOSED SYSTEM	
3.	THEORETICAL ANALYSIS	5
	1. BLOCK DIAGRAM	
	2. HARDWARE/SOFTWARE DESIGN	
4.	EXPERIMENTAL INVESTIGATIONS	6
5	FLOWCHART	7
6.	RESULT	8
7.	ADVANTAGES AND DISDVANTAGES	10
8.	APPLICATIONS	11
9.	CONCLUSION	12
10	FUTURE SCOPE	13
11	BIBLIOGRAPHY	14
	APPENDIX	15
	A. SOURCE CODE	

1. INTRODUCTION

1.1 Overview

Detecting problems on the wall surfaces of high-rise structures, such as cracks and flakes, is a critical responsibility of building maintenance. These defects, if left undetected and untreated, can have a significant impact on a building's structural integrity and aesthetic appeal. Building owners and maintenance agencies must use timely and cost-effective methods of building condition surveys to replace the manual survey's time-and labor-intensive approach.

Clients are increasingly looking for fast and effective means to quickly and frequently survey and communicate the condition of their buildings so that essential repairs and maintenance work can be done in a proactive and timely manner before it becomes too dangerous and expensive. Traditional methods for this type of work commonly comprise engaging building surveyors to undertake a condition assessment, which involves a lengthy site inspection to produce a systematic recording of the physical condition of the building elements, including cost estimates of immediate and projected long-term costs of renewal, repair, and maintenance of the building.

1.2 Purpose

We are using CNN pre-trained model VGG16 to assess the type of building defect on the given parameters in this project to detect building defects such as cracks, flakes, and roof flaws. The project's goal is to create an application that can detect the type of building defect. The model captures the video frame with an integrated webcam, compares it to the pre-trained model, and the type of construction issue is identified and shown in the OpenCV window, prompting an emergency.

2. LITERATURE SURVEY

2.1 Existing problem

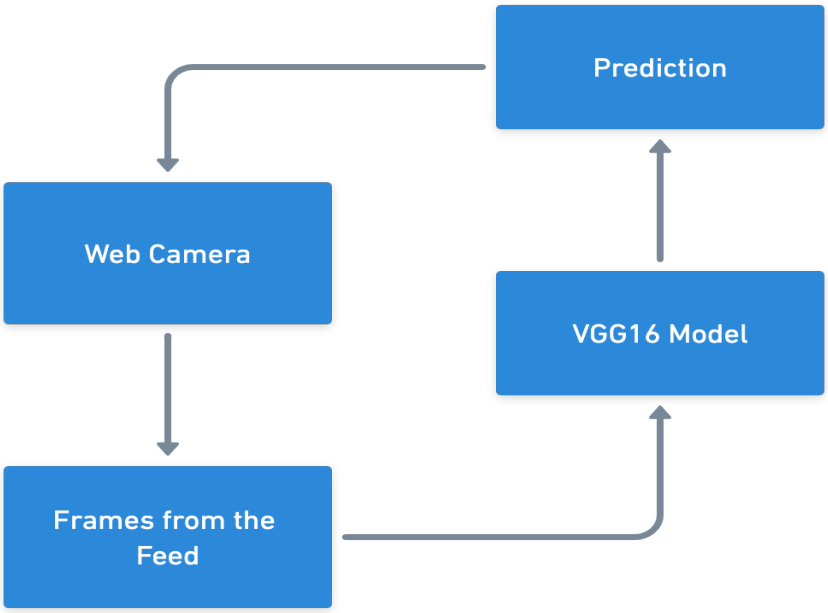
Detecting problems on the wall surfaces of high-rise structures, such as cracks and flakes, is a critical responsibility of building maintenance. These defects, if left unnoticed, can have a huge impact on the buildings. Building owners, associations, and maintenance teams must use time-efficient and cost-effective methodologies for building condition surveys to replace the manual survey's time and labor cost approach. Clients are increasingly searching for quick and effective ways to scan and report the status of their facilities on a regular basis so that necessary repairs and maintenance work may be done before it becomes too unsafe or expensive. Traditional methods for this type of work typically include hiring building surveyors to conduct a condition assessment, which entails a lengthy site inspection to produce a systematic recording of the physical condition of the building elements, as well as cost estimates for immediate and projected long-term costs of renewal, repair, and maintenance of the building.

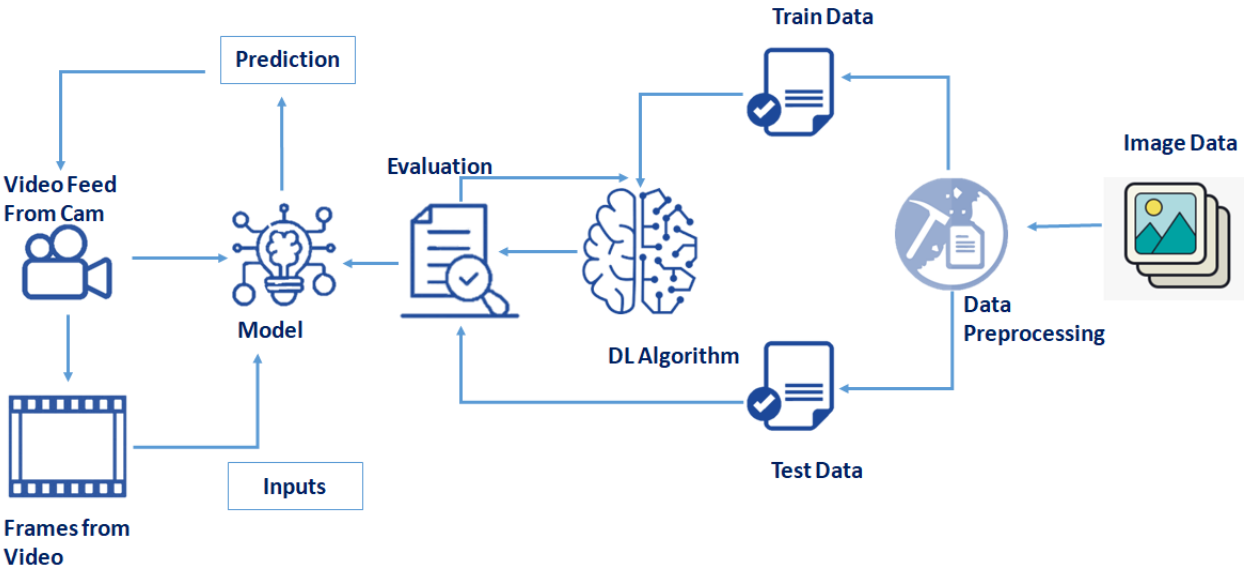
2.2 Proposed solution

We are using CNN pre-trained model **VGG16** to assess the kind of building defect on the supplied parameters in this project, identifying building flaws such as **cracks, flakes, and roof defects**. The project's goal is to create an application that can detect the type of construction fault. The model captures the video frame using an integrated webcam, and the video frame is compared to the pre-trained model, and the kind of construction flaw is detected and shown in the OpenCV window.

3. THEORETICAL ANALYSIS

3.1 Block Diagram





3.2 Hardware/Software Designing

Hardware Requirements:

1. Windows/Linux/Mac OS
2. Memory (RAM): Minimum 1 GB; Recommended 4 GB or above
3. Ethernet connection (LAN) OR a wireless adapter (Wi-Fi)

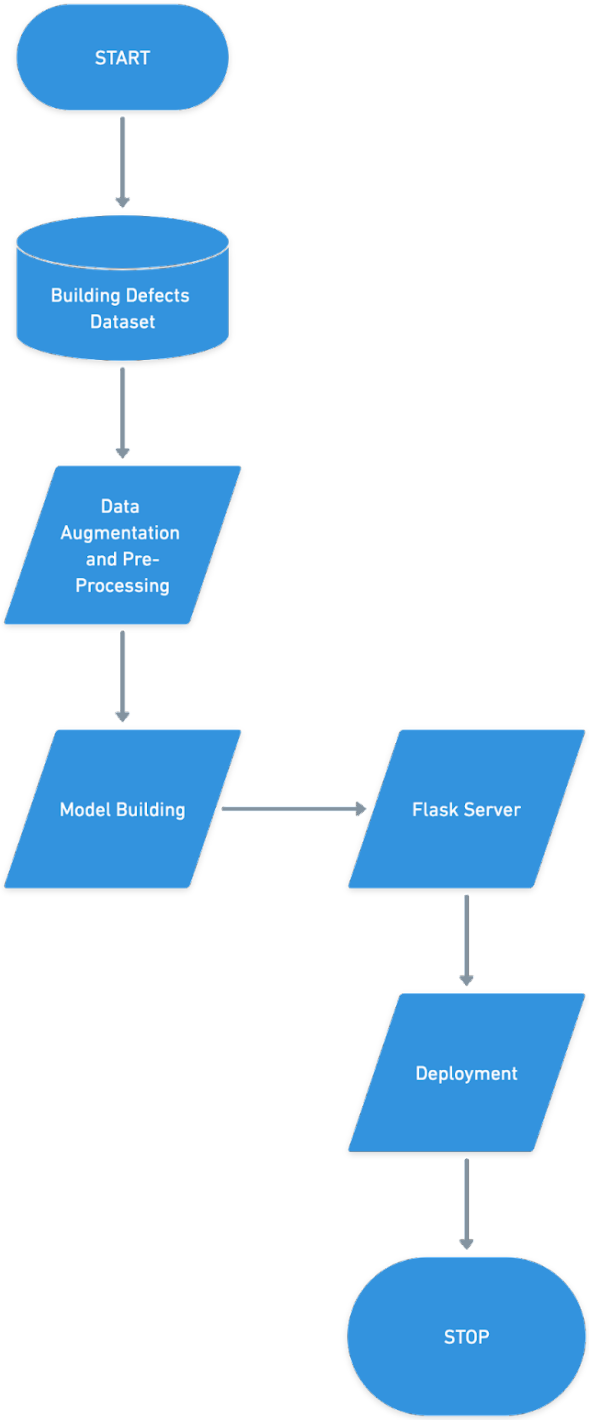
Software Requirements:

1. IBM Watson Studio
2. Flask Framework
3. OpenCV
4. Tensorflow
5. Keras
6. HTML
7. CSS
8. Javascript

4. EXPERIMENTAL INVESTIGATIONS

VGG16 network is improved by using large convolution kernel instead of small convolution kernel and reducing some fully connected layers to reduce the complexity and parameters of the model. Then, the high-dimensional abstract feature data output by the improved VGG16 is input into the convolution neural network (CNN) for training, so as to output the expression types with high accuracy. It is a very good architecture for benchmarking on a particular task. The number 16 in the name VGG refers to the fact that it is 16 layers deep neural network (VGGnet). This means that VGG16 is a pretty extensive network and has a total of around 138 million parameters. Even according to modern standards, it is a huge network.

5. FLOWCHART


START
Building Defects Dataset
Data Augmentation and Pre-Processing
Model Building
Flask Server
Deployment
STOP

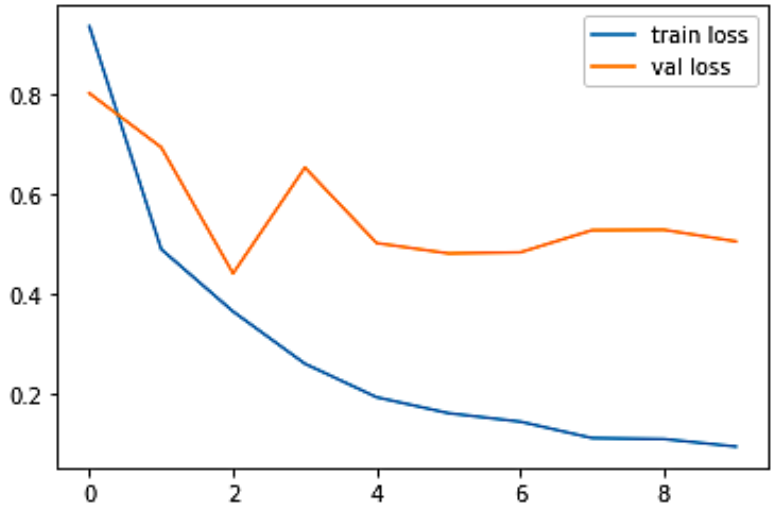
6. RESULT

- The VGG16 model training was done on Jupyter Notebook with 10 epochs for training and the following layers added:

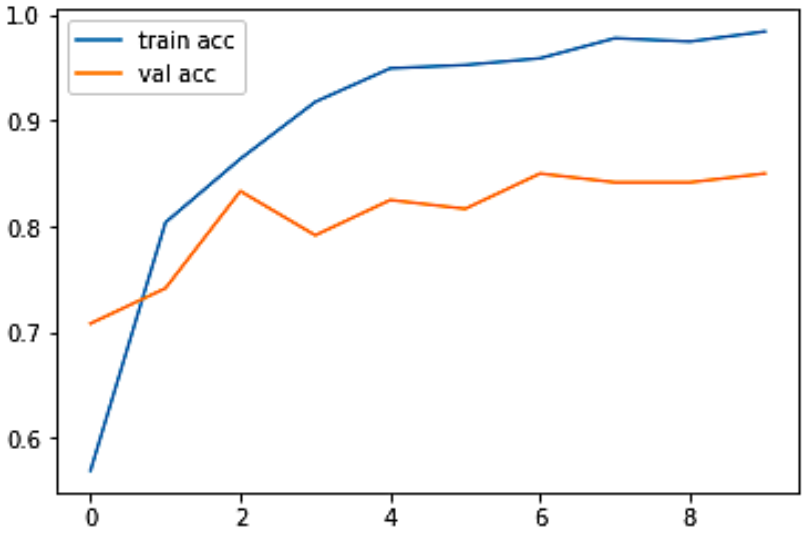
```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0

 dense (Dense)               (None, 3)                 75267

=================================================================
Total params: 14,789,955
Trainable params: 75,267
Non-trainable params: 14,714,688
```

- After a run of 10 epochs, the model achieved around 98.42% accuracy.

1. The loss plot is given below:



The accuracy plot is given below:



A user-interface was developed on a Flask server to capture live video feed from the webcam of the system and feed it into the pre-trained VGG16 model, which would produce the output.

7. ADVANTAGES AND DISADVANTAGES

ADVANTAGES:

- Features are automatically deduced and optimally tuned for the desired outcome. Features are not required to be extracted ahead of time. This avoids time-consuming machine learning techniques.
- Massive parallel computations can be performed using GPUs and are scalable for large volumes of data. Moreover, it delivers better performance results when the amount of data is huge.
- Our application will capture the image in real-time, allowing the user to capture the image quickly and discover a suitable solution.
- Our application is user friendly and available 24/7 and does not have any geographic constraints
- The proposed model has proven to be robust and able to accurately detect and localize building defects

DISADVANTAGES:

- It requires a very large amount of data to perform better than other techniques.
- It is extremely expensive to train due to complex data models. Moreover, deep learning requires expensive GPUs and hundreds of machines. This cost the users.
- It is not easy to comprehend output based on mere learning and requires classifiers to do so. Convolutional neural network-based algorithms perform such tasks.
- Our application only detects construction faults and does not offer solutions for their correction.
- The focus is on the automated detection and localization of key defects arising from dampness in buildings from images. Firstly, multiple types of defects are not considered at once. This means that the images considered by the model belong to only one category. Secondly, only the images with visible defects are considered. Thirdly, consideration of extreme lighting and orientation, e.g., low

lighting and too bright images, are not included in this study. In future works, however, these limitations will be considered to be able to get closer to the concept of a fully automated detection.

8. APPLICATIONS

Clients with various assets are increasingly requiring a detailed understanding of each of their operational assets in order to successfully manage their portfolio and improve company performance. Researchers have been experimenting with the application of a range of soft computing and machine learning-based detection approaches in an attempt to raise the level of automation of asset condition assessment in recent years.
Our current work will be turned into a software application that will detect flaws in real time using visual sensors such as drones. Our application can be utilized in a variety of settings, including homes and industries. In the home, our application can be used to detect cracks, flakes, and roofs that can be identified and repaired in advance, protecting against dangerous events; in factories and industries, our application can be used to detect cracks in chemical containers that can be repaired in advance; and in schools and universities, our application can be used to maintain the building, which helps in lowering the cost of repairs in the future.

9. CONCLUSION

The goal of this project is to create a deep learning-based system for detecting and localizing important construction faults from photographs. This study is part of a larger project on building asset condition assessment. The proposed method divides photos into four categories: mold, stain, and paint deterioration, which includes peeling, blistering, flaking, and crazing; and a fourth group called "Normal" when no flaws are found. We used various augmentation approaches to build a larger dataset in order to achieve acceptable resilience. A random 20% of the training data was picked for the validation set. After 10 epochs, the network had a training set accuracy of 97.83 percent with a 0.0572 loss and a validation set accuracy of 98.86 percent with a 0.042 loss. The overall accuracy of the evaluation test was 87.50 percent, with 90 percent of images containing mold correctly classified, 82 percent of images containing deterioration, 89

percent of images containing stain, and 99 percent of normal images correctly classified. The availability of large-labeled datasets that may be utilized to train a network for this type of problem was the key challenge during this endeavor. To get around this, we employed picture augmentation techniques to create synthetic data for a dataset that was large enough to train our model.

10. FUTURE SCOPE

The difficulties and limitations that we encountered in the paper will be addressed. The current paper had to set a number of constraints, the first of which is that various sorts of flaws are not examined at the same time. This signifies that the model only looked at photos from a single category. Second, only photos with evident flaws are taken into account. Third, extreme lighting and orientation, such as low illumination and very brilliant images, are not taken into account in this study. These restrictions will be considered in future work in order to get closer to the concept of totally automated detection. Our current work will be developed into a software application to do real-time defect detection employing visual sensors, including drones, by fully meeting these problems and limits. Additional models that can detect other faults in construction, such as fractures, structural movements, spalling, and corrosion, will be included in the task. Our long-term objective includes the creation of a huge, open-source database of various building and construction problems that will help global research on asset condition assessment.

11. BIBLIOGRAPHY

1.https://towardsdatascience.com/convolutional-neural-network-a-step-by-step-guide-a8b4c88d6943

2.https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c#:~:text=VGG16%20is%20a%20convolution%20neural,vision%20model%20architecture%20till%20date.

3.https://pyimagesearch.com/2018/07/19/opencv-tutorial-a-guide-to-learn-opencv/

4.https://www.analyticsvidhya.com/blog/2020/04/how-to-deploy-machine-learning-model-flask/

5.https://towardsdatascience.com/camera-app-with-flask-and-opencv-bd147f6c0eec

APPENDIX

A. SOURCE CODE

Image Data Agumentation

```
# Use the Image Data Generator to import the images from the dataset
from tensorflow.keras.preprocessing.image import ImageDataGenerator
#performing data agumentation on train data
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)
#performing data agumentation on test data
test_datagen = ImageDataGenerator(rescale = 1./255)
```

Model Building

```
# re-size all the images to this
IMAGE_SIZE = [224, 224]

train_path = 'data/train_set'
valid_path = 'data/test_set'

# Import the Vgg 16 library as shown below and add preprocessing layer to the front of VGG
# Here we will be using imagenet weights
vgg16 = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)
# don't train existing weights
for layer in vgg16.layers:
    layer.trainable = False
# create a model object
model = Model(inputs=vgg16.input, outputs=prediction)
```

```
# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',metrics=['accuracy'])
Fit the model
# fit the model
# Run the cell. It will take some time to execute
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=10,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set))
# plot the loss
import matplotlib.pyplot as plt
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
plt.savefig('LossVal_loss')
```

FLASK

```
from flask import Flask,render_template,request, Response
#import os
from tensorflow.keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing import image
from camera1 import VideoCamera
app = Flask(__name__,template_folder="templates",static_url_path='/static') # initializing a flask
app
# camera=cv2.VideoCapture(0)
model=load_model('model_building_defects_vgg16.h5')
print("Loaded model from disk")
```

```
def gen(camera):
    while True:
        frame=camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n')
@app.route('/', methods=['GET'])
def index():
    return render_template('home.html')
@app.route('/upload', methods=['GET', 'POST'])
def upload():
        return render_template("upload1.html")
@app.route('/video')
def video():
    return Response(gen(VideoCamera()),mimetype='multipart/x-mixed-replace;
boundary=frame')
    if __name__ == '__main__':
      app.run(host="0.0.0.0",port=5001,debug=False)
```

Open CV

```
while True:
    (grabbed, frame) = vs.read()
    if not grabbed:
        break
    if W is None or H is None:
        (H, W) = frame.shape[:2]
    output = frame.copy()
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    frame = cv2.resize(frame, (224, 224))
    x=image.img_to_array(frame)
    x=np.expand_dims(frame, axis=0)
    img_data=preprocess_input(x)
    result = np.argmax(model.predict(img_data), axis=-1)
    index=['crack','flakes','roof']
    result=str(index[result[0]])
```