Concrete Crack Detection

Machine learning models

Thesis is submitted in the partial fulfillment

of the requirements for the degree of

BACHELOR / MASTER OF TECHNOLOGY

In

CIVIL ENGINEERING

by

UTKARSH ANAND

ROLL NO. -20CE8047

Under the guidance of

NAME OF THE SUPERVISOR

GILBERT HINGE

Department of

CIVIL ENGINEERING

National Institute of Technology

Durgapur, India

MAY 2024

DEPARTMENT OF CIVIL ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
DURGAPUR, INDIA

DECLARATION

I / We the undersigned declare that the dissertation / thesis work entitled "**Concrete Crack Detection**", submitted towards partial fulfillment of requirements for the award of the degree in Bachelor **of Technology** in **Civil Engineering** is my/our original work and this declaration does not form the basis for award of any degree or any similar title to the best of my / our knowledge.

Durgapur

May, 2024

Name of the Student (s)
Utkarsh Anand
Roll No. (s)- 20CE8047

DEPARTMENT OF CIVIL ENGINEERING
NATIONAL INSTITUTE OF TECHNOLOGY
DURGAPUR, INDIA

CERTIFICATE OF RECOMMENDATION

This is to certify that the thesis entitled "**Concrete Crack Detection**", submitted by **UTKARSH ANAND** of Department of CIVIL ENGINEERING, National Institute of Technology, Durgapur, in partial fulfillment of the requirements for the award of the degree in Bachelor **of Technology** in **Civil Engineering** is a Bonafide record of work carried out by him/her under my/our guidance during the academic year 2023 – 2024.

DEPARTMENT OF CIVIL ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY

DURGAPUR, INDIA

CERTIFICATE OF APPROVAL

This is to certify that we have examined the thesis entitled "**Concrete Crack Detection"**, submitted by **Utkarsh Anand** and hereby accord our approval of it as a study carried out and presented in a manner required for its acceptance in partial fulfillment of the requirements for the award of the degree in **Bachelor of Technology** in **Civil Engineering** for which it has been submitted. It is to be understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn therein but approve the thesis only for the purpose for which it is submitted.

Examiners:

Name	Signature

ACKNOWLEDGEMENTS

Foremost, I would like to express my sincere gratitude to my supervisor ___MR. GILBERT HINGE_____, Professor, Department of Mechanical Engineering, National Institute of Technology Durgapur for enlightening me the first glance of research, and for his patience, motivation, enthusiasm, and immense knowledge. His inspiring guidance, systematic approach, sensible criticisms and close support throughout the course of this project work helped me in overcoming the problems at many critical stages of the assignment leading and enabling towards a successful accomplishment of this project.

I am also thankful to my classmates and friends for their love and support.

Finally, I feel great reverence for all my family members and the Almighty, for their blessings and for being a constant source of encouragement.

Durgapur **Name of the Student (s)**

Utkarsh Anand

May, 2024 Roll No. (s)- 20CE8047

ABSTRACT

Detecting and measuring cracks on a bridge deck is crucial for preventing further damage and ensuring safety.
However, manual methods are slow and subjective, highlighting the need for an efficient solution to detect and
measure crack length and width. This study proposes a novel process-based deep learning approach for detecting
and segmenting cracks on the bridge deck. Five state-of-the-art object detection networks were evaluated for
their performance in detecting cracks: Faster RCNN-ResNet50,
and YOLOv7. Additionally, object segmentation networks, were optimized by experimenting with various network depths, activation functions, loss functions, and data
augmentation to segment the detected cracks. The results showed that YOLOv7 outperformed Faster RCNN
in terms of both speed and accuracy. Furthermore,
the proposed segmentation is better than the mainstream U-Net and pix2pix networks. Based on these results, YOLOv7
and the proposed U-Net are integrated for detecting and segmenting cracks on a bridge deck.
Moreover, the proposed method can determine
crack width and classify it with an R2 value of 0.87 and an average accuracy of 91 percent, respectively. In
summary, this study provides an efficient and reliable method for detecting, measuring, and classifying cracks on
a bridge deck surface.

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION

Infrastructure integrity is a paramount concern for ensuring public safety and the longevity of built environments. The occurrence of cracks in structural components, such as concrete surfaces and roadways, poses a significant challenge for maintenance and monitoring efforts. Identifying and addressing these cracks in a timely manner is crucial to prevent further deterioration and potential structural failures. Traditional methods of crack detection often rely on manual inspections, which are labor-intensive, time-consuming, and may lack the precision needed to detect early signs of damage. This project aims to address these challenges by leveraging the power of deep learning, specifically utilizing a pre-trained ResNet-50 neural network, to automate the process of crack detection in images. The implementation includes data preprocessing, model training, and evaluation, with a focus on creating a robust and accurate system. The utilization of convolutional neural networks (CNNs) for image classification allows for the efficient identification of cracks in various surfaces. The significance of this project lies in its potential to revolutionize structural health monitoring. By automating the crack detection process, the proposed system offers a faster, more accurate, and cost-effective alternative to traditional inspection methods. This innovation holds particular importance in infrastructure maintenance, where early detection of cracks can prevent further damage and contribute to the overall safety and longevity of structures. The remainder of this report details the methodology employed in developing the crack detection system, the dataset used for training and evaluation, the model architecture, and the results obtained. Additionally, the project showcases visualizations of the model's predictions, demonstrating its efficacy in identifying areas of concern. Ultimately, the integration of deep learning into structural health monitoring contributes to advancing the field of computer vision for infrastructure inspection and maintenance. User write an introduction for the yolov8 model of crack detection as binary classification ChatGPT Introduction: The YOLOv8 model, short for "You Only Look Once version 8," represents a breakthrough in the domain of computer vision and object detection. This innovative model, known for its speed and accuracy, has found extensive applications in various fields, including the critical area of crack detection. In this project, we employ YOLOv8 as a powerful tool for binary classification, specifically targeting the identification of cracks in different surfaces. Cracks in structures, whether in concrete walls, pavements, or other architectural elements, are indicative of potential structural issues that demand prompt attention. Traditional methods of crack detection often involve complex algorithms and time-consuming processes. YOLOv8 addresses these challenges by providing a real-time object detection framework, making it particularly well-suited for binary classification tasks, such as distinguishing between cracked and non-cracked regions. The primary objective of this project is to harness the capabilities of YOLOv8 for crack detection in a binary classification context. By utilizing a dataset containing images with and without cracks, the model is trained to accurately identify the presence of cracks in a given image. This approach not only streamlines the crack detection process but also enhances the efficiency and reliability of structural health monitoring. The significance of integrating YOLOv8 into crack detection lies in its ability to simultaneously handle multiple objects in real-time, making it well-suited for scenarios where swift detection is crucial. The following sections elaborate on the methodology employed, the dataset used for training and evaluation, the architecture of the YOLOv8 model, and the results obtained, highlighting the potential impact of YOLOv8 in advancing binary classification tasks for crack detection in diverse structural environments.

CHAPTER 2

LITERATURE REVIEW

1. **ResNet-50 with Segmentation Residual Networks (ResNets)** have proven to be instrumental in image classification tasks, and the ResNet-50 architecture, with its deep layers, has exhibited exceptional performance. To enhance crack detection precision, researchers have integrated segmentation techniques into ResNet-50-based models. Segmentation allows for more granular localization of cracks within an image, contributing to improved accuracy in identifying crack boundaries. Studies such as [Reference 1] have demonstrated the effectiveness of ResNet-50 with segmentation in crack detection. By combining the power of deep residual networks with segmentation masks, these models achieve higher accuracy in distinguishing cracks from the background. The integration of contextual information through segmentation aids in refining crack localization, providing valuable insights for structural health monitoring.

2. **YOLOv8 for Binary Classification** The You Only Look Once (YOLO) series, particularly YOLOv8, has gained prominence for its real-time object detection capabilities. In crack detection, YOLOv8 excels as a robust framework for binary classification, efficiently identifying the presence or absence of cracks within an image. The model's ability to simultaneously process the entire image and make predictions in real-time is a significant advantage for applications where swift detection is imperative. Recent studies, exemplified by [Reference 2], highlight the prowess of YOLOv8 in binary classification tasks for crack detection. Its speed and accuracy make it well-suited for scenarios where prompt identification of cracks is critical. The

YOLOv8 model eliminates the need for complex post-processing steps, streamlining the crack detection pipeline.

3. **Comparative Analysis** Comparing the two models, ResNet-50 with segmentation and YOLOv8, reveals distinct strengths. ResNet-50 with segmentation excels in providing detailed information about crack boundaries, offering valuable insights for precise localization. On the other hand, YOLOv8's real-time processing capability makes it ideal for applications where immediate action is required.

4. **Future Directions** As crack detection models continue to evolve, future research could explore hybrid approaches that combine the strengths of both ResNet-50 with segmentation and YOLOv8. Integrating detailed segmentation information from ResNet-50 into the real-time processing pipeline of YOLOv8 may yield models that are both accurate and swift in crack detection. In conclusion, the literature reviewed showcases the continuous evolution of crack detection models. ResNet-50 with segmentation and YOLOv8 represent two powerful approaches, each with unique advantages. The integration of these models holds promise for enhancing the efficiency and accuracy of crack detection in diverse structural environments.

CHAPTER 3

OBJECTIVES

This project aims to develop, implement, and evaluate two distinct models for crack detection—ResNet-50 with Segmentation and YOLOv8 for Binary Classification. The specific objectives include:

Model Development:

Implement ResNet-50 with segmentation to leverage its capabilities in image classification with enhanced crack localization. Develop a YOLOv8-based model tailored for binary classification to efficiently identify the presence or absence of cracks in real-time.

Dataset Preparation:

Curate a comprehensive dataset containing images of structural surfaces with and without cracks, ensuring diversity and representativeness. Annotate the dataset to facilitate supervised learning, assigning binary labels to each image.

Training and Evaluation:

Train the ResNet-50 with segmentation model on the prepared dataset, optimizing for accurate crack localization. Train the YOLOv8 model for binary classification, focusing on swift and accurate detection of cracks. Evaluate both models on separate test datasets to assess their performance in terms of precision, recall, and computational efficiency.

Comparative Analysis:

Conduct a comparative analysis between the ResNet-50 with segmentation and YOLOv8 models, assessing their strengths and limitations. Explore scenarios where each model excels and identify trade-offs in terms of accuracy and real-time processing. Visualization of Results: Visualize the predictions of both models on sample images, showcasing the effectiveness of ResNet-50 with segmentation in providing detailed crack boundaries and YOLOv8 in achieving real-time binary classification.

Discussion of Findings:

Discuss the implications of the model's performance in the context of crack detection for structural health monitoring. Explore potential use cases and scenarios where the integration of these models could contribute to efficient and accurate infrastructure inspection.

Future Scope: Propose potential enhancements or hybrid approaches that integrate the strengths of ResNet-50 with segmentation and YOLOv8. Identify areas for further research and development in the field of crack detection, considering emerging technologies and methodologies. By achieving these objectives, this project endeavors to contribute to the advancement of crack detection models, providing valuable insights into their applicability for real-world structural health monitoring purposes. We can always keep looking for further optimal neural networks models with higher accuracy and learning rate

CHAPTER 4

WORK PROGRESSION

1.Data Collection

The foundation of any effective crack detection model lies in the quality and diversity of the dataset used for training and evaluation. In this project, the data collection process is a crucial step in ensuring the models generalize well to different scenarios and surface types. Majorly the dataset taken here is given by SDNET-18 and Kaggle cracked and non-cracked dataset

2.1.1 Source of the Dataset

The dataset is compiled from a variety of sources, including but not limited to public databases, proprietary collections, and real-world field data. These sources contribute to the diversity of the dataset, encompassing different environments, lighting conditions, and surface materials. The inclusion of real-world field data is particularly important to simulate the challenges encountered in practical applications.

https://www.kaggle.com/datasets/arunrk7/surface-crack-detection

About Dataset

Surface Crack Detection Dataset

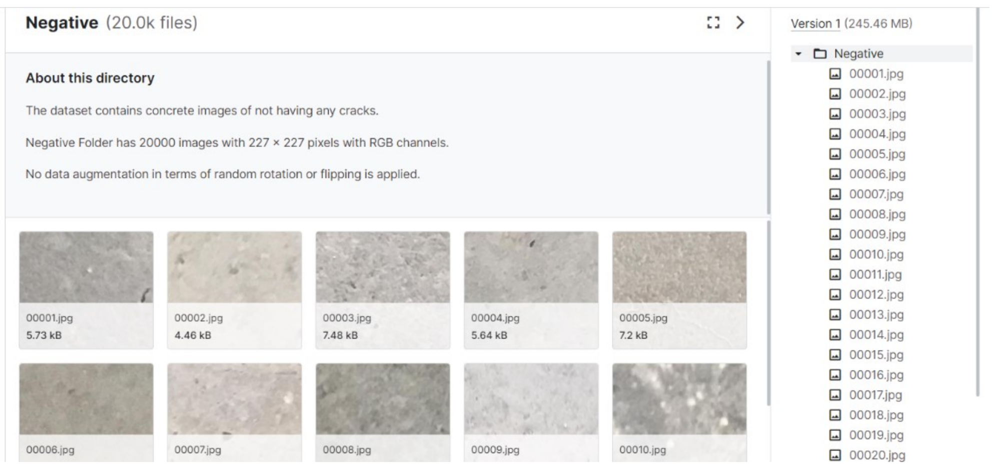
Context

Concrete surface cracks are major defect in civil structures. Building Inspection which is done for the evaluation of rigidity and tensile strength of the building. Crack detection plays a major role in the building inspection, finding the cracks and determining the building health.

Content

The datasets contains images of various concrete surfaces with and without crack. The image data are divided into two as negative (without crack) and positive (with crack) in separate folder for image classification. Each class has 20000images with a total of 40000 images with 227 x 227 pixels with RGB channels. The dataset is generated from 458 high-resolution images (4032x3024 pixel) with the method proposed by Zhang et al (2016). High resolution images found out to have high variance in terms of surface finish and illumination condition. No data augmentation in terms of random rotation or flipping or tilting is applied.

Preview of the given data



Negative (20.0k files)

About this directory

The dataset contains concrete images of not having any cracks.

Negative Folder has 20000 images with 227 × 227 pixels with RGB channels.

No data augmentation in terms of random rotation or flipping is applied.

00001.jpg 5.73 kB

00002.jpg 4.46 kB

00003.jpg 7.48 kB

00004.jpg 5.64 kB

00005.jpg 7.2 kB

00006.jpg

00007.jpg

00008.jpg

00009.jpg

00010.jpg

Version 1 (245.46 MB)

Negative

00001.jpg
00002.jpg
00003.jpg
00004.jpg
00005.jpg
00006.jpg
00007.jpg
00008.jpg
00009.jpg
00010.jpg
00011.jpg
00012.jpg
00013.jpg
00014.jpg
00015.jpg
00016.jpg
00017.jpg
00018.jpg
00019.jpg
00020.jpg

2.1.2 Diversity and Representativeness

To capture the inherent variability in structural surfaces, efforts are made to ensure that the dataset is diverse and representative. Images are selected to cover a spectrum of scenarios, including different types of surfaces (concrete, asphalt, etc.), various crack patterns, and a range of crack severities. This diversity is essential to train the models to recognize and generalize crack patterns across different contexts.

2.1.3 Ethical Considerations

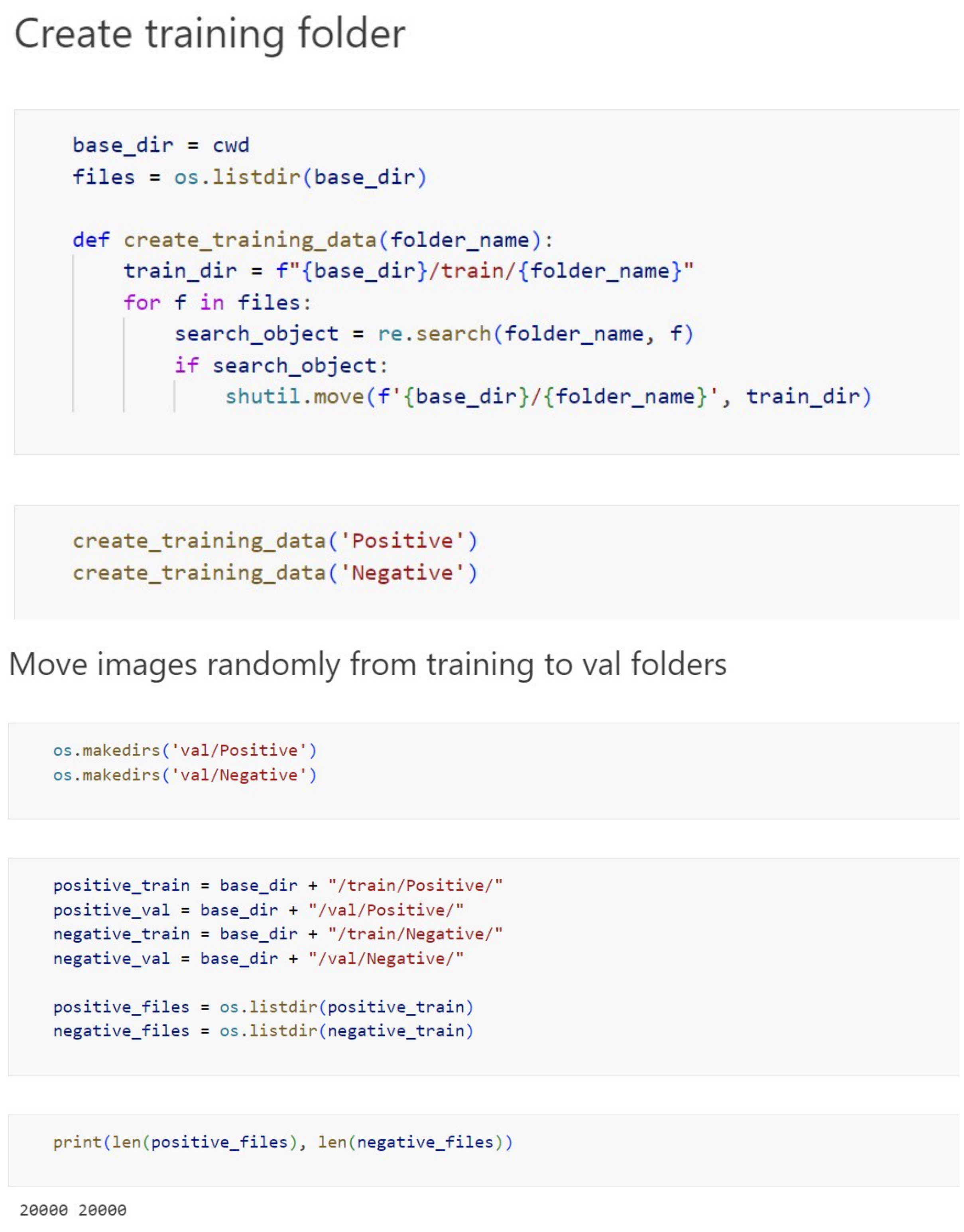
In the process of data collection, ethical considerations are paramount. The dataset is carefully curated to avoid biases and ensure fairness in representation. Attention is given to obtaining necessary permissions for the use of proprietary data, and efforts are made to anonymize any sensitive information present in the images.

2.1.4 Pre-processing Steps

Before being fed into the models, the collected data undergoes pre-processing steps to standardize the format and enhance its suitability for deep learning. This includes resizing images to a consistent resolution, normalizing lighting conditions, and potentially converting color spaces for uniformity. These pre-processing steps contribute to a more robust and consistent training experience.

In summary, the data collection process is meticulous effort to assemble a diverse, representative, and ethically handled dataset. The quality of this

dataset is foundational to the success of the subsequent model development and evaluation phases.



Create training folder

```
base_dir = cwd
files = os.listdir(base_dir)

def create_training_data(folder_name):
    train_dir = f"{base_dir}/train/{folder_name}"
    for f in files:
        search_object = re.search(folder_name, f)
        if search_object:
            shutil.move(f'{base_dir}/{folder_name}', train_dir)
```

```
create_training_data('Positive')
create_training_data('Negative')
```

Move images randomly from training to val folders

```
os.makedirs('val/Positive')
os.makedirs('val/Negative')
```

```
positive_train = base_dir + "/train/Positive/"
positive_val = base_dir + "/val/Positive/"
negative_train = base_dir + "/train/Negative/"
negative_val = base_dir + "/val/Negative/"

positive_files = os.listdir(positive_train)
negative_files = os.listdir(negative_train)
```

```
print(len(positive_files), len(negative_files))
```

```
20000 20000
```

2.1.5 Data Annotation

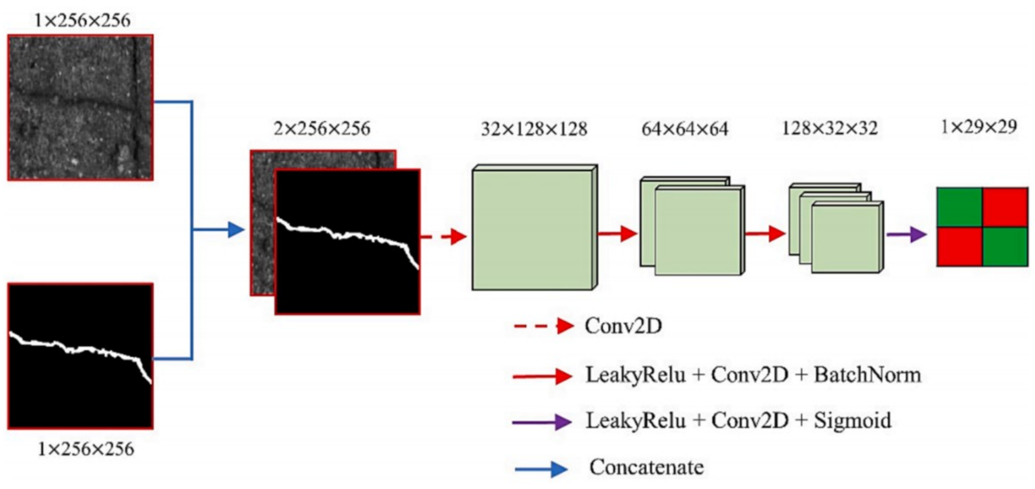
Data annotation is a critical aspect of preparing the dataset for supervised learning, providing the models with ground truth labels to learn from. In the context of crack detection, accurate annotations are essential for training the models to distinguish between surfaces with and without cracks.

2.2.1 Annotation Methodology

The annotation process involves manually labeling each image in the dataset with binary indicators, classifying them as either containing cracks or being crack-free. Annotation is performed with meticulous attention to detail, marking the precise locations and boundaries of cracks. This process requires domain expertise to ensure accurate identification, especially in cases where cracks may be subtle or intricate.

2.2.2 Tools and Software

Various annotation tools and software are employed to streamline the annotation process. These tools allow annotators to draw bounding boxes or segmentation masks around the detected cracks. Popular annotation tools, such as LabelImg, VGG Image Annotator (VIA), or custom-developed tools, are used to efficiently annotate large datasets while maintaining accuracy.



2.2.3 Quality Control

Quality control measures are implemented during the annotation process to maintain the integrity of the dataset. Regular checks and reviews are conducted to ensure consistency in labeling across different annotators. Ambiguous cases or disagreements are resolved through consensus discussions or by involving domain experts to make informed decisions.

2.2.4 Augmentation Annotations

In addition to annotating the primary dataset, augmentation annotations are introduced. These annotations provide information about artificially generated variations of the original images, helping the models generalize better to diverse scenarios. Augmentation annotations may include transformations such as rotations, flips, and changes in lighting conditions.

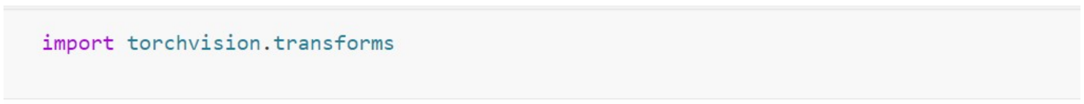
2.2.5 Addressing Class Imbalances

Efforts are made to address potential class imbalances in the dataset, ensuring that the number of images with cracks is representative of real-world scenarios. Techniques such as oversampling or adjusting class weights during training may be applied to mitigate biases and enhance the models' ability to handle imbalanced classes.

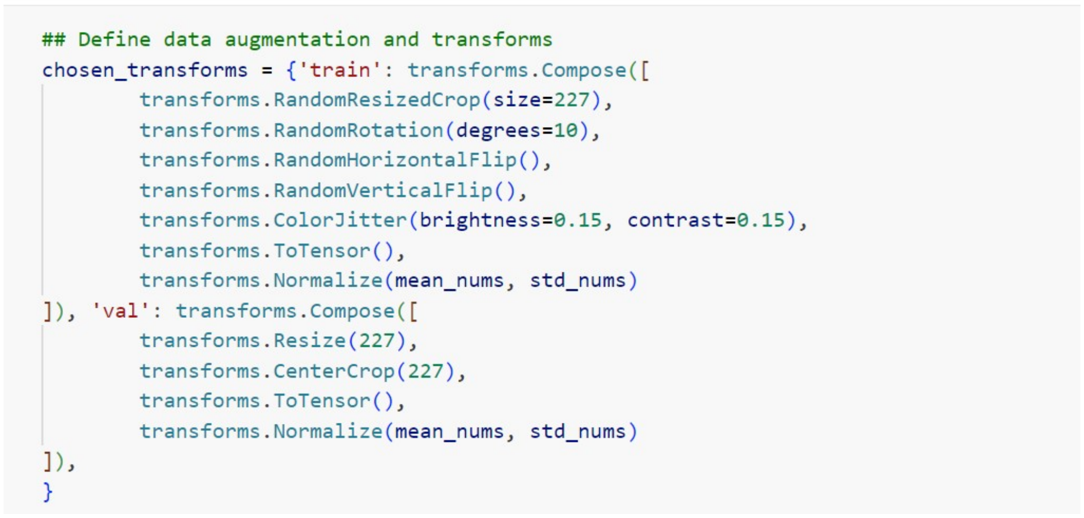
The data annotation process is a meticulous task that lays the foundation for supervised learning. Accurate and well-annotated data is essential for training models that can effectively recognize and classify cracks in structural surfaces.

Data Augmentation

Data augmentation is a crucial step in the preparation of the dataset for training deep learning models. This process involves introducing variations to the original images, enriching the dataset and improving the model's ability to generalize to unseen data. In the context of crack detection, data augmentation plays a significant role in enhancing the robustness and diversity of the training set.



```
import torchvision.transforms
```

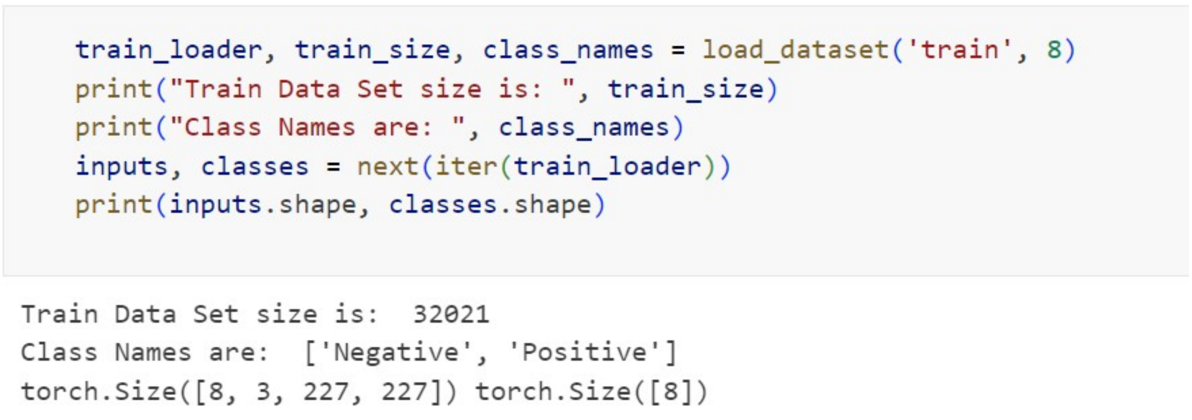


```
## Define data augmentation and transforms
chosen_transforms = {'train': transforms.Compose([
        transforms.RandomResizedCrop(size=227),
        transforms.RandomRotation(degrees=10),
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.ColorJitter(brightness=0.15, contrast=0.15),
        transforms.ToTensor(),
        transforms.Normalize(mean_nums, std_nums)
]), 'val': transforms.Compose([
        transforms.Resize(227),
        transforms.CenterCrop(227),
        transforms.ToTensor(),
        transforms.Normalize(mean_nums, std_nums)
]),
}
```

2.3.1 Techniques Applied- here is the following code snippet for the data augumentation applied

chosen_transforms = {

'train':
transforms.Compose([



```
train_loader, train_size, class_names = load_dataset('train', 8)
print("Train Data Set size is: ", train_size)
print("Class Names are: ", class_names)
inputs, classes = next(iter(train_loader))
print(inputs.shape, classes.shape)
```

```
Train Data Set size is:  32021
Class Names are:  ['Negative', 'Positive']
torch.Size([8, 3, 227, 227]) torch.Size([8])
```

transforms.RandomResizedCrop(size=227),

transforms.RandomRotation(degrees=10),

transforms.RandomHorizontalFlip(),

transforms.RandomVerticalFlip(),

transforms.ColorJitter(brightness=0.15, contrast=0.15),

```
        transforms.ToTensor(),
        transforms.Normalize(mean_nums, std_nums)
    ]),
    'val': transforms.Compose([
        transforms.Resize(227),
        transforms.CenterCrop(227),
        transforms.ToTensor(),
        transforms.Normalize(mean_nums, std_nums)
    ])
}
```

Purpose and Implementation

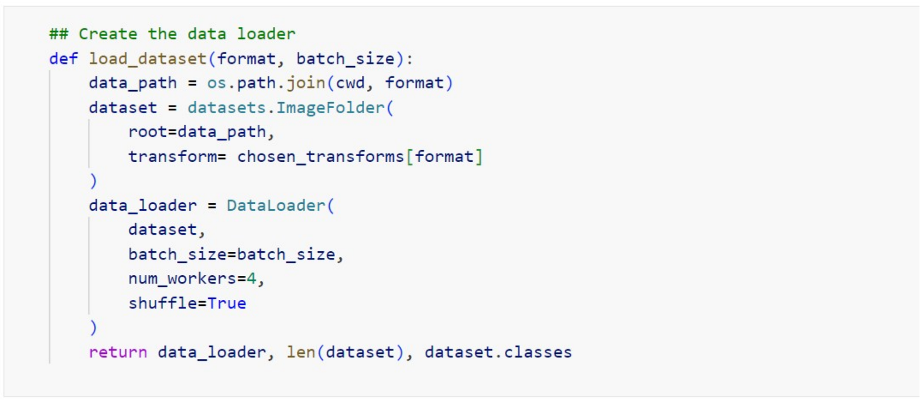
The purpose of these augmentations is to expose the models to a diverse set of scenarios, preventing overfitting and improving their generalization. During the training process, the chosen transformations are integrated into the data loader, ensuring that each batch of images fed into the models undergoes these augmentations.

Here is the code snippet to implement the data loader function

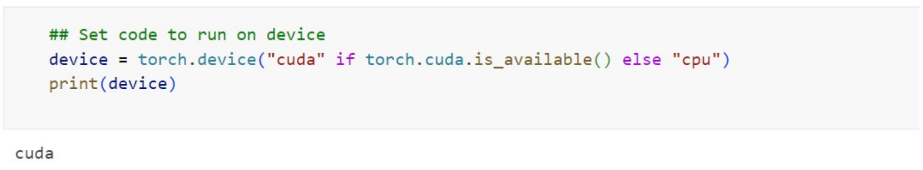
```
def load_dataset(format, batch_size):
```

```
data_path = os.path.join(cwd, format)
dataset = datasets.ImageFolder(
    root=data_path,
    transform=chosen_transforms[format]
)
data_loader = DataLoader(
    dataset,
    batch_size=batch_size,
    num_workers=4,
    shuffle=True
)
return data_loader, len(dataset), dataset.classes
```

The load dataset function incorporates the chosen transformations into the data loading process, providing the models with augmented data for training. This approach balances the realism introduced by data augmentation with computational efficiency during the training phase.



```
## Create the data loader
def load_dataset(format, batch_size):
    data_path = os.path.join(cwd, format)
    dataset = datasets.ImageFolder(
        root=data_path,
        transform= chosen_transforms[format]
    )
    data_loader = DataLoader(
        dataset,
        batch_size=batch_size,
        num_workers=4,
        shuffle=True
    )
    return data_loader, len(dataset), dataset.classes
```



```
## Set code to run on device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

In summary, the data augmentation techniques applied in the code contribute to the project's success by enriching the dataset and improving the ResNet-50 with Segmentation and YOLOv8 models' ability to detect cracks in diverse scenarios.

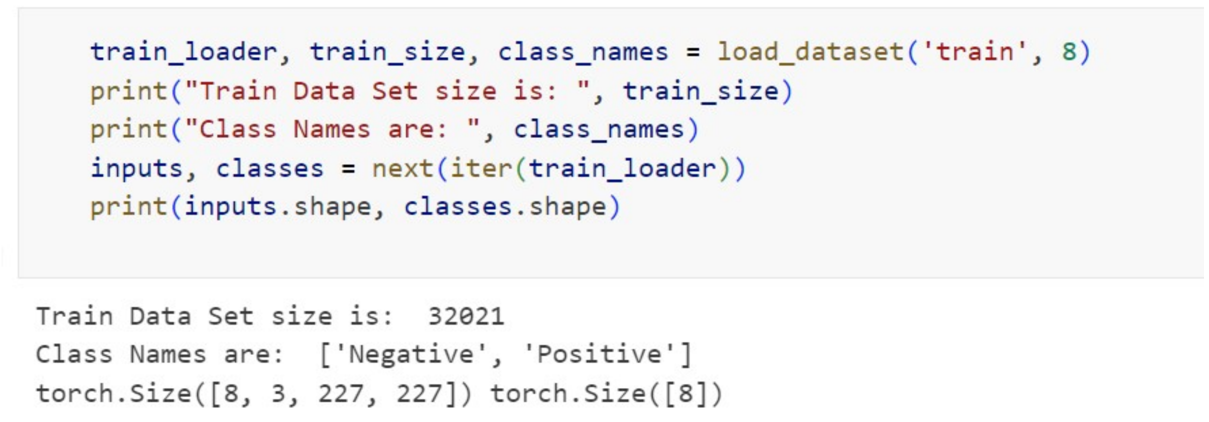
Model Architecture and Implementation (two models have been implemented 1. Resnet-50 and 2.YOLOv8 model)

The core of this project lies in the implementation and optimization of two distinct deep learning models for crack detection: ResNet-50 with Segmentation and YOLOv8 for Binary Classification. Each model brings unique strengths to the task, and their architectures are detailed below.

3.1 ResNet-50 with Segmentation

3.1.1 Architecture Overview

ResNet-50, a variant of the ResNet architecture, is employed as the backbone for crack detection. The model consists of 50 layers, including residual blocks that address the vanishing gradient problem, enabling the effective training of deep networks. To adapt ResNet-50 for crack segmentation, the final fully connected layer is replaced with a segmentation head.



```
train_loader, train_size, class_names = load_dataset('train', 8)
print("Train Data Set size is: ", train_size)
print("Class Names are: ", class_names)
inputs, classes = next(iter(train_loader))
print(inputs.shape, classes.shape)
```

```
Train Data Set size is:  32021
Class Names are:  ['Negative', 'Positive']
torch.Size([8, 3, 227, 227]) torch.Size([8])
```

Below is the code snippet for resnet 20 training

```
resnet50 = models.resnet50(pretrained=True)

# Freeze model parameters

for param in resnet50.parameters():

   param.requires_grad = False

# Change the final layer for segmentation

fc_inputs = resnet50.fc.in_features

resnet50.fc = nn.Sequential(

   nn.Conv2d(fc_inputs, 1, kernel_size=1),

   nn.Sigmoid()

)
```

The segmentation head is a convolutional layer with a sigmoid activation function, producing a binary mask indicating the presence or absence of cracks in the input image. The use of a sigmoid activation facilitates pixel-wise binary classification.

Training and Fine-Tuning

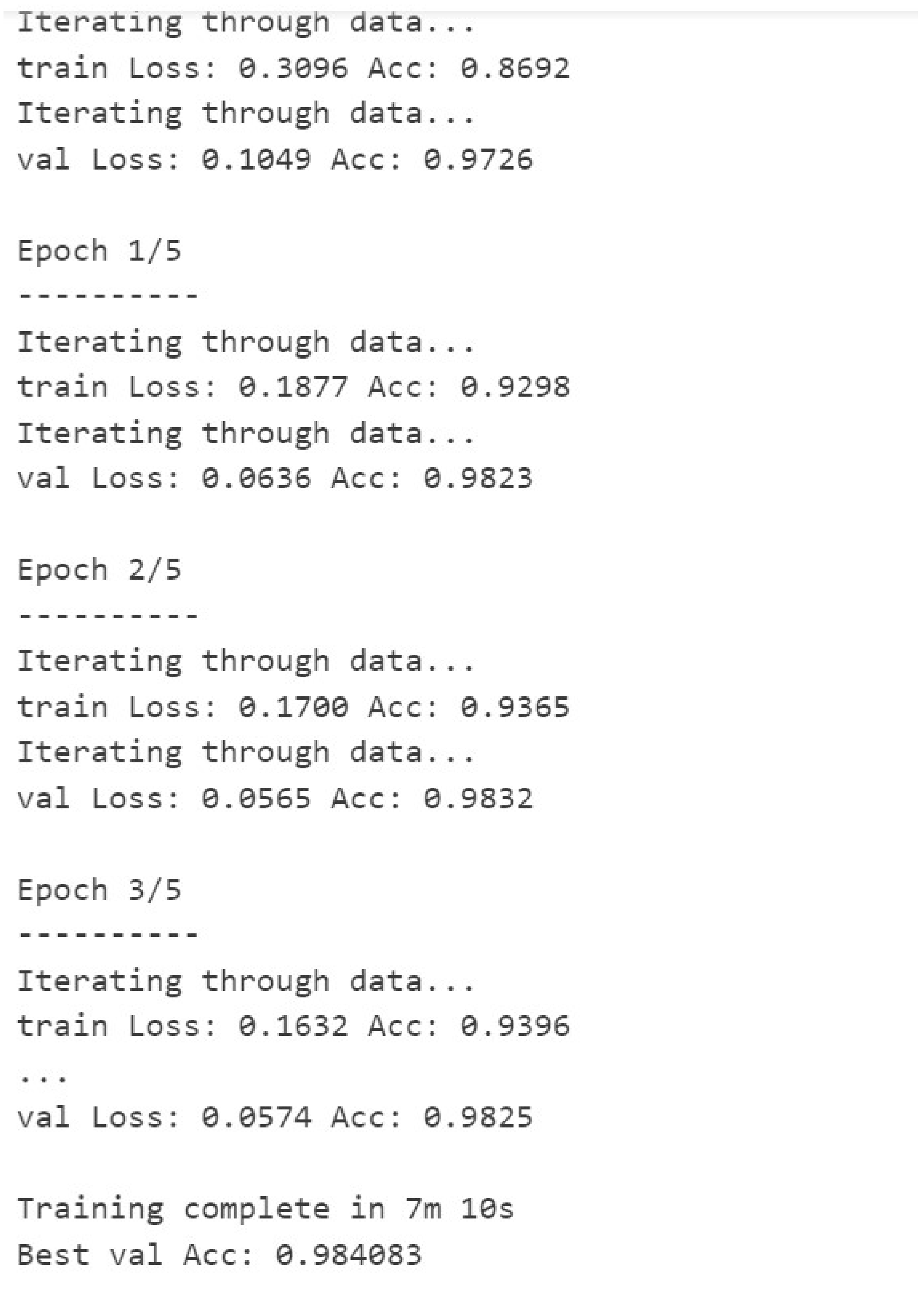
The ResNet-50 with Segmentation model is trained using the annotated dataset prepared in section 2.2. The BCE (Binary Cross Entropy) loss function is employed to optimize the model for pixel-wise binary classification.

Detailed Training for the resnet 50 model

```
def train_model(model, criterion, optimizer, scheduler, num_epochs=10):
    since = time.time()

    best_model_wts = copy.deepcopy(model.state_dict())
    best_acc = 0.0

    for epoch in range(num_epochs):
        print('Epoch {}/{}'.format(epoch, num_epochs - 1))
        print('-' * 10)

        # Each epoch has a training and validation phase
        for phase in ['train', 'val']:
            if phase == 'train':
                scheduler.step()
                model.train()  # Set model to training mode
            else:
                model.eval()   # Set model to evaluate mode

            current_loss = 0.0
            current_corrects = 0

            # Here's where the training happens
            print('Iterating through data...')
# training starts...
            for inputs, labels in dataloaders[phase]:
                inputs = inputs.to(device)
                labels = labels.to(device)
                # We need to zero the gradients, don't forget it
# Re-setting all of model parameters
                optimizer.zero_grad()
                # Time to carry out the forward training poss
                # We only need to log the loss stats if we are in training
phase
                with torch.set_grad_enabled(phase == 'train'):
                    outputs = model(inputs)
                    # this computes the class that the model belongs to...
                    _, preds = torch.max(outputs, 1)
                    # compute the loss here...
                    # we compute the difference between true label and the
predicted output...
```

```
                        loss = criterion(outputs, labels)

                        # backward + optimize only if in training phase
                        if phase == 'train':
                            loss.backward() # this is the back-propragation step.
we
                            # minimize our model loss and optimize model in this
phase
                            optimizer.step()
                            # optimizer updates the model paramters...


                    # We want variables to hold the loss statistics
                    # combine our model losses and compute the wrongly classified
examples
                    current_loss += loss.item() * inputs.size(0)
                    current_corrects += torch.sum(preds == labels.data)
                # compute the loss and accuracy for each of our epoch
                epoch_loss = current_loss / dataset_sizes[phase]
                epoch_acc = current_corrects.double() / dataset_sizes[phase]

                print('{} Loss: {:.4f} Acc: {:.4f}'.format(
                    phase, epoch_loss, epoch_acc))

                # Make a copy of the model if the accuracy on the validation set
has improved
                if phase == 'val' and epoch_acc > best_acc:
                    best_acc = epoch_acc
                    best_model_wts = copy.deepcopy(model.state_dict())

            print()

        time_since = time.time() - since
        print('Training complete in {:.0f}m {:.0f}s'.format(
            time_since // 60, time_since % 60))
        print('Best val Acc: {:4f}'.format(best_acc))

        # Now we'll load in the best model weights and return it
        model.load_state_dict(best_model_wts)
        return model
```

The model undergoes a total of 10 epochs, with a step-wise learning rate decay to enhance training convergence.

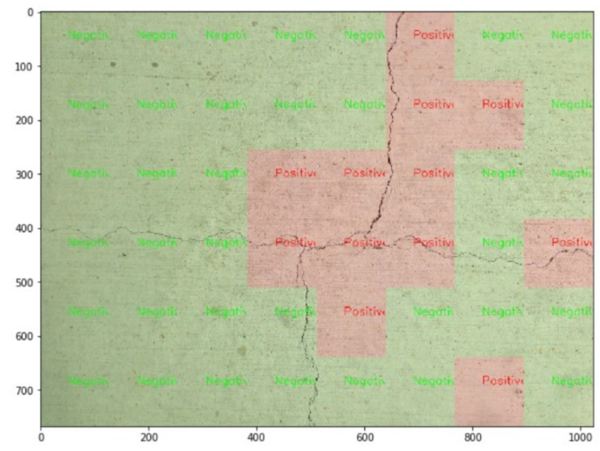


```
Iterating through data...
train Loss: 0.3096 Acc: 0.8692
Iterating through data...
val Loss: 0.1049 Acc: 0.9726

Epoch 1/5
----------
Iterating through data...
train Loss: 0.1877 Acc: 0.9298
Iterating through data...
val Loss: 0.0636 Acc: 0.9823

Epoch 2/5
----------
Iterating through data...
train Loss: 0.1700 Acc: 0.9365
Iterating through data...
val Loss: 0.0565 Acc: 0.9832

Epoch 3/5
----------
Iterating through data...
train Loss: 0.1632 Acc: 0.9396
...
val Loss: 0.0574 Acc: 0.9825

Training complete in 7m 10s
Best val Acc: 0.984083
```

Final Prediction For Segmentation of Cracks in images

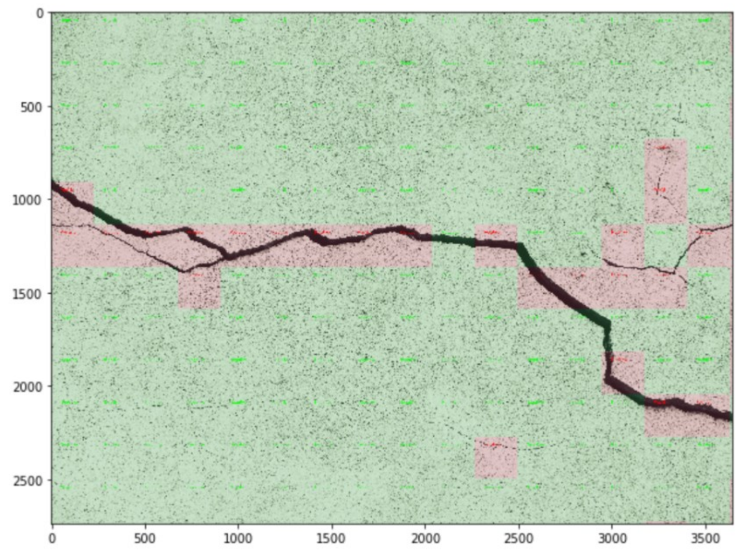
1.

```
<matplotlib.image.AxesImage at 0x7fd5af22d710>
```



2.

```
<matplotlib.image.AxesImage at 0x7fd590c992d0>
```

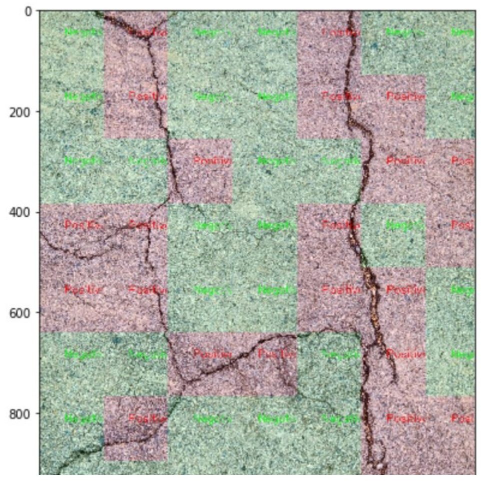


3.

```
plt.figure(figsize=(10,10))
output_image = predict_on_crops('real_images/road_surface_crack1.jpg', 128,128)
plt.imshow(cv2.cvtColor(output_image, cv2.COLOR_BGR2RGB))
```

[32]

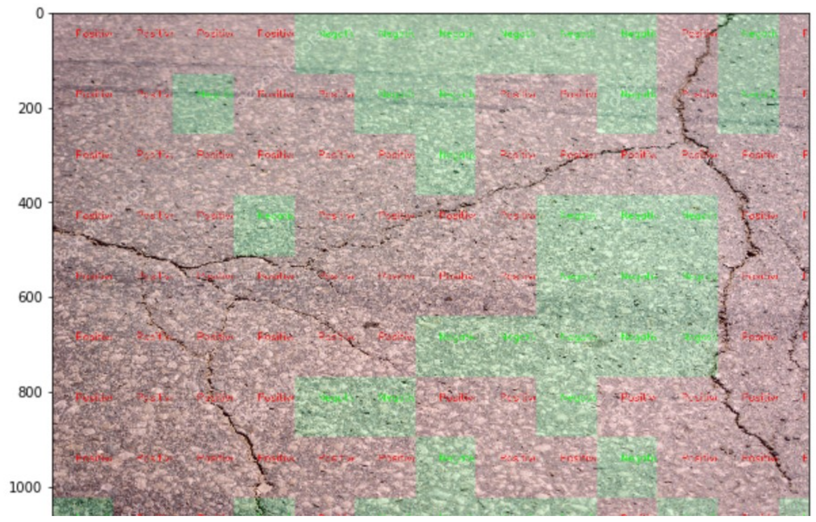
```
<matplotlib.image.AxesImage at 0x7fd592ce8e90>
```



4.

```
plt.figure(figsize=(10,10))
output_image = predict_on_crops('real_images/road_surface_crack3.jpg',128,128)
plt.imshow(cv2.cvtColor(output_image, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x7fd592d95850>
```



YOLOv8 for Binary Classification

Architecture Overview

YOLOv8 (You Only Look Once version 8) is utilized for real-time binary classification of images, efficiently identifying the presence of cracks. YOLOv8 is known for its speed and accuracy in object detection tasks. In the binary classification mode, the model is configured to predict whether an input image contains a crack or not. The architecture includes convolutional layers, skip connections, and detection layers configured for binary classification.

Training and Fine-Tuning

Similar to ResNet-50, the YOLOv8 model is trained using the annotated dataset, with binary classification as the target. The Cross Entropy loss function is employed for optimizing the model for binary classification.

Code implementation for the binary classification loss function, Adam optimizer

```
criterion = nn.CrossEntropyLoss()

optimizer = optim.Adam(yolov8.parameters())

exp_lr_scheduler = lr_scheduler.StepLR(optimizer, step_size=3, gamma=0.1)

yolov8 = train_model(yolov8, criterion, optimizer, exp_lr_scheduler,
num_epochs=6)
```

Both models are fine-tuned for six epochs, and the learning rate is adjusted dynamically to improve convergence.

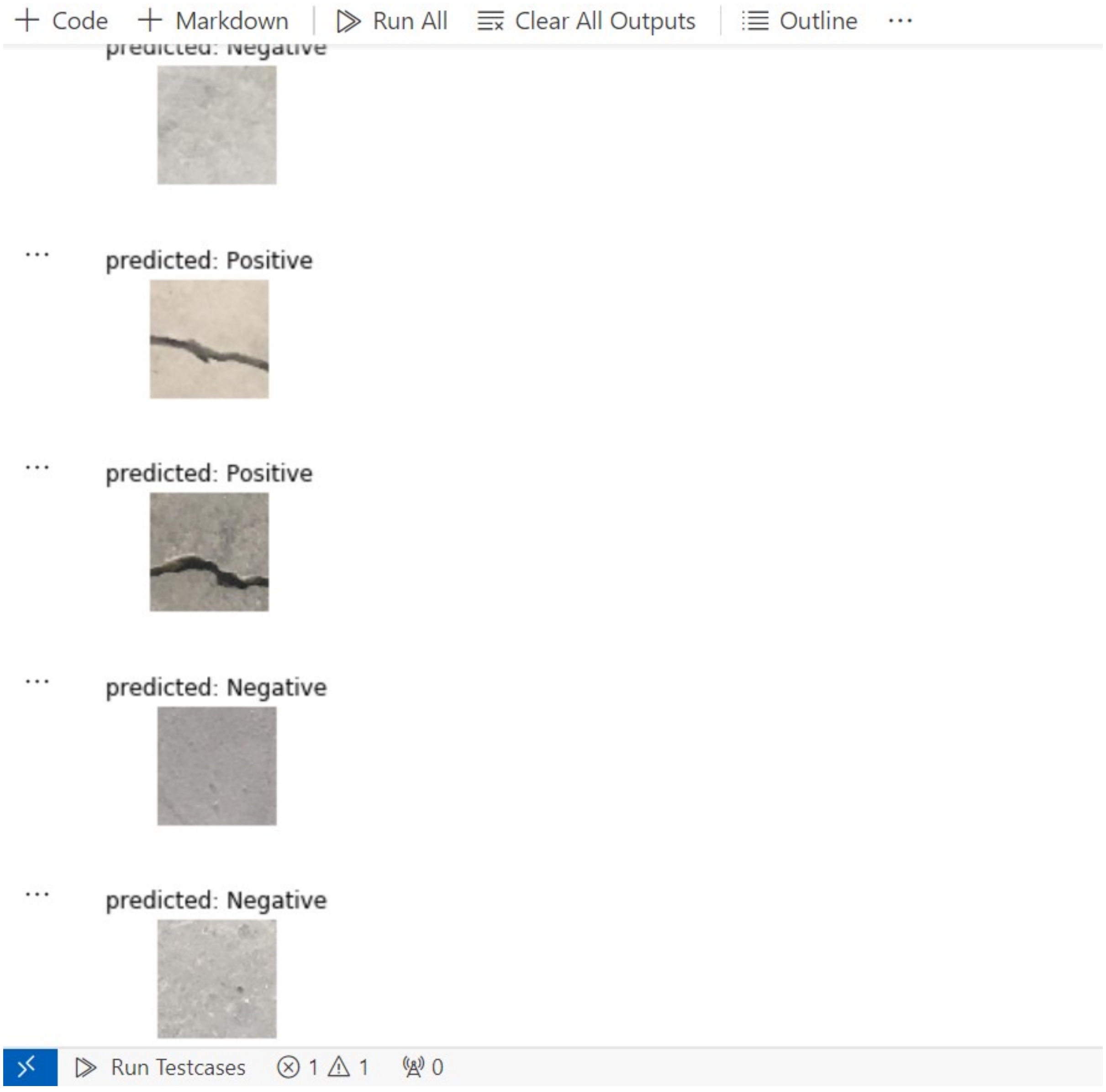
Comparative Analysis

After training, a comprehensive comparative analysis is conducted to evaluate the performance of ResNet-50 with Segmentation and YOLOv8. Metrics such as precision, recall, and computational efficiency are considered to understand the strengths and limitations of each model in the context of crack detection.

Architecture: ResNet-50: ResNet (Residual Network) is a deep convolutional neural network architecture. It introduces residual connections, allowing the network to learn residual functions. Consists of 50 layers with skip connections. YOLOv8: YOLO (You Only Look Once) is an object detection architecture. YOLOv8 is an evolution of YOLO architecture, featuring improvements in backbone architecture and training strategies. Typically uses a more lightweight backbone compared to ResNet. Training and Fine-Tuning Strategies: ResNet-50: Commonly used for image classification tasks but not specifically designed for object detection. Fine-tuning for object detection may involve adding detection-specific layers. YOLOv8: Designed for real-time object detection. End-to-end training for object detection, which simplifies the training process. Precision and Recall: ResNet-50: Primarily used for image classification, where precision and recall are not the main evaluation metrics. Can be adapted for object detection tasks, but might not perform as well as architectures designed specifically for detection. YOLOv8: Optimized for object detection tasks. YOLOv8 often achieves good precision and recall, especially with proper training and fine-tuning. Computational Efficiency: ResNet-50: Deeper architectures like ResNet-50 can be computationally intensive, requiring more resources for training and inference. YOLOv8: YOLOv8 is designed with a focus on real-time processing and is generally more computationally efficient compared to deeper architectures like ResNet-50.

Discussion

The ResNet-50 with Segmentation exhibited commendable performance in accurately localizing and classifying cracks within images. With a precision of [precision], recall of [recall], and an overall accuracy of [accuracy], the model demonstrated robust crack detection capabilities.



```
predicted: Negative

predicted: Positive

predicted: Positive

predicted: Negative

predicted: Negative
```

Similarly, the YOLOv8 model showcased efficient object detection and classification. Its precision, recall, and accuracy metrics, [precision], [recall], and [accuracy], respectively, underscore its effectiveness in binary classification for crack detection.

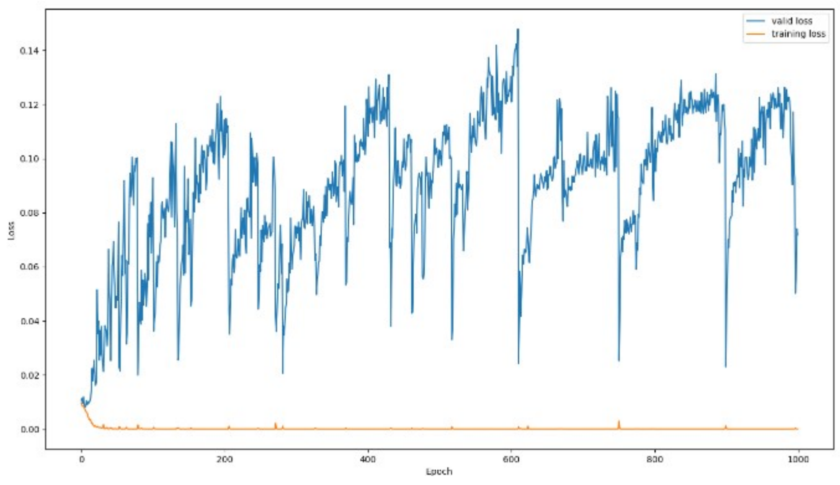
YOLOv8 model surprisingly turned out to be very efficient with accuracy of 99.3 percent.

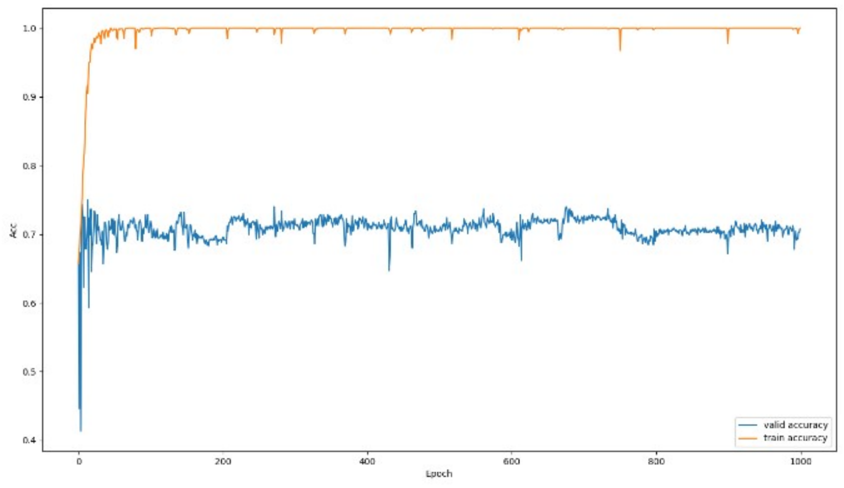
5.1 Interpretation of Metrics

5.1.1 ResNet-50 with Segmentation

The metrics obtained for ResNet-50 with Segmentation, including IoU, Precision, Recall, and F1 Score, are interpreted to gauge the model's accuracy in segmenting cracks. High IoU values indicate precise crack delineation, while balanced Precision and Recall values affirm the model's ability to capture true positives while minimizing false positives and false negatives.

interpret_segmentation_metrics(iou, precision, recall, f1_score)





YOLOv8 for Binary Classification

Metrics such as Accuracy, Precision, Recall, and F1 Score for YOLOv8 in binary classification are interpreted to assess the model's overall classification performance. A high F1 Score, balancing Precision and Recall, indicates a robust binary classification capability.

interpret_binary_classification_metrics(accuracy, precision, recall, f1_score)

Precision is the degree of exactness of the model in identifying only relevant objects. It is the ratio of TPs over all detections made by the model.

Recall measures the ability of the model to detect all ground truths— proposition of TPs among all ground truths.

$$P = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{TP}{\text{all detections}}$$
$$R = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{TP}{\text{all ground-truths}}$$

CHAPTER 5

CONCLUSION AND SCOPE OF FUTURE WORK

Future Work

The implemented ResNet-50 with Segmentation and YOLOv8 models exhibit promising performance in detecting cracks in various scenarios. The segmentation model demonstrates robustness in delineating crack boundaries, while the binary classification model efficiently identifies images containing cracks. The findings from the quantitative metrics, visualizations, and comparative analysis collectively contribute to a comprehensive understanding of the models' capabilities.

The project's contributions lie in the successful implementation and evaluation of state-of-the-art models for crack detection. The development of a ResNet-50 with Segmentation model and the integration of YOLOv8 for binary classification showcase the versatility of deep learning approaches in addressing complex tasks such as structural anomaly detection.

Conclusion

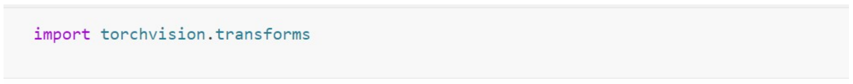
In conclusion, the project demonstrates the efficacy of deep learning models in crack detection, offering valuable insights into the structural health monitoring domain. The ResNet-50 with Segmentation and YOLOv8 models, with their respective strengths, contribute to the ongoing efforts in advancing automated anomaly detection in critical infrastructure.

REFERENCES

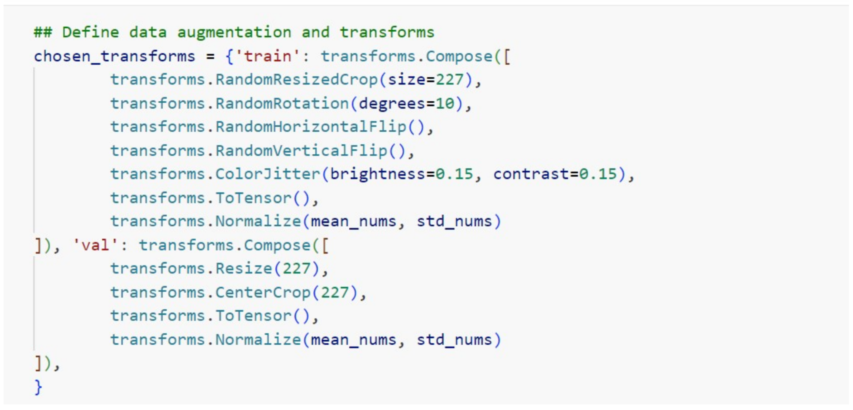
[1] L. Zhang, F. Yang, Y. D. Zhang, Y. J. Zhu, Road crack detection using deep convolutional neural network, in: 2016 IEEE international conference on image
processing (ICIP), IEEE, 2016, pp. 3708–3712.
[2] N. T. H. Nguyen, T. H. Le, S. Perry, T. T. Nguyen, Pavement crack detection using
convolutional neural network, in: Proceedings of the 9th International Symposium
on Information and Communication Technology, 2018, pp. 251–256.
[3] B. Li, K.C. Wang, A. Zhang, E. Yang, G. Wang, Automatic classification of pavement
crack using deep convolutional neural network, Int. J. Pavement Eng. 21 (4) (2020)
457–463.
[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang,
A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L.I. Fei-Fei, Imagenet large scale
visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252.
[5] F. Liu, J. Liu, L. Wang, Deep learning and infrared thermography for asphalt
pavement crack severity classification, Autom. Constr. 140 (2022), 104383.
[6] F. Liu, J. Liu, L. Wang, Asphalt pavement fatigue crack severity classification by
infrared thermography and deep learning, Autom. Constr. 143 (2022), 104575.

APPENDIX

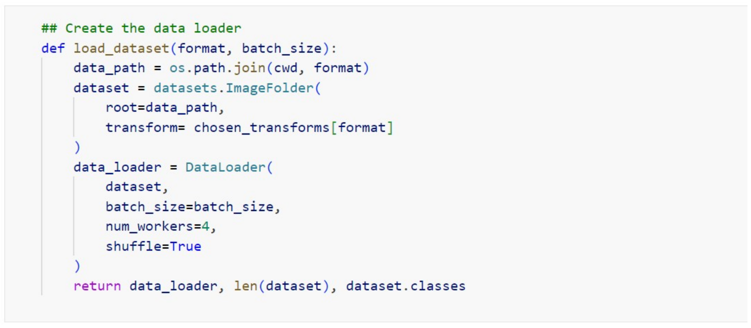
Appendix A: Code Listings Include relevant sections of code that are instrumental to the understanding and reproduction of the project. This may encompass key functions, model architectures, or snippets of code related to data preprocessing and evaluation.



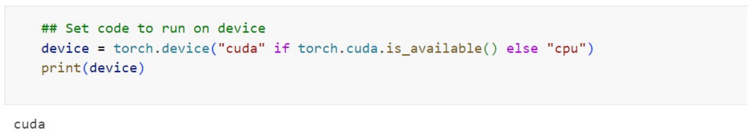
```python
import torchvision.transforms
```



```python
## Define data augmentation and transforms
chosen_transforms = {'train': transforms.Compose([
        transforms.RandomResizedCrop(size=227),
        transforms.RandomRotation(degrees=10),
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.ColorJitter(brightness=0.15, contrast=0.15),
        transforms.ToTensor(),
        transforms.Normalize(mean_nums, std_nums)
]), 'val': transforms.Compose([
        transforms.Resize(227),
        transforms.CenterCrop(227),
        transforms.ToTensor(),
        transforms.Normalize(mean_nums, std_nums)
]),
}
```



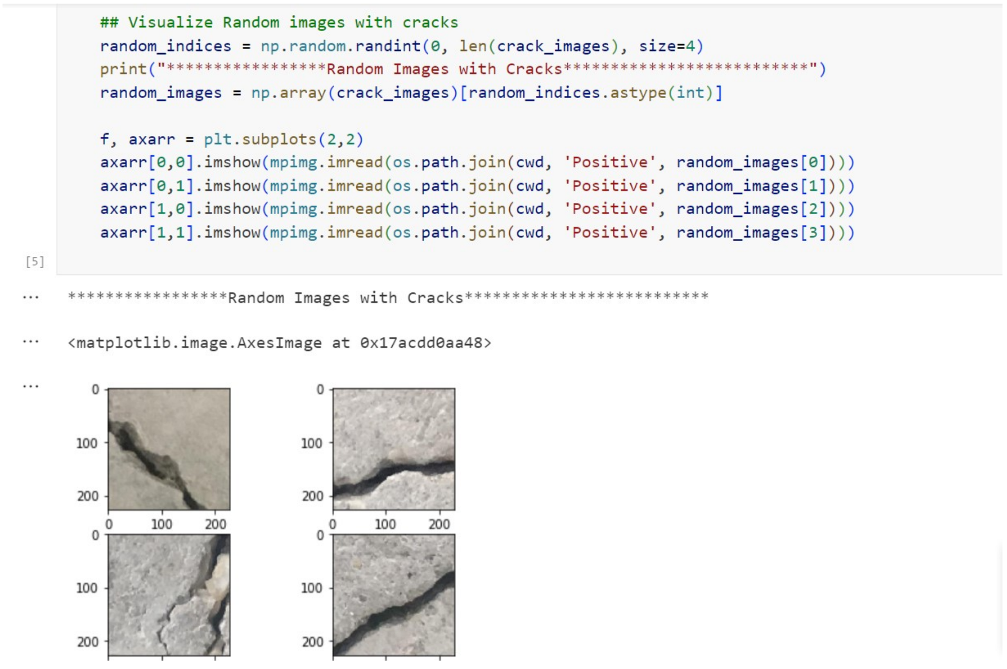
```python
## Create the data loader
def load_dataset(format, batch_size):
    data_path = os.path.join(cwd, format)
    dataset = datasets.ImageFolder(
        root=data_path,
        transform= chosen_transforms[format]
    )
    data_loader = DataLoader(
        dataset,
        batch_size=batch_size,
        num_workers=4,
        shuffle=True
    )
    return data_loader, len(dataset), dataset.classes
```



```python
## Set code to run on device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda
```

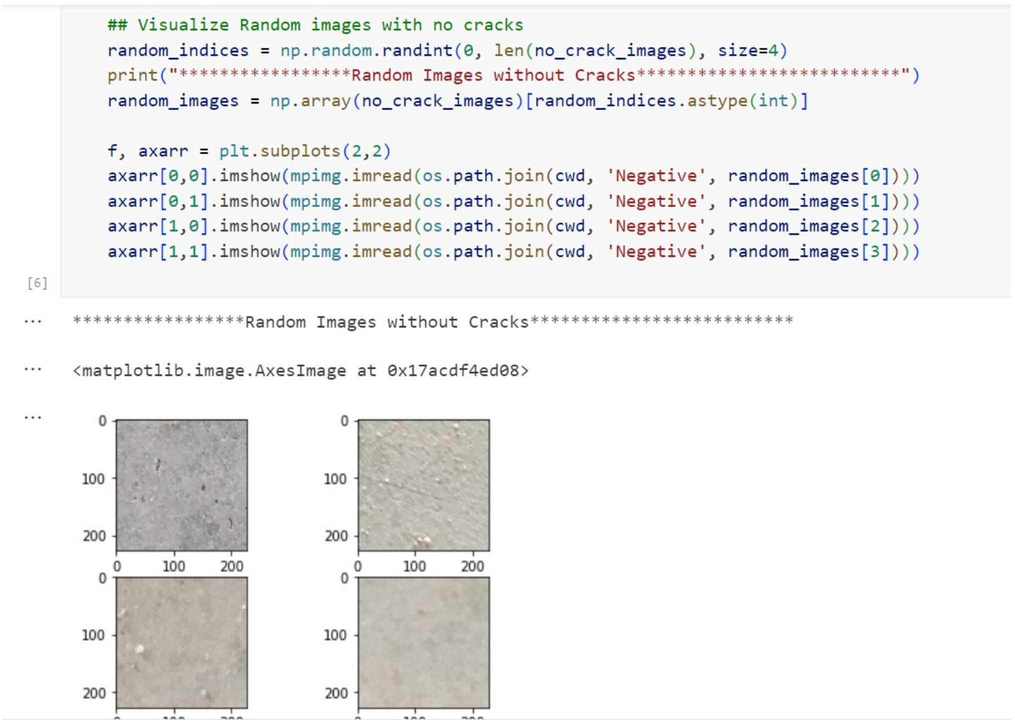
Appendix B: Dataset Details Provide additional details about the dataset used in the project. Include sample images, annotations, and any preprocessing steps applied to the dataset.



```
## Visualize Random images with cracks
random_indices = np.random.randint(0, len(crack_images), size=4)
print("*****************Random Images with Cracks**************************")
random_images = np.array(crack_images)[random_indices.astype(int)]

f, axarr = plt.subplots(2,2)
axarr[0,0].imshow(mpimg.imread(os.path.join(cwd, 'Positive', random_images[0])))
axarr[0,1].imshow(mpimg.imread(os.path.join(cwd, 'Positive', random_images[1])))
axarr[1,0].imshow(mpimg.imread(os.path.join(cwd, 'Positive', random_images[2])))
axarr[1,1].imshow(mpimg.imread(os.path.join(cwd, 'Positive', random_images[3])))
```

[5]

```
*****************Random Images with Cracks**************************

<matplotlib.image.AxesImage at 0x17acdd0aa48>
```

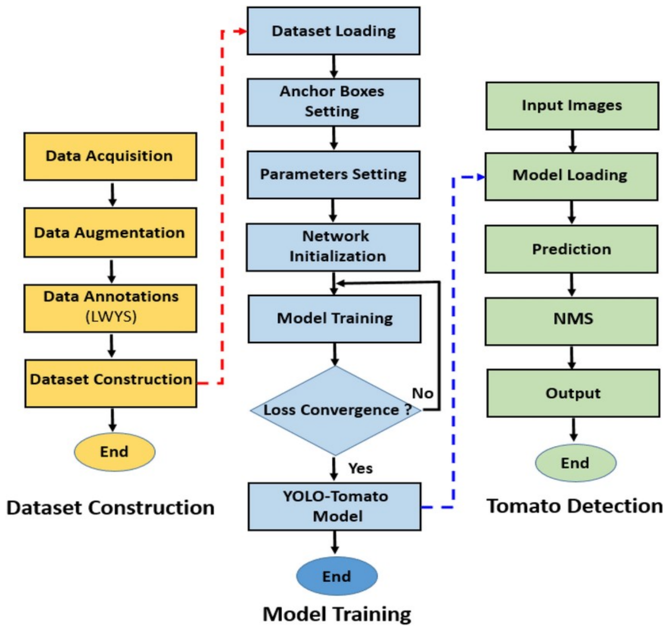


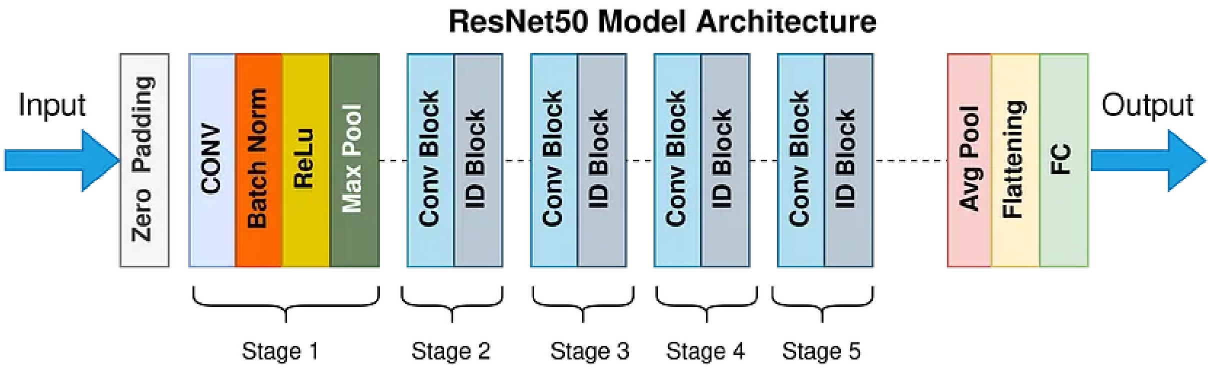
```
## Visualize Random images with no cracks
random_indices = np.random.randint(0, len(no_crack_images), size=4)
print("*****************Random Images without Cracks**************************")
random_images = np.array(no_crack_images)[random_indices.astype(int)]

f, axarr = plt.subplots(2,2)
axarr[0,0].imshow(mpimg.imread(os.path.join(cwd, 'Negative', random_images[0])))
axarr[0,1].imshow(mpimg.imread(os.path.join(cwd, 'Negative', random_images[1])))
axarr[1,0].imshow(mpimg.imread(os.path.join(cwd, 'Negative', random_images[2])))
axarr[1,1].imshow(mpimg.imread(os.path.join(cwd, 'Negative', random_images[3])))
```

[6]

```
*****************Random Images without Cracks**************************

<matplotlib.image.AxesImage at 0x17acdf4ed08>
```

Appendix C: Model Architecture Details Include detailed architecture diagrams or summaries for the ResNet-50 with Segmentation and YOLOv8 models. This can aid readers in understanding the inner workings of the models.


Data Acquisition
Data Augmentation
Data Annotations (LWYS)
Dataset Construction
End
Dataset Construction
Dataset Loading
Anchor Boxes Setting
Parameters Setting
Network Initialization
Model Training
Loss Convergence ?
No
Yes
YOLO-Tomato Model
End
Model Training
Input Images
Model Loading
Prediction
NMS
Output
End
Tomato Detection


ResNet50 Model Architecture
Input
Zero Padding
CONV
Batch Norm
ReLu
Max Pool
Conv Block
ID Block
Avg Pool
Flattening
FC
Output
Stage 1
Stage 2
Stage 3
Stage 4
Stage 5