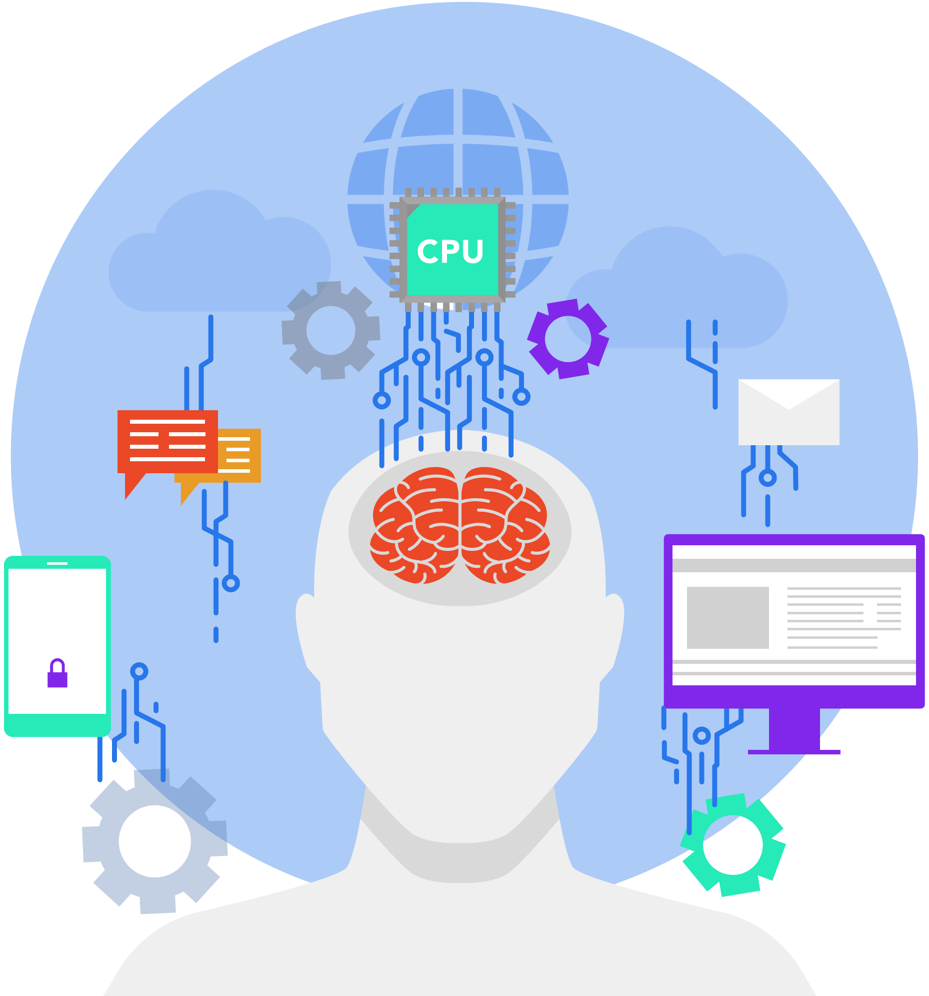


Surface Crack Detection P.S.-1

- Team The ChatGPT Cult

Machine Learning Infographics

1

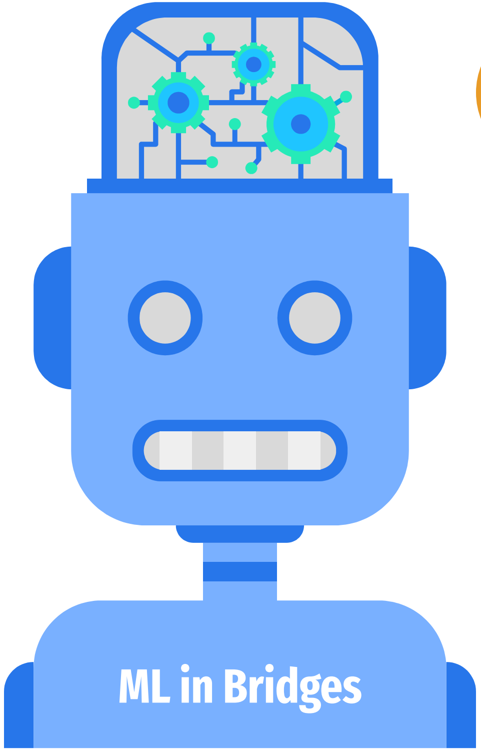
Data Collection

The construction company will collect the site photographs from the site cameras.

2

Data Preprocessing

The images will be preprocessed to resize them to a consistent size and format.



3

Model Selection

A deep learning model will be selected for the task of crack detection.

4

Model Training

The selected model will be trained on the collected and preprocessed images.

5

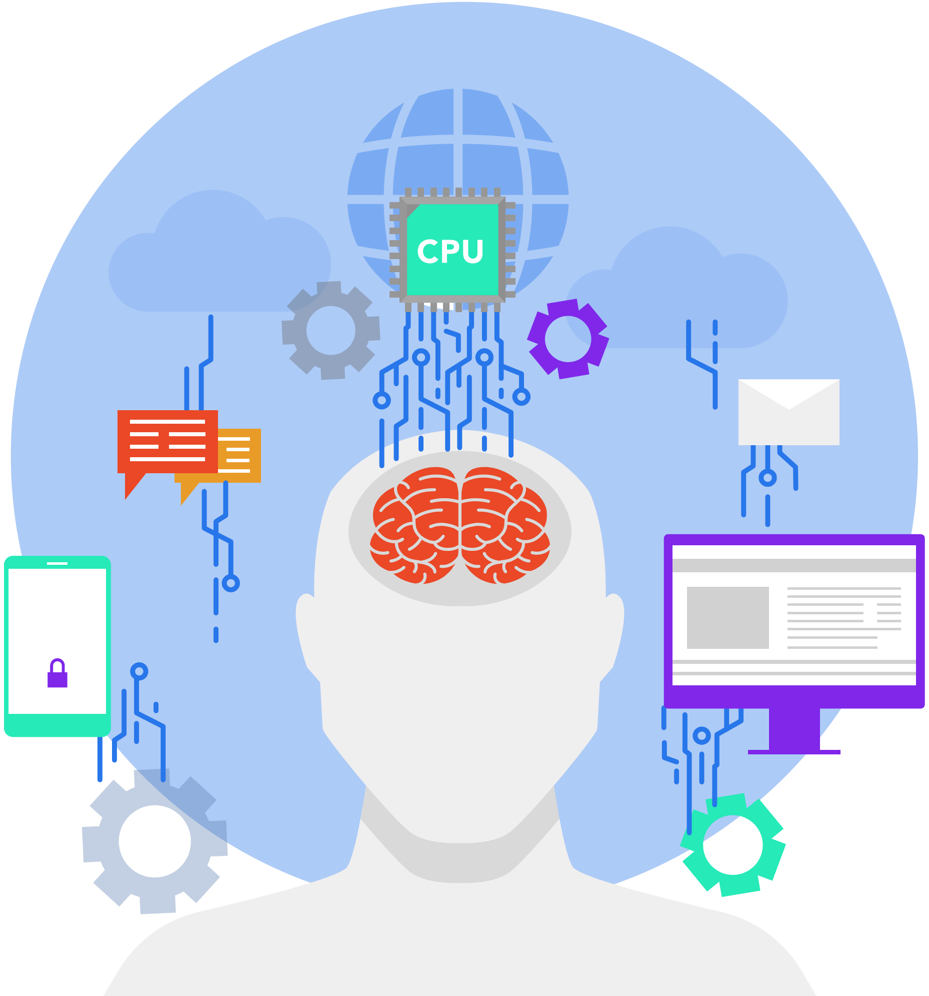
Model Deployment

The trained model will be deployed on the Smart camera device.

6

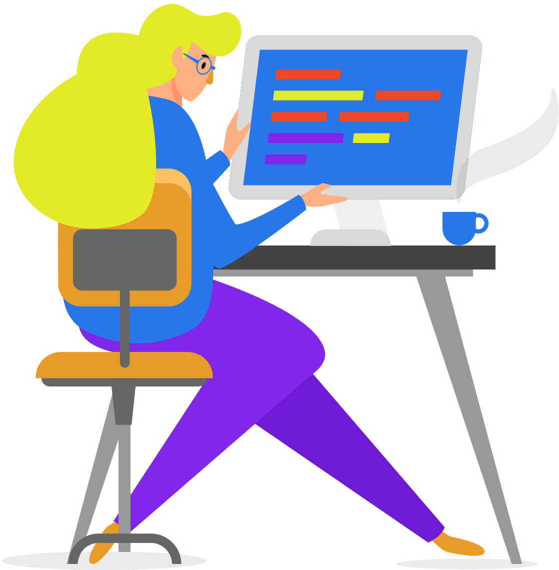
Dashboard

The results will be displayed on a dashboard and accessed by the construction company.



Model Training & Deployment

Section Overview



01 **Image and Data Preprocessing**
Images are reduced to std size

02 **Framework**
Transfer model selection

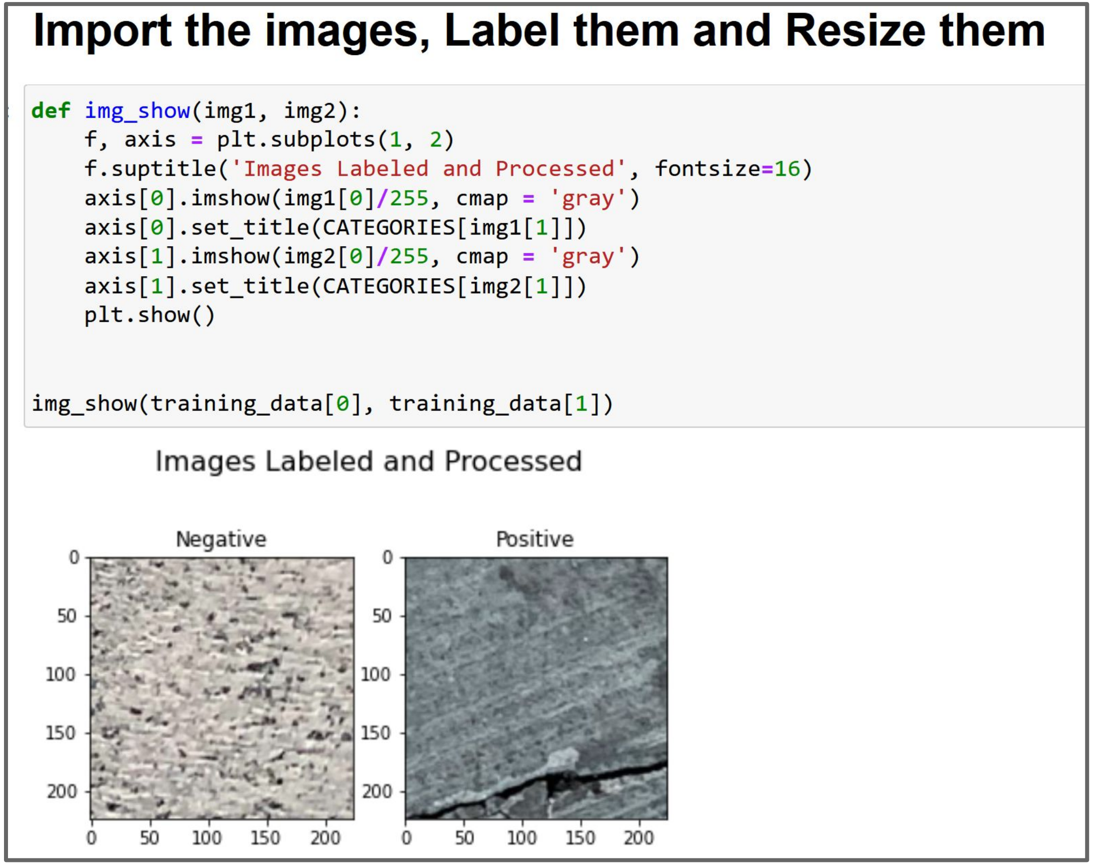
03 **Tweaking the Model**
Adding and removing layers to reduce overfitting

04 **Finalizing the Model**
Selecting the best performing model

05 **Deployment**
Convert the detection to a simple function

06 **Future Aspects?**
To implement this in cameras for ease of use

Image and Data Preprocessing



Import the images, Label them and Resize them

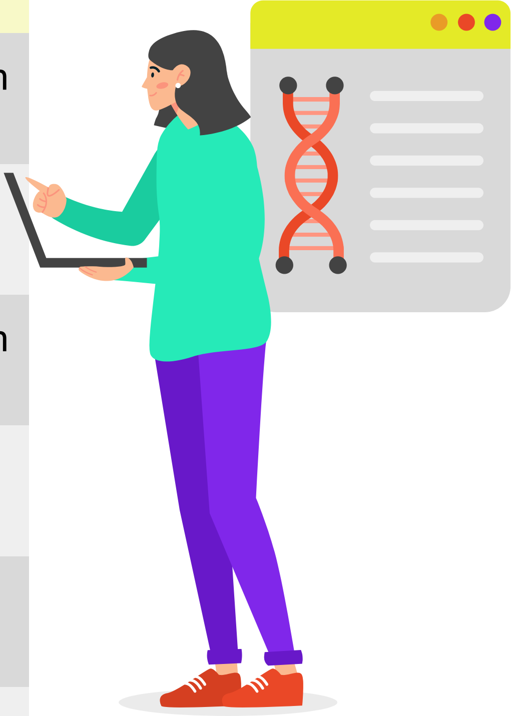
```
def img_show(img1, img2):
    f, axis = plt.subplots(1, 2)
    f.suptitle('Images Labeled and Processed', fontsize=16)
    axis[0].imshow(img1[0]/255, cmap = 'gray')
    axis[0].set_title(CATEGORIES[img1[1]])
    axis[1].imshow(img2[0]/255, cmap = 'gray')
    axis[1].set_title(CATEGORIES[img2[1]])
    plt.show()


img_show(training_data[0], training_data[1])
```

- Images are imported from the dataset provided
- These are categorized in training, testing and validation datasets
- Further these arrays are given a label and are stored in numpy arrays

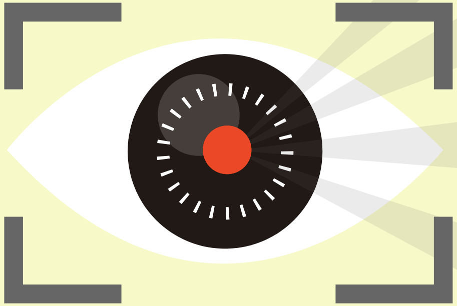
Transfer Learning Models

Models	
VGG-16	Deep convolutional image classification model.
ResNet-50	Deep residual learning image classification model
Inception-v3	Deep convolutional image classification model with factorized convolutions
MobileNet	Lightweight deep convolutional mobile model
DenseNet	Deep convolutional model with dense connections
EfficientNet	High-performance, yet efficient deep convolutional model



Framework

Why EfficientNet-B0?



Scalability

Due to its excellent architecture, expanding the scope of your project is not an issue

Architecture

Due to the use of the baseline network the Architecture is flexible and easier to understand for further enhancement of the model

Performance

EfficientNet models achieve both higher accuracy and better efficiency over existing CNN

Computation Cost

EfficientNet reaches at 97% acc for top-5 while being 8.4x smaller and 6.1x faster on Cpu inference

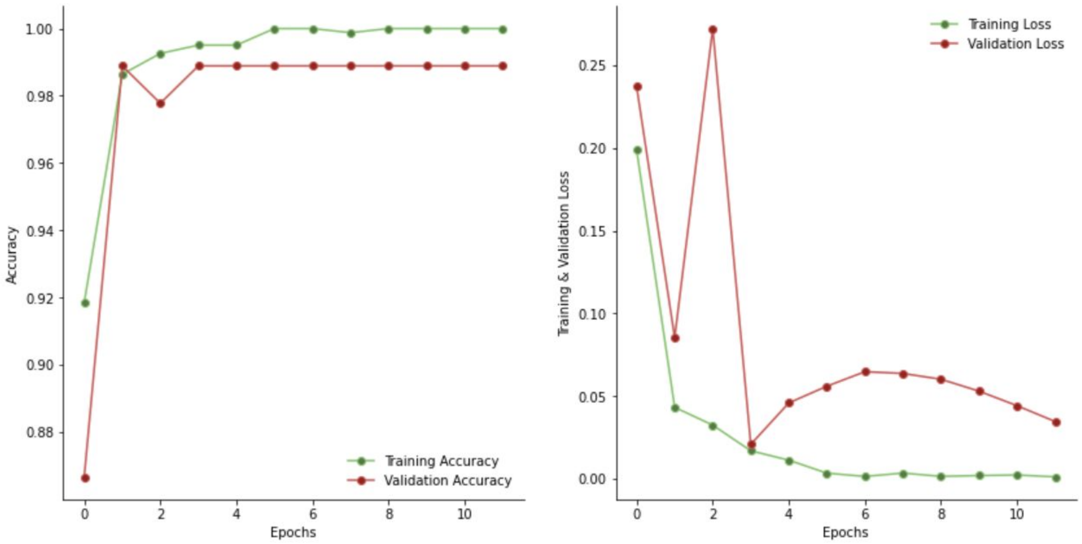
Tweaking the Model

- Overfitting was an issue in the initial runs as the accuracy for the training dataset was high, but the accuracy for the validation set was low
- To reduce this we reduced the number of neurons by 50 %
- This increased the rate of calculations as well as the overall accuracy of the validation set , Solving the problem of overfitting.

Finalizing the model

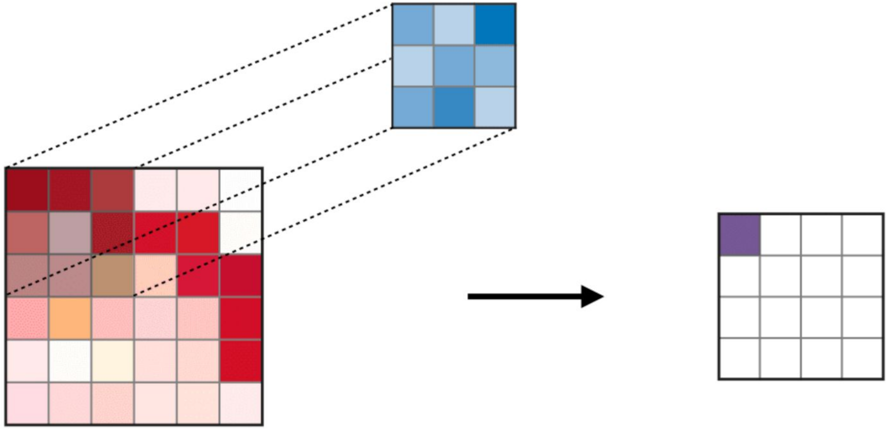
This part is self explanatory, the model with best performance was selected and finalized for implementation

Epochs vs. Training and Validation Accuracy/Loss



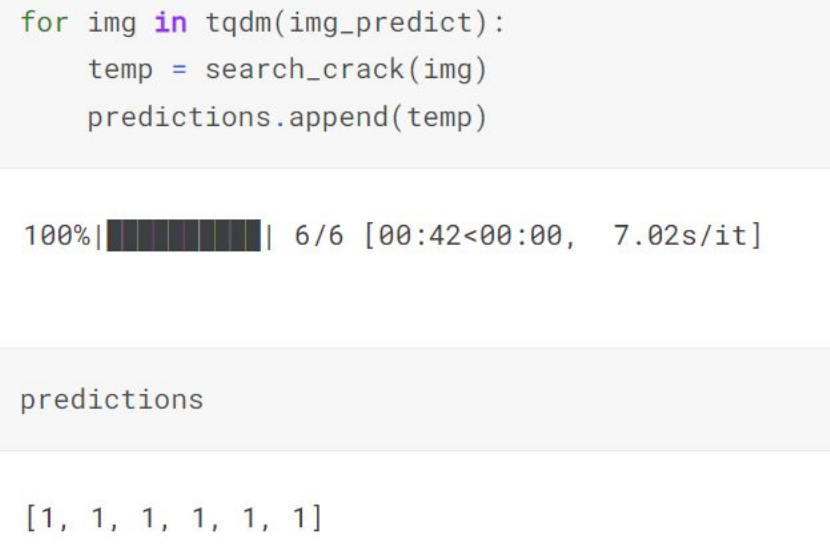
Deployment

- The model is deployed as a simple function which scans the image from the camera and divides it into multiple parts.
- The scanning of the frame begins from left to right and top to bottom and creates a new image for our model to predict
- Each of these images return a prediction of whether or not there is a crack
- Thus this function scrapes through all parts of the image to find a crack



Final Results

- The predictions dataset had a close to real life situation as the crack was tiny unlike the dataset used for training.
- Our function yet scrapes through the image for the crack and a similar approach can be used for many other problems.
- When the function is applied on the final prediction dataset, this is what we got:



```
for img in tqdm(img_predict):
    temp = search_crack(img)
    predictions.append(temp)
```

```
100%|██████████| 6/6 [00:42<00:00,  7.02s/it]
```

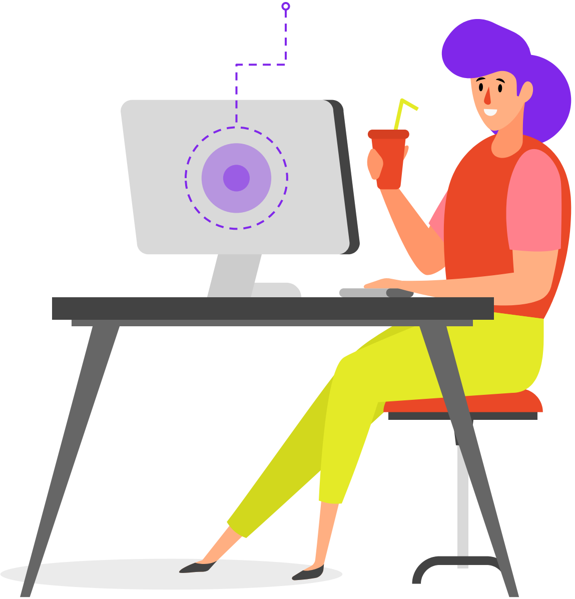
```
predictions
```

```
[1, 1, 1, 1, 1, 1]
```

Future Aspects?

AI

System configuration



01 Take the Photos

The client with collect the data of the suspected wall

Upload the Photo 02

The suspected photos should be uploaded on App

03 PreProcess the Image

The images will be send on the server and preprocessed to fit the Model requirement

Feed the image to Model 04

Then the image will be feeded as in input to the deployed model

05 Get The prediction

The Model will give the desired prediction

Display the Result 06

Then the computed result will be rendered in on the app Dashboard