# Smart India Hackathon Final Presentation

**Team EPOX**

**MK199**

SIH
SMART INDIA HACKATHON 2020

सत्यमेव जयते
Ministry of Rural Development Government of India

MK199: Automatic Assessment of Pavement condition

based on road photographs

# Key Aspects and requirements

Through EMARG and PMGSY-III, NRIDA has collected a vast collection of pictures of roads. These pictures are collected while doing inspection of roads or collection of PCI through visual inspections. An **AI assisted module** would be able to **automatically assess the picture and identify common issues** such as **shoulder clearance, potholes, road furniture** etc. Requirement is of a solution where there should be a provision to capture the chainage wise **pavement condition index**. Use of **open source software and existing neural network is encouraged. Train a machine learning model**, computer vision etc. which can **identify common issues with pavement based on photograph(s) per road alone**.

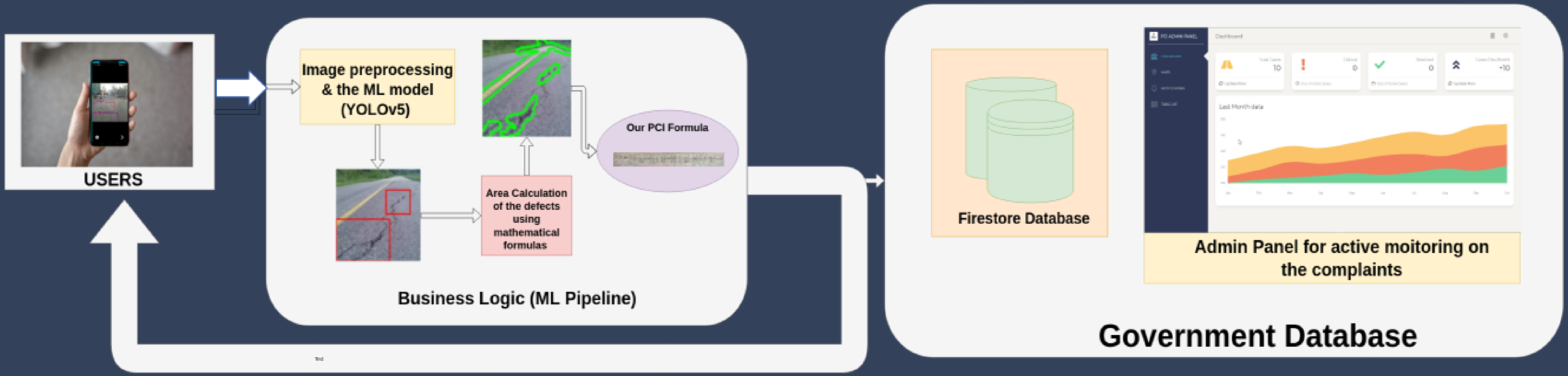
# Problems which our solution will solve

- Difficulty in **data interpretation** between the government, the users and the tech providers. We offer a **CONDUIT** between the above parties
- No clarity in the general public regarding the information which needs to be conveyed to the government when lodging a complain
- Inflexibility in number of features. The number of defects in  may vary according to the region and organization
- **Types of rural roads** may vary according to the region
- Human intervention required. Our method will introduce **complete automation** but will allow **fine tuning** according to the inspectors
- Efficiency in **M-R(Maintenance and Rehabilitation) and better utilization of resources and planning**
- No standardization of PCI

# Focus Points of Our Solution

- **Standardizing** PCI through mathematical justification
- Creating a **CONDUIT OF INFORMATION** between the users, the government and us.
- Empowering the people
- Extremely accessible and **user-friendly app**
- Keeping government database updated through crowd-sourcing
- Allowing fast and concise information access and maintenance for the govt. with our **dashboard**
- Reducing cost of M-R
- Using latest **SOTA** computer vision models to ensure ease in future proofing and further updates

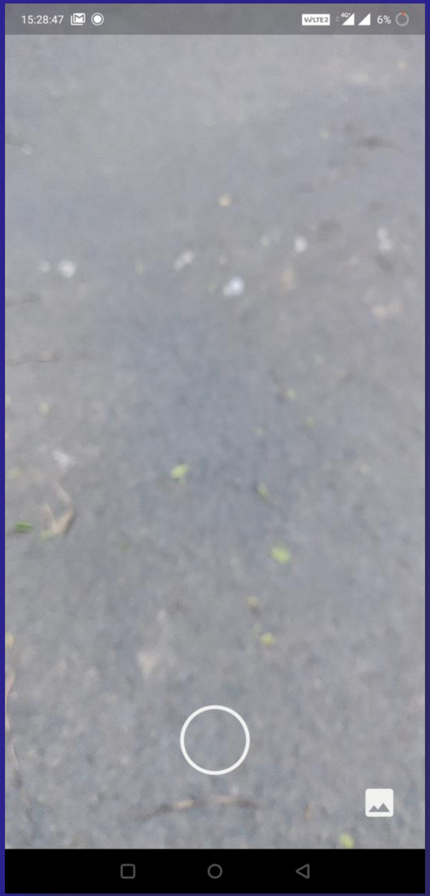
# Workflow/Pipeline



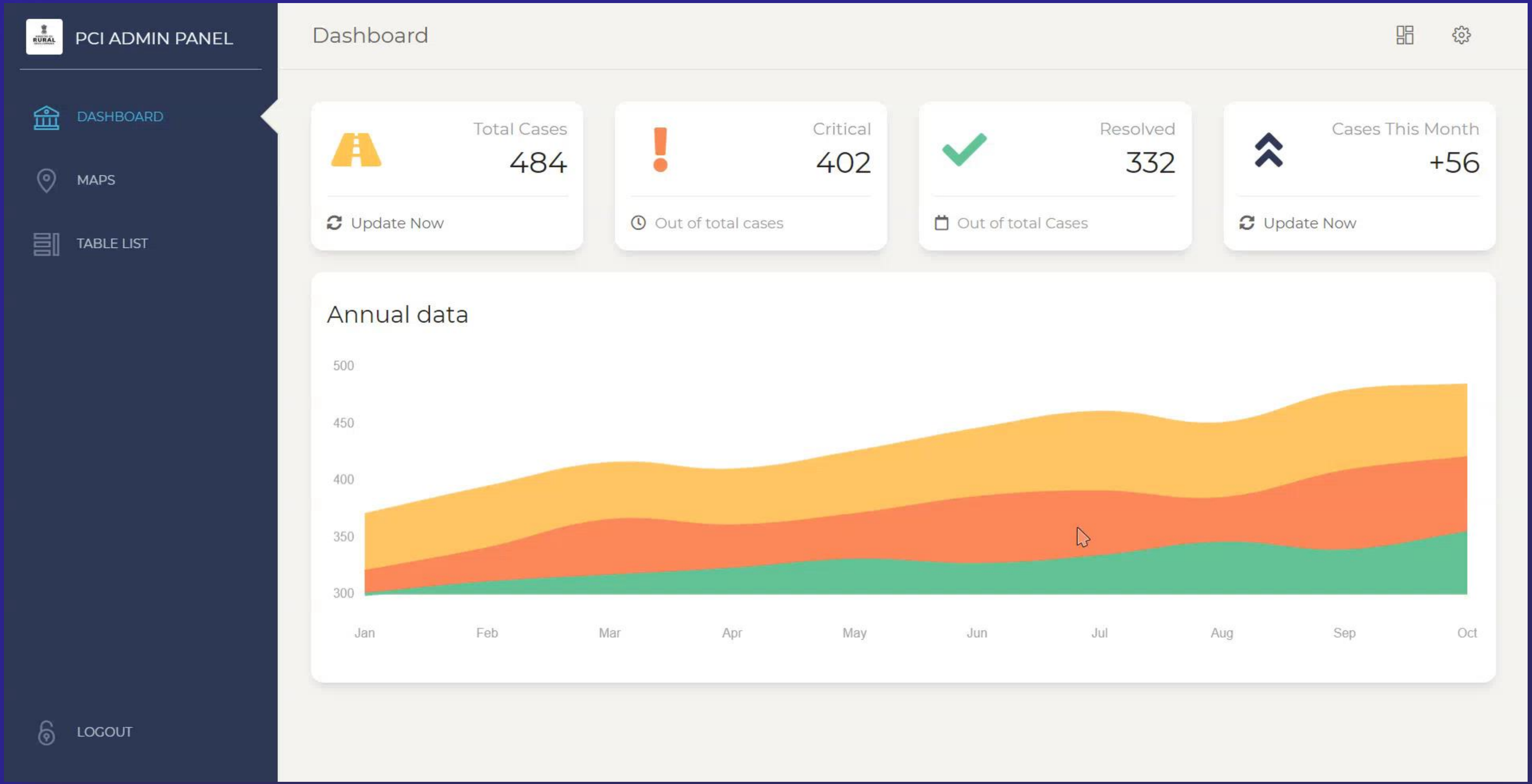
# UI / UX

App and Dashboard
The Conduit of Information

# UI / UX



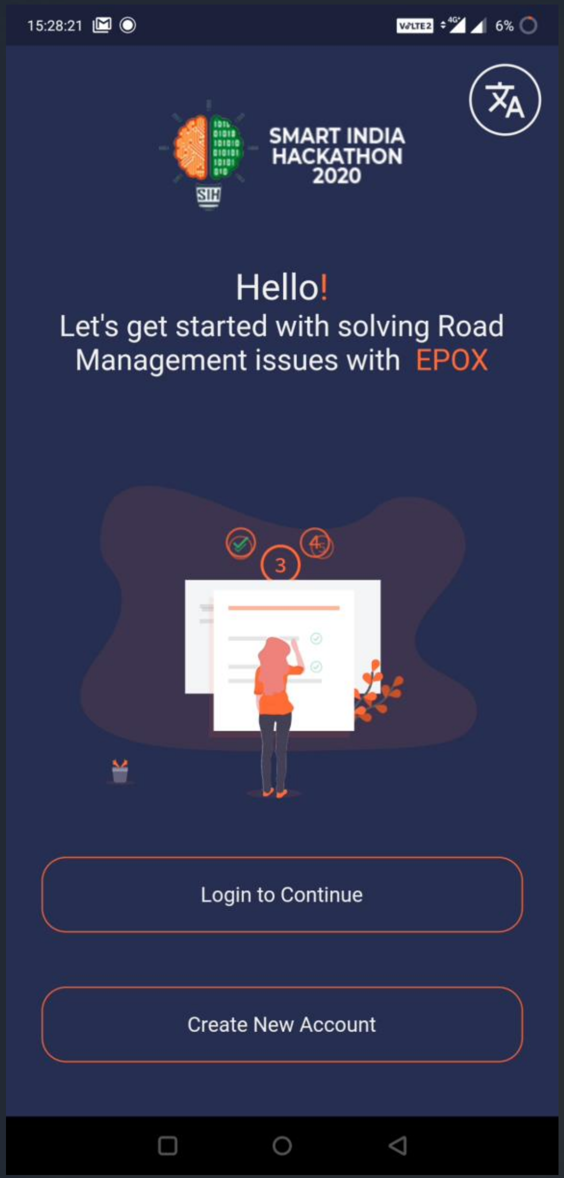

App



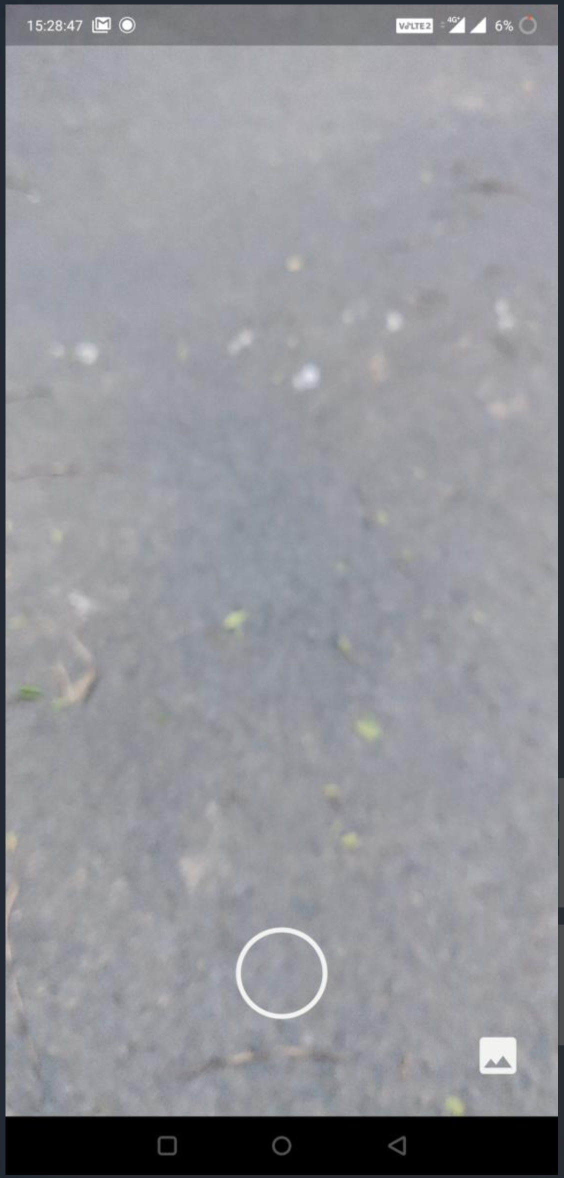

Dashboard

# The APP (*meri sadak*)

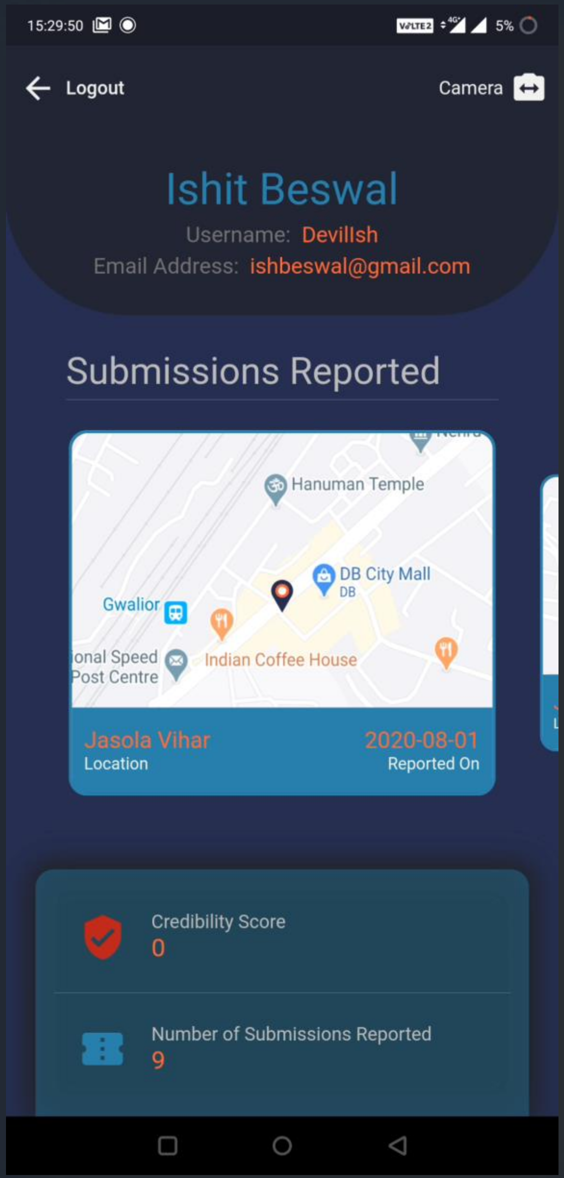
- Extremely accessible and **user-friendly**
- For **all strata** of society
- No skills needed
- **Multilingual** support
- User credibility score available
- Geo-Tagging
- Interface similar to apps like **SnapChat and Uber**
- Just snap a picture and you are **good to go!!!**




Localized for easy access

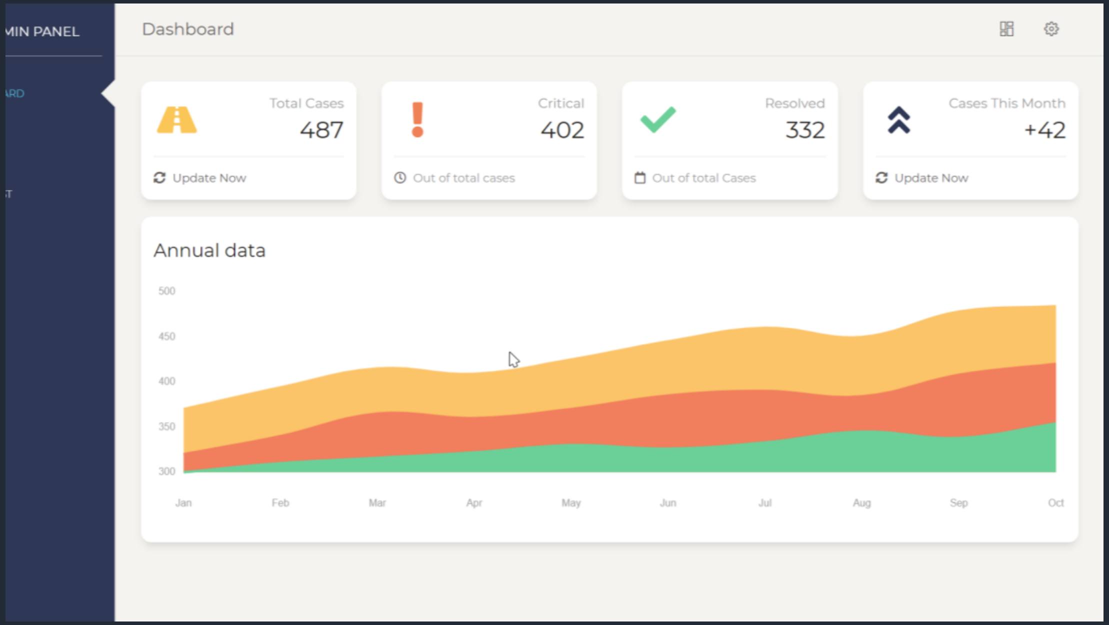



Familiar interface and controls



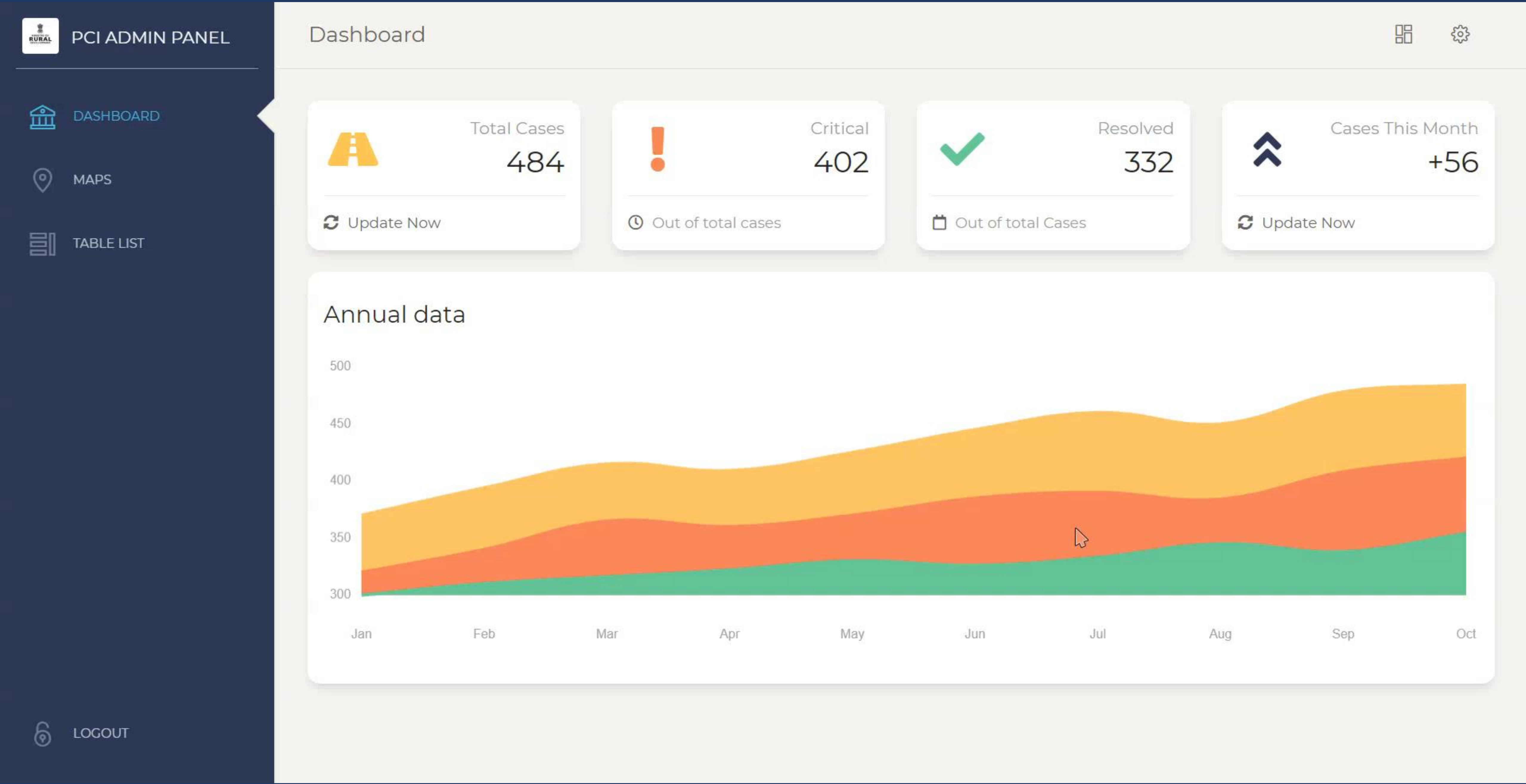

All info at a glance

# The Dashboard (PCI Admin Panel)



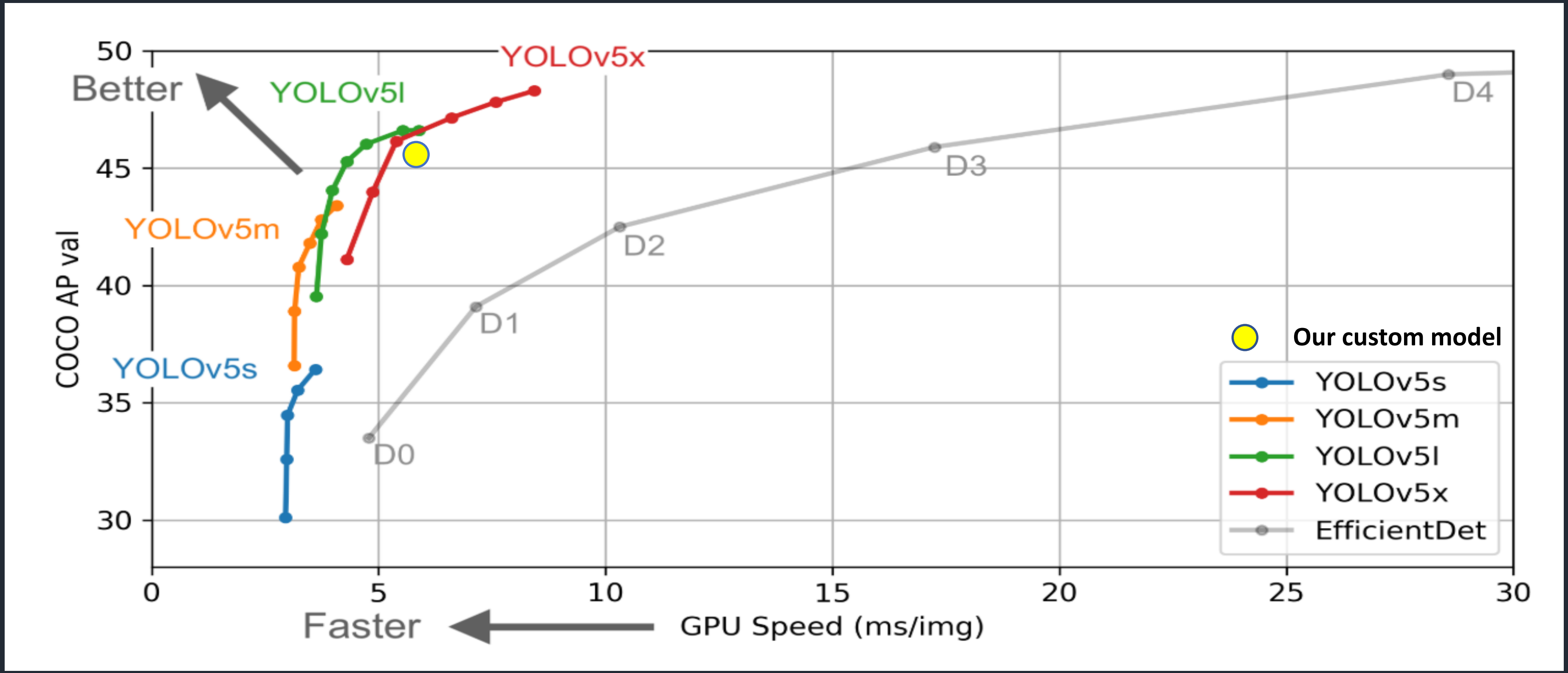

- Fast and concise information access
- Saves **time** of busy officials
- Database **updation and maintenance** done automatically
- Easy analysis of **trends**
- **Cost saving** in **maintenance and rehabilitation**
- Optimized **regional planning**
- Better **resource utilization**

# Admin panel Demo video



# YOLOv5 – State of the art computer vision algorithm

Achieves more in less time (compared to Google's EfficientDet Algorithm)



# OUR MODEL ARCHITECTURE

- **165 layer deep neural network**
- Uses State of the Art **Bi-FPN layers**
- Computes **6.86 million** parameters
- Our model has been trained for **16 hours on free resources (Google Colab )**
- Due to lack of computing power we couldn't use large or x-large models
- **We made our own custom YOLO architecture between the small and medium model**

# 165 Layer Deep Neural Network

```
                 from  n    params  module                                  arguments
  0                -1  1      3520  models.common.Focus                     [3, 32, 3]
  1                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
  2                -1  1     20672  models.common.Bottleneck                [64, 64]
  3                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
  4                -1  1    161152  models.common.BottleneckCSP             [128, 128, 3]
  5                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
  6                -1  1    641792  models.common.BottleneckCSP             [256, 256, 3]
  7                -1  1   1180672  models.common.Conv                      [256, 512, 3, 2]
  8                -1  1    656896  models.common.SPP                       [512, 512, [5, 9, 13]]
  9                -1  1   1905152  models.common.BottleneckCSP             [512, 512, 2]
 10                -1  1   1248768  models.common.BottleneckCSP             [512, 512, 1, False]
 11                -1  1     13851  torch.nn.modules.conv.Conv2d            [512, 27, 1, 1, 0]
 12                -2  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 13           [-1, 6]  1         0  models.common.Concat                    [1]
 14                -1  1    197120  models.common.Conv                      [768, 256, 1, 1]
 15                -1  1    313088  models.common.BottleneckCSP             [256, 256, 1, False]
 16                -1  1      6939  torch.nn.modules.conv.Conv2d            [256, 27, 1, 1, 0]
 17                -2  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
 18           [-1, 4]  1         0  models.common.Concat                    [1]
 19                -1  1     49408  models.common.Conv                      [384, 128, 1, 1]
 20                -1  1     78720  models.common.BottleneckCSP             [128, 128, 1, False]
 21                -1  1      3483  torch.nn.modules.conv.Conv2d            [128, 27, 1, 1, 0]
 22      [-1, 16, 11]  1         0  models.yolo.Detect                      [4, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]]]
Model Summary: 165 layers, 6.8692e+06 parameters, 6.8692e+06 gradients
```

# DATASET

- Using dataset of **Indian Roads** (around Noida region)
- We detect
  - Linear Cracks - Longitudinal and Transverse Cracks
  - Alligator Cracks
  - Potholes
  - Road Shoulders and furniture
- Has ~7000 images, divided into train, validation and test

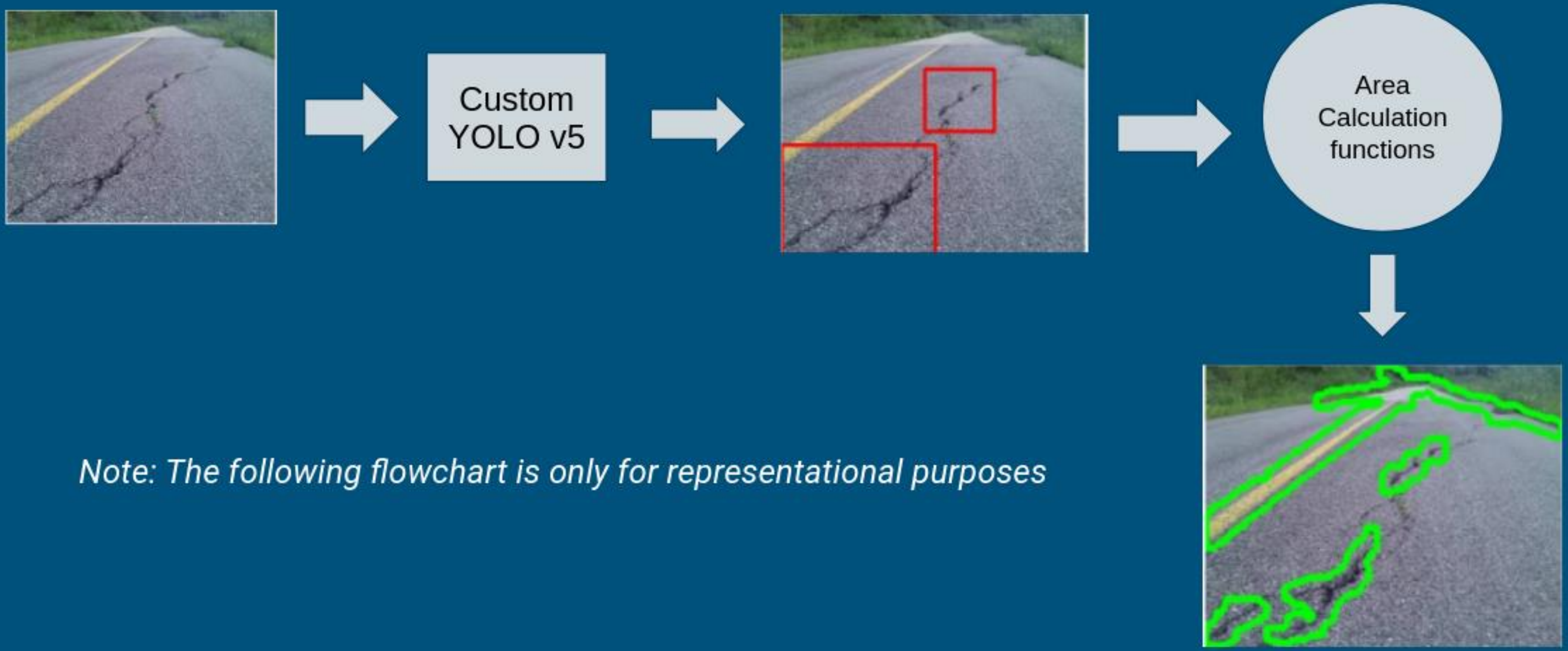
# The Algorithm and PCI calculation

Novel, lightweight and Independent

$$PCI = 100 \times \left[1 - \left\{1 - \frac{LCI}{100}\right\} \times 0.355\right] \times \left[1 - \left\{1 - \frac{TCI}{100}\right\} \times 0.355\right] \times \left[1 - \left\{1 - \frac{ACI}{100}\right\} \times 0.355\right] \times \left[1 - \left\{1 - \frac{PHI}{100}\right\} \times 0.262\right]$$

# Step 1: Calculating the area of the defects

The first step is to get a rough estimate of the area of the defect in question




*Note: The following flowchart is only for representational purposes*

# Step 1: Calculating the area of the defects

## Python Functions

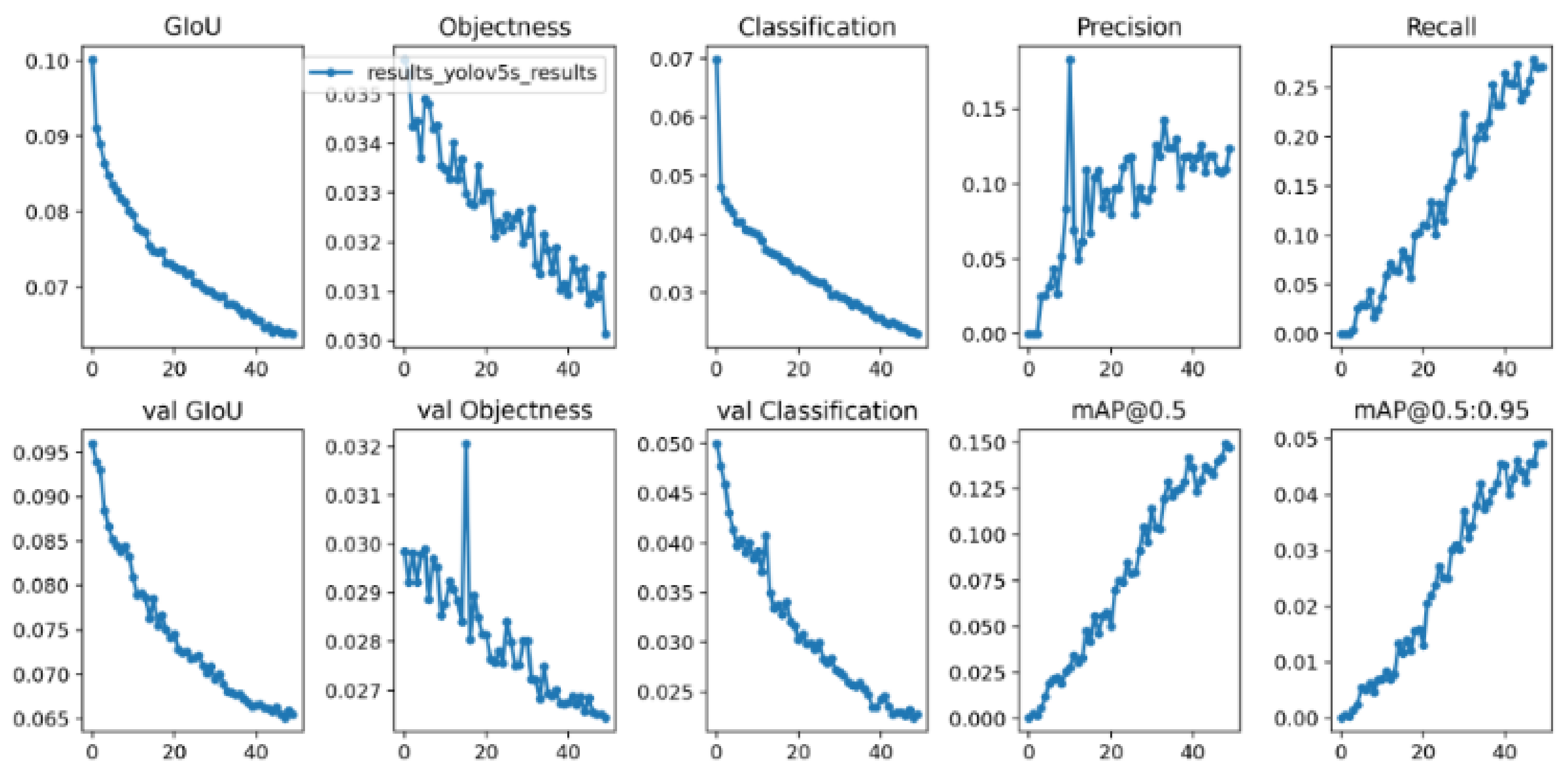
```
if class_num == 0:
  diam = 0.285 * w * h * 100             #calculating approx area
  if diam <= 2.41:
    dict["Linear Crack"]["Low"] = dict["Linear Crack"]["Low"] + diam
  elif diam > 2.42 and diam <= 4.80:
    dict["Linear Crack"]["Medium"] = dict["Linear Crack"]["Medium"] + diam
  else:
    dict["Linear Crack"]["High"] = dict["Linear Crack"]["High"] + diam

elif class_num == 1:
  diam = w * h * 100
  if diam <= 33.5:
    dict["Alligator Crack"]["Low"] = dict["Alligator Crack"]["Low"] + diam
  else:
    dict["Alligator Crack"]["High"] = dict["Alligator Crack"]["High"] + diam

elif class_num == 2:
  diam = 3.14 / 4 * max(w,h) * max(w,h)* 100
  if diam <= 26:
    dict["Potholes"]["Low"] = dict["Potholes"]["Low"] + diam
  elif diam > 26 and diam <= 52:
    dict["Potholes"]["Medium"] = dict["Potholes"]["Medium"] + diam
  else:
    dict["Potholes"]["High"] = dict["Potholes"]["High"] + diam

elif class_num == 3:
  diam = w * h * 100
  if diam <= 8.3:
    dict["Shoulders"]["Low"] = dict["Shoulders"]["Low"] + diam
  elif diam > 8.3 and diam <= 16.7:
    dict["Shoulders"]["Medium"] = dict["Shoulders"]["Medium"] + diam
  else:
    dict["Shoulders"]["High"] = dict["Shoulders"]["High"] + diam
```

# Step 2: Defining the Model Error Coefficients

Every Deep learning model produces error % it encounters during the training of the model

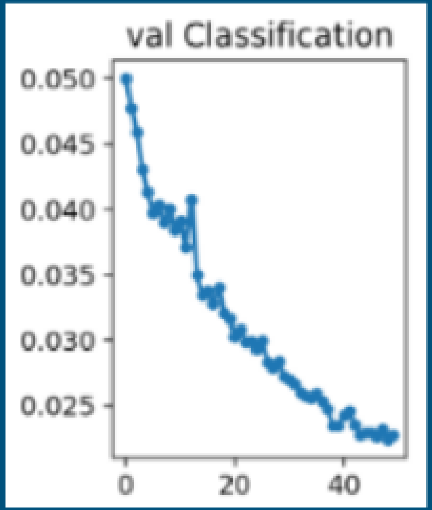
Various such metrics produced by our model

```
# we can also output some older school graphs if the tensor board isn't working for whatever reason...
from utils.utils import plot_results; plot_results()  # plot results.txt as results.png
Image(filename='./results.png', width=1000)  # view results.png
```

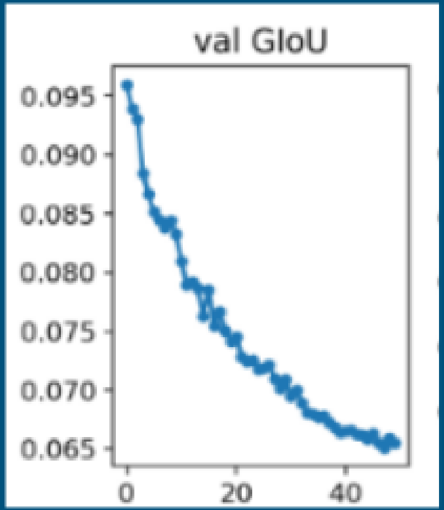


# Step 2: Defining the Model Error Coefficients

From this, we take the **val Classification and GIoU loss functions** for the individual classes/defects to get our **Model Error Coefficients**




+




=

```
#defining MAEs
MAE_ModErr = {
        "Linear Crack": {
                "Low": 5,
                "Medium": 5,
                "High": 10
        },
        "Alligator Crack": {
            "Low": 5,
            "High": 10
        },
        "Potholes": {
            "Low": 10,
            "Medium": 7,
            "High": 10
        },
        "Shoulders": {
            "Low": 10,
            "Medium": 5,
            "High": 10
        }}
```

# Step 3: Calculating the individual defect distress indexes

Taking the threshold PCI = 60 (baseline) and using the Model Error Coefficients (ME)

Individual Defects Cracking Indexes.

$$\text{Linear Crack Index (LCI)} = (100-40)\left[\frac{\%\,\text{low}}{ME_{low}} + \frac{\%\,\text{high}}{ME_{high}} + \frac{\%\,\text{medium}}{ME_{medium}}\right]$$

$$\text{Shoulder Index (SHI)} = (100-40)\left[\frac{\%\,\text{low}}{ME_{low}} + \frac{\%\,\text{high}}{ME_{high}} + \frac{\%\,\text{medium}}{ME_{medium}}\right]$$

$$\text{Pothole Index (PHI)} = (100-40)\left[\frac{\%\,\text{low}}{ME_{low}} + \frac{\%\,\text{high}}{ME_{high}} + \frac{\%\,\text{medium}}{ME_{medium}}\right]$$

$$\text{Alligator Cracking (ACI)} = (100-40)\left[\frac{\%\,\text{low}}{ME_{low}} + \frac{\%\,\text{High}}{ME_{high}}\right]$$

# Step 4: PCI calculation!!!

After we get the individual distress Indexes

$$PCI = 100 \times \left[1 - \left\{1 - \frac{LCI}{100}\right\} \times 0.355\right] \times \left[1 - \left\{1 - \frac{TCI}{100}\right\} \times 0.355\right] \times \left[1 - \left\{1 - \frac{ACI}{100}\right\} \times 0.355\right] \times \left[1 - \left\{1 - \frac{PHI}{100}\right\} \times 0.262\right]$$

## In python

```
PCI = 100 * ((1 - ((1 - (LCI / 100)) * 0.355)) * (1 - ((1 - (ACI / 100)) * 0.355))
             * (1 - ((1 - (PHI / 100))* 0.262)) * (1 - ((1 - (SHI / 100)) * 0.355)))
```

*Yogesh Shah, S.S Jain (2013), Development of Overall Pavement Condition Index for Urban Road Network
https://www.researchgate.net/publication/270848460_Development_of_Overall_Pavement_Condition_Index_for_Urban_Road_Network

# POPULAR OBJECT-DETECTION MODELS

- YOLO v5 : May 2020
- YOLO v4 :  April 2020
- EFFICIENT DET : Nov 2019
- DETECTRON 2 : 2019
- YOLO v3 : March 2018
- Faster RCNN : 2015
- Fast RCNN : 2015

Use the new
**Bi-FPN Layers**

Use only CNN layers
**Outdated**

# Questions we asked ourselves

- Will YOLOv5 remain the State-of-the-art few years down the line ?
- Is restricting our solution to a model really ensuring future proofing of the PCI formula ?

# No!!!

# MODEL ERROR COEFFICIENTS

```
MAE_ModErr = {"Linear Crack": {"Low": 5, "Medium": 5, "High": 10},
        "Alligator Crack": {"Low": 5, "High": 10},
        "Potholes": {"Low": 10, "Medium": 7, "High": 10},
        "Shoulders": {"Low": 10, "Medium": 5, "High": 10}}
```

- Uses the errors of the computer vision model to make corrections in the PCI formula
- Makes the PCI formula **model Independent!!!**

# PCI formula Benefits

- FUTURE PROOFED
- ALLOWS FEATURE INDEPENDENCE
- EMPIRICALLY TESTED
- SAVES MONEY IN THE LONG RUN AS LESS UPDATES REQUIRED
- ENTIRE SOLUTION IS QUITE CHEAP TO IMPLEMENT
- NO HUMAN INTERVENTION REQUIRED AT ANY STEP
- QUANTIFICATION OF COMPLAINTS

# X-FACTORS

- Future Proof solution
- Saves money in Maintenance and Rehabilitation
- Independent PCI calculation
- Cheap and Easy to implement
- Both government and user friendly
- Built on free software solely
- NO Language/Access Barrier

# Exploration of M&R

M&R Strategies based on PCI values

## BUILD-MEASURE-LEARN PROCESS

| PCI Value | Pavement Condition Rating | M&R Strategy | Suggested Maintenance Alternatives |
|---|---|---|---|
| 85-100 | Excellent | Routine Maintenance | Patching, Pothole filling, Crack sealing |
| 70-85 | Very good | Preventive Maintenance | Chip Seal, Micro -Surfacing, Thin Overlays, Fog Seal |
| 55-70 | Good | Rehabilitation | Thick overlays, Mill & Overlays, Full depth patching, Premix Carpet |
| 40-55 | Fair | | |
| 25-40 | Poor | | |
| 10-25 | Very Poor | Reconstruction | Cold in place recycling, Full depth reconstruction, Full depth reclamation |
| 0-10 | Failed | | |

*Cited from Development of Overall Pavement Condition Index for Urban Road Network*

# But how much would it cost in practice?

- In the actual implementation of our solution, the only cost the Government will have to Incur would be the **Server Costs**
- A rough estimate of the server cost to accommodate a user-base of **1 Million active monthly** users is done below

| Technology | Usage | Cost |
|---|---|---|
| AWS or GCP servers | For hosting the deep learning model and the website | ₹ 580 (approximately) (After the first free year) |
| Nvidia Tesla T4 GPU | For training bigger YOLOv5x model for ~2 days | ₹ 1260 (approximately) (one-time investment) |
| Firestore | For storing the user submissions and user details | ₹4500* (safe approximation) |
| | Total | **₹1260 (one time) + ₹5100 (rounded off)** |

*safe estimate by considering uploaded image with average size as 4 MB and 6500 GB worth of image upload on yearly

# TECHNOLOGY STACK

PyTorch

colab

- NumPy
- Scipi
- CudaToolKit
- Matplotlib
- Pycocotools
- Pillow
- Tensorboard
- YAML
- Py-yaml

OpenCV

Cloud Firestore

Flutter

# FUTURE IMPROVEMENTS

Working with video files

Improving the model performance even farther using deeper neural network

Collecting easy feedback of officials through PCI Admin Panel and fine-tuning the model

# SOCIAL IMPACT

- Empowering people by making them part of the solution
- No barriers in filing complaints
- Everyone from a simple rural farmer to a well-educated person can file complaints with the same ease

# Thank you

Created by Team EPOX of Shiv Nadar University