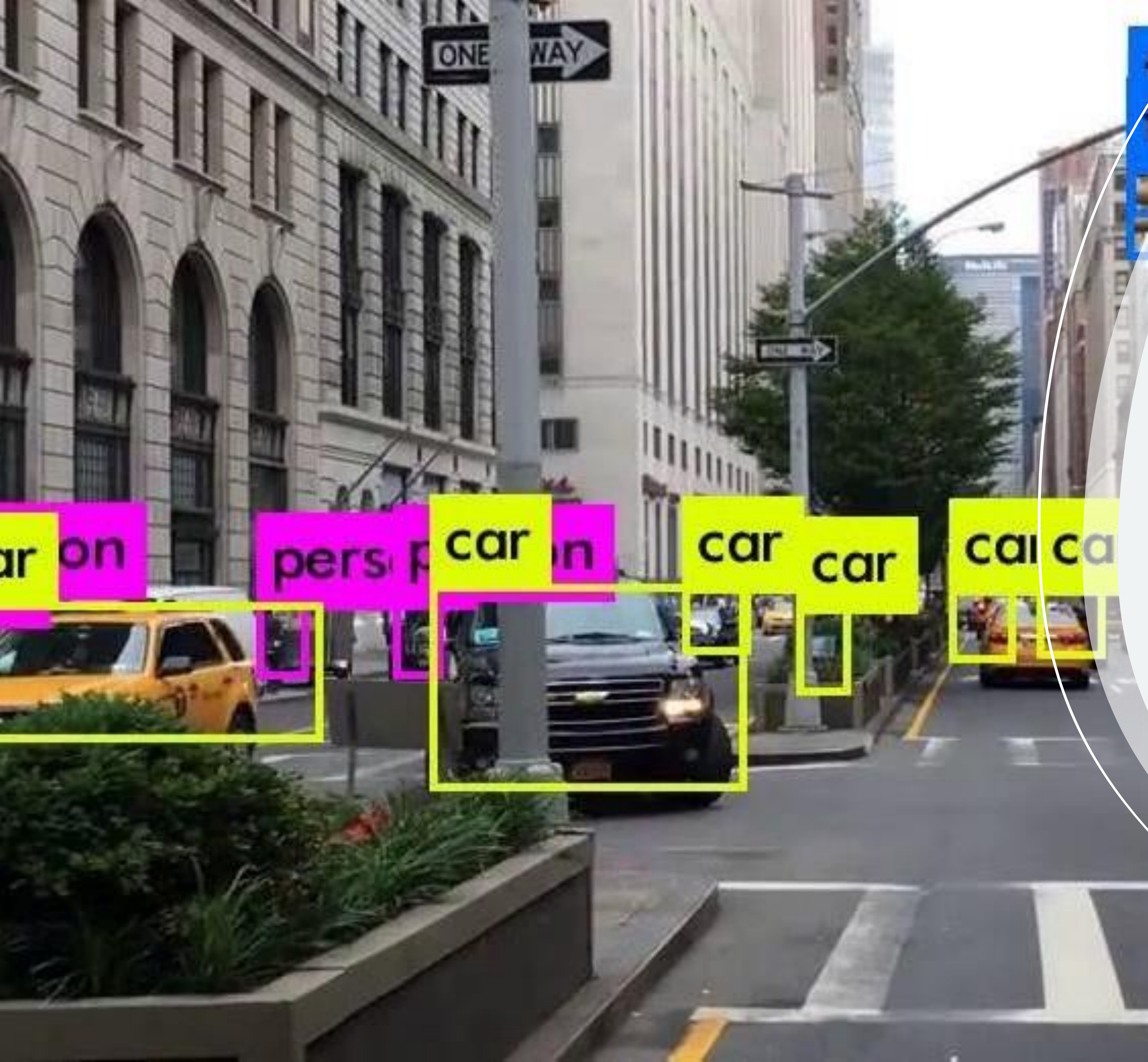


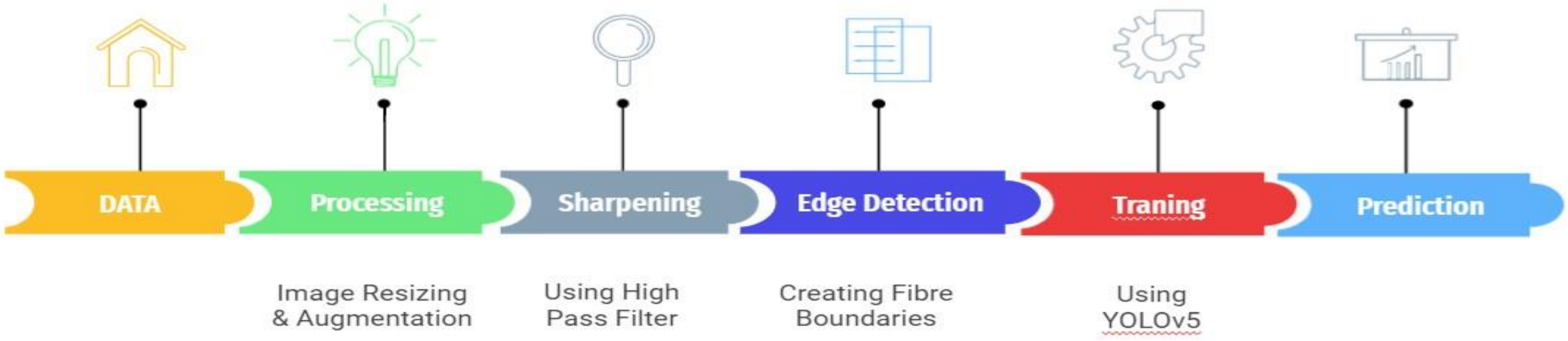
Hackathon
by PRITHVI AI

Saksham Soni

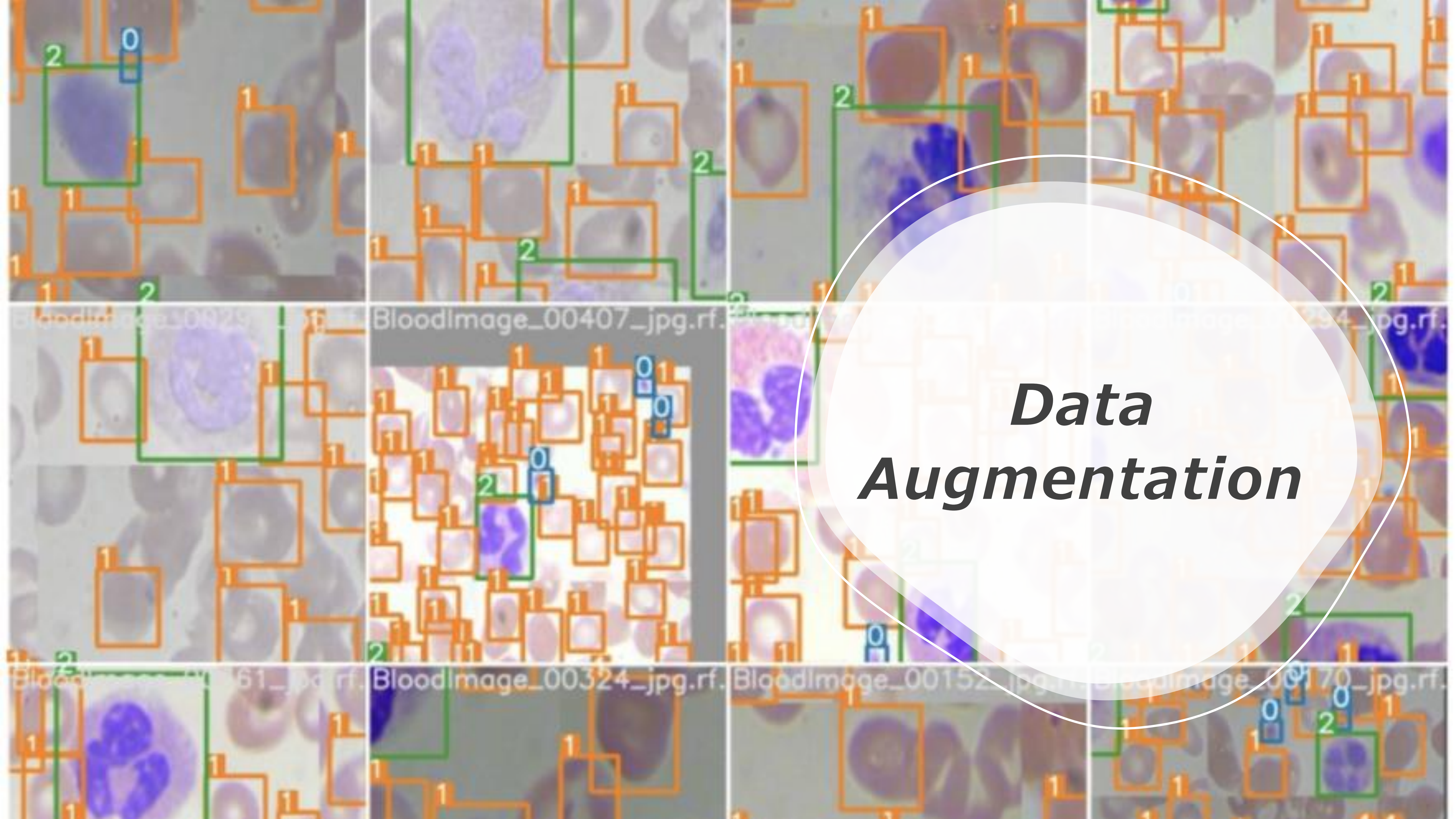
Rohit Kushwah

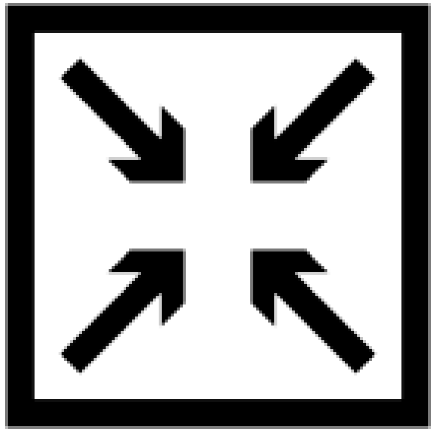
SAHIL

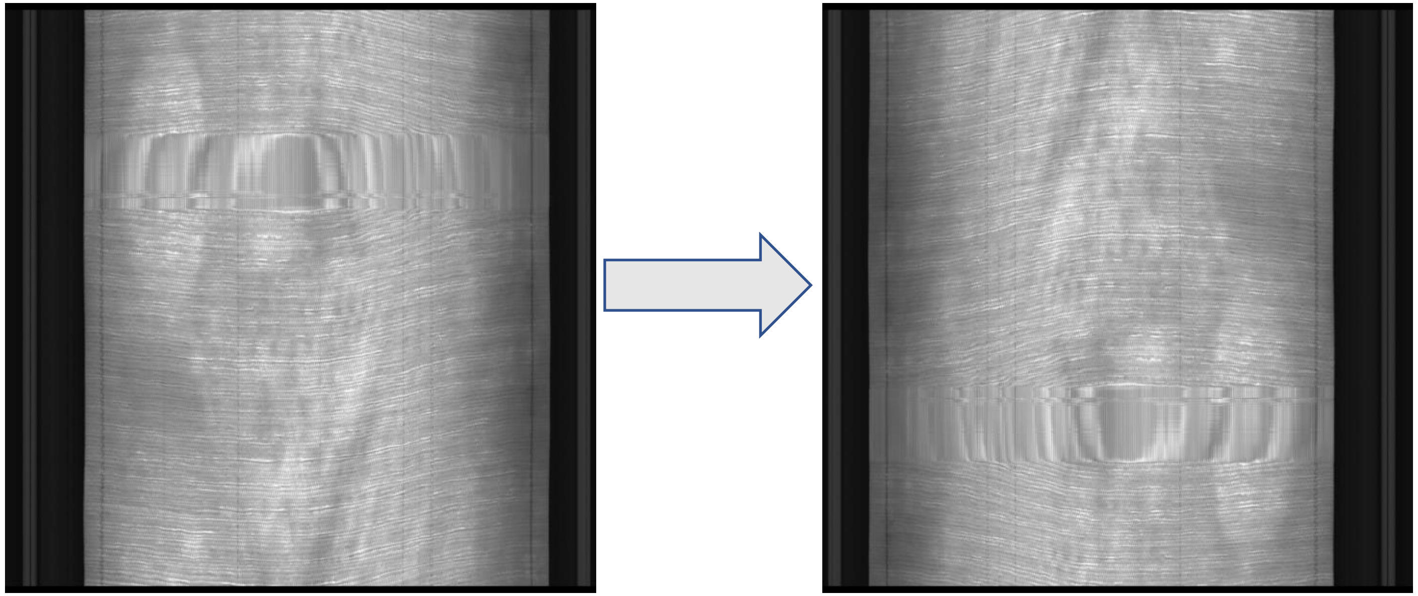
Algorithm Pipeline



Data Augmentation







Processing

- We converted the 1000*4096 image into 1024*1024 using symmetric reduction of height & width and applying padding

- We rotated each image by 180 degree to increase the size of the defective data images

Sharpening

Sharpening is used to sharpen & highlight the edges and make the transitioning of features and details more significant. It doesn't take into account whether it's highlighting the original features of the image or the noise associated with it. It is a process of differentiation.

-1	-1	-1
-1	9	-1
-1	-1	-1

Highpass Filter

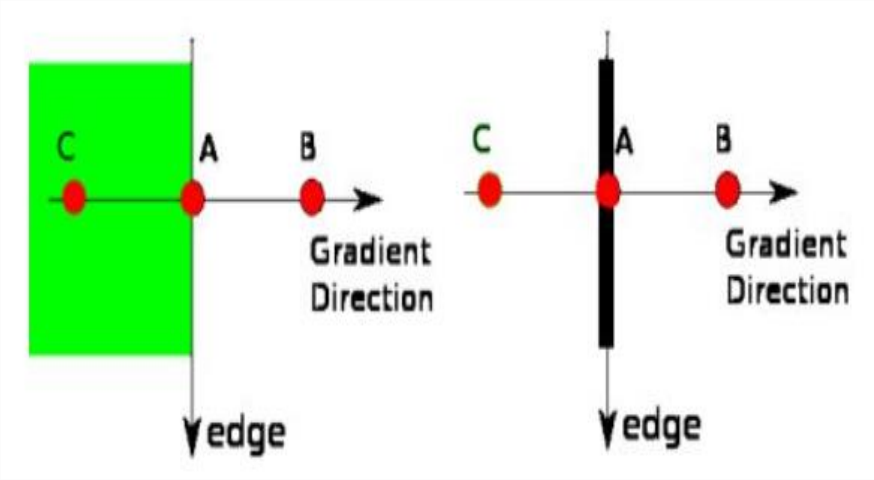
Spatial operation: taking difference between current and averaging (weighted averaging) of nearby pixels

- Can be interpreted as weighted averaging = linear convolution
- Can be used for edge detection

Edge Detection

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$



Noise Reduction

Edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a Gaussian filter

Finding Intensity Gradient of the Image

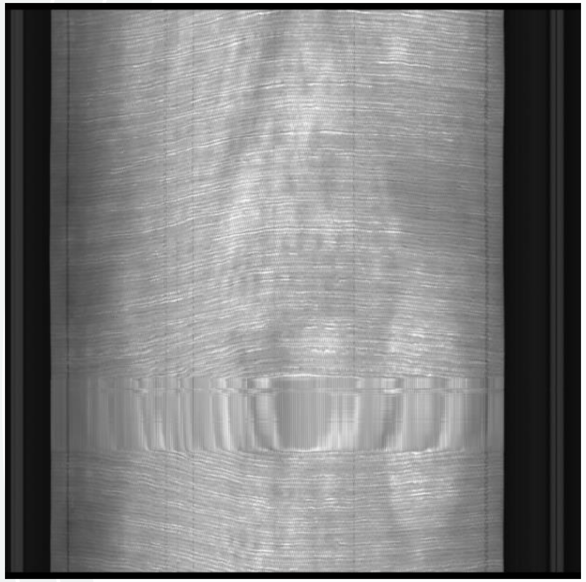
Smoothened image is then filtered with a Sobel kernel in both horizontal & vertical direction to get first derivative in horizontal direction & vertical direction. From these two images, we can find edge gradient & direction for each pixel. Gradient direction is always normal to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions.

Non-maximum Suppression

After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient

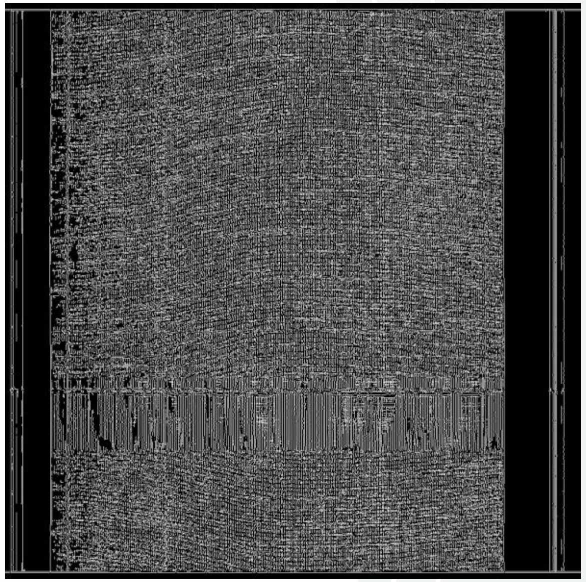
In short, the result you get is a binary image with "thin edges".

Image Preprocessing Output



Initial Image

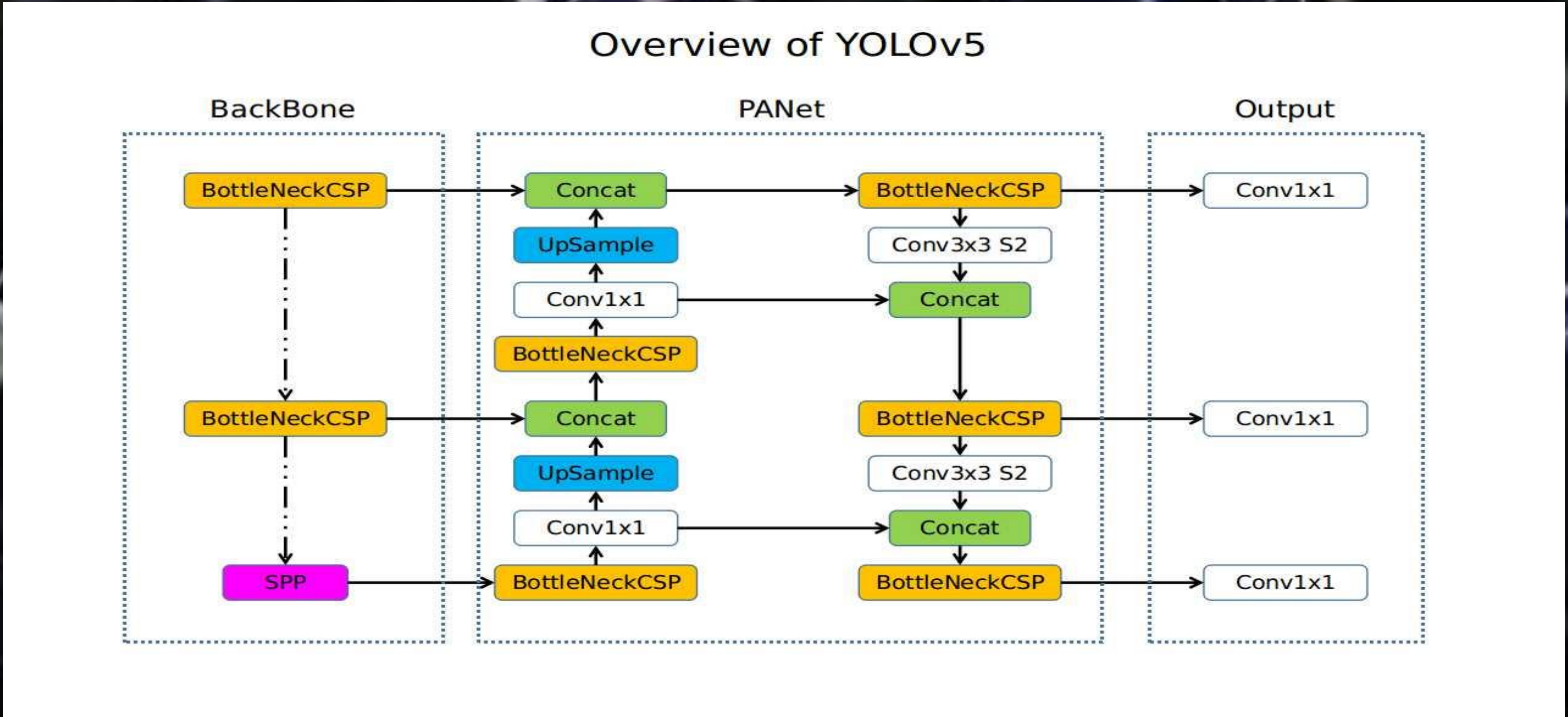
Sharpen ed Image



Edge Model of Image

Training Using Transfer Learning YOLOv5

Overview of the YOLO Architecture

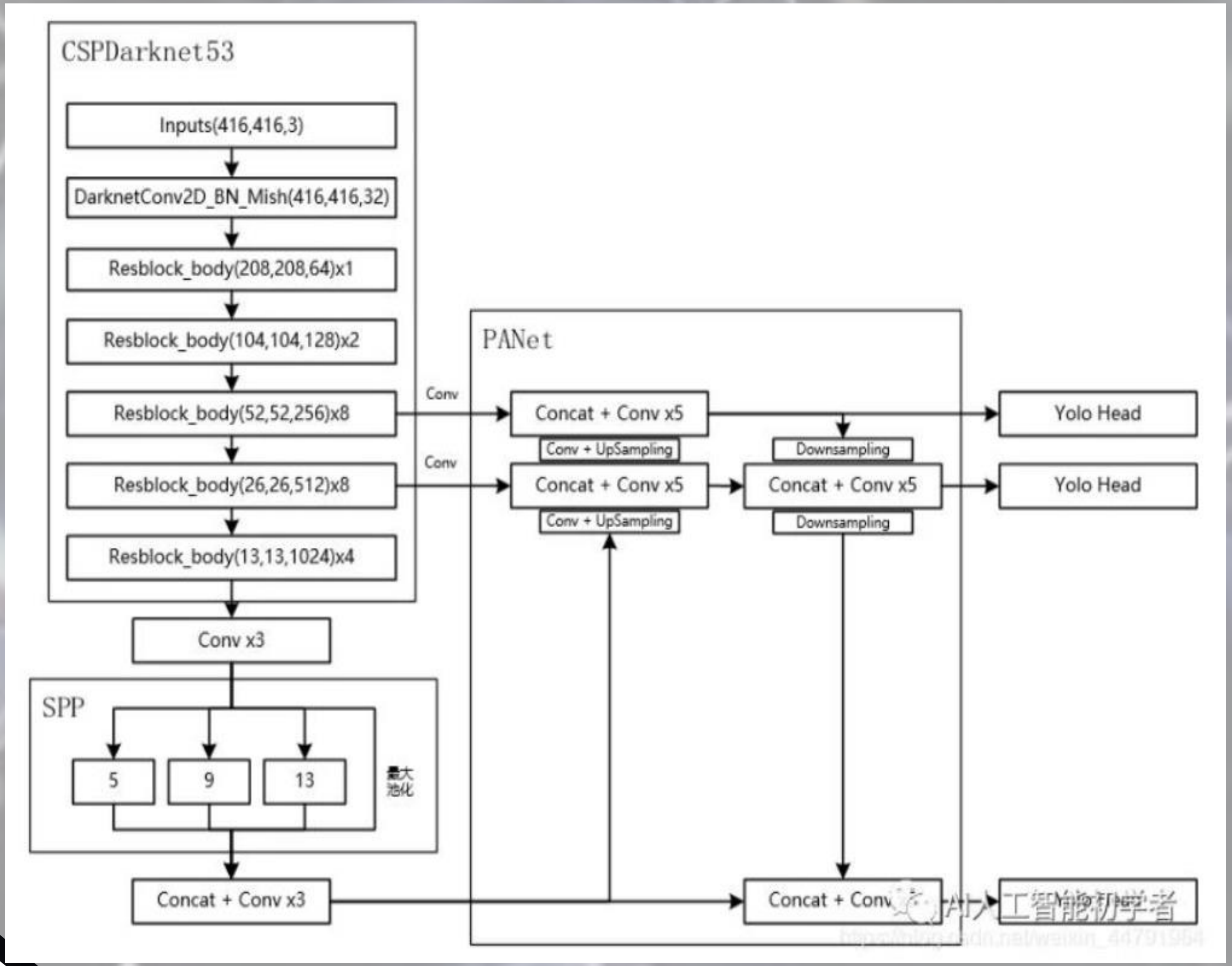


YOLO consists three main pieces

1) ***Backbone*** - A convolutional neural network that aggregates and forms image features at different granularities

2) ***Neck* -** A series of layers to mix and combine image features to pass them forward to prediction

3) ***Head* -** Consumes features from the neck and takes box and class prediction steps



Activation Function

The choice of activation functions is most crucial in any deep neural network. Recently lots of activation functions have been introduced like Leaky ReLU, mish, swish, etc.

YOLO v5 authors decided to go with the Leaky ReLU and Sigmoid activation function.

In YOLO v5 the Leaky ReLU activation function is used in middle/hidden layers and the sigmoid activation function is used in the final detection layer.

Optimization Function

For optimization function in YOLO v5, we have two options

- SGD
- Adam

In YOLO v5, the default optimization function for training is SGD

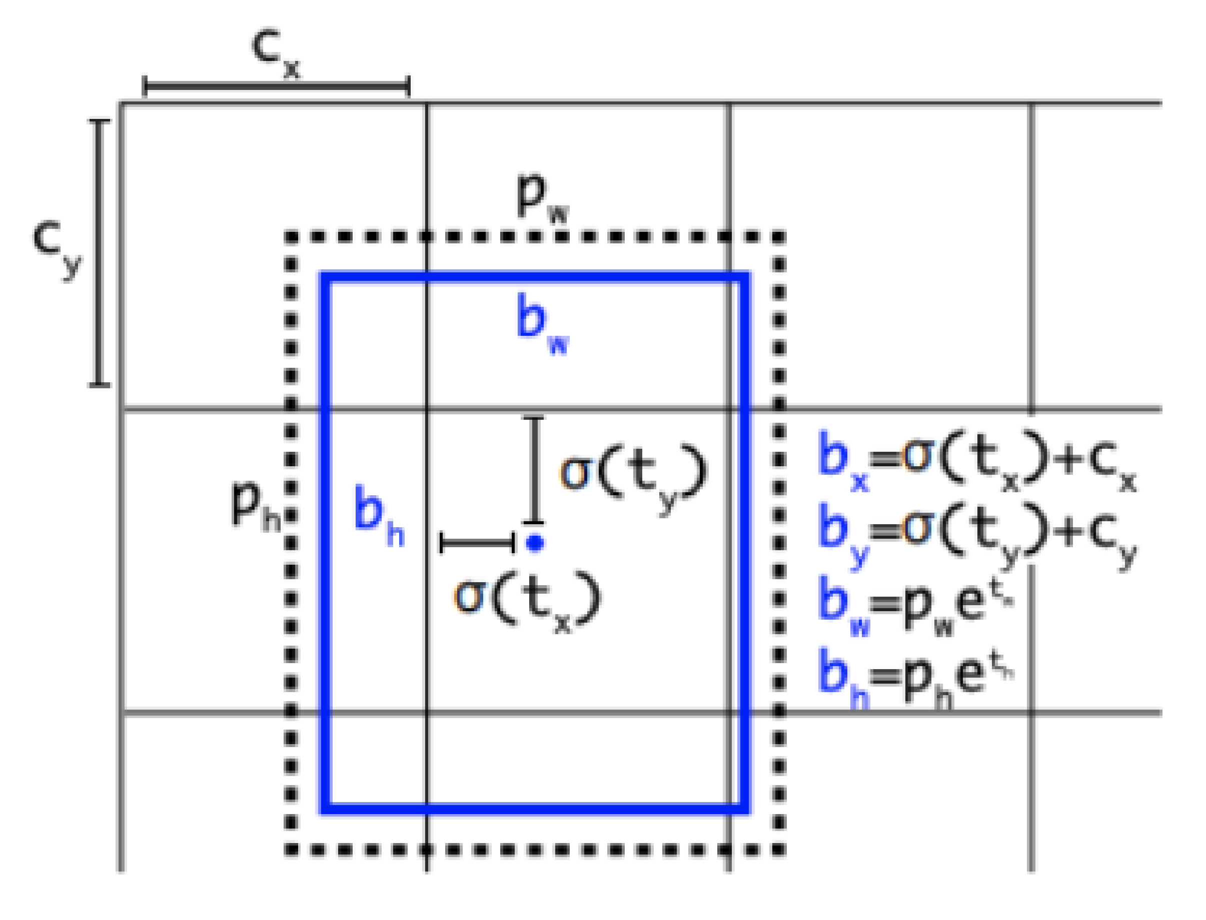
Cost Function or Loss Function

In the YOLO family, there is a compound loss is calculated based on objectness score, class probability score, and bounding box regression score.

Ultralytics have used Binary Cross-Entropy with Logits Loss function from PyTorch for loss calculation of class probability and object score.

We also have an option to choose the Focal Loss function to calculate the loss.

Auto Learning Bounding Box Anchors



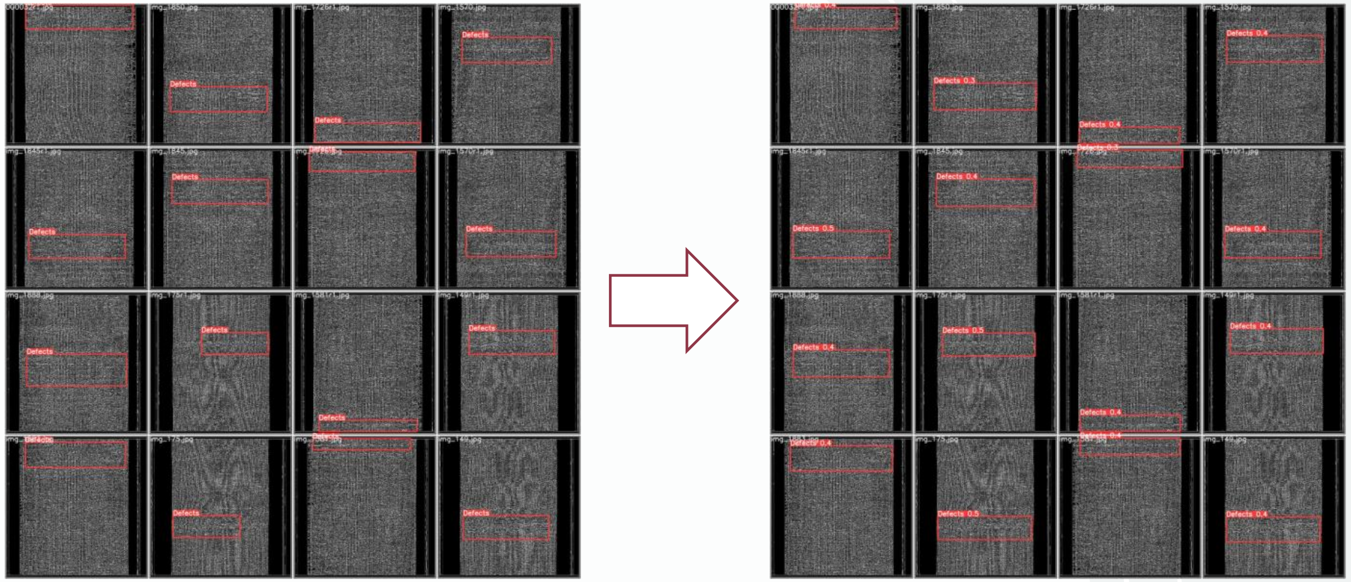
YOLO predicts bounding boxes as deviations from a list of anchor box dimensions

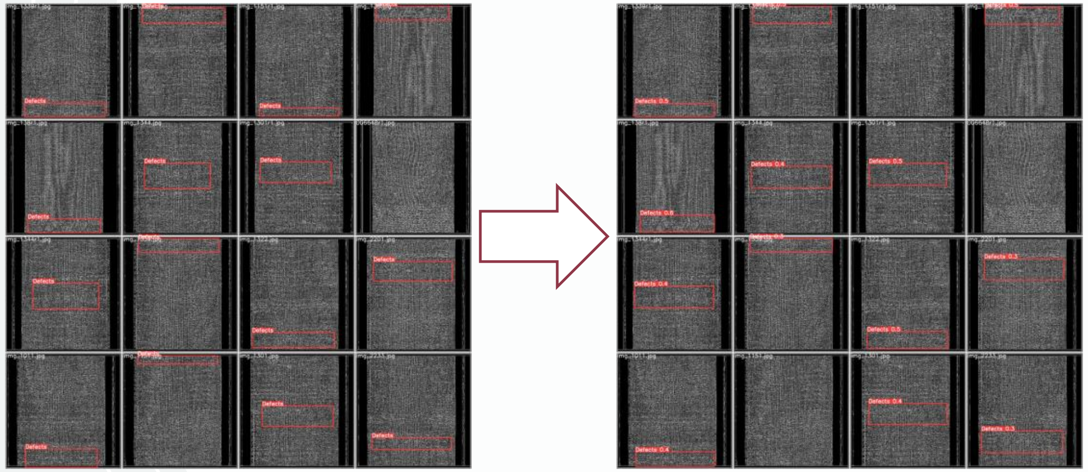
Idea of learning anchor boxes is based on the distribution of bounding boxes in custom dataset with K-means and genetic learning algorithms.

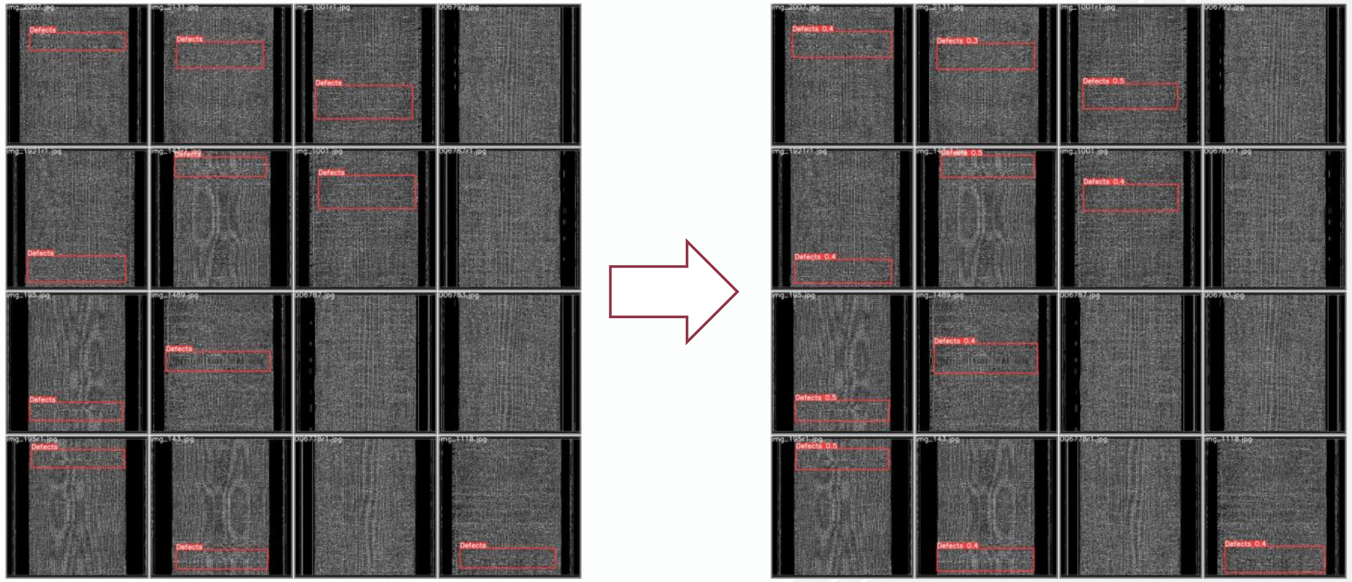
This is very important for custom tasks, because the distribution of bounding box sizes and locations may be dramatically different than the preset bounding box anchors in the COCO dataset

The maximum difference in anchor boxes may occur if we are trying to detect something like giraffes that are very tall and skinny or manta rays that are very wide and flat

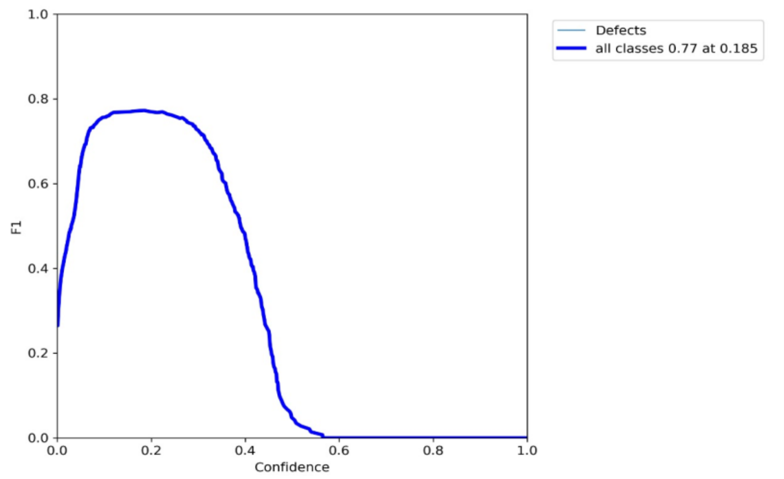
Validation Results

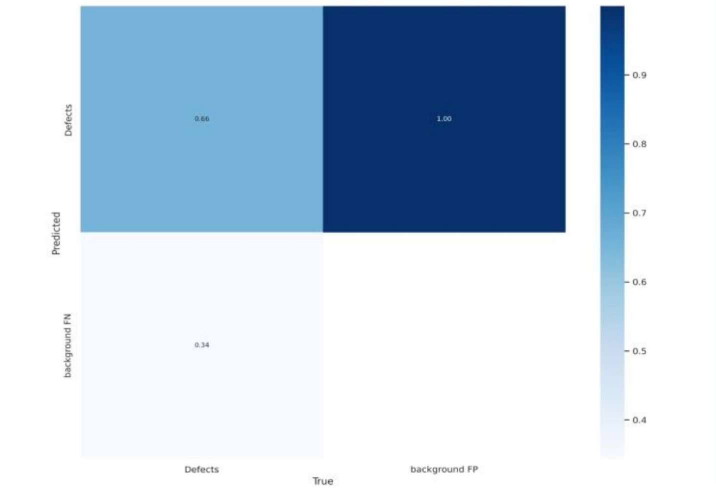


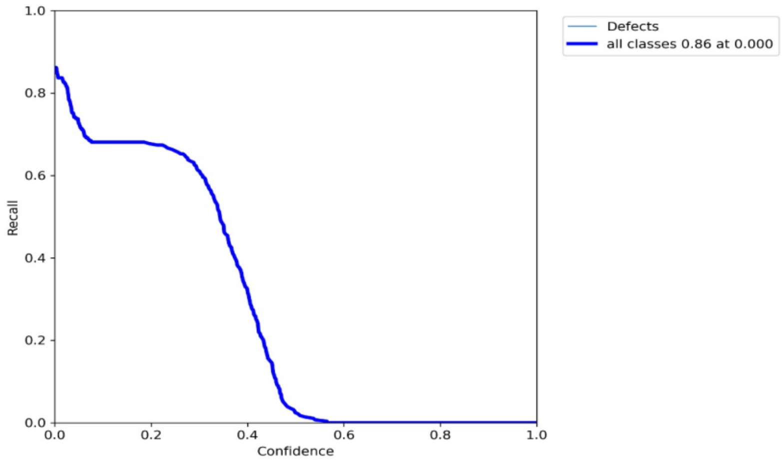


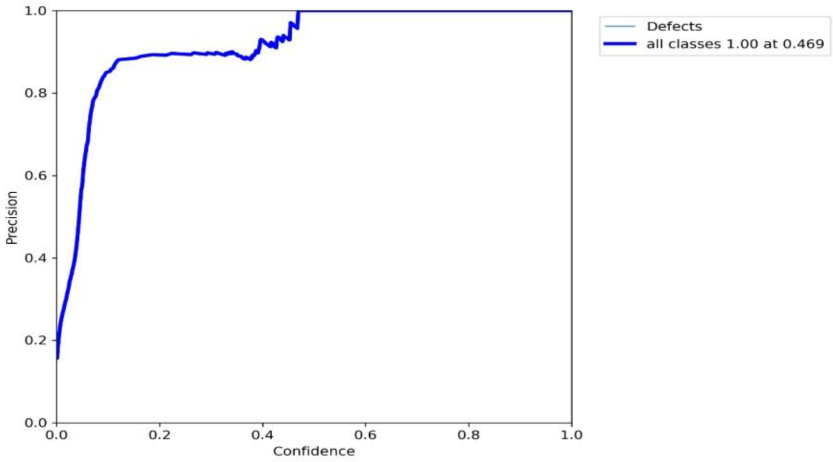


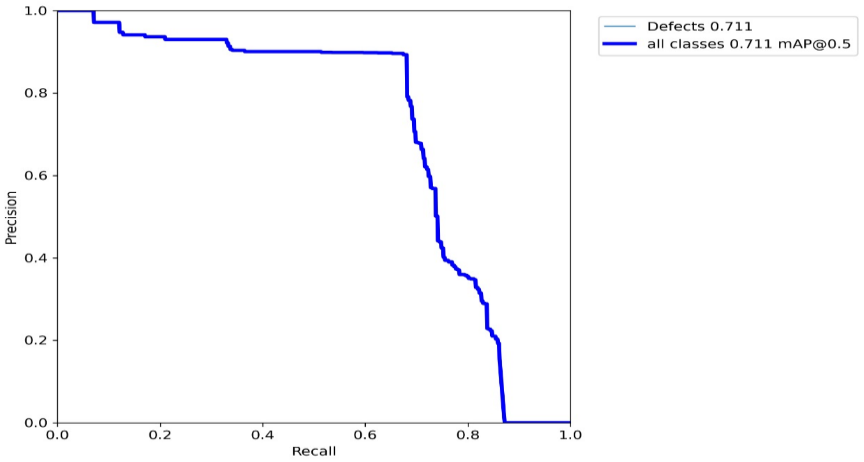
Evaluation Matrix











Thank You