Achira Labs Problem Set

By: Deva Dath Jagarlamudi

Part 1

Problem Statement:

4 shapes namely disc, gear, hexagon and square are given, and the goal is to generate images which have a scattered distribution of these shapes across the images.

Solution:

A python program by the name “gen_images.py” has been built for the cause, and it can be executed by the following command

python gen_images.py --input-folder *path_to_the_shapes_folder* --nout 1000 --out-dims 1024 --output-folder *path_to_the_folder_where_images_will_be_saved* --annotation-folder *path_to_the_folder_where_annotations_will_be_saved*

The environment needs to have Pillow and OpenCV libraries installed for the program to execute.

Details about the program are described below.

- The program takes in 6 input arguments which are
    - ***input-folder:*** Path to the folder where the shape files are located.
    - ***output-folder:*** Path to the folder where the generated images will be saved.
    - ***annotation-folder:*** Path to the folder where the annotations of the generated images are saved. These annotations will be in the YOLO format and can be used to train deep learning models.
    - ***nout:*** Number of output images that need to be generated
    - ***out-dims:*** The size or the dimension of the output images. Since the problem statement mentioned that the image size should be M x M, thereby

implying the shape of the image to be a square, only one integer value would suffice for this argument.

- ***shape-percent:*** This parameter describes the amount of area of the final image that will be covered by shapes. The image provided in the problem statement had 40 shapes in total and each shape was roughly 0.4% of the total area. Hence the area covered by shapes in the demo image is around 16% which is set as the default value for this argument. The user can feel free to change this value if need be.

- The problem statement states "*A given output image should contain k images of each type. Compute k based on the input image sizes and M. (Or make a reasonable assumption.)*". Since all the shapes should have equal number of count, and the shape-percent is fixed, k will be generated by the equation

  $$k = shape\ percent * \left(\frac{Area\ of\ the\ image}{\Sigma Area\ of\ individual\ shape}\right)$$

  All the shapes in this case have the same dimensions but the equation above is valid even if the shapes have different dimensions.

- The program executes by creating a blank black canvas and adding k number of every shape onto it. The locations of these shapes are randomized and kept track of, so that another shape will not be overlaid on top of it. The randomizer that is used to generate locations is used in such a way that it will never generate a location that will cause the shape to extend the canvas. Furthermore, these shapes will be scaled and rotated randomly within the thresholds provided before overlaying on the canvas.

- This entire program is built using fundamental mathematical logic. Hence, the possibility of the program malfunctioning is next to none, under the assumption that the logic is correct and takes all of the requirements into account.

The program has step wise explanations as comments for further explanation.

Part 2:

Problem Statement:

Train a deep learning model to detect the shapes and locations on these images.

Solution:

"Achira_Labs_Part_2.ipynb" contains the code used for the model training.

- I have generated 1000 images of size 1024 x 1024 in total, and split them into training, validation and test sets in the ratio 8:1:1.
- Due to the abundance of data available, and the uniformity of the objects of interest, I believed that a simpler and light model can do a good job when trained for sufficient epochs. Hence, I picked the YOLO version 8 nano model to train a multi class object detection model for this task.
- I started training the model with a batch size of 16 and intended it to train for 100 epochs. However, early stopping criteria stopped the model training as it saw no improvement after epoch number 59.
- Loss curves produced during training are shown below in Figure 1

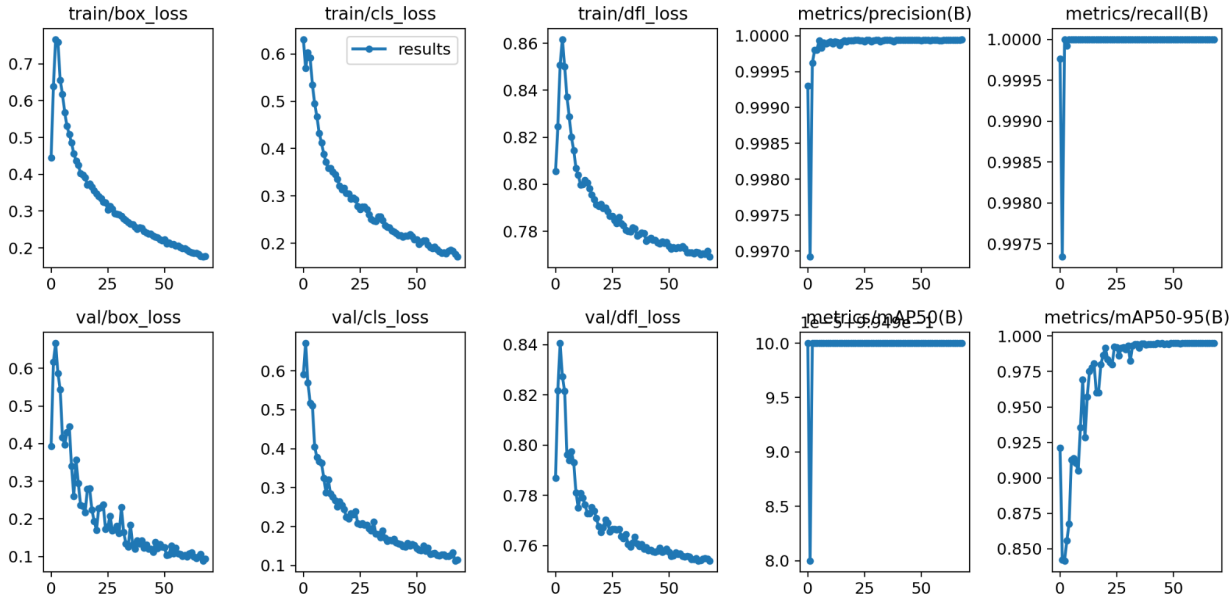


Figure 1: Loss curves and metrics during training.

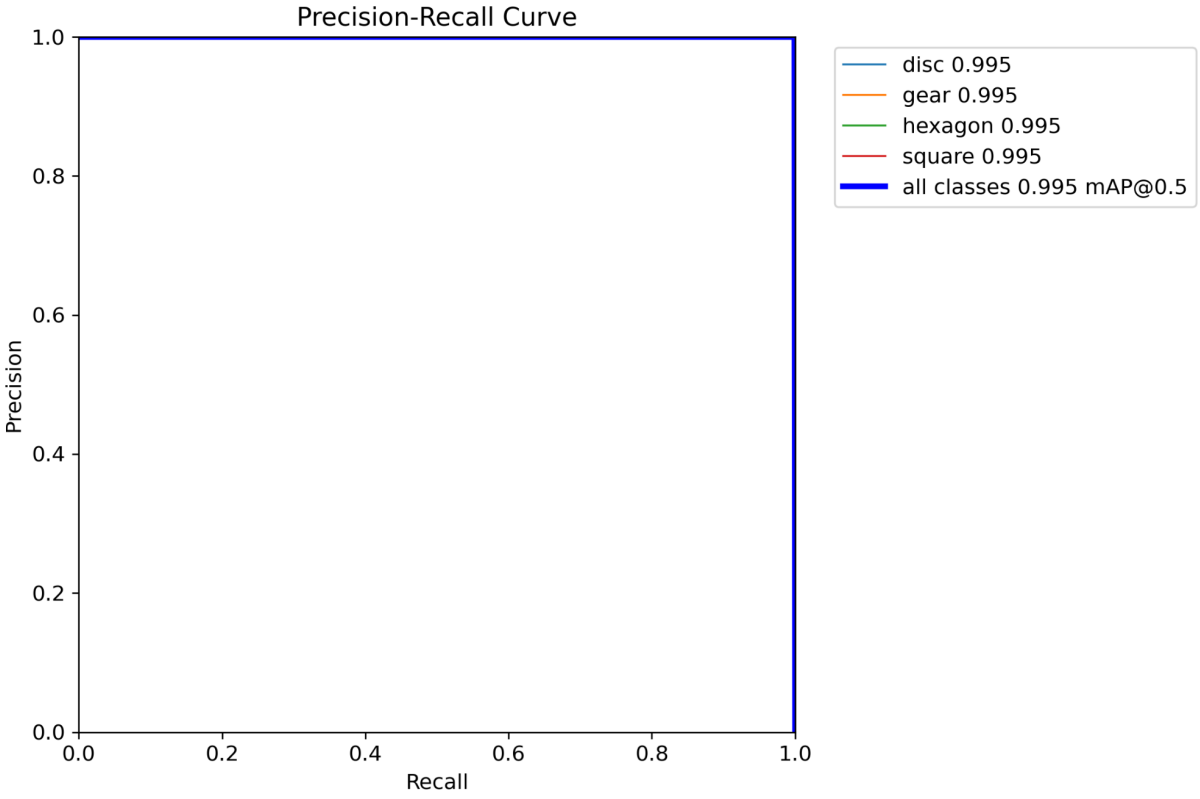


Figure 2: PR curve on validation set

Results:

"Result_analysis.py" is the program written to perform analysis on the inferences on the test set.

The environment needs to have numpy library installed for the program to execute.

- There are two parameters that control the metrics for evaluation. They are the confidence of the inference itself, and the intersection over union of the bounding box inferred by the model and that of the ground truth.
- I have calculated the true positives, false positives and false negatives for all the 4 classes at different confidence thresholds and with an IOU threshold of 95%, and the results are displayed in the table below.

Confidence = 0.9	Precision	Recall	F1-Score
Disc	1	1	1
Gear	1	1	1
Hexagon	1	1	1
Square	1	1	1

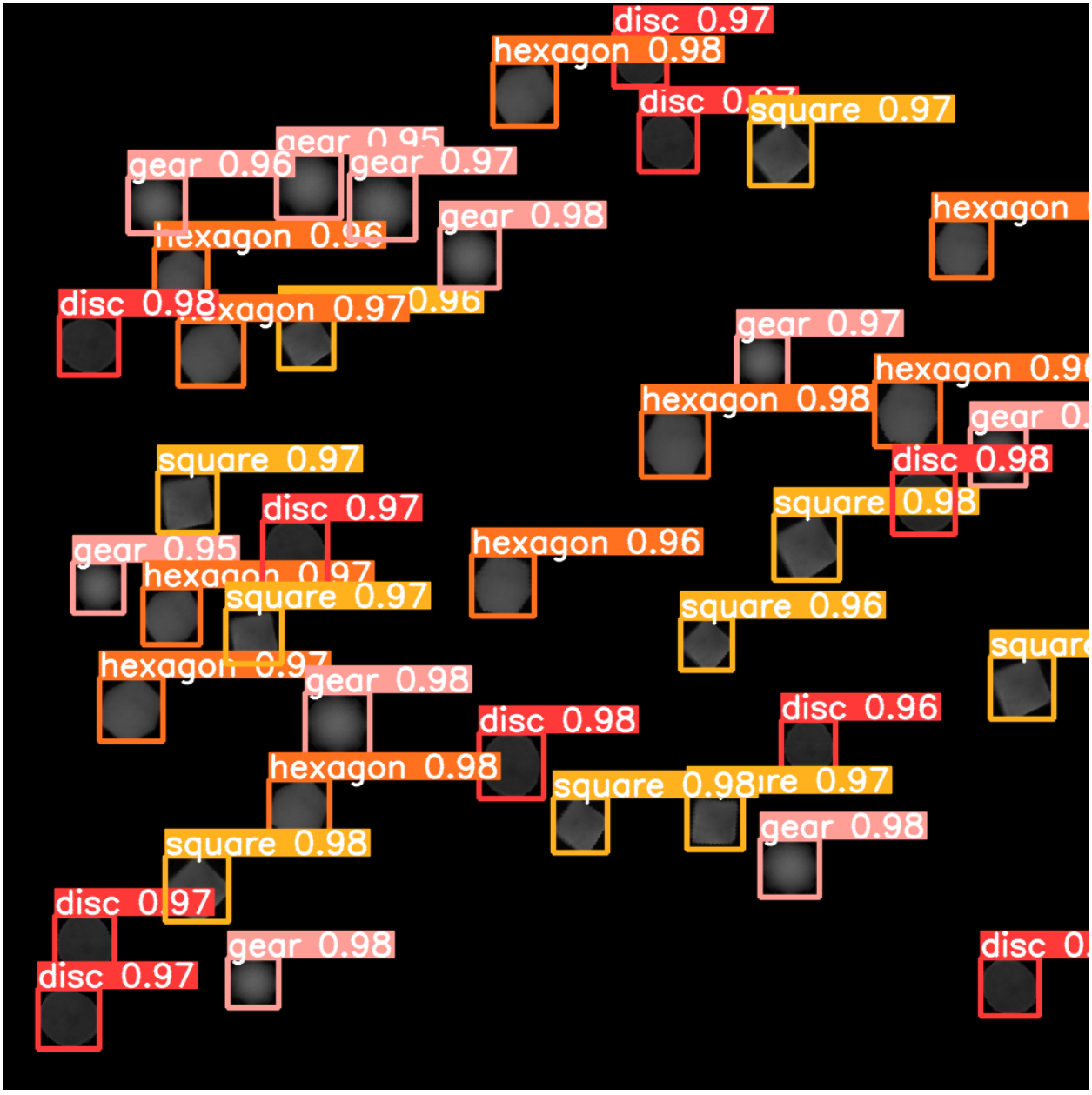
Confidence = 0.95	Precision	Recall	F1-Score
Disc	0.995	0.995	0.995
Gear	0.913	0.913	0.913
Hexagon	0.985	0.985	0.985
Square	0.977	0.977	0.977

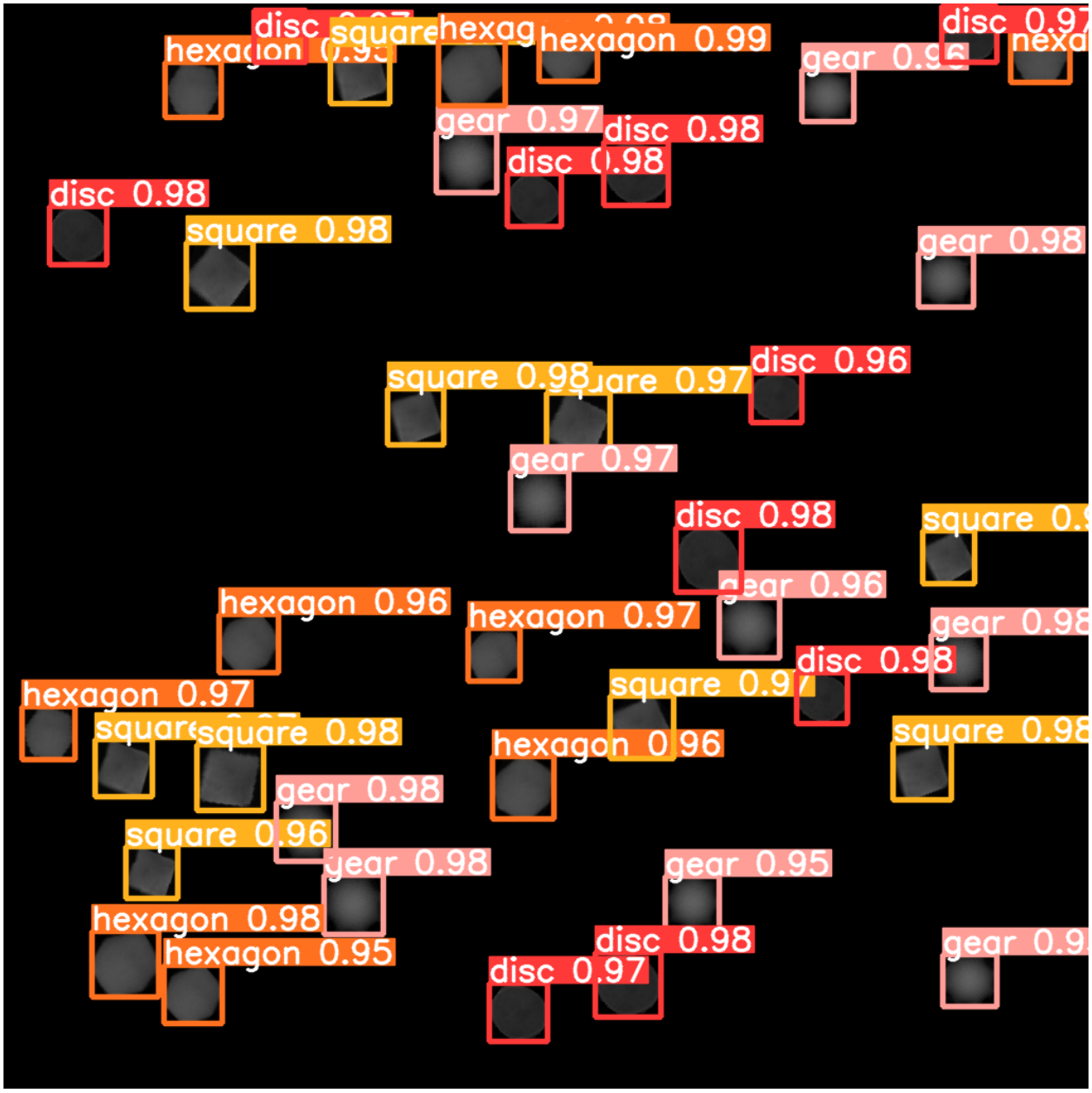
Confidence = 0.98	Precision	Recall	F1-Score
Disc	0.189	0.189	0.189
Gear	0.128	0.128	0.128
Hexagon	0.168	0.168	0.168
Square	0.184	0.184	0.184

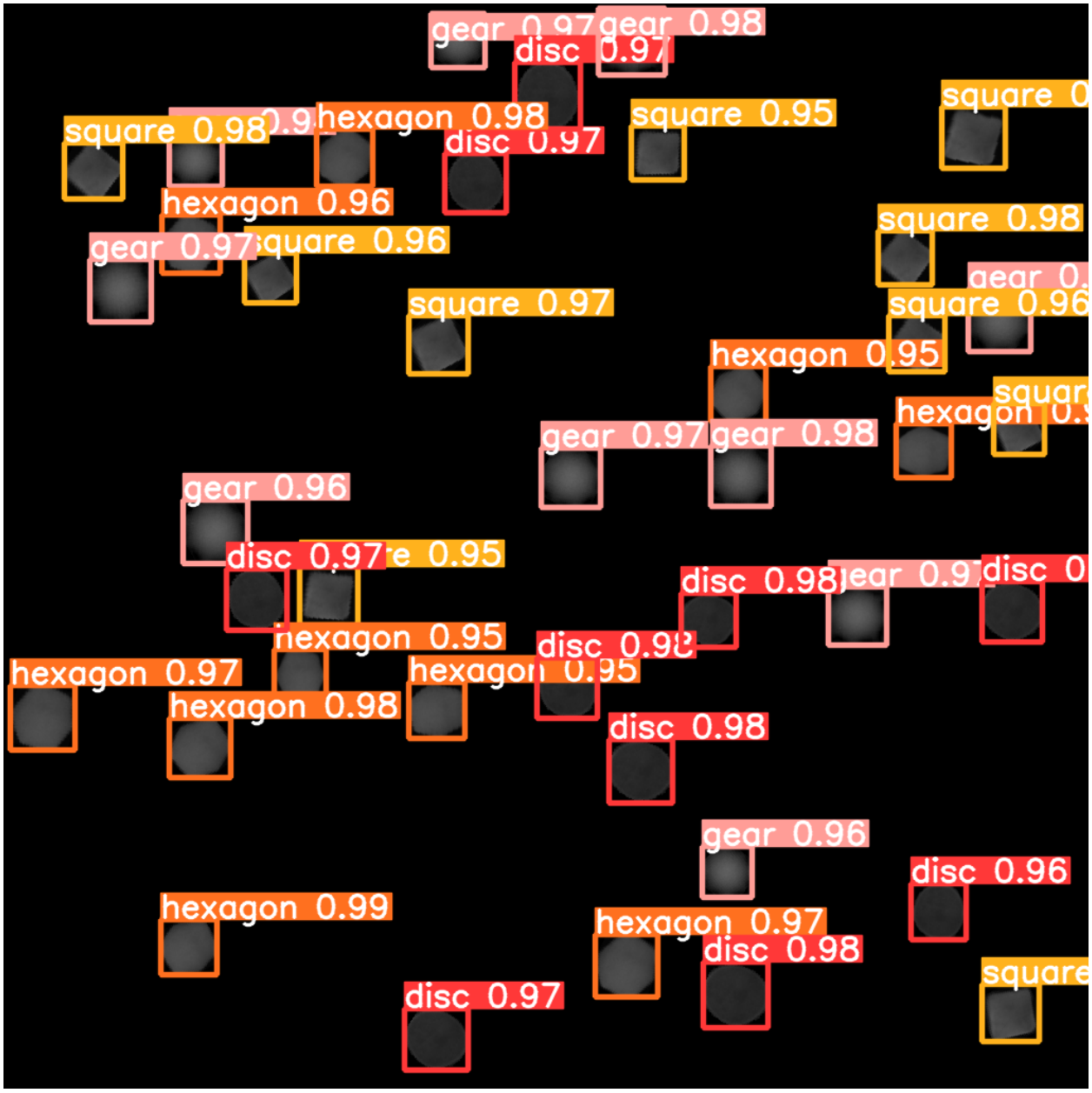
- We can notice that the precision and recall are the same value for every confidence threshold. This indicates that the model correctly identifies every shape. However, the confidence with which it identifies varies across the shape.
- It can also be observed that the detections on gear are less confident compared to the other classes. Since the number of gears in the training data are the same as the other classes, the reason for the poorer performance could be coming from the complexity of the shape compared to others. Increasing the number of gears in the training images could lead to a better performance. Also, a more complex model

could improve the results. This hypothesis is being made because of the fact that the current model has reached a saturation point in training and improving the epochs would not cause any improvement.

- Other metrics like Average precision and mean average precision could also be used to analyze the data. Since, there are no false detections, the precision recall curve would always be a straight line and these metrics would not add any insight to our analysis.
- A couple of test images and their inferences are added below for visualization.




hexagon 0.99
gear 0.97
disc 0.98
disc 0.98
disc 0.98
square 0.98
gear 0.98
square 0.98
disc 0.96
gear 0.97
disc 0.98
hexagon 0.96
hexagon 0.97
gear 0.96
gear 0.98
disc 0.98
hexagon 0.97
square 0.98
hexagon 0.96
square 0.98
gear 0.98
square 0.96
gear 0.98
gear 0.95
hexagon 0.98
hexagon 0.95
disc 0.98
disc 0.97


gear 0.97
gear 0.98
disc 0.97
square 0.98
hexagon 0.98
square 0.95
disc 0.97
hexagon 0.96
gear 0.97
square 0.96
square 0.97
square 0.98
square 0.96
hexagon 0.95
gear 0.97
gear 0.98
gear 0.96
disc 0.97
disc 0.98
hexagon 0.95
hexagon 0.97
hexagon 0.98
disc 0.98
gear 0.96
disc 0.96
hexagon 0.99
hexagon 0.97
disc 0.98
disc 0.97